T.Y.B.Sc. (IT)

SEMESTER - V (CBCS)

ENTERPRISE JAVA

SUBJECT CODE : USIT506

Prof. Suhas Pednekar
Vice Chancellor
University of Mumbai, Mumbai.

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai.

Prof. Prakash Mahanwar
Director
IDOL, University of Mumbai.

Programe Co-ordinator : **Shri. Mandar L. Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai – 400098.

Course Co-ordinator : **Ms. Gouri S. Sawant**
Assistant Professor B.Sc.IT, IDOL,
University of Mumbai- 400098.

Editor : **Dr Vinayak Pujari**
Assistant Professor,
D. Y. Patil Engineering College, Kolhapur.

Course Writers : **Mr. Umesh Waghmare**
Assistant Professor,
MKSSS's K.B. Joshi Institute of Information technology, Pune.

: **Ms. Sujata Rizal**
Assistant Professor, SM Shetty College Powai, Hiranandani Powai.

: **Ms. Sherilyn Kevin Kuruthukulangara**
Assistant Professor, Thakur College of Science and Commerce,
Thakur Village, Kandivali.

: **Ms. Ifrah Rizwan Kampoo**
Assistant Professor, D.G Ruparel College.

: **Ms. Fatima Shaikh**
Assistant Professor, Jai Hind College, Churchgate.

July 2022, Print I

Published by
Director
Institute of Distance and Open Learning,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP COMPOSED AND PRINTED BY
Mumbai University Press
Vidyanagari, Santacruz (E), Mumbai - 400098.

CONTENTS

*****

Syllabus

B. Sc. (InformationTechnology)		**Semester – V**	
Course Name: Enterprise Java		**Course Code: USIT506 (Elective II)**	
Periods per week (1 Period is 50 minutes)		**5**	
Credits		**2**	
		Hours	**Marks**
Evaluation System	**Theory Examination**	**2½**	**75**
	Internal	**-**	**25**

Unit	Details	Lectures
I	**Understanding Java EE:** What is an Enterprise Application? What is java enterprise edition? Java EE Technologies, Java EE evolution, Glassfish server **Java EE Architecture, Server and Containers:** Types of System Architecture, Java EE Server, Java EE Containers. **Introduction to Java Servlets:** The Need for Dynamic Content, Java Servlet Technology, Why Servlets? What can Servlets do? **Servlet API and Lifecycle:** Java Servlet API, The Servlet Skeleton, The Servlet Life Cycle, A Simple Welcome Servlet **Working with Servlets:** Getting Started, Using Annotations Instead of Deployment Descriptor. **Working with Databases:** What Is JDBC? JDBC Architecture, Accessing Database, The Servlet GUI and Database Example.	**12**
II	**Request Dispatcher:** Resquestdispatcher Interface, Methods of Requestdispatcher, Requestdispatcher Application. **COOKIES:** Kinds of Cookies, Where Cookies Are Used? Creating Cookies Using Servlet, Dynamically Changing the Colors of A Page **SESSION:** What Are Sessions? Lifecycle of Http Session, Session Tracking With Servlet API, A Servlet Session Example **Working with Files:** Uploading Files, Creating an Upload File Application, Downloading Files, Creating a Download File Application. **Working with Non-Blocking I/O:** Creating a Non-Blocking Read Application, Creating The Web Application, Creating Java Class, Creating Servlets, Retrieving The File, Creating index.jsp	**12**
III	**Introduction To Java Server Pages:** Why use Java Server Pages? Disadvantages Of JSP, JSP v\s Servlets, Life Cycle of a JSP Page, How does a JSP function? How does JSP execute? About Java Server Pages	**12**

	Getting Started With Java Server Pages: Comments, JSP Document, JSP Elements, JSP GUI Example. **Action Elements:** Including other Files, Forwarding JSP Page to Another Page, Passing Parameters for other Actions, Loading a Javabean. **Implicit Objects, Scope and El Expressions:** Implicit Objects, Character Quoting Conventions, Unified Expression Language [Unified El], Expression Language.	
IV	**Introduction To Enterprise Javabeans:** Enterprise Bean Architecture, Benefits of Enterprise Bean, Types of Enterprise Bean, Accessing Enterprise Beans, Enterprise Bean Application, Packaging Enterprise Beans **Working with Session Beans:** When to use Session Beans? Types of Session Beans, Remote and Local Interfaces, Accessing Interfaces, Lifecycle of Enterprise Beans, Packaging Enterprise Beans, Example of Stateful Session Bean, Example of Stateless Session Bean, Example of Singleton Session Beans. **Working with Message Driven Beans:** Lifecycle of a Message Driven Bean, Uses of Message Driven Beans, The Message Driven Beans Example. **Interceptors:** Request and Interceptor, Defining An Interceptor, AroundInvoke Method, Applying Interceptor, Adding An Interceptor To An Enterprise Bean, Build and Run the Web Application. **Java Naming and Directory Interface**: What is Naming Service? What is Directory Service? What is Java Naming and Directory interface? Basic Lookup, JNDI Namespace in Java EE, Resources and JNDI, Datasource Resource Definition in Java EE.	**12**
V	**Persistence, Object/Relational Mapping And JPA:** What is Persistence? Persistence in Java, Current Persistence Standards in Java, Why another Persistence Standards? Object/Relational Mapping, **Introduction to Java Persistence API:** The Java Persistence API, JPA, ORM, Database and the Application, Architecture of JPA, How JPA Works? JPA Specifications. **Writing JPA Application:** Application Requirement Specifications, Software Requirements, The Application Development Approach, Creating Database and Tables in Mysql, creating a Web Application, Adding the Required Library Files, creating a Javabean Class, Creating Persistence Unit [Persistence.Xml], Creating JSPS, The JPA Application Structure, Running the JPA Application. **Introduction to Hibernate:** What is Hibernate? Why Hibernate? Hibernate, Database and The Application, Components of Hibernate, Architecture of Hibernate, How Hibernate Works? **Writing Hibernate Application:** Application Requirement Specifications, Software Requirements, The Application	**12**

	Development Approach, Creating Database and Tables in Mysql, creating a Web Application, Adding the Required Library Files, creating a Javabean Class, Creating Hibernate Configuration File, Adding a Mapping Class, Creating JSPS, Running The Hibernate Application.	

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Java EE 7 For Beginners	Sharanam Shah, Vaishali Shah	SPD	First	2017
2.	Java EE 8 Cookbook: Build reliableapplications with the most robust and mature technology for enterprise development	Elder Moraes	Packt	First	2018
3.	Advanced Java Programming	Uttam Kumar Roy	Oxford Press		2015

*****

UNIT I

1

UNDERSTANDING JAVA EE

Unit Structure

1.1 OBJECTIVES

1) Java EE Provides More Flexible Technology.

2) Java EE is a collection of API where students/professionals are able to design server side applications.

3) Students/professionals are able to originally design and develop applications in a thin-client-tiered environment.

4) Java EE applications are hosted on application servers (WebSphere, GlassFish, WildFly, Apache Tomcat etc….)

5) When we design & develop Java EE applications to use popular design patterns [MVC].

6) Java EE support for Enhanced Extensibility.

7) Java EE provides a powerful API for Strong and Dynamic Web Programming.

1.2 INTRODUCTION TO JAVA EE

The Java EE stands for Java Enterprise Edition, which was earlier known as J2EE and is currently known as Jakarta EE in JDK latest version. The Java EE provides a platform for Student/professionals with enterprise features including- distributed computing and web services. Java EE types of applications are usually run on microservers or application servers. Areas where Java EE is used are e-commerce, accounting, banking information systems.

1.3 BASIC CONCEPTS RELATED TO JAVA EE

1.3.1 Specification of Java EE:

Java EE has various specifications which are useful in Designing & Developing applications and web pages, reading and writing from the database in various transactional ways and also managing distributed queues in network communication. Java EE API such as Enterprise JavaBeans, connectors, Servlets, Java Server Pages and various web services.

1.3.2 What is an Enterprise Application?:

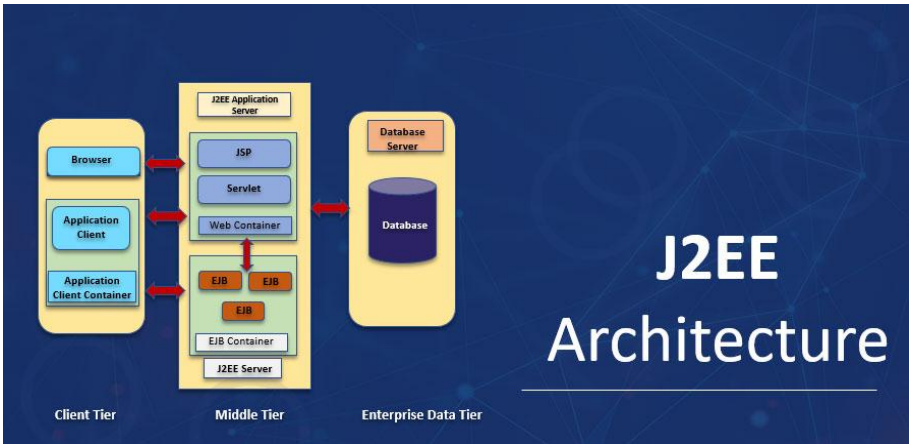
In Java EE we can design the Enterprise /Business/Commercial level Applications. Java EA is a large software system platform designed to operate in a corporate / business or government level environment. Java EE Applications are complex,scalable, component-based, distributed and mission critical.

Java EA consists of a large number of programs with shared business applications and organizational modeling utilities designed for unparalleled functionalities.

Enterprise Application's software is a critical component of any computer-based information system. Enterprise level Application software ultimately enhances their efficiency and productivity through various levels of functionality in business.

1.3.3 What is java enterprise edition?:

Java EE frameworks provide common design patterns rarely used in development of Java EE Application, and add into reusable class libraries. These class libraries are implemented to access the database for various processing ways- security, transaction processing, screen layout, data validation,object construction, caching, and other development/Programming related tasks so that Java EE developers can focus on the purely business logic .



1.3.3 A Figure shows Java EE Architecture.

Above Figure indicate flow of java EE applications first layer as an client where they can be send & receive responses from server side(Java Enterprise Edition Programs) ,second layer is an middle Tier [Combination of java's web enabled API's] where it can be acting as a Controller of client & Database .

Java Enterprise Edition (Java EE) technology provides services to enterprise applications using a multi-layer architecture. Java EE applications/projects are web-enabled and Java based, which means they may be written once and deployed on any container supporting the powerful execution in Java EE environment.

Following are The three most common Java EE design patterns also known as MVC model focus on:

1) business(Model)

2) presentation(View)

3) persistence layers(Controllers).

1.3.3.1 Business [Model]:

Model/Business Logic represents an object or JAVA POJO[purely old java object] carrying data. It can also have logic to update controllers if its data changes vary to Action.Model represents the state of the application i.e. data. Model containing business logic.

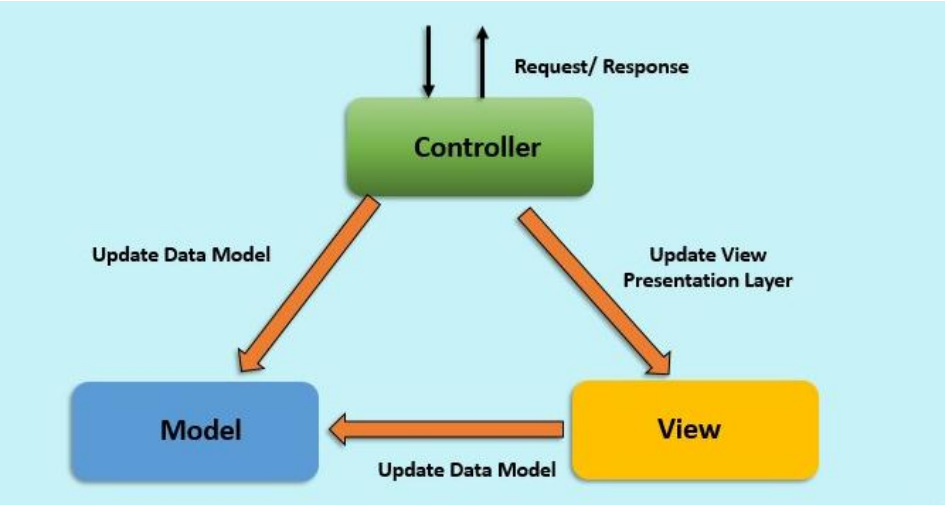
1.3.3.2 Presentation [View]:

View represents the visualization of the data and represents on the user screen that model contains processed data/result oriented. View is an represents the presentation i.e. UI(User Interface)/front end..

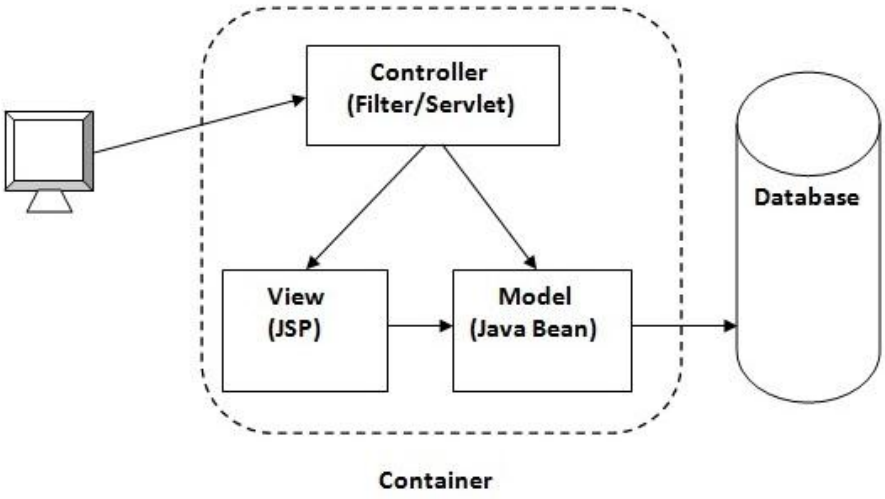
1.3.3.3 Persistence layers [Controllers]:

Controllers are handled on both model and view. It controls the data flow into the model object and updates the view whenever data changes/requirements change . It keeps view and model separate.

In MVC Controller acts as an interface/Intermediate between View and Model. Controller Handling all the incoming requests from Model & View as per request redirect to automatically as per request & response preference .



1.3.3 A Figure shows Java EE design patterns Communication.



1.3.3 B Figure shows Java EE design patterns Communication[MVC].

1.3.4 JAVA EE TECHNOLOGIES.

Java EE is actually a collection of various technologies and API[Application Programming Interface] for the Java EE platform designed & Developed to support "Enterprise" levels Applications which can generally be classed as large-scale, Multi-tier , distributed, transactional and highly-available applications designed to support mission-critical/handling critical processes in business requirements.

- **Following are list of java EE Technologies:**

1. JDBC:

JDBC stands for Java Database Connectivity.JDBC is a Java API used to connect and execute the sql query with their relevant database. JDBC can

handle backend & frontend transactions the JDBC API uses JDBC drivers to connect with the database/backend to front end or java application .

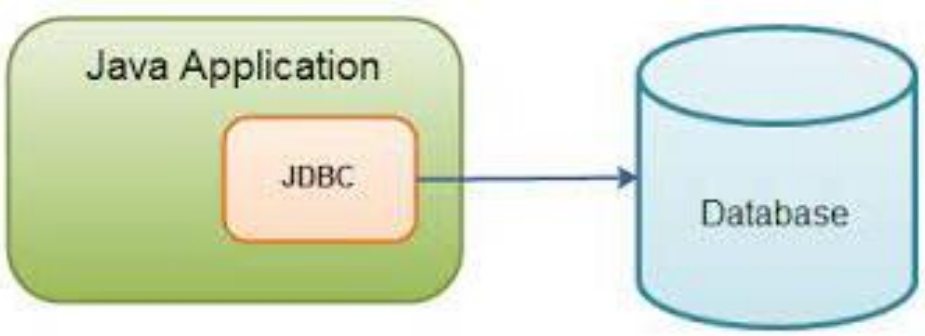


Fig: shows JDBC for Database connectivity with java application.

There are four types of JDBC drivers:

A) JDBC-ODBC Bridge Driver.

B) Native Driver.

C) Network Protocol Driver.

D) Thin Driver.

2. JNDI:

Java Naming and Directory Interface is the name of the interface in the Java programming language. It is an API(Application Program Interface) that works with servers for fetching files from a database using naming conventions. The naming convention(Name of class,interface etc….) can be a single phrase or a word. It can also be incorporated in a socket to implement socket programming, using servers transferring data files or flat files in a project.

JNDI is used in web pages in browsers/clients where there are instances of many directories/files. JNDI provides users in Java related various facilities to search objects in Java using the Java coding language.

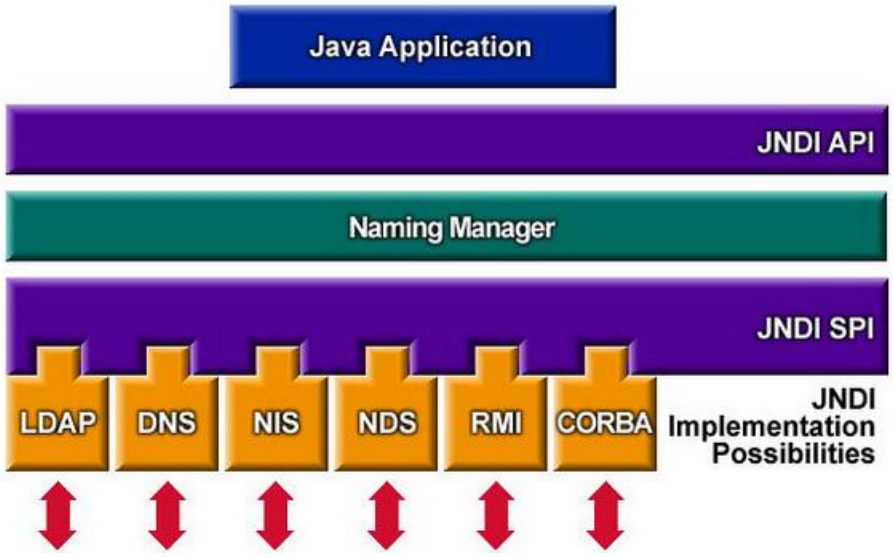


Fig: shows Architecture of JNDI.

3. EJB:

EJB stands for Enterprise Java Bean. It is a specification provided by Sun Microsystems/Oracle corporation or IBM to develop robust,secure and scalable distributed applications using Java EE.

To run the Java EJB applications, you should install an application server(EJB Container) such as Weblogic,Jboss,Websphere, Glassfish etc.

It performs Following Transaction of EJB application:

1. life cycle management.
2. security.
3. transaction management.
4. object pooling.

- **Use of EJB:**

1) EJB Application needs Remote Access. In other words, it is distributed.

2) EJB Application needs to be scalable. Java EJB applications support load balancing, clustering and fail-over.

3) EJB Application needs encapsulated business logic. Java EJB application is separated from presentation and persistent layer.

- **Types of Enterprise Java Bean:**

There are 3 types of enterprise beans in java.

1) Session Bean:

Session bean contains business logic that can be invoked by local, remote or web service clients/browsers.

2) Message Driven Bean:

Message Driven Bean also Like Session Bean, it contains the business logic but it is invoked by passing messages/message communication.

3) Entity Bean:

IN Entity Bean It encapsulates the state that can be persisted in the database. Currently it is known as or it is replaced with JPA (Java Persistence API).

4) RMI:

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed applications in java. The RMI allows an object to invoke methods on an object running in another JVM.

The RMI provides remote communication between the applications using two objects stub and skeleton.

- **Understanding stub and skeleton:**

  In client server communication RMI uses stub and skeleton objects for communication with the remote object for both objects are accessing purpose .

Note: A remote object is an object whose method can be invoked from another JVM.

Let's understand the stub and skeleton objects:

1) stub:

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It implement a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshalls) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result from the remote JVM.
4. Stub reads (unmarshalls) the return value or exception.
5. Stub finally returns the value to the caller .

2) skeleton:

The skeleton is an instance/object, and acts as an interface for the server side object. All the incoming requests are routed through skeleton. When the skeleton receives the incoming request, it does the following operations:

1. Skeleton reads the parameter for the remote method.
2. Skeleton invokes the method on the actual remote object as per communication.
3. Skeleton writes and transmits (marshalls) the result to the caller.

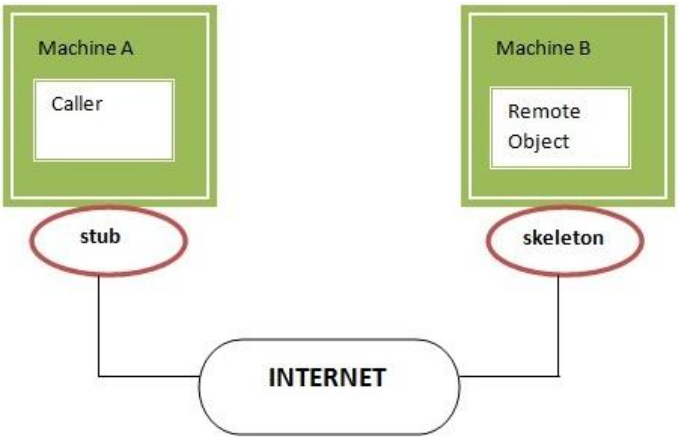


Fig: Shows communication of stub & skeleton.

5) JSP:

In Java EE uses JSP technology to create web applications just like Servlet technology. It can be treated as an enhancement to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

In the JSP page containing HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate the code of designing and development. It provides some additional features such as implicit objects,Expression Language, Custom Tags, etc.

6) Java servlets:

Servlet technology is used to create a web application (Executes/resides on server side and generates a dynamic web page & response to browser).

Servlet is a technology that would be robust and scalable because of the Java language. Before Servlet, to use one common interface CGI (Common Gateway Interface) scripting language was common as a server-side programming language.

- What is a Servlet?

  Following are the Servlet can be described in many ways
- Servlet is a technology which is used to create and design a dynamic web application.
- Servlet consists of an API that provides various interfaces and classes.
- Servlet is an interface that must be implemented for creating any Servlet class.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- In java EE Servlet is a web component that is deployed on the server to create a dynamic web page.

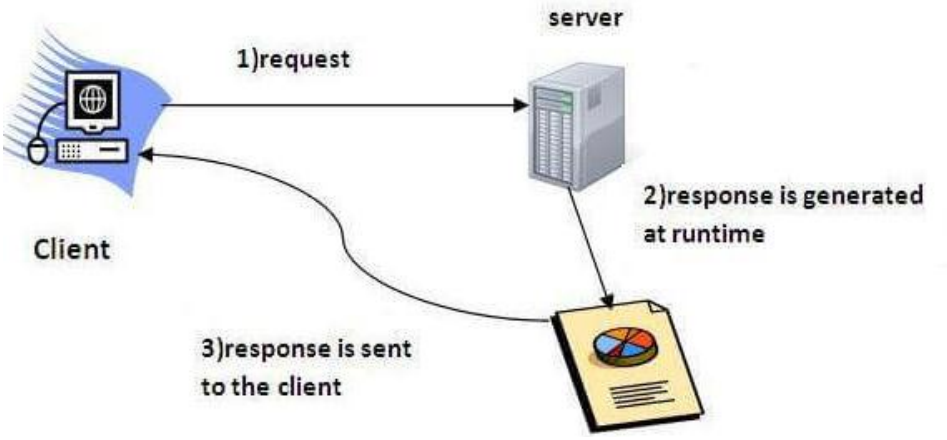


Fig: Using servlet interface implementing client server communication.

7) XML:

XML (Extensible Markup Language) is a very popular simple text-based/marked up language that can be used as an interface of communication between different applications. It is considered as a standard technique to transport and store data. JAVA provides excellent support and a rich set of libraries to modify, parse and inquire XML documents.

8) JMS:

MS (Java Message Service) is an API that provides the facility to create, send and read messages.JMS is also known as a messaging service. It provides loosely coupled, reliable and asynchronous communication.

Use of Java Messaging Service:

- In JMS Service messaging is a technique to communicate applications or software components.
- JMS service is mainly used to send and receive messages from one application to another.
- Generally, the user sends a message to the application. But, if we want to send messages from one application to another, we need to use the JMS API.
- Consider a scenario, one application A is running in INDIA and another application B is running in the UK. To send a message from A application to B, we need to use JMS.

9) Java IDL:

In Java EE IDL stands for (Interface Definition Language) is a technology for distributed objects-that is, objects can communicate/interact on different platforms across a network. Java IDL is similar to RMI (Remote Method Invocation), But in IDL which supports distributed objects written entirely in the Java programming language.

10) JTS:

In Java The JTS stands for Java Topology Suite (JTS) is an open source Java API/ library that provides an object model for planar geometry together with a set of fundamental geometric functions.

JTS is specifically designed to be used as a core component of vector-based geomatics software such as GIS-Geographical Information Systems. Now JTS also be used as a general-purpose library providing algorithms in computational geometry in java applications.

11) JTA:

The Java Transaction API (JTA) allows for applications to perform distributed transactions, that is, transactions that access and update data on two or more networked computer resources. The JTA specifies standard Java interfaces between a transaction manager and the other network component.

12) Java Mail:

In Java EE Technology JavaMail is an API that is used to compose, write and read electronic messages i.e (to send & receive emails).

These JavaMail API provides services related to protocol-independent and platform-independent frameworks for sending and receiving mail through the network.

The javax.mail and javax.mail.activation packages contain the core classes of JavaMail API.

13) JAF:

JFA (Java Framework Architecture) is an API included in the Software Development Kit(SDK) for designing software applications in Indian languages. It consists of a set of Java components ,Interfaces and supporting classes which enable the creation of content in Indian scripts(Language). The scripts supported are Devanagari,Oriya , Kannada,Assamese, Bengali, Tamil, Gujarati, Punjabi, Malayalam,Telugu and English.

1.3.5 Java EE evolution:

Java EE Formerly called as a J2EE, the first version of Java EE platform was officially released in December 1999 with 10 specifications. Among these specifications, there were Servlets and JavaServer Pages (JSP) for data presentation, Enterprise JavaBeans (EJB) for the management of persistent data and secure transactions, remote access to business services through RMI-IIOP protocol (Remote Method Invocation over Internet Inter-ORB Protocol), and the JMS (Java Message Service) specification, which was used to send messages.

More effort and many contributions, early versions of Java EE were too much complex and difficult to implement because JEE provides more

specification than J2EE. This leading to much criticism/complexity of code and caused the rise of competing frameworks such as Spring Framework.

1.3.6 Glassfish Server:

GlassFish is an application server/web server started by Sun Microsystems for Java Enterprise Edition which is now owned & managed by Oracle corporation. It is a free software that is released under two free software licenses. The one is a common development and distribution license and the other is GNU general public license. Sun Microsystems launched the project on June 6, 2005.

Requirement:

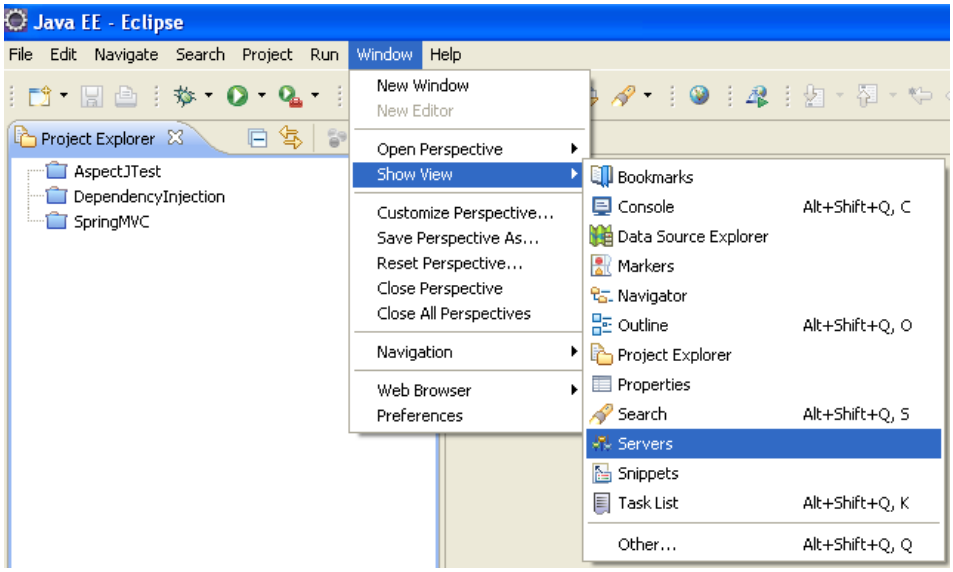
1. JDK must be installed on the system.
2. Windows OS
3. Login as an admin.

Steps to Install Glassfish web server in Eclipse:

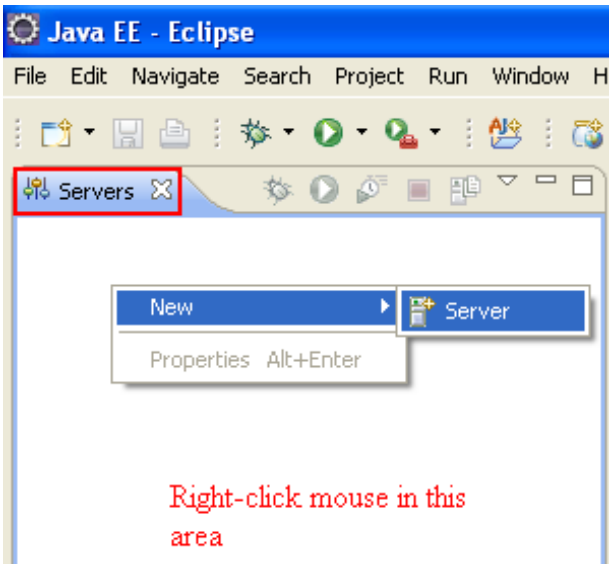
1. Open Eclipse.
2. Go to Help > Eclipse Marketplace.
3. Search for GlassFish.
4. Click on Install.
5. Glassfish Tools and oracle. eclipse. tools. glassfish is selected. Click Confirm.

Steps to configure Glassfish server with eclipse:

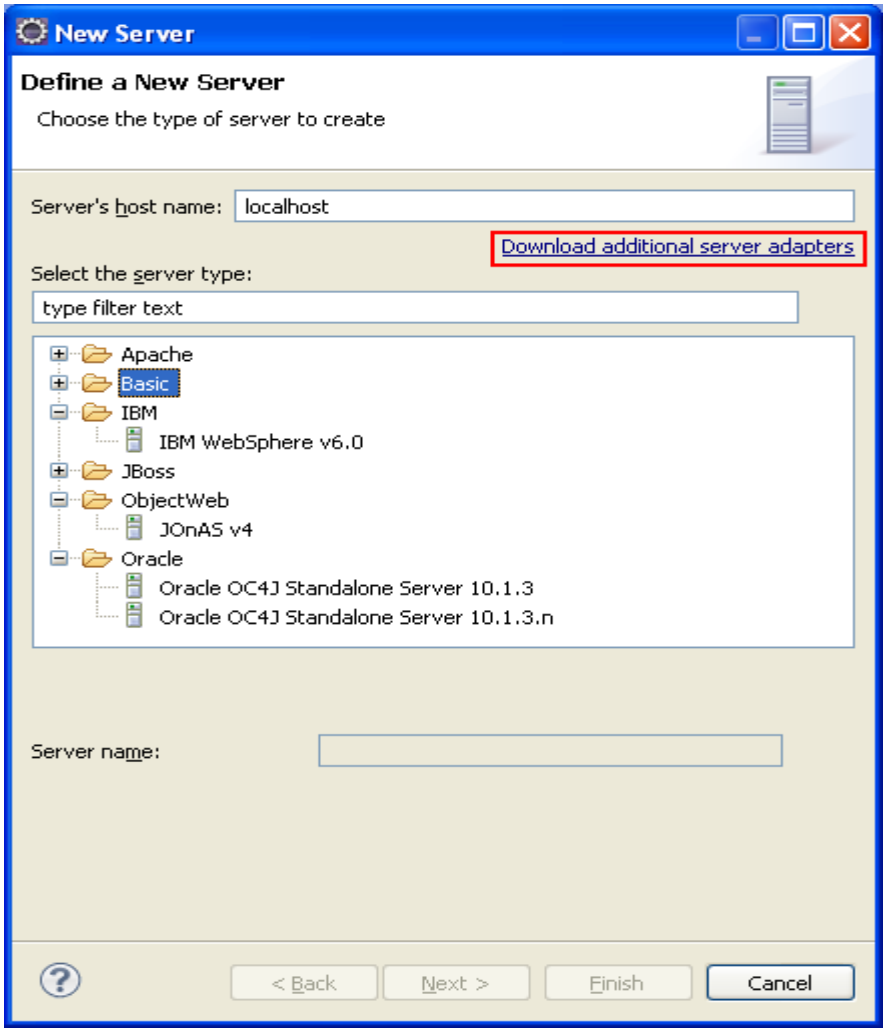
Step-1 launch Eclipse IDE. Click Window menu > select Show View sub menu > select Servers.



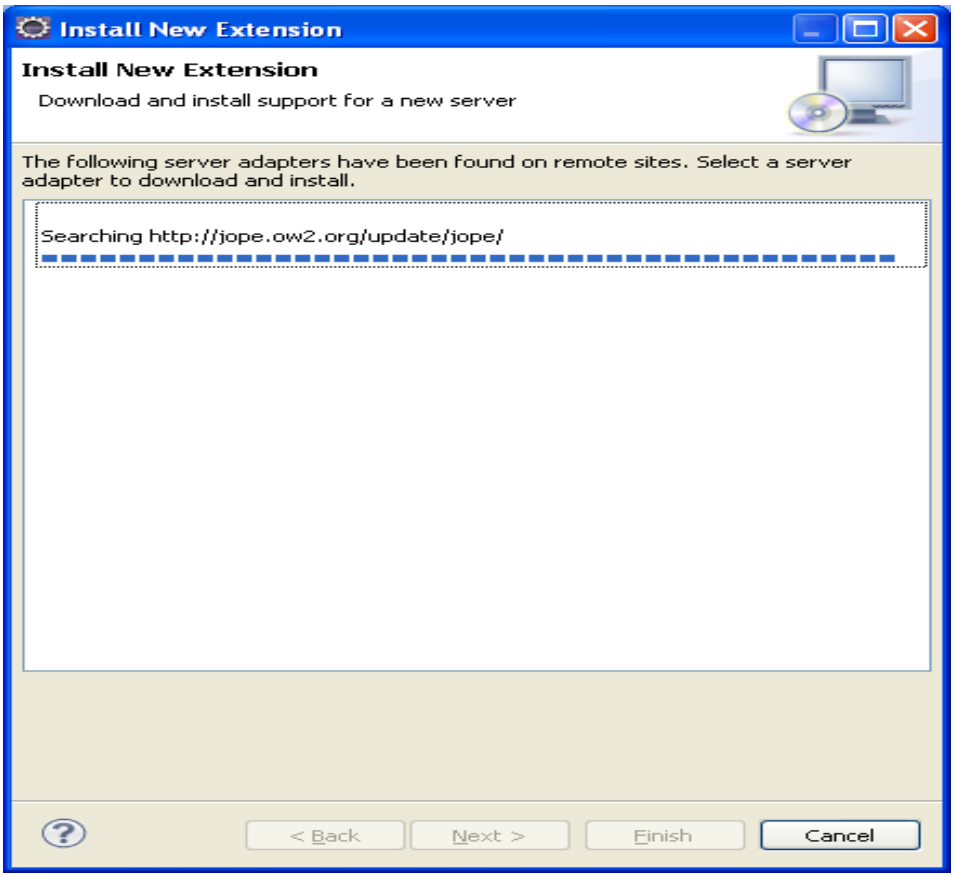
Or, right click mouse anywhere in the Servers page > select New menu > select Server sub menu.



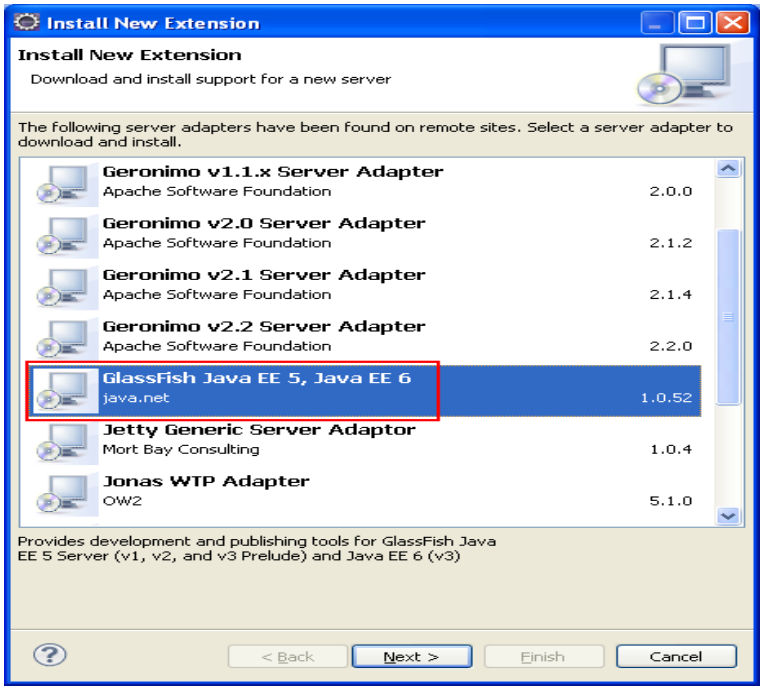
Step-2: in the New Server page, notice in the middle box which list the available servers. We are going to download the Glassfish server which is not in the list. Type the Server's host name (if needed). In this case we are using 'localhost'. Then, click the Download additional server adapters link.



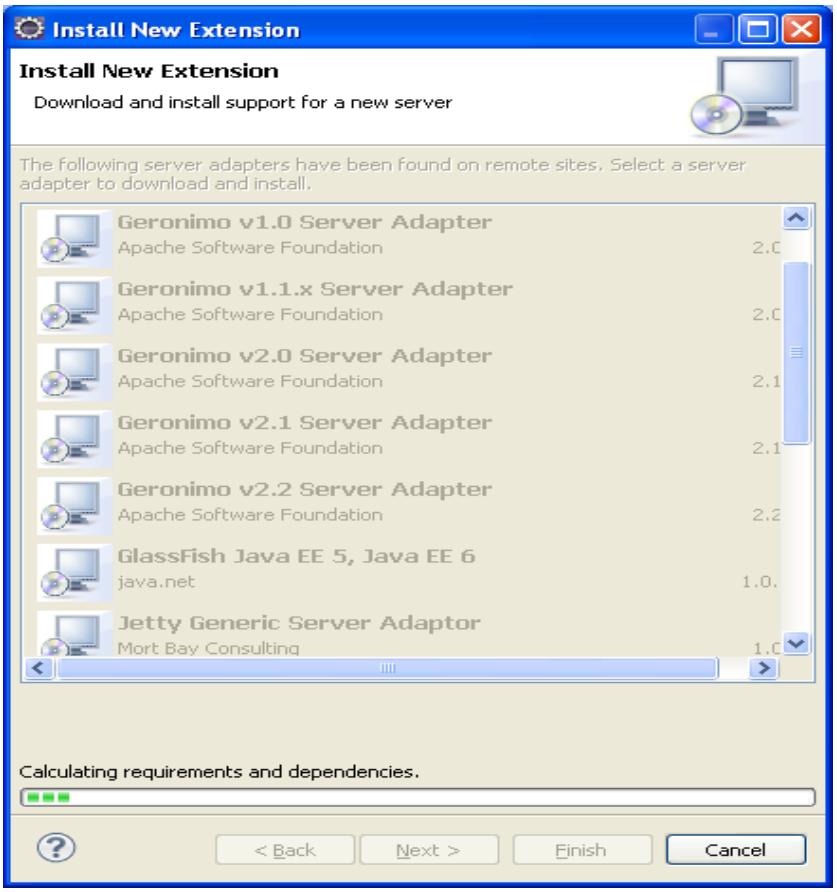
The Install New Extension wizard will begin, searching available server adapters which are available from the update servers.



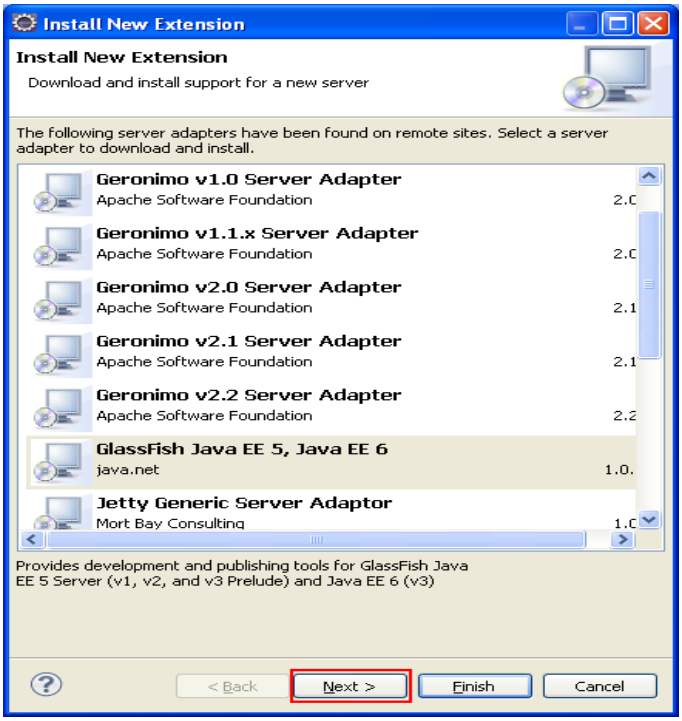
Step-3 from the list of available server adapter, select Glassfish server and click Next.



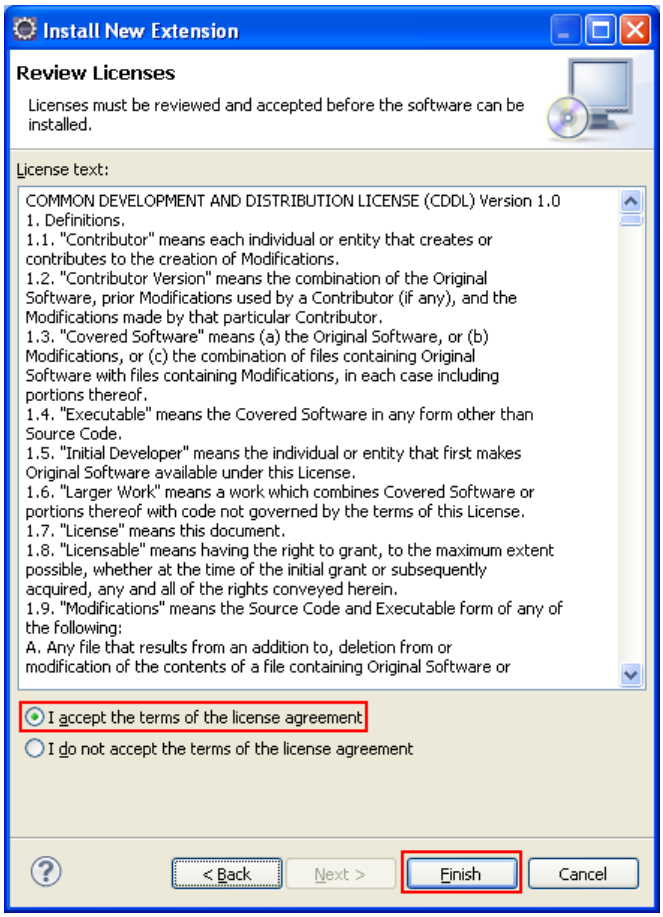
The download requirements and dependencies process will begin.



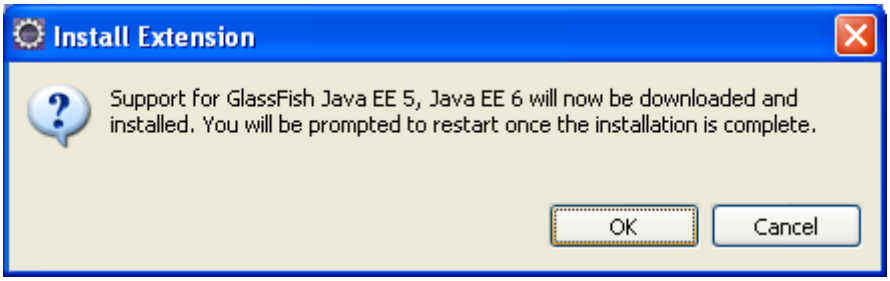
After the download requirement and dependencies process is completed, click Next.



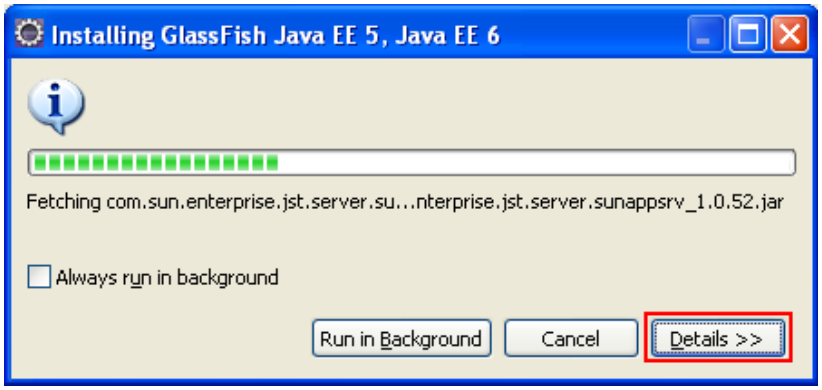
Step-4 Accept the license agreement and click Finish.



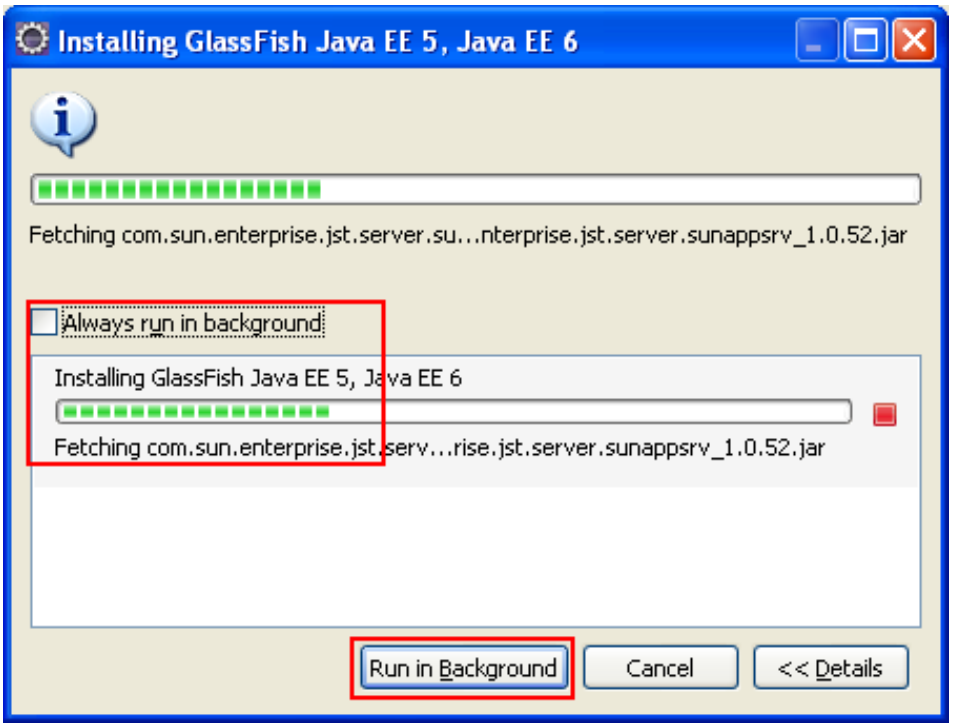
Step -5 Click OK for the server adapter download confirmation prompt window.



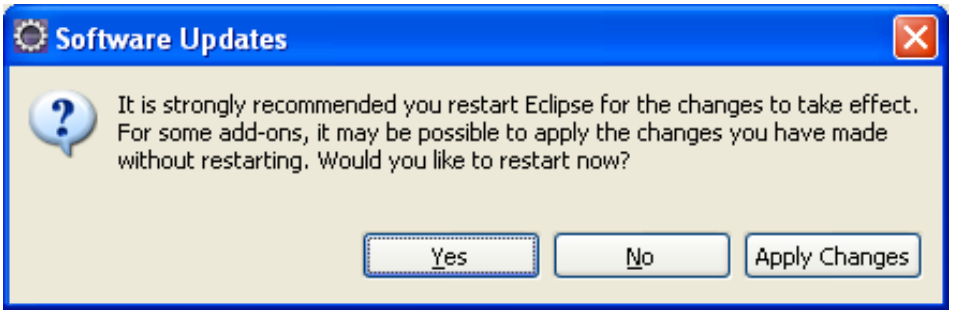
The Glassfish download and installation will begin. Click Details for the details process.



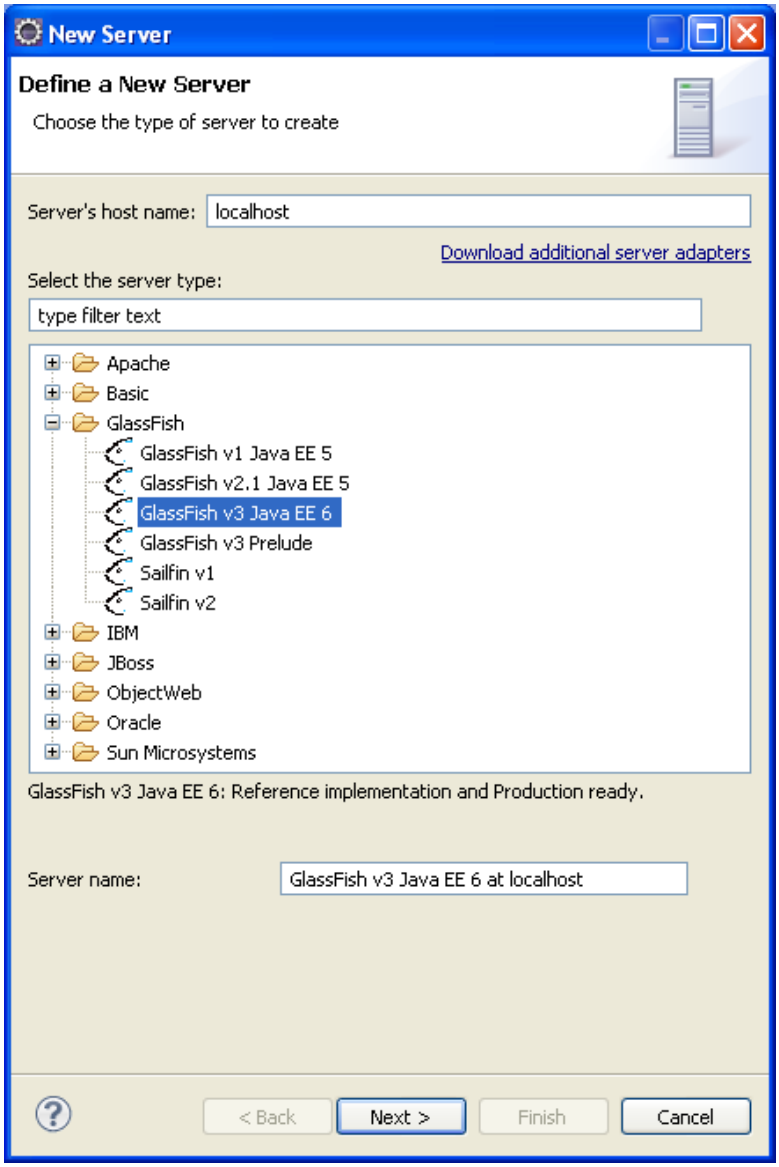
The task can be set to be done at the background by clicking the Run in Background button.



Step-6 restart Eclipse in order the new downloaded and installed Glassfish server take effect.



Step-7 Now, in the New Server page, Glassfish server will be visible in the list of the available servers that are associated with Eclipse and ready to be used.



1.4 SUMMARY

We can use the Java EE tools for implementing new features in applications that are structured around modules with different purposes, such as web sites, web applications and Enterprise applications. When you use Java Enterprise Edition components, you can create distributed, secure applications with transactional support.

1.5 QUESTIONS

Q.1 Define java EE?

Q.2 Applications of Java EE.

Q.3 Explain Design Pattern [MVC] in Java EE.

Q.4 Explain what technologies are used in Java EE.

Q.5 Define Java Enterprise Application.

Q.6 Explain Servlet & JSP.

Q.7 Define Web Server?

Q.8 Explain steps to configure glassfish web server in eclipse.

Q.9 Define JavaMail API.

Q.10 Explain Architecture of JEE.

1.6 REFERENCE FOR FURTHER READING

1. Java EE 6 Enterprise Architect Exam Guide ,Author: PaulAllen , Publisher: McGraw-Hill
2. The Complete Reference -Java Enterprise Edition (Black Book) ,Author:Herbert schildt.
3. Java EE 7 The Big Picture by - Dr.Danny Coward Publisher- Oracle press.
4. Advanced Java by-Balaguruswamy .
5. The Java Ee 7 Tutorial by-Ricardo Cervera-Navarro

*****

2

JAVA EE ARCHITECTURE, SERVER AND CONTAINERS

Unit Structure

2.1 OBJECTIVES

1) Java EE Provides Simplified coding Technique.
2) Java EE Provides Flexible Architecture where we can design in various perspective levels..
3) students/professionals are able to originally design and develop & Deploy their application on server.
4) Java EE Server for implementation of Enterprise Application.
5) Java EE Provides various Containers to transform Business logic etc...
6) Java EE provides a powerful API for Strong and Dynamic Web Programming.

2.2 JAVA ENTERPRISE SYSTEM ARCHITECTURE

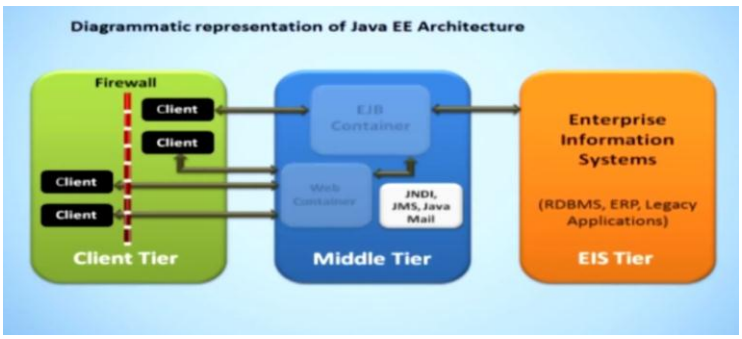


Fig: 2.2 Java Enterprise System Architecture.

A) Java Enterprise System deployments based Architecture:

To discuss java EE architectural concepts upon which Java Enterprise System deployments are based.

Java Enterprise Edition is a framework in which Java Enterprise System deployment architectures are analyzing along with the following three ways/Dimensions:

- Logical tiers.
- Infrastructure service levels,
- Quality of service.

These three dimensions, following figure, help to clarify the architectural functions of Java Enterprise System components while designing and developing projects. In Java EE The three-dimensional framework is the key to designing successful Design & deployment architectures for business software solutions.

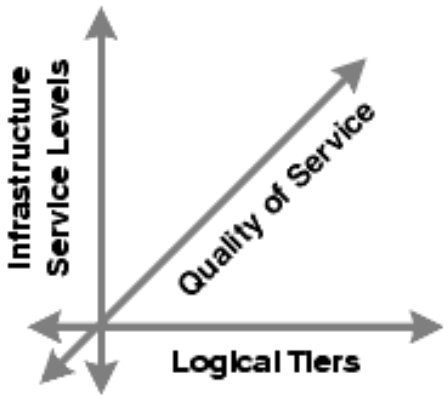


Figure 2.3-A Three Dimensions of Java Enterprise System Architectural Framework

1) Logical Tiers:

In Java EE standard architecture for distributed applications separates application logic into a number of tiers. These tiers signify a logical and physical organization of components into an ordered chain/processing sequence of service providers and consumers in network.

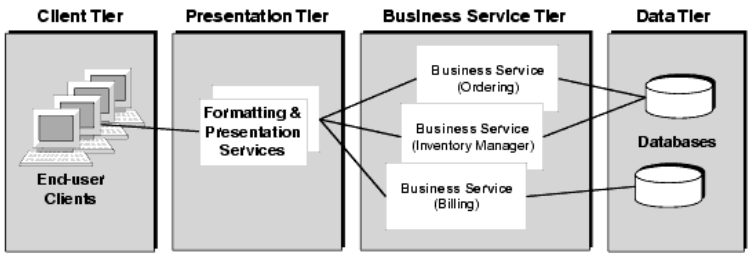


Figure 2.3.1 Logical Tiers for Distributed Enterprise Applications

I) Client Tier:

The client tier consists of application logic/Business logic accessed directly by an end user/client side through a user interface. The logic in the client tier could include browser-based clients.

II) Presentation Tier:

The presentation tier consists of application logic/Business logic that prepares data as per request/response for sending to the client tier and processes requests from the client tier to get the back-end business logic as per concern of request & response. The logic in the presentation tier typically consists of J2EE components such as Java Servlet components or JSP components that prepare data for HTML or XML sending/receiving or that receive requests for processing from client to server or vice versa. This tier includes various services - secure, personalized, and customized access to business services in the business service tier.

III) Business Service Tier:

The business service tier consists of actual required Business logic that performs the main functionality of the applications such as processing data, implementing business rules, coordinating multiple users, and managing external resources such as databases or legacy systems as per request & response from the network. In J2EE components can be assembled to deliver complex business services/business processes such as an inventory service or tax calculation service etc.

The various implementations of business services encapsulate specific application functionality that can reside and run on a particular computing node/client.

IV) Data Tier:

The data tier consists of data used by business logic. The data can be persistent application data stored in a database management system. The data can also include data feeds from external sources or data accessible from legacy systems.

2) Infrastructure Service Levels:

The interacting software components of distributed enterprise applications require an underlying set of infrastructure services that allows the distributed components to communicate with each other on network communication i.e client to server & server to client, coordinate their work, implement secure access, and so forth. This set of distributed services constitutes an infrastructure upon which distributed components can be design & built.

Distributed Infrastructure Services:

Distributed infrastructure services distributed at many different levels.

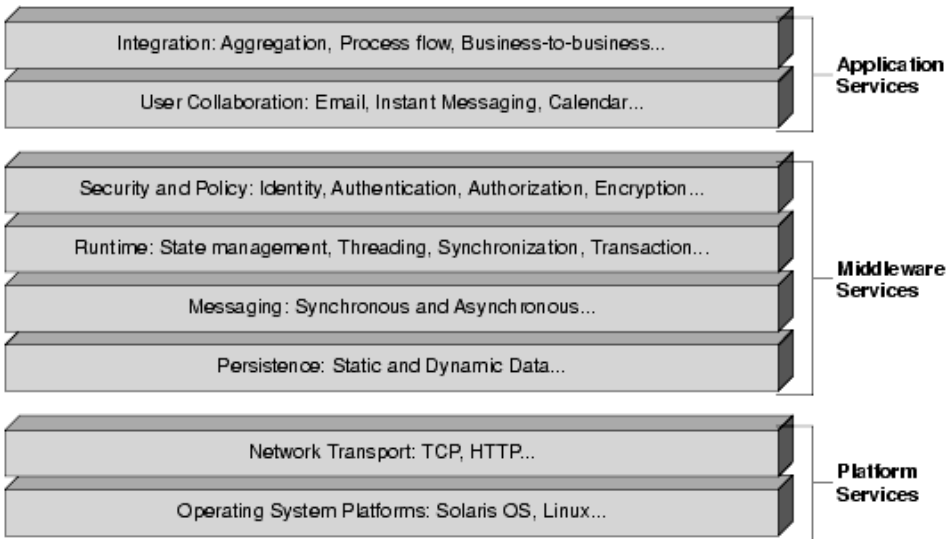


Figure 2.3.2 Distributed Infrastructure Service Levels

The Infrastructure Service Levels in the above Figure reflect a general dependence of the various distributed services on one another, from the lowest-level operating system services to the highest-level application and integration services in Java Enterprise.

Following are the list of Levels from bottom to top:

I) Operating system platform:

It Provides the basic support for any process running on a computing node. The operating system manages physical devices as well as memory, threads, and other resources necessary to support the Java Virtual Machine .

II) Network transport:

Network transport Provides basic networking support for communication between distributed application components running on different computing nodes/Network connected different nodes or clients. All These services include support for protocols such as TCP and HTTP/ HTTPS.

III) Persistence:

It Provides support for accessing/fetching and storing both static data (such as user, directory, or configuration information) and dynamic application data (information that is frequently being updated).

VI) Messaging:

Messaging Layer Provides support for both synchronous and asynchronous communication between application components in client and server. Synchronous messaging is real-time sending and receipt of

messages; it includes remote method invocation (RMI) between J2EE components and SOAP interactions with web services. Asynchronous messaging is communication in which the sending of a message does not depend on the readiness of the consumer to immediately receive it. Asynchronous messaging specifications, for example, Java Message Service (JMS) and ebXML, It will support guaranteed reliability and other messaging semantics.

VII) Runtime:

It Provides support required by any distributed component model, such as the J2EE or CORBA models. In addition to the remote method invocation needed for tightly coupled distributed components, runtime services include component state (life-cycle) management, thread pool management, synchronization (mutex locking), persistence services, distributed transaction monitoring, and distributed exception handling. In a J2EE environment, these runtime services are provided by EJB, web, and message-driven bean (MDB) containers in an application server or web server.

VIII) Security and policy:

Provides support for secure access to application resources. These services include support for policies that govern group or role-based access to distributed resources, as well as single sign-on capabilities. The enhancement of authentication in Single sign-on allows a user's authentication to one service in a distributed system to be automatically applied to other services (J2EE components, business services, and web services) in the system.

IX) User collaboration:

It Provides services that play a key role in supporting direct communication between users and collaboration among users in enterprise and Internet environments. And also, all these services are application-level business services, normally provided by standalone servers (such as an e-mail server).

X) Integration:

It Provides the services that aggregate existing business services, either by providing a common interface for accessing them through a network, as in a portal, or by integrating them through a process engine that coordinates them within a production workflow. Integration can also take place as business-to-business interactions between different enterprises/ various enterprise applications.

3) Quality of Service Layer:

The previous two architectural Levels (logical tiers and infrastructure service levels) largely define the logical/business logical aspects of architecture, namely which components are needed to interact in what are the ways where to deliver services to end users/clients.

As internet and E-commerce services now become more critical to handle business operations/services, scalability, performance, security, availability and serviceability of these services has become a key requirement of large-scale, high-performance deployment architectures in java enterprise applications.

Following are the List of Quality Services required in Java EE Architecture.

1) Performance:

To check out measurement of response time and latency with respect to user load conditions.

2) Availability:

A measuredly system's resources and services are accessible to end users/clients, often expressed as the uptime of a desired system.

3) Security:

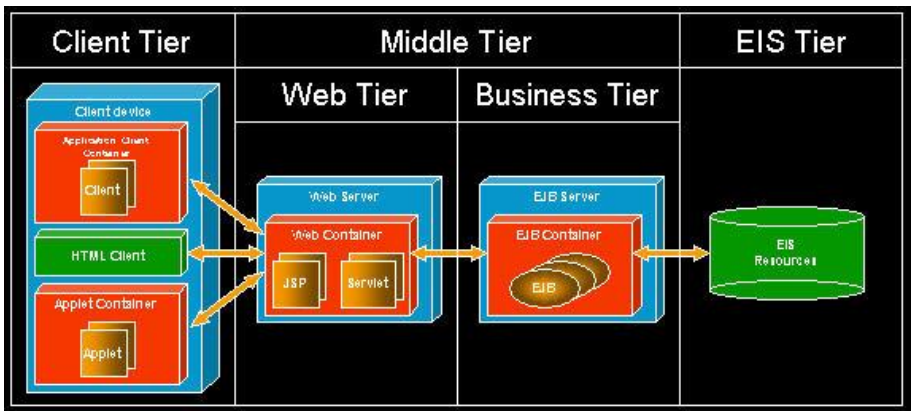
Regarding security, a complex combination of factors that describe the integrity of a system and its users. Security includes authentication and authorization of users as well as the secure transport of information through the network.

4) Scalability:

In Enterprise applications the ability to add capacity (and users) to a deployed system over time. Scalability typically involves adding resources to the system but should not require changes to the deployment architecture while designing java applications.

B) Java EE Development Architecture:

Following are Java EE provides an environment for development and deployment of web-based enterprise applications using multi-tier architecture.



The above diagram demonstrates J2EE multi-tier architecture that encompasses several J2EE containers each including its own J2EE components.

Following a list of overview about the different tiers of J2EE architecture:

1**) Client Tier:**

In JEE Architecture Components of Client Tier will run in the client devices / containers. Client Tier components are standalone or web based java applications, static and dynamic HTML pages, and applets.

2) Middle Tier:

I) Web Tier: In JEE the web tier components namely JSP and Servlets execute with the help of J2EE web server in a web container.

II) Business Tier: In this tire integrate purely Business logic with Enterprise Java Beans (EJB) are the business tier components that are executed within the EJB container using J2EE Application Server/web server.

3) EIS Tier:

EIS (Enterprise Information Systems) tier follows operations related to application data that are stored in a database. EIS tier may also include ERP's varois big operations or legacy systems.

2.4 JAVA EE SERVER

A Java EE server is a server application that implements the Java EE platform APIs and provides the standard Java EE services for Design & developing Applications. Java EE servers are sometimes called application servers or web servers, because they allow you to serve application data to clients, same processing on web servers serve web pages to web browsers. Java EE servers host several application component types that correspond to the tiers in a multi-tiered application. The Java EE server provides services to these components in the form of a container.

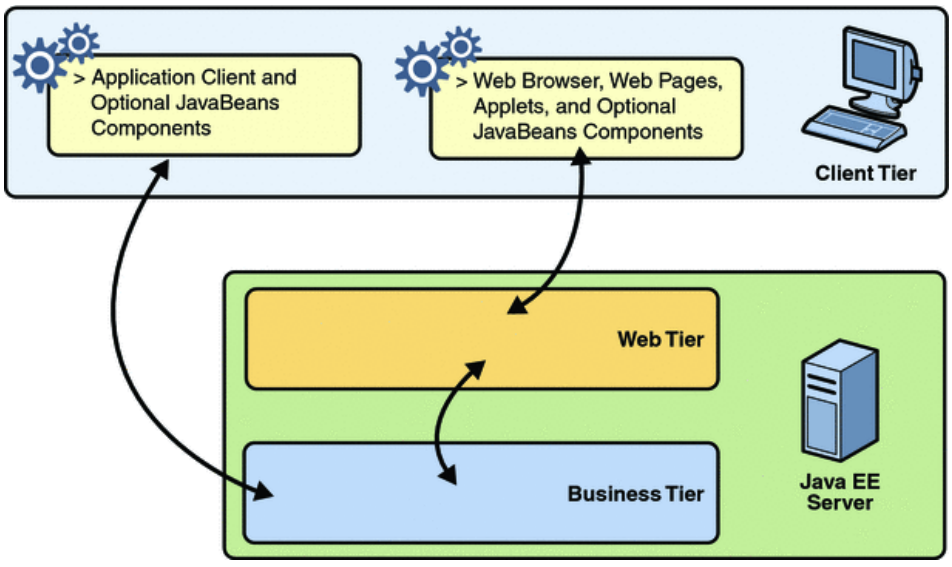


Fig: 2.4 Java EE Communication Server:

2.4.1 Java EE Containers:

Java EE containers are the interface between the component and the lower-level functionality of the application server provided by the platform to support that component and API . The functionality of the container is defined by the platform to provide more productivity, and is different for each component type. The server allows the different component types to work together to provide the enhancement of functionality in an enterprise application.

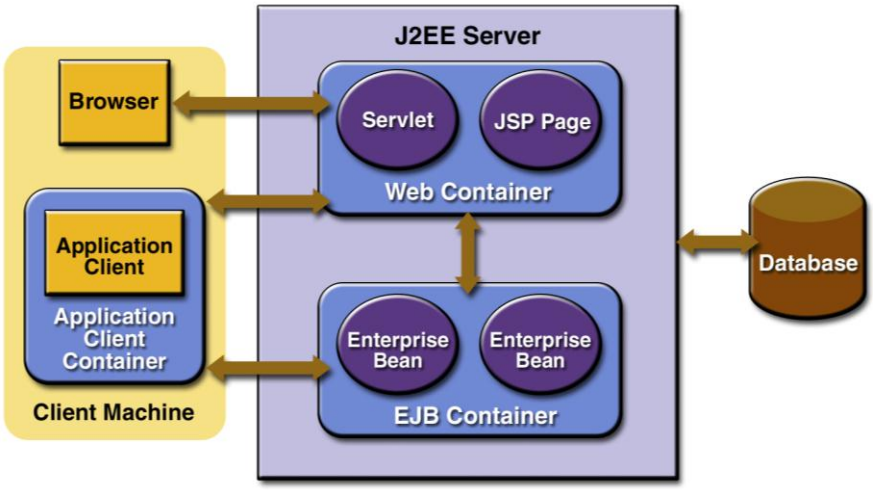


Fig: 2.4.1 Java EE Containers

2.4.2 The Web Container:

The web container is the interface between web components or API and the web server. A web component can be a JSP page, or a JavaServer Faces Facelets page, and servlet. The web container manages the processes of Request & Response in the component's lifecycle , dispatches requests to application components, and provides interfaces to context data, such as information about the current request.

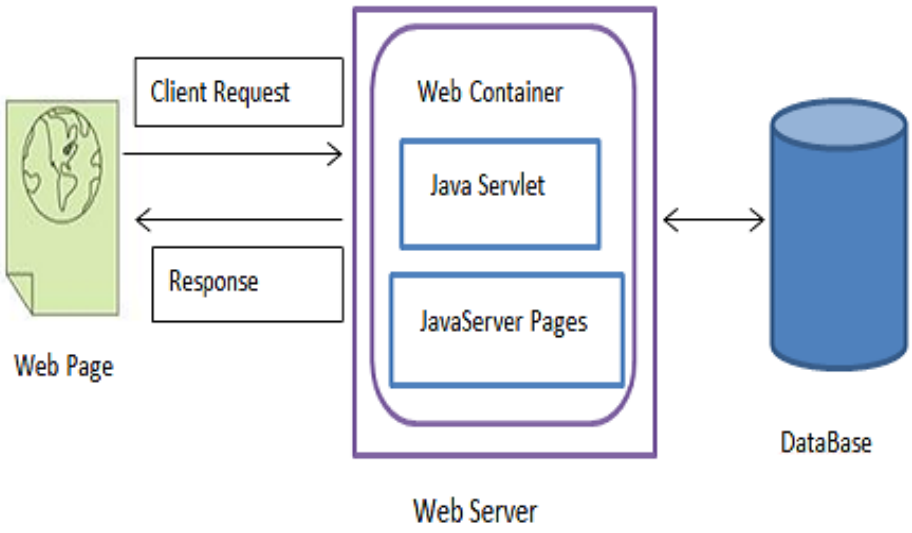


Fig: 2.4.2 TheWeb Container:

2.4.3 The Application Client Container:

The application client container is the interface between Java EE applications and clients/browsers , which are special Java SE applications that use Java EE server components with more API's , and the Java EE server. The application client container runs on the client machine, and is the gateway between the client application and the Java EE server components that the client/browsers uses.

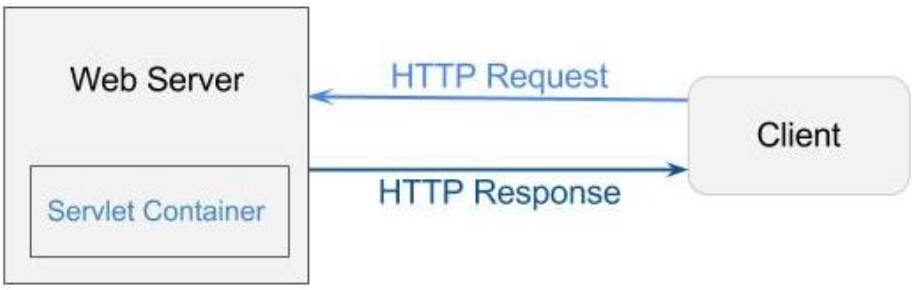


Fig: 2.4.3 The Application Client Container.

2.4.4 The EJB Container:

The EJB container is the interface between enterprise java beans, which provide the business logic in a Java EE application to manage data transportation in critical condition with the Java EE server. The EJB container runs on the Java EE server or web server and manages the execution of an application's enterprise beans in Business Logic to Database and viceversa.

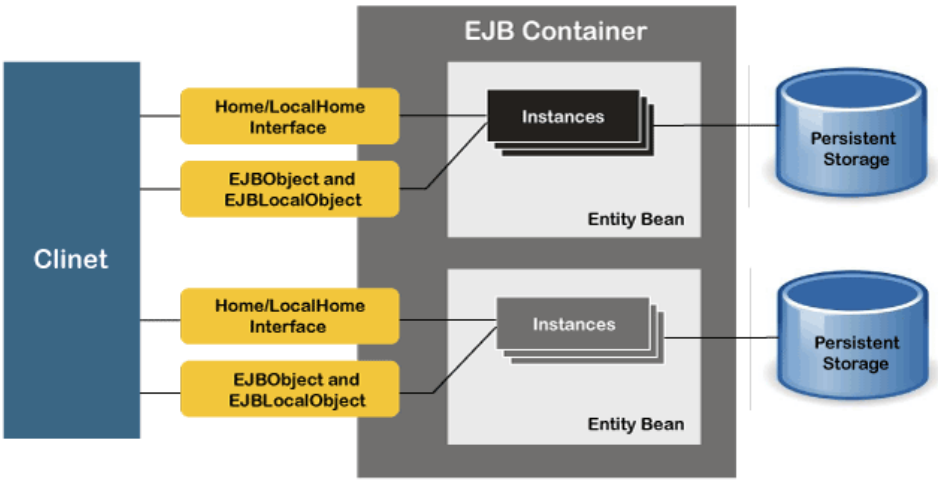


Fig: 2.4.4 The EJB Container.

2.5 JAVA EE CONTAINERS

Containers easily manage their transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent.

Java EE architecture provides better service to write business logic that is organized into reusable components.

Container Services:

Containers are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container.

Key points of Container:

- In Java EE security models configure a web component or EJB so that system resources are accessed at client side/browsers only by authorized users.
- The Java EE transaction model provides relationships among methods/functions that communicate a single transaction so that all methods/functions in single transaction are treated as a single unit.
- The Java EE can be remotely connectivity model manages low-level communications between clients and EJB

Following diagrams shows The deployment process installs Java EE application components in the Java EE containers as illustrated in Following Figure.

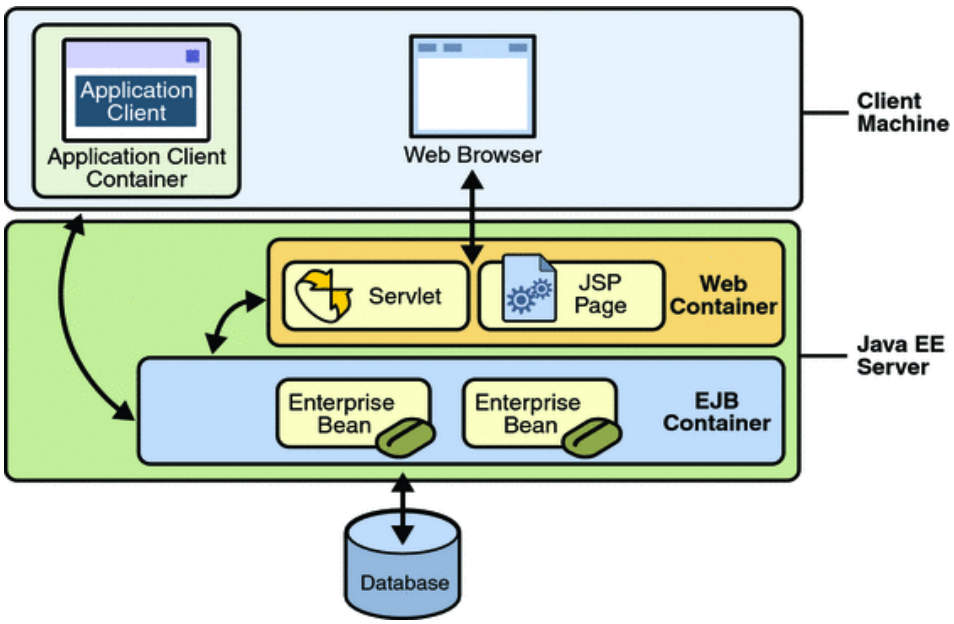


Figure 2.5 Java EE Server and Container.

Following are the list of Types of Container:

1) Enterprise JavaBeans (EJB) container:

EJB containers can manage the execution of enterprise beans for Java EE applications/ web applications & services. EJB and their container run on the Java EE server/web server.

2) Web container:

Here Web containers can manage the execution of JSP page and servlet components for Java EE applications. Web components and their containers run on the Java EE server/web server.

3) Application client container:

Java EE Application client container can manage the execution of application client components. Java EE Application clients and their container run on the client/browser.

4) Applet container:

Manages the execution of applets/java programs executed on client. Applet container consists of a web browser and Java Plug-in running on the client.

2.6 QUESTION

1 Explain in detail the container with types.

2 Explain Java EE Development Architecture.

3 Difference between Apache tomcat & Glassfish server.

4 Explain Java EE Deployment Architecture.

5 Explain Quality of Service layer in Java EE Deployment Architecture.

6 Define the web container.

7 Define the EJB container.

2.7 SUMMARY

Web server is a computer where the web content is stored. Basically a web server is used to host the web sites. In java EE Containers are the interface between a component and the low-level, platform-specific functionality that supports the component. Java Enterprise Edition is a set of various specifications, for enterprise features such as distributed computing, Security, Powerful API's and web services.

2.8 REFERENCE FOR FURTHER READING:

1. Java EE 6 Enterprise Architect Exam Guide ,Author: PaulAllen, Publisher: McGraw-Hill

2. The Complete Reference -Java Enterprise Edition (Black Book), Author:Herbert schildt.

3. Java EE 7 The Big Picture by - Dr.Danny Coward Publisher- Oracle press.

4. Advanced Java by-Balaguruswamy .

5. The Java Ee 7 Tutorial by-Ricardo Cervera-Navarro

*****

3

INTRODUCTION TO JAVA SERVLETS

Unit Structure

3.1 OBJECTIVES

1) Java servlet provides server side coding technique.
2) Java servlet is used to develop & design web applications in a web server.
3) Using servlet for students/professionals are able to originally design and develop & Deploy their application on server.
4) Servlet is a technology which is used to create a web application executed at server side.
5) Java servlet provides a powerful API for best Dynamic Web Programming.

3.2 THE NEED FOR DYNAMIC CONTENT

3.2.1 Introduction:

Dynamic content refers to web content that changes based on the behaviour, preferences, and interests of the user. It refers to websites as well as e-mail content and is generated at the moment a user requests a page. Dynamic content is personalized and adapts based on the data you have about the user and on the access time, its goal being to deliver an engaging and satisfying online experience for the visitor.

Dynamic content (adaptive content) refers to web content that changes based on the behaviour, preferences, and interests of the user. It refers to websites as well as Email content and is generated at the moment a user requests a page. Dynamic content is personalized and adapts based on the data you have about the user and on the access time, its goal being to deliver an engaging and satisfying online experience for the visitor.

A server-side dynamic web page is a web page whose construction is controlled by an application server to process server-side scripts dynamically. In server-side scripting programming, an parameters determine how the assembly of every new web page proceeds as per user request, including the setting up of more client-side processing

This content can be displayed in a variety of different forms on the Web. The way it is usually presented is based on the type of website you are on. However, things like pictures, text, videos, newsletters, and other web forms are often great examples of this content being used.

3.2.2 Dynamic content vs static content:

Static content has not changed on the internet. This is because it's much easier to implement than the dynamic text alternatives on the Web. However, the downside is that static content is not personalized and thus it reduces the performance of the website. Dynamic content has various benefits as follows:

Difference between Static and Dynamic Web Pages

Sr. No	Static Web Page	Dynamic Web Page
1.	In static web pages, Pages will remain same until someone changes it manually.	In dynamic web pages, Content of pages are different for different visitors.
2.	Static Web Pages are simple in terms of complexity.	Dynamic web pages are complicated.
3.	In static web pages, Information are change rarely.	In dynamic web page, Information are change frequently.
4.	Static Web Page takes less time for loading than dynamic web page.	Dynamic web page takes more time for loading.
5.	In Static Web Pages, database is not used.	In dynamic web pages, database is used.
6.	Static web pages are written in languages such as: HTML, JavaScript, CSS, etc.	Dynamic web pages are written in languages such as: CGI, AJAX, ASP, ASP.NET, etc.
7.	Static web pages does not contain any application program	Dynamic web pages contains application program for different services.

8.	Static web pages require less work and cost in designing them.	Dynamic web pages require comparatively more work and cost in designing them.

- it makes for a more user-friendly experience on the web/Internet.
- it helps increase vital current data, latest information or instant info.
- Within page layout properly & accurate displays data.
- Once has been upload, live and active, you don't need to tend to it anymore

3.2.3 Areas to implement Dynamic Content:

1) Newsletters and Emails:

Newsletters and emails are probably the most basic and classic forms of dynamic content being presented as per requirement. Customized emails or customized form fields for specific users have been around on the web for a long time. Presenting your dynamic content this way is always a good choice because data is frequently changed. And some newsletter plugins will display its content/data as per updated content you re-purpose.

2) Landing Pages:

A landing page is built specifically to target a certain thing or targeted audience. Whether it is a product or a service that has to be reached, the landing page acts as an informational doorway to what you are selling to an accurate customer. Take it further by presenting a landing page that uses dynamic content to display different information and items based on who is viewing it as per search & updated data.

3) Articles:

Articles can be used to display dynamic content based on the device/platforms that is being used to read the article. This type of content can be displayed properly on any device and screen size or resolution. This is why your website should be mobile friendly and responsive web content. Giving the user viewable content no matter where they access it would be displayed in proper layout to any device or screen.

4) Forms and Purchases:

As per every user requirement or organization operation perspective Different form and purchase fields or other information can be displayed, based on how the checkout is going to be on demand , who the user is, and what their interests are different users makes the entire experience much better for them.

5**) Product Pages:**

In E-commerce online stores will use product pages to cross-sell and match related items for the site user where the user chooses the products.

This is dynamic content at work using certain things to match up what we think or what the users need as per the user wants to see. Online store organization bounce offers like sales and coupons or other special recommendations as well.

6) Website Ads:

Ads on websites, social media and on Google can use Digital marketing or Digital advertisement using dynamic content technique. These ads display content on what users search and interests the website or Application. This makes ad interaction and click-throughs much more likely where users easily interact with.

3.3 Java Servlet Technology:

Servlet is a technology used to create a java web application (placed at server side and generates a dynamic web page as per client request).

Servlet is a technology that is robust and scalable and secure because of the Java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. There were many disadvantages of CGI. Some of these disadvantages are discussed below.

Servlet API is a collection of various interfaces and classes in the Servlet API such as Servlet, ServletResponse,ServletRequest GenericServlet, HttpServlet... etc.

3.3.1 Use of Servlet:

Servlet technology can be described in many ways, As follows.

- Servlet is a technology which is used to create a java web application.
- Servlet is an API collection of various interfaces and classes.
- Servlet is an interface that must be implemented for creating any Servlet page.

- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

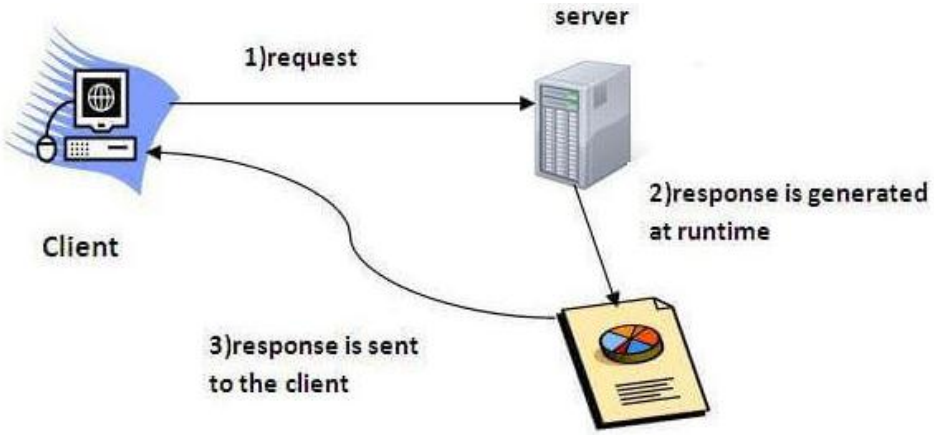


Fig: 3.3 Communication of Java Servlet Technology.

What is a web application?

A web application is a server side application which is accessible from the web/web browser. A web application is composed of web components like Servlet, Filter & JSP, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

3.3.2 CGI (Common Gateway Interface):

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each & every request, it starts a new process or creates a new Thread for every request. i.e. Number request coming to CGI, a number of newly processes or threads to create. CGI will get more load & responses are slow or its processing task may be heavy.

The **Common Gateway Interface (CGI)** provides the middleware between WWW servers and external databases and information sources. The World Wide Web Consortium (W3C) defined the Common Gateway Interface (CGI) and also defined how a program interacts with a Hyper Text Transfer Protocol (HTTP) server. The Web server typically passes the form information to a small application program that processes the data and may send back a confirmation message. This process or convention for passing data back and forth between the server and the application is called the common gateway interface (CGI).

Features of CGI:

- It is a very well defined and supported standard.

- CGI scripts are generally written in either Perl, C, or maybe just a simple shell script.
- CGI is a technology that interfaces with HTML.
- CGI is the best method to create a counter because it is currently the quickest
- CGI standard is generally the most compatible with today's browsers.

Advantages of CGI:

- The advanced tasks are currently a lot easier to perform in CGI than in Java.
- It is always easier to use the code already written than to write your own.
- CGI specifies that the programs can be written in any language, and on any platform, as long as they conform to the specification.
- CGI-based counters and CGI code to perform simple tasks are available in plenty.

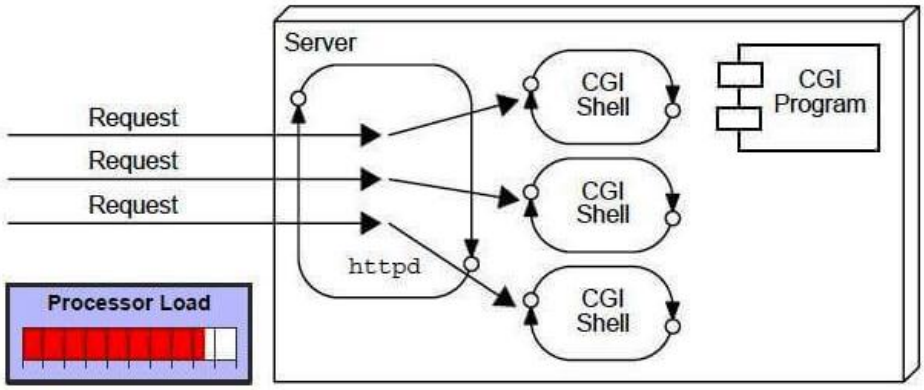


Fig: 3.3.2 CGI (Common Gateway Interface) Communication flow.

Disadvantages of CGI:

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time to send the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent languages e.g. C, C++, perl.
4. CGI facing Network Traffic, because of heavy load.
5. Number request, Number of newly processes so CGI contain heavy load.

3.3.3 Servlet Technology & its Advantages:

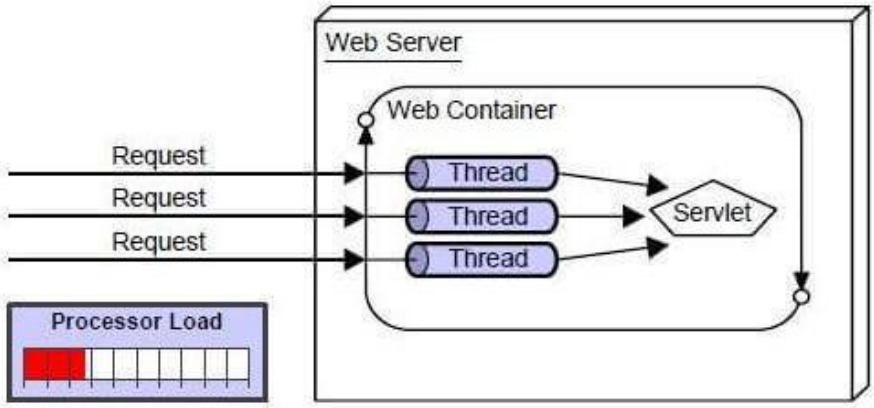


Fig: 3.3.3 Servlet Technology Communication flow.

As per comparison of Servlet & CGI there are many advantages of Servlet over CGI. The web container/Servlet will create threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes happening in Network such as they share a common memory area of the server, are lightweight, and cost of communication between the threads are low.

Shortly after the Web began to be used for delivering services, service providers recognized the need for dynamic content. Applets, one of the earliest attempts toward this goal, focused on using the client platform to deliver dynamic user experiences. At the same time, developers also investigated using the server platform for the same purpose. Initially, Common Gateway Interface (CGI) server-side scripts were the main technology used to generate dynamic content. Although widely used, CGI scripting technology had many shortcomings, including platform dependence and lack of scalability. To address these limitations, Java Servlet technology was created as a portable way to provide dynamic, user-oriented content.

The advantages of Servlet are as follows:

1. **Better performance**: No of processes is converted into threads so, because it creates a thread for each request, not process.

2. **Portability:** Servlet programs execute any platform because it uses Java language.

3. **Robust:** JVM manages Servlet programs, so we don't need to worry about the memory leak, garbage collection, etc.

4. **Secure:** because it uses java language.

Difference between Servlet and CGI

Servlet	CGI(Common Gateway Interface)
Servlets are portable and efficient.	CGI is not portable
In Servlets, sharing data is possible.	In CGI, sharing data is not possible.
Servlets can directly communicate with the webserver.	CGI cannot directly communicate with the webserver.
Servlets are less expensive than CGI.	CGI is more expensive than Servlets.
Servlets can handle the cookies.	CGI cannot handle the cookies.

3.4 WHY SERVLETS?

Today's Web applications trends it to creating dynamic web pages i.e the ones which have the capability to change the site contents according to the time/response or are able to generate the contents according to the request received by the client. If you like coding in Java, then you will be happy to know about Java. There also exists a way to generate dynamic web pages and that way is Java Servlet API. first understand the need for server-side extensions i.e why use servlet to develop dynamic applications.

A servlet is a Java Programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. It is also a web component that is deployed on the server to create a dynamic web page.

Fig: How servlet Work.

In this figure you can see, a client sends a request to the server and the server generates the response, analyses it and sends the response to the client.

Servlets are the Java programs that run on the Java-enabled web server or application server[Apache server, Glassfish server, etc….]. They are used to handle the request obtained from the web server/Client, process the request, produce the response, then send a response back to the web server/web browser.

Properties of Servlets are as follows:

- Servlets work on the server-side.
- Servlets are capable of handling complex requests obtained from the web server.

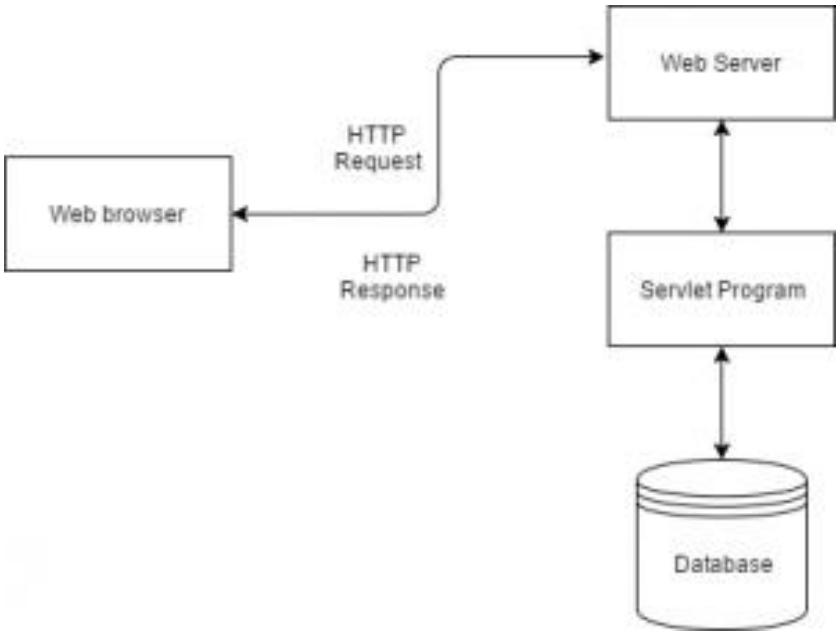


Fig: 3.4 Servlet Architecture

Execution of Servlets basically involves following basic steps:

1. The clients/browsers send the request to the web server.
2. The web server receives the request from clients.
3. The web server passes the request to the corresponding servlet container.
4. The servlet processes the request and generates the response in the form of output.
5. The servlet sends the response back to the web server.

6. The web server sends the response back to the client and the client browser displays it on the screen.

3.5 WHAT CAN SERVLETS DO?

Introduction:

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a request coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI).

But Servlets offer several advantages in comparison with the CGI:

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.

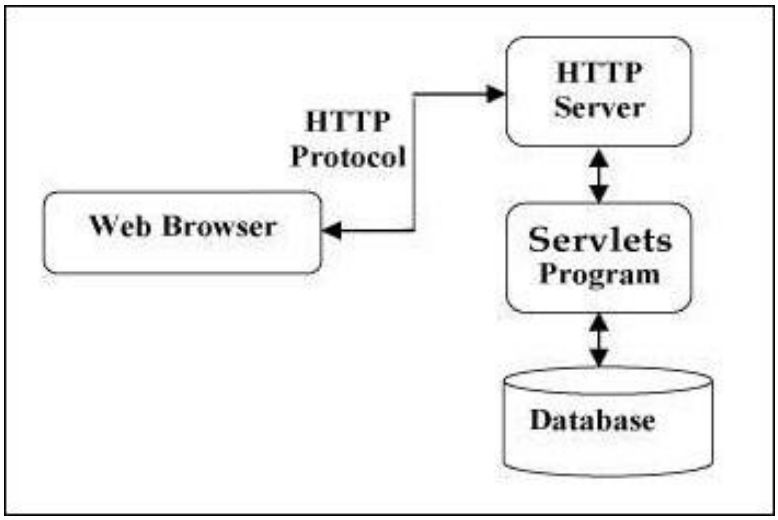


Fig: Servlet Architecture

Servlets Tasks:

Servlets perform the following major tasks

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

1) Dynamic website:

Dynamic website is a collection of dynamic web pages whose content changes dynamically. It accesses content from a database or Content Management System (CMS). Therefore, when you alter or update the content of the database, the content of the website is also altered or updated.

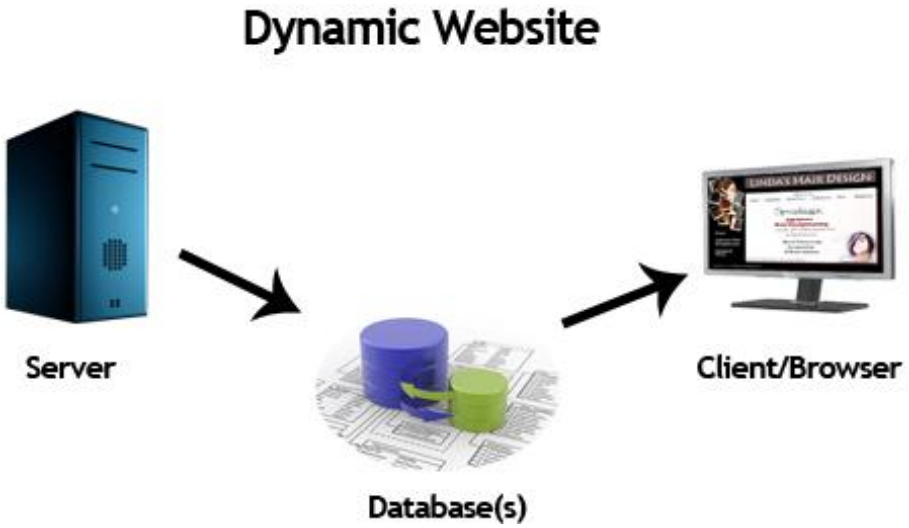
Dynamic websites are those websites that changes the content or layout with every request to the webserver. These websites have the capability of producing different content for different visitors from the same source code file. There are two kinds of dynamic web pages i.e. client side scripting and server side scripting. The client-side web pages changes according to your activity on the web page. On the server-side, web pages are changed whenever a web page is loaded.

Dynamic websites use client-side scripting or server-side scripting, or both to generate dynamic content.

Dynamic websites use client-side scripting or server-side scripting, or both to generate dynamic content.

Client side scripting generates content at the client computer on the basis of user input. The web browser downloads the web page from the server and processes the code within the page to render information to the user.

In server side scripting, the software runs on the server and processing is completed in the server then plain pages are sent to the user.



2) The server-side extensions:

In Java server side extensions i.e. servlet or JEE are nothing but the technologies that are used to create dynamic Web pages. In servlet Technology to provide the facility of dynamic Web pages, Web pages need a container or Web server. To complete this requirement, independent Web server providers offer some essentials solutions in the form of APIs(Application Programming Interface).

3) Handling Request & Response:

In Servlet Technology for the purpose of communication to use HTTP protocol. The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative and hypermedia information systems. HTTP Request & Response are the data communication protocol used to establish communication between client and server.

HTTP is a stateless TCP/IP based communication protocol, which is used to send & receive data like image files, query results, HTML files etc on the World Wide Web (WWW) with the default port being TCP 80. It provides a standardized way for computers to communicate with each other in a network.

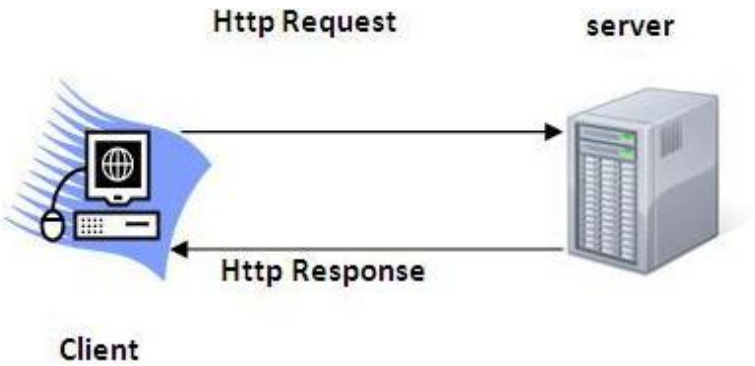


Fig:A- Handling Http Request & Response

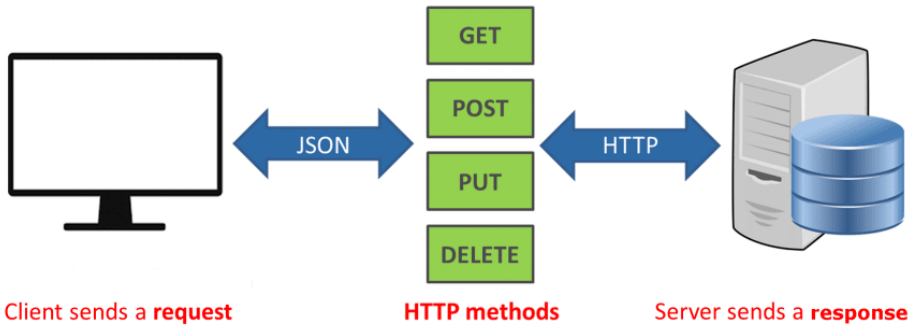


Fig:B- Handling Http Request & Response.

The Basic Characteristics of HTTP (HyperText Transfer Protocol):

- HTTP protocol allows web servers and browsers to exchange data over the web.
- HTTP a request response protocol.
- HTTPuses the reliable TCP connections by default on TCP port 80.
- HTTP is stateless means each request is considered as the new request. In other words, the server doesn't recognize the user by default.

HttpServlet class provides specialized methods that handle the various types of HTTP requests. A servlet developer typically overrides one of these methods. These methods are **doDelete( ), doGet( ), doHead( ), doOptions( ), doPost( ), doPut( ),** and **doTrace( ).** However, the GET and POST requests are commonly used when handling form input. The doPost( ) method is overridden to process any HTTP POST requests that are sent to this servlet. It uses the getParameter( ) method of HttpServletRequest to obtain the selection that was made by the user.

4) Filtering Requests and Responses:

In Servlet Technology to provide a filter object that can transform the header and content (or both) of a request or response. In the Servlet Filters objects differ from web components in that filters usually do not themselves create a response. Instead a filter object provides functionality that can be "attached" to any kind of web resource with servlet objects.

In Servlet filter Object can perform following tasks:

- Filter object Query the request and act accordingly.
- Block the request-and-response pair from passing any further transaction.
- Filter objects can modify the request headers and data. You do this by providing a customized version of the request.

- In Filter Modify the response headers and data. You do this by providing a customized version of the response.

- Filter Objects are Interact with various external resources.

5) Sharing Information:

In servlet its mechanism is objects communicate in through a network. There are several ways they can do this. In Java Servlet class can use private helper objects (for example, JavaBeans components), Servlet class (beans) can share objects that are attributes of a public scope or private, they can use a database, and they can invoke other web resources.

In servlet its mechanism is objects communicate in through a network. There are several ways they can do this. In Java Servlet class can use private helper objects (for example, JavaBeans components), Servlet class (beans) can share objects that are attributes of a public scope or private, they can use a database, and they can invoke other web resources.

The Java Servlet technology mechanisms that can be allowed to access a component to invoke other web resources are described in Invoking Other Web Resources.

6) Accessing the Web Context:

In The Servlet context in which web components execute is an object that implements the ServletContext interface. You retrieve the web context using the getServletContext method. The web context provides methods for accessing:

- Initialization parameters.
- Resources associated with the web context.
- Object-valued attributes.
- Logging capabilities.

3.6 QUESTIONS

1. Explain what is Dynamic Content?
2. Explain what is needed to be Design Dynamic Content.
3. Explain the features of Servlet.
4. Explain in detail Why Servlet is popular as compared to CGI.
5. Difference between Servlet Vs CGI.
6. Explain Use of Servlet in Web Application.
7. Define Servlet API.

8. Explain in brief What can servlet do?
9. Explain in detail Servlet Architecture.
10. Define Request & Response Methodology.

3.7 SUMMARY

A server-side dynamic web page is a web page whose construction is controlled by an application server to process server-side scripts dynamically. In server-side scripting programming, parameters determine how the assembly of every new web page proceeds as per user request, including the setting up of more client-side processing.

Dynamic content is personalized and adapts based on the data you have about the user and on the access time, its goal being to deliver an engaging and satisfying online experience for the visitor.

Servlet is a technology used to create a java web application (placed at server side and generates a dynamic web page as per client request).

Servlet is a technology that is robust and scalable and secure because of the Java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. There were many disadvantages of CGI. Some of these disadvantages are discussed below.

The client-side web pages changes according to your activity on the web page. On the server-side, web pages are changed whenever a web page is loaded.

Dynamic website is a collection of dynamic web pages whose content changes dynamically so we can design such a dynamic application using servlet. It accesses content from a database or Content Management System (CMS).

3.8 REFERENCE FOR FURTHER READING

1. The Complete Reference -Java Enterprise Edition (Black Book) , Author:Herbert schildt.
2. Java EE 7 The Big Picture by - Dr.Danny Coward Publisher- Oracle press.
3. Advanced Java by-Balaguruswamy .
4. The Java Ee 7 Tutorial by-Ricardo Cervera-Navarro.

*****

4

SERVLET API AND LIFECYCLE

Unit Structure

4.1 OBJECTIVES

1) Java Servlet Provides Server side Programming techniques.
2) Java Servlet Provides Flexible Architecture where we can communicate easily through a web server.
3) Java Servlet for implementation of communication between client & server.
4) Java Servlet Provides a mechanism i.e handles the request & response services.
5) Java Servlet provides a powerful API for Strong and Dynamic Web Programming.
6) Students/professionals are able to originally design and develop & deploy their application on server.

4.2 JAVA SERVLET API

- In Java Servlet Containing API i.e the javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet API Where to design java web applications.
- The javax.servlet package contains so many interfaces and classes that are used by the servlet or web container programming.
- The javax.servlet.http package contains interfaces and classes that are responsible for handling HTTP related Requests & Responses from client to server & viceversa.

In Java EE Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement for handling processes.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly) Without servlet interface servlet class will not be generate.

In the Javax.servlet package, the ServletRequest Interface is used to handle client requests to access a servlet. It provides the information of a servlet like, parameter names, content type, content length and values.

In the Javax.servlet package, the ServletResponse interface defines an object to help a Servlet in sending a response to the client/browser. It has various methods that help communicate a servlet to respond to the client requests.

In the Javax.servlet package, the RequestDispatcher interface provides the facility of dispatching the request to another resource , be it html, servlet or jsp. This interface can also be used to include the content of another resource.

Following are the interfaces of javax.servlet package:

There are many interfaces in the javax.servlet package. They are as follows:

1. Servlet Interface.
2. ServletRequest.
3. ServletResponse.
4. RequestDispatcher.
5. ServletConfig:
6. ServletContext
7. ServletRequestListener
8. ServletRequestAttributeListener
9. ServletContextListener
10. ervletContextAttributeListener

Following are the List of Classes in javax.servlet package

1. GenericServlet.
2. ServletInputStream.
3. ServletOutputStream.

4. ServletRequestWrapper.
5. ServletResponseWrapper.
6. ServletRequestEvent.
7. ServletContextEvent.
8. ServletRequestAttributeEvent.
9. ServletContextAttributeEvent.
10. ServletException.
11. UnavailableException.

Following are the List of Interfaces in javax.servlet.http package:

1) HttpServletRequest.
2) HttpServletResponse.
3) HttpSession.
4) HttpSessionListener.
5) HttpSessionAttributeListener.
6) HttpSessionBindingListener.
7) HttpSessionActivationListener.

Following are the List of Classes in javax.servlet.http package:

1) HttpServlet.
2) Cookie.
3) HttpServletRequestWrapper
4) HttpServletResponseWrapper
5) HttpSessionEvent.
6) HttpSessionBindingEvent.

4.3 THE SERVLET SKELETON

Servlet development: a skeleton Servlet, Once you have set up your Servlet environment, the first step is generally to write a test Servlet using code such as the skeleton shown in this example.

To write a basic Servlet, you generally:

- overwrite HttpServlet;
- overwrite the doGet() and doPut().

In servlet there are two methods: doGet() and doPut(). These methods will be called in response to GET and POST requests from the user's web browser/client side. Unless you specifically need your web application to respond differently to the two types of request.

Servlet code looks as follows:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class BasicServlet extends HttpServlet
 {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
{
    res.setContentType("text/html");
   PrintWriter pw = res.getWriter();
   // ... output page to pw...
  }
  public void doPost(HttpServletRequest req, HttpServletResponse res)
  throws ServletException, IOException
{
  doGet(req, res);
  }
 }
```

The above example doesn't actually output any HTML, but it shows the basic anatomy of a servlet class. Class extends HttpServlet and must provide implementations of the two methods. These methods correspond to HTTP get() and post() methods respectively. Developers can generally make doPost() simply pass the request to doGet(). Servlets containing both types of requests essentially look the same. For example, parameters will be extracted from either a URL or POSTed data.

4.4 THE SERVLET LIFE CYCLE

In Java Servlet, The web container maintains the life cycle of a servlet instance/Object.

Following are the Stages of life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.

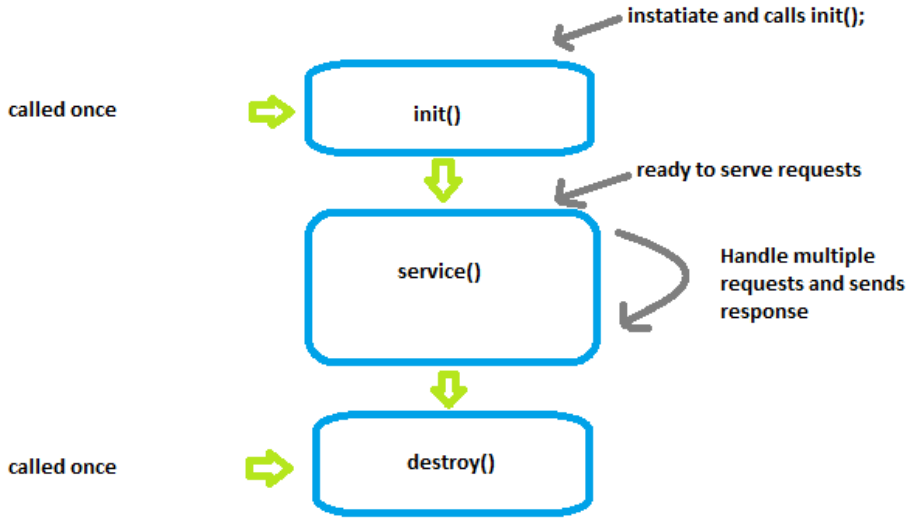


Fig: 4.4 Stages of servlet life cycle .

As displayed in the above diagram, there are three states of a servlet namely: new, ready and end.

- The servlet is in a new state if the servlet instance is created.
- After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks.
- When the web container invokes the destroy() method, it shifts to the end state.

1) Servlet class is loaded:

In servlet while executing the servlet class first stage is i.e classloader is responsible for loading the servlet class into RAM. servlet class is loaded when the first request comes from the web container.

2) Servlet instance is created:

After loading the servlet class web container creates the instance of a servlet t class. The servlet instance/object is created only once in the servlet life cycle; the second time request is not created.

3) init method is invoked:

The web container/web server calls the init() method only once after creating the servlet instance. Basically the init() method is used to initialize the servlet. init() method is the content of the life cycle of the javax.servlet.Servlet interface.

Syntax:

public void init(ServletConfig config)

4) service method is invoked:

The web container/web server calls the service method each time when a request for the servlet is received. If the servlet class or object is not initialized, again it follows the first three steps as described above then calls the service method. If the servlet is initialized, it calls the service method.

Syntax:

public void service(ServletRequest request, ServletResponse response)

5) destroy method is invoked:

The web container/web server calls the destroy method before removing the servlet instance from the RAM. destroy() method gives the servlet an opportunity to clean up any resource for example memory, thread etc.

4.5 SIMPLE WELCOME SERVLET PROGRAM

There are 6 steps to create a servlet Program example.

Following are These steps are required for writing the Servlet Programs.

The servlet example can be created by three ways:

- Implementing Servlet interface.
- Inheriting GenericServlet class.
- Inheriting HttpServlet class.

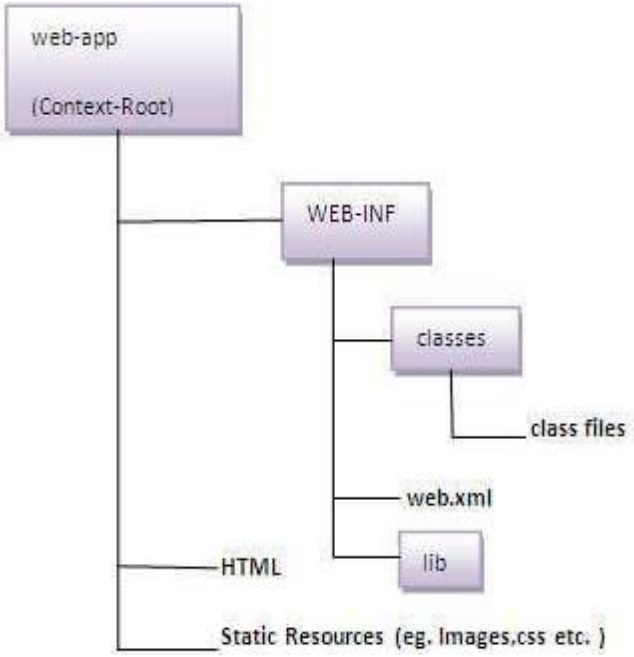
The most used approach is by extending HttpServlet in the desired servlet file because it provides http request/response, specific methods such as doGet(), doPost(), doHead().

❖ **Steps to Create & Execute servlet Program:**

1) Create a directory structures:

The directory structure defines where to put the different types of files so that the web container may get the information and respond to the client.

The Sun Microsystem defines a unique standard to be followed by all the server vendors. Let's see the directory structure that must be followed to create the servlet.



As you can see, the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

2) Create a Servlet file:

There are three ways to create the servlet

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost doHead() etc.

In this servlet example we are going to create a servlet class that will be extended from HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. Notice that get request is the default request

Create File : SampleServlet.java

```
import javax.servlet.http.*;

import javax.servlet.*;

import java.io.*;

public class SampleServlet extends HttpServlet//servlet class is extends from Httpservlet class.

{

public void doGet(HttpServletRequest request,HttpServletResponse response)

throws ServletException,IOException

{

response.setContentType("text/html");//setting the content type

PrintWriter pw=response.getWriter();//get the stream to write the data

//writing html in the stream

pw.println("<html><body>");

pw.println("Welcome to servlet");

pw.println("</body></html>");

pw.close();//closing the stream

}

}
```

3) Compile the servlet:

For compiling the Servlet, a jar file is required to be loaded. Different Servers provide different jar files:

Jar files	Desired Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Put the java file in any folder. After compiling the java file, paste the class file of servlet in WEB-INF/classes directory.

4) Create the deployment descriptor (web.xml file):

The deployment descriptor is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here are some necessary elements to run the simple servlet program.

web.xml file

<web-app>

<servlet>

<servlet-name>umesh</servlet-name>

<servlet-class>SampleServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>umesh</servlet-name>

<url-pattern>/welcome</url-pattern>

</servlet-mapping>

</web-app>

Description of the content of web.xml file:

There are too many elements in the web.xml file. Here is the illustration of some elements that are used in the above web.xml file. The elements are as follows:

1. <web-app> represents the whole application.
2. <servlet> is a sub element of <web-app> and represents the servlet.
3. <servlet-name> is a sub element of <servlet> that represents the name of the servlet.
4. <servlet-class> is a sub element of <servlet> that represents the class of the servlet.
5. <servlet-mapping> is a sub element of <web-app>. It is used to map the servlet.
6. <url-pattern> is a sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

5) Start the Server and deploy the project:

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

❖ One Time Configuration for Apache Tomcat Server

You need to perform 2 tasks:

1. We have to set JAVA_HOME or JRE_HOME in the environment variable (It is required to start the server).

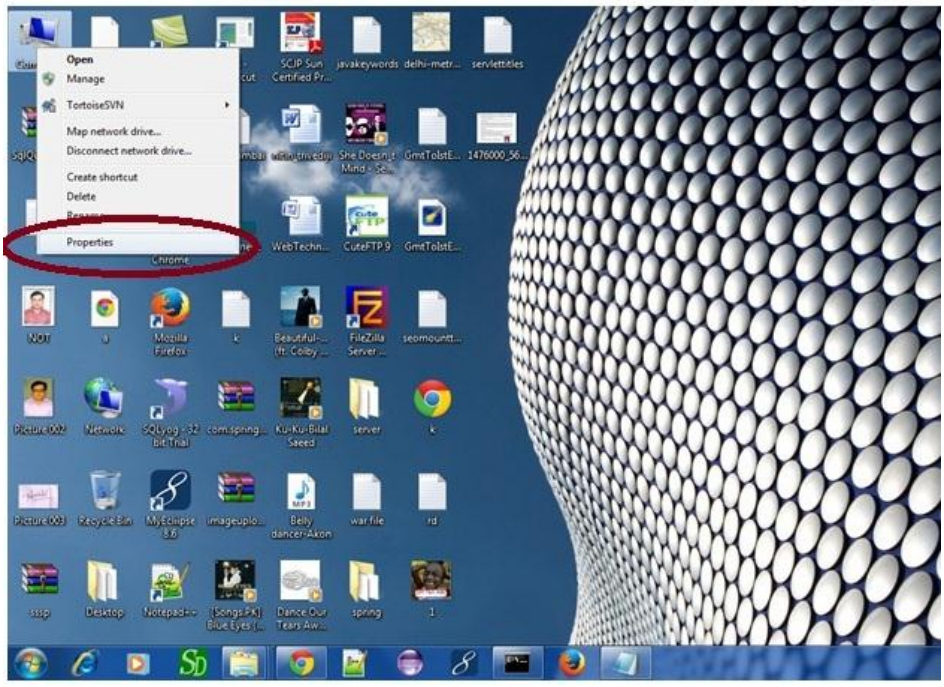
2. To Change the port number of tomcat (optional). It is required if another server is running on the same port (8080).

1) How to set JAVA_HOME in the environment variable:

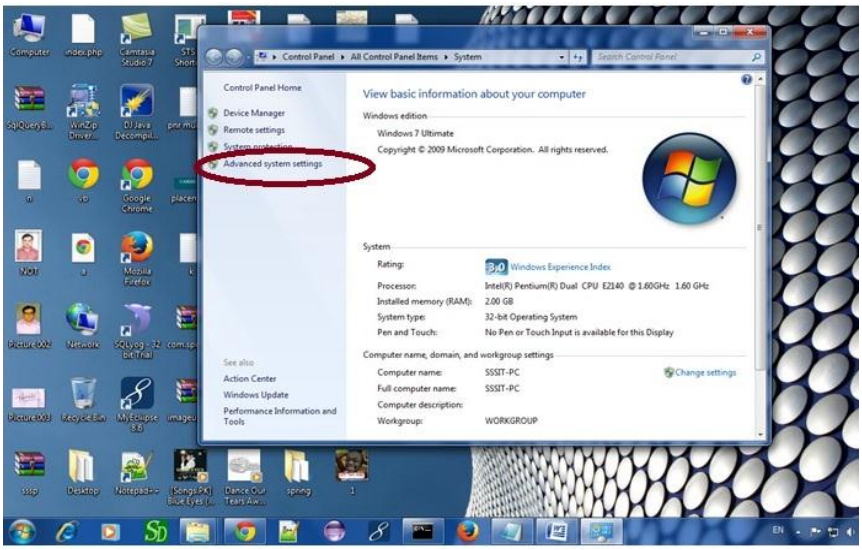
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

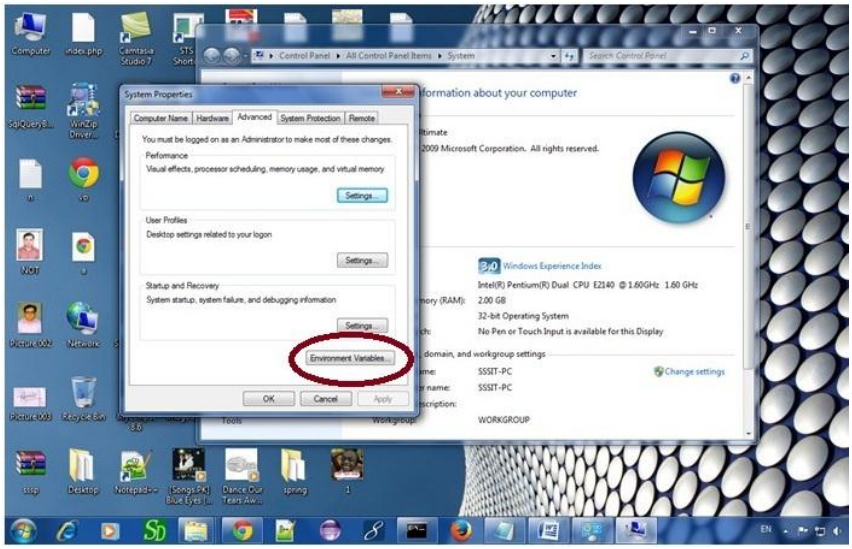
Right click on my computer/ThisPC -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> Then click on ok

Go Computer properties:

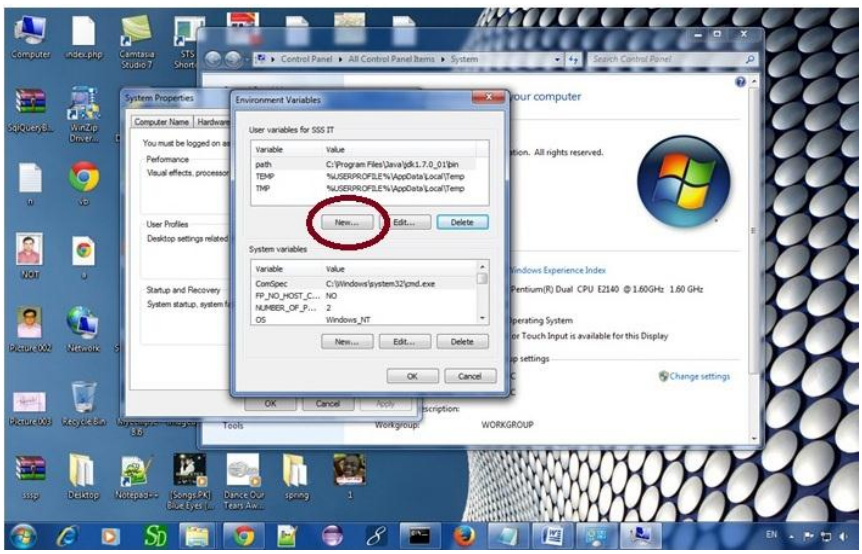


Click on advanced system settings:

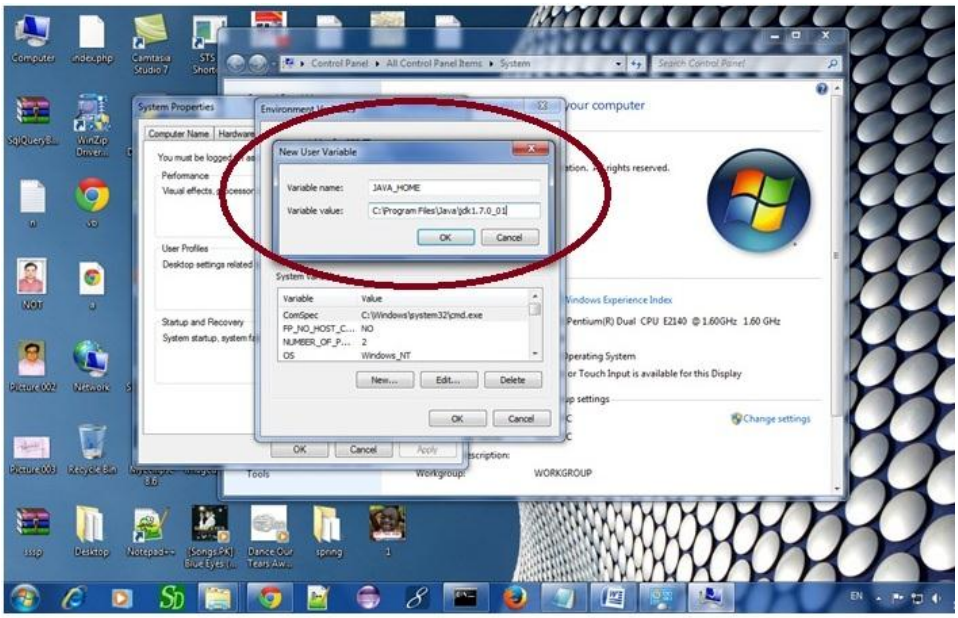


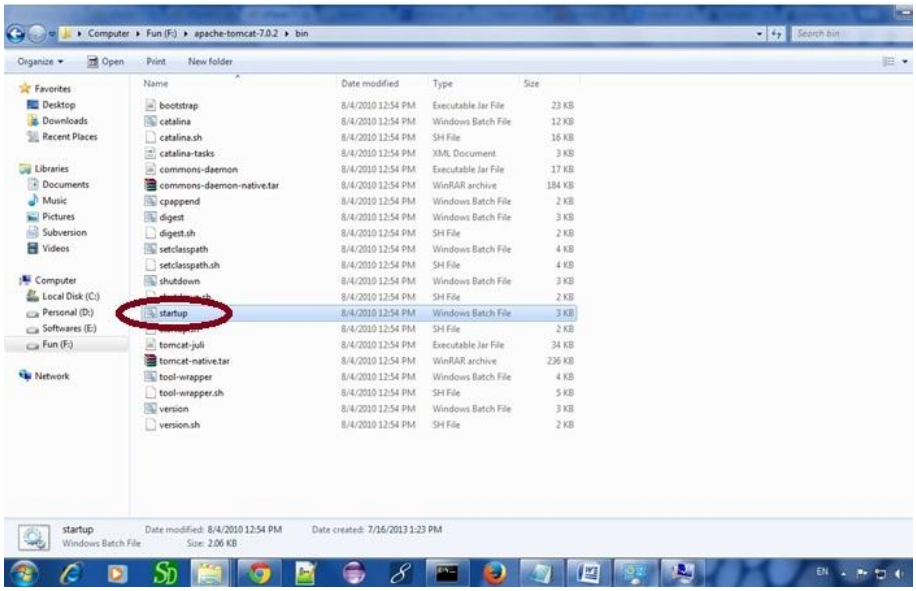


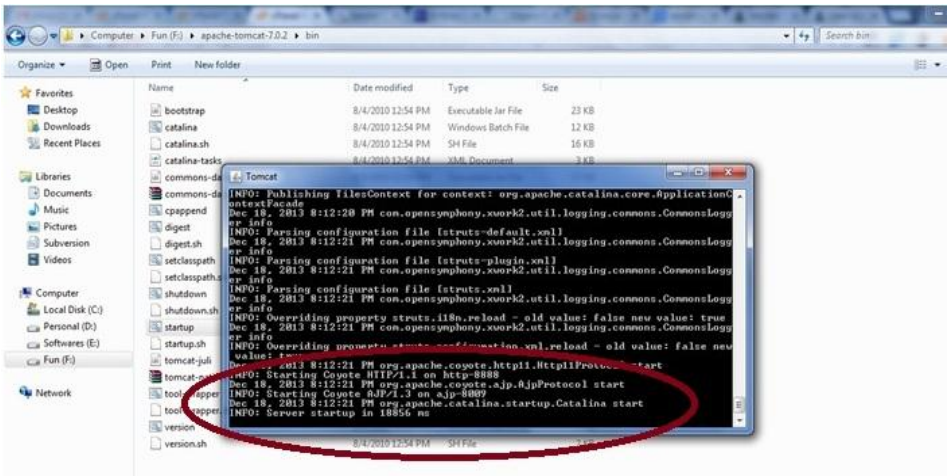
Click on the new tab of user variable or system variable whenever required:



JAVA_HOME in variable name and paste the path of jdk folder in variable value until bin directory:







Finally, the Apache server is started successfully.

2) How to change port number of apache tomcat:

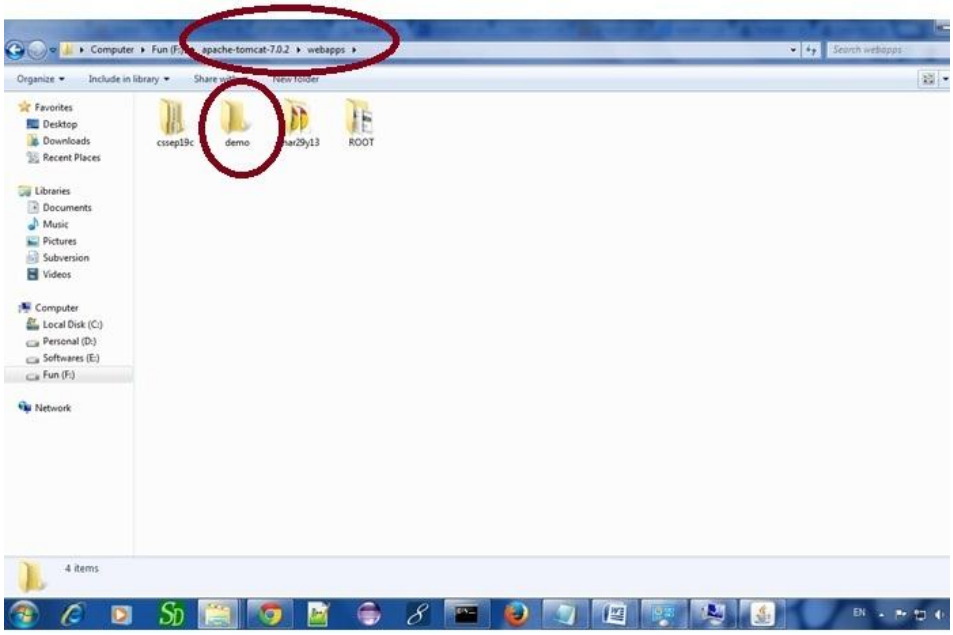
TO change port number is required if there is another server running on the same system with the same port number.

Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

By manually we can change the port no. Open server.xml file in notepad. It is located inside the apache-tomcat/conf directory. TO Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. E.g replace it by 9999 and save this file.

5) How to deploy the servlet project:

Copy the project and paste it in the webapps folder under apache tomcat.



You can also create a war file, and paste it inside the webapps directory. To do so, you need to use a jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

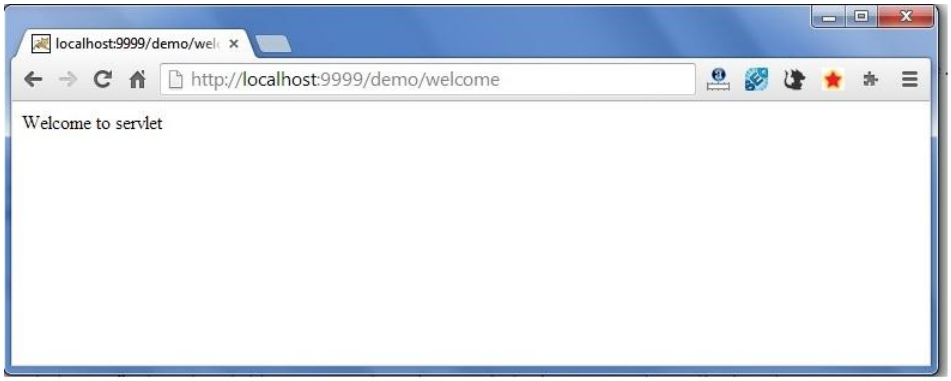
projectfolder> jar cvf myproject.war

Creating a war file has the advantage that moving the project from one location to another takes less time.

6) How to access the servlet class:

Now Open browser and write in the address bar http://hostname:portno/contextroot/urlpatternofservlet. For example:

http://localhost:9999/demo/welcome



2) Example of Student Registration form in servlet:

In this example, we have created the three pages.

- register.html
- Register.java
- web.xml

1) StudentRegister.html:

In this web page, we have input from the user using text fields and Dropdown list. The information entered by the user is forwarded to the Register servlet.

```
<html>
<body>
<form action="/Register" method="post">
Enter Student Name:<input type="text" name="userName"/><br/><br/>
Enter Password:<input type="password" name="userPass"/><br/><br/>
Enter Email Id:<input type="text" name="userEmail"/><br/><br/>
Select Country:
<select name="userCountry">
<option>India</option>
<option>USA</option>
<option>Australia</option>
 <option>Other</option>
</select>
<br/><br/>
<input type="submit" value="register"/>
</form>
```

```
</body>
</html>
```

2) Register.java:

This Java Servlet class File is designed to receive all the data entered by the user and store it into the database. Here, we are performing the database logic.

```
import java.io.*;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
 public class Register extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String uname=request.getParameter("userName");
String upass=request.getParameter("userPass");
String uemail=request.getParameter("userEmail");
String uc=request.getParameter("userCountry");
PrintWriter out = response.getWriter();
 out.println("<p>User name=" +uname + "</p>");
 out.println("<p>User pass=" +upass+ "</p>");
 out.println("<p>User email=" +uemail+ "</p>");
 out.println("<p>User country=" +uc+ "</p>");
out.close();
}
}
```

3) web.xml file:

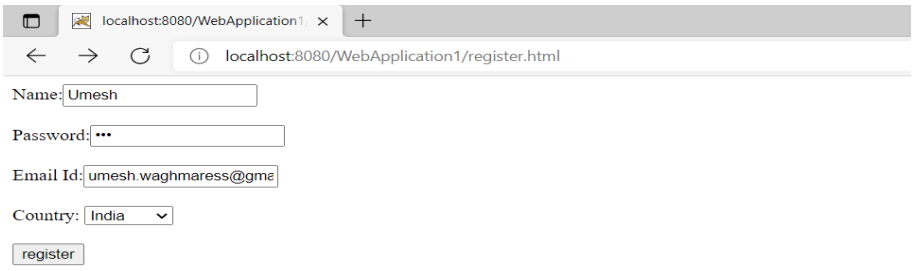
Web.xml is the configuration file, to provide information about the servlet.

```
<web-app>
  <servlet>
<servlet-name>Register</servlet-name>
```

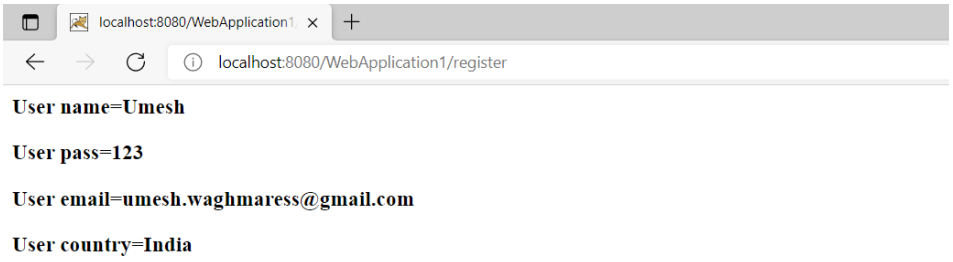
```
<servlet-class>Register</servlet-class>
</servlet>
  <servlet-mapping>
<servlet-name>Register</servlet-name>
<url-pattern>/register</url-pattern>
</servlet-mapping>
  <welcome-file-list>
    <welcome-file>StudentRegister.html</welcome-file>
</welcome-file-list>
  </web-app>
```

Output:

1) Input Screen - StudentRegister.html:



2) Output Screen - Register,java// servlet file executed



4.6 QUESTIONS

1. Define Servlet?
2. Explain in detail the servlet life cycle.
3. Explain API of Servlet.
4. Differentiate between servlet & CGI.

5. Why use servlets?
6. Explain servlet interface.
7. Explain Role of servlet in web application.
8. Explain stages of executing servlet programs.
9. Explain javax.servlet package.
10. Explain javax.servlet.http package.

4.7 SUMMARY

In java servlet technology is used to create a web application (basically resides at server side and generates a dynamic web page as per request).

Java Servlet technology is secure, robust and scalable because of the Java language features. Before Servlet, CGI (Common Gateway Interface) this scripting language was used for server-side programming language. In the servlet API there are many interfaces and classes such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse.

4.8 REFERENCE FOR FURTHER READING

1. The Complete Reference -Java Enterprise Edition (Black Book) , Author:Herbert schildt.
2. Java EE 7 The Big Picture by - Dr.Danny Coward Publisher- Oracle press.
3. Advanced Java by-Balaguruswamy.
4. The Java Ee 7 Tutorial by-Ricardo Cervera-Navarro.

*****

5

WORKING WITH SERVLETS

Unit structure

5.1 OBJECTIVES

1) Java Servlet Provides Server side Programming techniques.
2) In Java Servlet for implementation of XML file or Deployment Descriptor file.
3) We can use Annotation " @webservlet" for there is no requirement for a Deployment Descriptor file i.e web.xml .
4) Java Servlet provides a powerful API for Strong and Dynamic Web Programming.
5) students/professionals are able to originally design and develop & Deploy their application on server.

5.2 JAVA ANNOTATIONS

In java Annotations is a tag that represents the metadata(Data about Data) i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML i.e web.xml file (Deployment Descriptor) and Java marker interfaces.

- **Built-In Java Annotations:**

There are several built-in annotations in Java. Some annotations are applied to Java code and some to other annotations.

Built-In Java Annotations used in Java code:

1. @Override
2. @SuppressWarnings
3. @Deprecated

- **Let's Discuss with Built-In Annotations:**

1) @Override:

@Override annotation assures that the subclass method is overriding the parent class method. If it is not so, a compile time error occurs.

Sometimes, we make silly mistakes such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurity that method is overridden.

```
class Student
{
void learnSomething()
{
System.out.println("Learning  something");
}
}
  class Result extends Student
{
@Override
void learnsomething()
{
System.out.println("Result----”);
}//should be override learnSomething
}
  class TestStudent
{
public static void main(String args[])
{
Student a=new Result();
a.learnSomething();
}
}
```

Output:

Compile Time Error

2) @SuppressWarnings:

@SuppressWarnings annotation: is used to suppress warnings issued by the compiler.

```
import java.util.*;
```

```
class Employee
{
@SuppressWarnings("unchecked")
public static void main(String args[])
{
ArrayList emplist=new ArrayList();
list.add("Umesh");
list.add("Datta");
list.add("Monu");
for(Object obj1:emplist)
System.out.println(obj1);
}
}
```

At Compile Time:

Now no warning at compile time.

If you remove the @SuppressWarnings("unchecked") annotation, it will show a warning at compile time because we are using a non-generic collection.

3) @Deprecated:

@Deprecated annotation marks that this method is deprecated so the compiler prints a warning. It informs users that it may be removed in the future versions. So, it is better not to use such methods.

```
class A
{
void mfun(){System.out.println("hello m");
}
@Deprecated
void nfun()
{
System.out.println("hello n");
}
}
class TestAnnotation3
{
public static void main(String args[])
{
A a=new A();
```

```
a.nfun();
}
}
```

At Compile Time:

Note: Test.java uses or overrides a deprecated API.

5.3 GETTING STARTED WITH SERVLET

Servlet is the key component that forms a typical Java EE application, beside JSP, EJB, XML and other related technologies.

A Java EE application can be packaged/Archived in a WAR file (Web ARchive) in order to be deployed on a web server/Application server. A web server that can run Java servlets is called a servlet container. The most popular and widely used servlet containers are - Apache Tomcat, JBoss, Glassfish etc.

In Java EE servlet is a simple Java class that extends either:

- javax.servlet.GenericServletclass for generic client-server protocol.
- javax.servlet.http.HttpServlet class for HTTP protocol communication purpose.

Java servlet is mostly used for handling HTTP requests & response , by overriding the HttpServlet's doGet(), doPost() methods to handle GET and POST methods to response, respectively.

The servlet container supplies an HttpServletRequest object and HttpServletResponse object for dealing with the handling request and response .

Servlet is usually used in conjunction with JSP for generating dynamic content based on client's requests.

- **Annotations in servlet:**

In Servlet Annotation represents the metadata. It will be prefix "@" symbol in Servlet with Annotation,

When you use annotation (@WebServlet), deployment descriptor (web.xml file) is not required. If we want to execute a servlet using annotations you should have a tomcat7 and above web server to execute the servlet. As it will not run in the previous versions of tomcat. @WebServlet annotation is used to map the servlet with the specified name(URL).

In Java Servlet uses the deployment descriptor (web.xml file) for deploying/hosting your application into a web server. In java Servlet API 3.0 has introduced a new package called " javax.servlet.annotation " . It provides annotation types which can be used for annotating a servlet class.

If you use annotation, then the deployment descriptor (web.xml) is not required. But you should use tomcat7 or any latest version of tomcat.

Annotations can replace equivalent XML file configuration in the web deployment descriptor file (web.xml) such as servlet declaration,servlet class,servlet url and servlet mapping. Servlet containers will process the annotated classes at deployment time.

Following are The list of annotation types introduced in Servlet 3.0.:

Sr. No.	Annotation & Description
1	**@WebServlet:** To declare a servlet.
2	**@WebInitParam:** To specify an initialization parameter.
3	**@WebFilter:** To declare a servlet filter.
4	**@WebListener:** To declare a WebListener
5	**@HandlesTypes:** To declare the class types that a ServletContainerInitializer can handle.
6	**@HttpConstraint:** This annotation is used within the ServletSecurity annotation to represent the security constraints to be applied to all HTTP protocol methods for which a corresponding HttpMethodConstraint element does NOT occur within the ServletSecurity annotation.
7	**@HttpMethodConstraint:** This annotation is used within the ServletSecurity annotation to represent security constraints on specific HTTP protocol messages.
8	**@MultipartConfig:** Annotation that may be specified on a Servlet class, indicating that instances of the Servlet expect requests that conform to the multipart/form-data MIME type.
9	**@ServletSecurity:** This annotation is used on a Servlet implementation class to specify security constraints to be enforced by a Servlet container on HTTP protocol message.

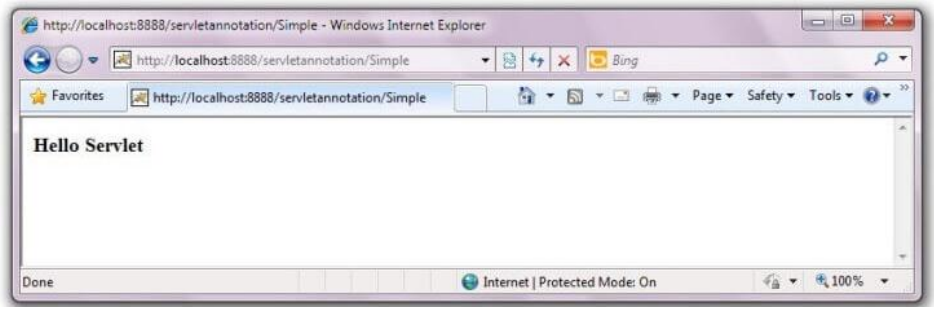
5.4 USING ANNOTATIONS INSTEAD OF DEPLOYMENT DESCRIPTOR

1) A Program of simple servlet by annotation without web.xml file (Deployment Descriptor):

SimpleExample.java:

```
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/SimpleExample")  //Annotations
public class SimpleExample extends HttpServlet
{
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out=response.getWriter();
out.print("<html><body>");
out.print("<h3>Hello Servlet</h3>");
out.print("</body></html>");
}
}
```

Output:



5.5 -SERVLET PROGRAM

2) A Program Design Student Registration process using servlet by annotation without web.xml file:

TO Designing a servlet we have to create the three pages.

1. register.html
2. Register.java

1) StudentRegister.html:

In this page, we have input from the user using text fields and combobox. The information entered by the user is forwarded to the Register servlet.

```
<html>
<body>
<form action="/Register" method="post">
 Enter Student Name:<input type="text" name="userName"/><br/><br/>
Enter Password:<input type="password" name="userPass"/><br/><br/>
Enter Email Id:<input type="text" name="userEmail"/><br/><br/>
Select Country:
<select name="userCountry">
<option>India</option>
<option>USA</option>
<option>Australia</option>
 <option>Other</option>
</select>
<br/><br/>
<input type="submit" value="register"/>
</form>
</body>
</html>
```

2) Register.java:

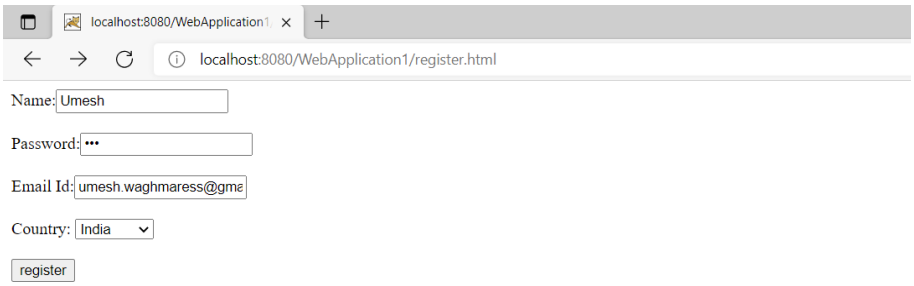
This servlet class receives all the data entered by the user and stores it into the database. Here, we are performing the database logic. But you may separate it, which will be better for the web application.

```
import java.io.*;
import javax.servlet.http.*;
import java.io.IOException;
```

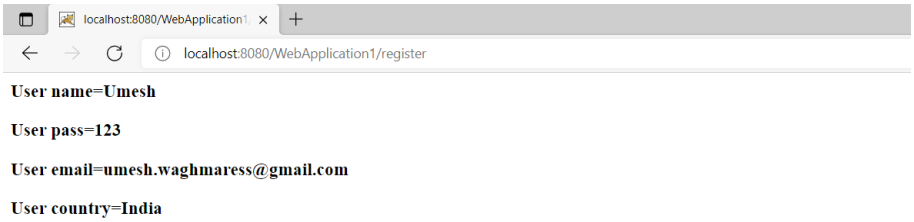
```
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
@WebServlet("/register")
public class Register extends HttpServlet
{
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String uname=request.getParameter("userName");
String upass=request.getParameter("userPass");
String uemail=request.getParameter("userEmail");
String uc=request.getParameter("userCountry");
PrintWriter out = response.getWriter();
out.println("<p>User name=" +uname + "</p>");
out.println("<p>User pass=" +upass+ "</p>");
out.println("<p>User email=" +uemail+ "</p>");
out.println("<p>User country=" +uc+ "</p>");
}
}
```

Output:

1) Input screen: StudentRegister.html



2) Output Screen : Register.java:



5.6 QUESTIONS

1. Explain what is the use of "annotation" in Java.
2. Explain in detail @WebServlet annotation in java.
3. What is needed to use annotations except web.xml.
4. Explain @override annotations in java.
5. Explain @Deprecated annotations in java.
6. Explain in detail the flow of executing annotations.
7. List of annotations in servlet 3.0
8. Write a servlet program for calculate simple & compound interest.
9. Write a servlet program for Display entered Username .

5.7 SUMMARY

Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.

Annotations in Java are used to provide additional information, so it is an alternative option for XML and Java marker interfaces.

The servlet container supplies an HttpServletRequest object and HttpServletResponse object for dealing with the handling request and response.

In Servlet Annotation represents the metadata. It will be prefix "@" symbol in Servlet with Annotation,

When you use annotation (@WebServlet), deployment descriptor (web.xml file) is not required. If we want to execute a servlet using annotations you should have a tomcat7 and above web server to execute the servlet.

5.8 REFERENCE FOR FURTHER READING

1. The Complete Reference -Java Enterprise Edition (Black Book), Author:Herbert schildt.
2. Java EE 7 The Big Picture by - Dr. Danny Coward Publisher- Oracle press.
3. Advanced Java by-Balaguruswamy.

*****

6

WORKING WITH DATABASES

Unit structure

6.1 OBJECTIVES

1) JDBC is useful for A DataSource object that is used to establish connections.
2) Using Objects of JDBC classes & interfaces to be executing SQL Statements.
3) Extracting metadata of a data source via JDBC driver.
4) JDBC Driver Manager can also be used to establish a connection between Java & Database .
5) Using JDBC API we can Design & Develop GUI Web Application in Java.

6.2 WHAT IS JDBC

JDBC stands for Java Database Connectivity JDBC basically used in java programming for create/implement connection between java application & database. Java API provides JDBC Driver is a software component / module that enables java applications to connect with the database. JDBC is a Java API to connect and execute the SQL query with the database. JDBC API uses JDBC drivers to connect with the database and java application.

Following are the types of JDBC drivers:

1. JDBC-ODBC Bridge Driver,
2. Native Driver,
3. Network Protocol Driver, and

4. Thin Driver:

In a java application we can use JDBC API to access tabular/structured data stored in any relational database. With the help of JDBC API, we can save, update, delete and fetch data from the database. JDBC like Open Database Connectivity (ODBC) provided by Microsoft.

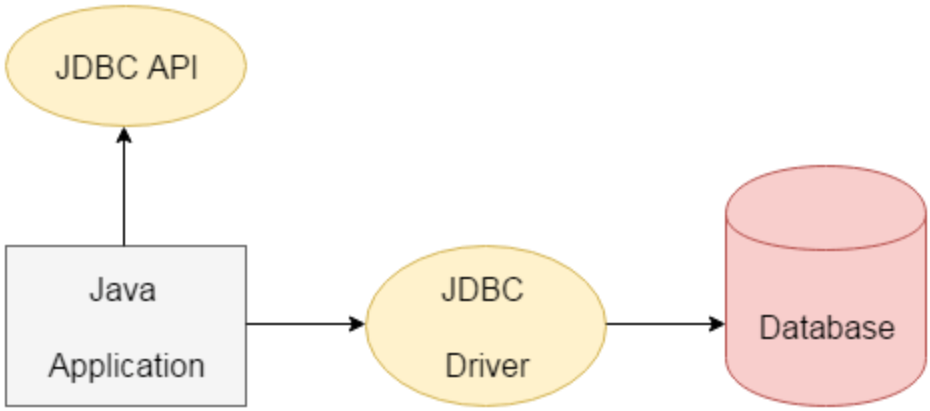


Fig: 6.2 JDBC Connection

The java.sql package contains classes and interfaces for JDBC API.

How to import java.sql package in to java application

E.g Import java.sql.*;

Following are the list of popular interfaces of JDBC API :

1. Driver interface:

In JDBC API the JDBC Driver interface provides powerful implementations of the abstract classes provided by the JDBC API for connection. JDBC driver must provide implementations of the java.sql. CallableStatement, PreparedStatement, Connection, Statement, ResultSet.

2. Connection interface:

In JDBC API a Connection interface is the session between a Java application and database. The Connection interface is a factory of PreparedStatement, Statement, and DatabaseMetaData, that means objects of Connection can be used to get the object of Statement and DatabaseMetaData.

Commonly used methods of Connection interface:

A) public Statement createStatement():

In Java creates a statement object that can be used to execute SQL queries.

B) public Statement createStatement(int resultSetType,int resultSetConcurrency):

Creates a Statement object that will generate ResultSet objects with the given type.

C) public void close():

closes the connection and Releases a JDBC resources immediately.

1. Statement interface:

The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides a factory method to get the object of ResultSet.

Following are the important methods of Statement interface:

A) public ResultSet executeQuery(String sql):

In Java is used to execute SELECT query. It returns the object of ResultSet when executing the query.

B) public int executeUpdate(String sql):

In Java is used to execute specified query, it may be create, drop, insert, update, delete.

C) public boolean execute(String sql): is used to execute queries that may return multiple results from a database.

1. PreparedStatement interface:

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized queries.

example of parameterized query:

String sql="insert into emp values(?,?,?)";

In the above example , we are passing parameters (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement?:

- **Improves performance:** The performance of the application will be faster if you use PreparedStatement interface because the query is compiled only once.

Methods of PreparedStatement interface.

Following are the important methods of PreparedStatement interface are given below:

Method	Description
public void setInt(int paramIndex, int value)	In This method sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	In This method sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	In This method sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	In This method sets the double value to the given parameter index.
public int executeUpdate()	In This method executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	In This method executes the select query. It returns an instance of ResultSet.

2. CallableStatement interface:

In JDBC API CallableStatement interface is used to call the stored procedures and functions. We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled.

Suppose you need to get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

Following are the differences between stored procedures and functions:

Procedure	Function
is used to perform business logic.	is used to perform calculations.
must not have the return type.	must have the return type.
may return 0 or more values.	may return only one value.
We can call functions from the procedure.	Procedure cannot be called from function.
Procedure supports input and output parameters.	Function supports only input parameters.
Exception handling using try/catch block can be used in stored procedures.	Exception handling using try/catch can't be used in user defined functions.

3. ResultSet interface:

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, the cursor points to the first row.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method as well as we can make this object as updatable by:

Following are the Commonly used methods of ResultSet interface:

Methods	Description
1) public boolean next():	This method is used to move the cursor to the one row next from the current position.
2)public boolean previous():	This method is used to move the cursor to the one row previous from the current position.
3) public boolean first():	This method is used to move the cursor to the first row in the result set object.
4) public boolean last():	This method is used to move the cursor to the last row in the result set object.
5) public boolean absolute(int row):	This method is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	This method is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	This method is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	This method is used to return the data of the specified column name of the current row as int.

9) public String getString(int columnIndex):	This method is used to return the data of the specified column index of the current row as String.
10) public String getString(String columnName):	This method is used to return the data of the specified column name of the current row as String

Following are the a popular classes of JDBC API:

DriverManager class:

In JDBC API the DriverManager class acts as an interface between user and drivers for implementing bridge between Java Application & Database . It keeps track of the drivers that are available and handles establishing a connection between a database and the driver.

Following are the methods of DriverManager class:

Method	Description
1) public static void registerDriver(Driver driver):	This method is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	This method is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	This method is used to establish the connection with the specified url.
4) public static Connection getConnection(String url, String userName, String password):	This method is used to establish the connection with the specified url, username and password.

6.3 JDBC ARCHITECTURE

The JDBC API supports both two-tier and three-tier architecture of sql processing models for database access data .

6.3.1 Two-tier Architecture:

In the JDBC two-tier architecture model, a Java applet or application talks directly to the data source. This requires a JDBC driver that can communicate with the particular data source/database being accessed.

A client sends a request to execute the database or other data source, and the results of those statements are sent back to the user. The data source may be located on another machine to which the user is connected via a network. This is referred to as a client/server configuration where they can communicate to each other .

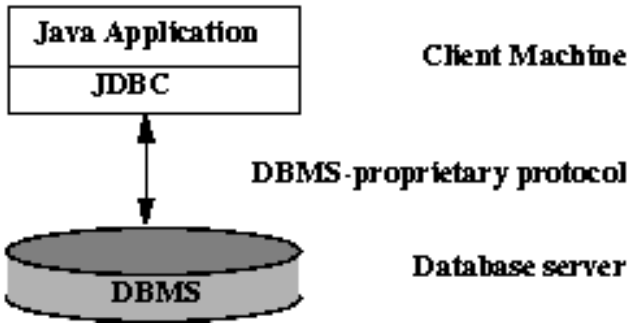


Figure 1: 6.3.1 Two-tier Architecture for Data Access.

6.3.2 Three-tier Architecture:

In Java JDBC is a three-tier architecture model, commands are sent to a "middle tier" of services, which then sends the commands to the data source/database. The data source /database processes the commands and sends the results back to the middle tier, which then sends them to the user.

In this three-tier model is very attractive because the middle tier makes it possible to maintain control over access and the kinds of updates that can be made to corporate data. One more advantage is that it simplifies the deployment of javabapplications. Three-tier architecture can provide performance advantages.

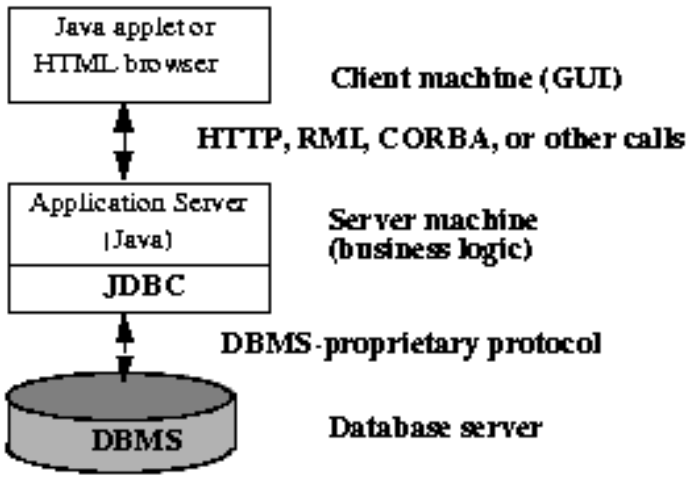


Figure 2: 6.3.2 Three-tier Architecture for Data Access.

The JDBC API is being used more and more in the middle tier of a three-tier architecture. In JDBC various features that make JDBC a server technology are its support for connection pooling, distributed transactions. The JDBC API is also what allows access to a data source/database from a Java middle tier.

6.4 ACCESSING DATABASE

A) Insert Records in Database(MySql):

In this example how to insert records in a table using the JDBC application.

Required Steps:

The following are the steps to create a new Database using JDBC application:

1. **Import the packages:** import keywords specify that you include the packages containing the JDBC classes needed for database programming. Most often, using import java.sql.* will suffice.

2. **Register the JDBC driver:** Requires that you initialize a driver so you can open a communications channel with the database.

3. **Open a connection:** Open() method used in the DriverManager.getConnection() method to create and open a Connection object, which represents a physical connection with a database .

1. **Execute a query:** execute() method used for an object of type Statement for building and submitting an SQL statement to insert records into a desired table.

2. Cleaning up the environment with resources automatically closes the resources.

Sample Code

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;
public class InsertData
 {
String DB_URL = "jdbc:mysql://localhost/Umesh";
String USER = "guest";
String PASS = "guest123";
public static void main(String[] args)
```

```
{
// Open a connection
Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
Statement stmt = conn.createStatement();
Try
{
// Execute a query
System.out.println("Inserting records into the table...");
String sql = "insert into  UserRegistration VALUES (100, 'Umesh', 'Waghmare', 30)";
stmt.executeUpdate(sql);
sql = "insert into  UserRegistration VALUES (101, 'Shreya', 'Patil', 25)";
stmt.executeUpdate(sql);
sql = "insert into  UserRegistration VALUES (102,'Nilima', 'Adagale', 30)";
stmt.executeUpdate(sql);
sql = "insert into  UserRegistration VALUES(103, 'Mimu', 'Mittal', 25)";
stmt.executeUpdate(sql);
System.out.println("Inserted records into the table...");
}
catch (SQLException e)
{
e.printStackTrace();
}
}
}
```

Now let us compile the above example as follows −

```
C:\>javac InsertData.java
C:\>
```

When you run InsertData, it produces the following result −

```
C:\>java InsertData
Inserting records into the table...
Inserted records into the table...
C:\>
```

B) Retrieve/Access Records from Database(MySql):

In java JDBC API specify to select/ fetch records from a table using JDBC Driver/application.

Following are the Required Steps:

The following steps are required to create a new Database using JDBC application:

1. **Import the packages:** import keywords that include the packages containing the JDBC classes needed for database related programming. E.g import java.sql.*;
2. **Open a connection:** Open() method used in the DriverManager.getConnection() method to create a Connection object, which represents a physical connection with a database .
3. **Execute a query:** executeQuery() method used as an object of type Statement for building and submitting an SQL statement to select (i.e. fetch/access) records from a table.
4. **Extract Data:** SQL query is executed successfully , you can fetch/access records from the table.
5. **Clean up the environment:** try with resources automatically.

Sample Code

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class AccessData
 {
String DB_URL = "jdbc:mysql://localhost/umesh";
String USER = "guest";
String PASS = "guest123";
String QUERY = "SELECT id, first, last, age FROM Registration";
 public static void main(String[] args)
{
// Open a connection
Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
Statement stmt = conn.createStatement();
```

```
ResultSet rs = stmt.executeQuery(QUERY);
try {
while(rs.next())
   //Display values
System.out.print("ID: " + rs.getInt("id"));
System.out.print(", Age: " + rs.getInt("age"));
System.out.println(", Last: " + rs.getString("last"));
}
}
catch (SQLException e)
{
e.printStackTrace();
}
}
}
```

Execute example as follows:

```
C:\>javac AccessData.java
C:\>
```

When you run AccessData, it produces the following result:

```
C:\>java AccessData
ID: 100, Age: 30, First: Umesh, Last: Waghmare
ID: 101, Age: 25, First: Shreya, Last: Patil
ID: 102, Age: 30, First: Nilima, Last: Adagale
ID: 103, Age: 25, First: Mimu, Last: Mittal
```

6.5 THE SERVLET GUI AND DATABASE EXAMPLE:

To start with basic concept, let us create a simple table and create few records in that table as follows −

Create Table

To create the Employees table in TEST database, use the following steps –

Step 1:

Open a Command Prompt and change to the installation directory as follows −

```
C:\>
```

C:\>cd Program Files\MySQL\bin

C:\Program Files\MySQL\bin>

Step 2:

Login to database as follows

C:\Program Files\MySQL\bin>mysql -u root -p

Enter password: ********

mysql>

Step 3:

Create the table Student in teststudent database as follows −

mysql> use teststudent;

mysql> create table Student (

id int not null,

age int not null,

first varchar (255),

last varchar (255)

);

Query OK, 0 rows affected (0.08 sec)

mysql>

Create Data Records

Finally you create few records in Student table as follows −

mysql> INSERT INTO Student VALUES (100, 30, 'Umesh', 'Waghmare');

Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Student VALUES (101, 25, 'Shreya', 'Patil');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Student VALUES (102, 30, 'Nilima', 'Adagale');

Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Student VALUES (103, 28, 'Mimu', 'Mittal');

Query OK, 1 row affected (0.00 sec)

mysql>

Accessing a Database

Here is an example which shows how to access teststudent database using Servlet.

// Loading required libraries

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;
 public class DatabaseAccess extends HttpServlet
 {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
// JDBC driver name and database URL
String JDBC_DRIVER = "com.mysql.jdbc.Driver";
String DB_URL="jdbc:mysql://localhost/teststudent";
//  Database credentials
String USER = "root";
String PASS = "root";
// Set response content type
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "Database Result";
 out.println(
"<html>\n" +
"<head><title>" + title + "</title></head>\n" +
"<body bgcolor = \"#f0f0f0\">\n" +
"<h1 align = \"center\">" + title + "</h1>\n");
try
{
// Register JDBC driver
Class.forName("com.mysql.jdbc.Driver");
  // Open a connection
Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
// Execute SQL query
Statement stmt = conn.createStatement();
String sql;
```

```
sql = "SELECT id, first, last, age FROM Student";
ResultSet rs = stmt.executeQuery(sql);
// Extract data from result set
while(rs.next())
{
//Retrieve by column name
int id  = rs.getInt("id");
int age = rs.getInt("age");
String first = rs.getString("first");
String last = rs.getString("last");
//Display values
out.println("ID: " + id + "<br>");
out.println(", Age: " + age + "<br>");
out.println(", First: " + first + "<br>");
out.println(", Last: " + last + "<br>");
}
out.println("</body></html>");
// Clean-up environment
rs.close();
stmt.close();
conn.close();
}
catch(Exception e)
 {
//Handle errors for Class.forName
e.printStackTrace();
}
}
}
```

Now let us compile above servlet and create following entries in web.xml

```
....
<servlet>
<servlet-name>DatabaseAccess</servlet-name>
<servlet-class>DatabaseAccess</servlet-class>
</servlet>
```

<servlet-mapping>

<servlet-name>DatabaseAccess</servlet-name>

<url-pattern>/DatabaseAccess</url-pattern>

</servlet-mapping>

Now call this servlet using URL http://localhost:8080/DatabaseAccess which would display following response −

Database Result:

ID: 100, Age: 30, First: Umesh, Last: Waghmare

ID: 101, Age: 25, First: Shreya, Last: Patil

ID: 102, Age: 30, First: Nilima, Last: Adagale

ID: 103, Age: 25, First: Mimu, Last: Mittal

Example of Fetching Result for the given country:

Here, you will learn how to fetch results for the given country. I am assuming that there is a table as given below:

```
CREATE TABLE `registeruser` (
  `uname` varchar(50) NOT NULL,
  `upass` varchar(45) DEFAULT NULL,
  `email` varchar(45) DEFAULT NULL,
  `country` varchar(45) DEFAULT NULL
);
```

We are assuming there are many records in this table. In this example, we are getting the data from the database in servlet and Display on screen.

In this example, we have create three files:

1. index.html
2. Search.java
3. web.xml

1) index.html:

This page gets rollno from the user and forwards this data to servlet which is responsible to show the records based on the given country.

```
<html>
<body>
<form action="servlet/Search">
Enter your Country:<input type="text" name="country"/><br/>
```

```
<input type="submit" value="search"/>
</form>
</body>
</html>
```

2) Search.java:

This is the servlet file which gets the input from the user and maps this data with the database and prints the record for the matched data. In this page, we are displaying the column name of the database along with data, so we are using the ResultSetMetaData interface.

```
import java.io.*;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
public class Search extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
{
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String country=request.getParameter("country");
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/teststudent","root","root");
//here teststudent is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from registeruser where country like +country+");
while(rs.next())
System.out.println(rs.getString(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}
catch(Exception e)
{
```

```
System.out.println(e);
}
}
}
```

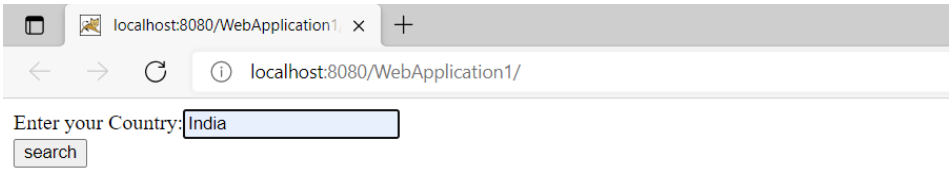
3) web.xml file

This is the configuration file which provides information of the servlet to the container.

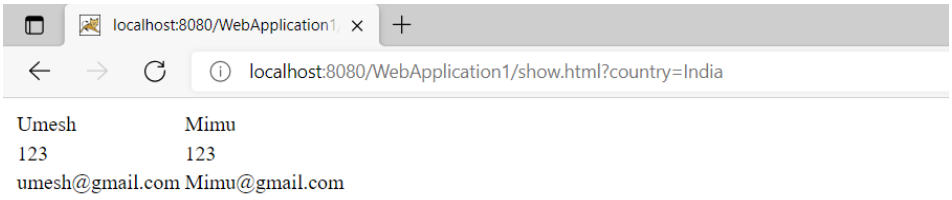
```
<web-app>
<servlet>
<servlet-name>Search</servlet-name>
<servlet-class>Search</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Search</servlet-name>
<url-pattern>/servlet/Search</url-pattern>
</servlet-mapping>
</web-app>
```

Output:

1) Input Screen



2) Output Screen:



6.7 SUMMARY

JDBC is basically used in java programming for creating/implementing connections between java applications & databases. In Java API JDBC Driver is a software component that enables java applications to connect with the database. JDBC is a Java API to connect and execute the SQL

query with the database. JDBC is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database.

6.6 QUESTIONS

1 What is the use of JDBC?

2 Explain Architecture of JDBC.

3 Explain Two Tier Architecture.

4 Explain types of JDBC drivers.

5 Explain Three Tier Architecture.

6 Discuss steps to connect java to a database.

7 Design a Employee registration (HTML form) insert records in the EmployeeDetails table and display all records.

8 Design Result (HTML form) and a servlet program for calculating student results with grades and display on browser.

6.8 REFERENCE FOR FURTHER READING

1. The Complete Reference -Java Enterprise Edition (Black Book), Author:Herbert schildt.

2. Java EE 7 The Big Picture by - Dr.Danny Coward Publisher- Oracle press.

3. Advanced Java by-Balaguruswamy .

*****

UNIT II

7

REQUEST DISPATCHER

Unit Structure

7.0 OBJECTIVES

The servlet container forms a RequestDispatcher object, which is used as a cover around a server resource located in a particular path or named. This link is intended for wrapping servlets, but the servlet container can create RequestDispatcher items to wrap any type of app.

7.1 REQUESTDISPATCHER INTERFACE

- A RequestDispatcher is an extremely important Java class that allows you to "include" content in a request / response or to "forward" a request / response to a resource. As a typical example, a servlet can use a RequestDispatcher to include or forward a request / response to a JSP.

- A RequestDispatcher object can forward a client's request to a resource or include the resource itself in the response to the client. A resource can be another servlet, an HTML file or a JSP file, and so on.

- You can also think of a RequestDispatcher object as a container for the resource located on a given path given as an argument to the getRequestDispatcher method.

- To construct a RequestDispatcher object, you can use the ServletRequest.getRequestDispatcher () method or the ServletContext.getRequestDispatcher () method. They both do the same thing, but impose slightly different constraints on the path of the topic. For the former, it looks for the resource in the same web application that the invocation servlet belongs to, and the path name specified can be relative to the invocation servlet. For the latter, the path name must start with '/' and is interpreted in relation to the root of the web application.

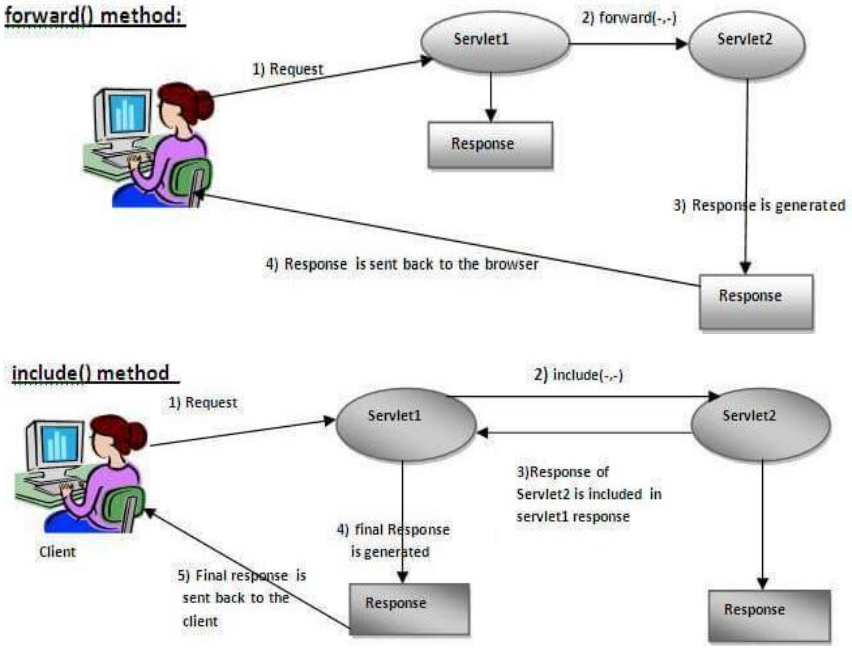
7.2 METHODS OF THE REQUESTDISPATCHER INTERFACE

- public void forward (ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:

  Forwards a request for a servlet to another resource (servlet, JSP file, or HTML file) on the server.

- public void include (ServletRequest request, ServletResponse response) throws ServletException, java.io.IOException:

Include the content of a resource (servlet, JSP page, or HTML file) in the response.

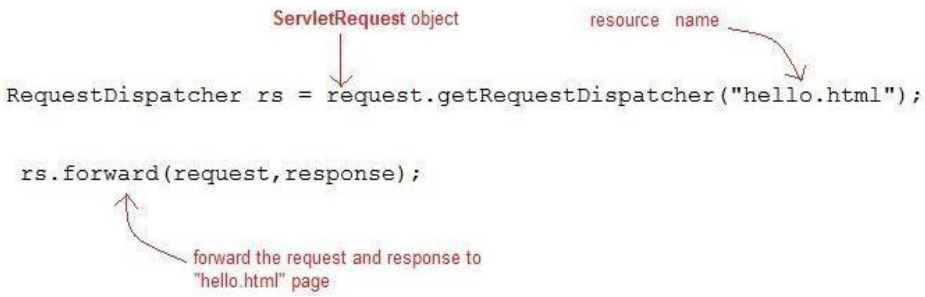


How to get the RequestDispatcher object:

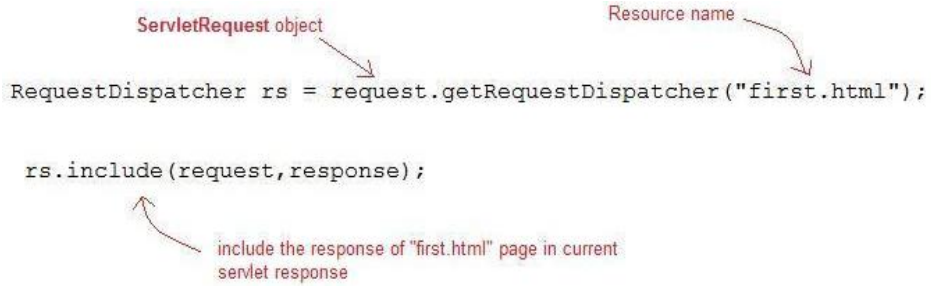
- The getRequestDispatcher () method of the ServletRequest interface returns the RequestDispatcher object.

How to get a RequestDispatcher object:

- RequestDispatcher rs = request.getRequestDispatcher ("hello.html"); rs.forward (request, response);



RequestDispatcher rs = request.getRequestDispatcher ("hello.html"); rs.include (request, response);

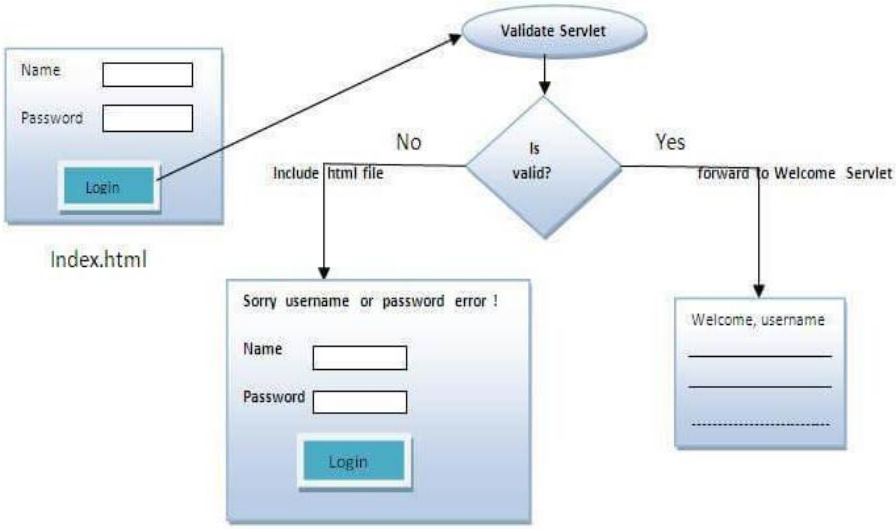


7.3 REQUEST THE SENDER'S REQUEST

- In this example, we are validating the password entered by the user. If the password is a servlet, it will forward the request to the WelcomeServlet; otherwise it will display an error message: sorry, wrong username or password!

In this example, we have created the following files:

- **index.html file:** to obtain information about the user.
- **Access file.java:** a servlet class to process the response. If the password is admin123, it will send the request to the welcome servlet.
- **WelcomeServlet.java file:** a servlet class to display the welcome message.
- **web.xml file:** a deployment descriptor file that contains servlet information.



index.html:

- <form method = "post" action = "Validation"> Name: <input type = "text" name = "user">

- Password: <input type = "password" name = "pass">
- <input type = "submit" value = "submit"> </form>

Validate.java:

```
import java.io. *;
import javax.servlet. *;
import javax.servlet.http. *;
Validate public class extends HttpServlet {
protected void doPost (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
 response.setContentType ("text / html; character set = UTF-8");
 PrintWriter out = response.getWriter ();
 to deal {
 String name = request.getParameter ("user");
 Password string = request.getParameter ("pass");
 if (password.equal ("admin123"))
 {
 RequestDispatcher rd = request.getRequestDispatcher ("Welcome");
 rd.forward (request, response);
 } the rest
 {
 out.println ("<font color = 'red'> <b> You entered an incorrect password </b> </font>");
 RequestDispatcher rd = request.getRequestDispatcher ("index.html");
 rd.include (request, response);
 }
 } Long last {
 out.close ();
 }
}
}
```

Welcome.java:

```
import java.io. *;
import javax.servlet. *;
import javax.servlet.http. *;
The public class welcome extends HttpServlet
```

```
{
protected void doPost (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException
{
 response.setContentType ("text / html; character set = UTF-8");
PrintWriter out = response.getWriter ();
to deal
{
out.println ("<h2> Welcome user </h2>");
}
Long last
{
out.close (); }}}
```

7.4 SUMMARY

The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the ways of servlet collaboration.

7.5 REFERENCES

- https://www.javatpoint.com/java-tutorial
- https://www.tutorialspoint.com/java/index.htm
- https://www.geeksforgeeks.org/java/
- https://www.oracle.com/in/java/technologies/java-ee-glance.html
- https://developers.redhat.com/topics/enterprise-java
- https://www.javacodegeeks.com/enterprise-java-tutorials

7.6 QUESTIONS

1. Explain the RequestDispatcher interface with its methods and diagram.
2. How to get the RequestDispatcher object with syntax.
3. Provide an example to validate the password entered by the user.

*****

8

COOKIES

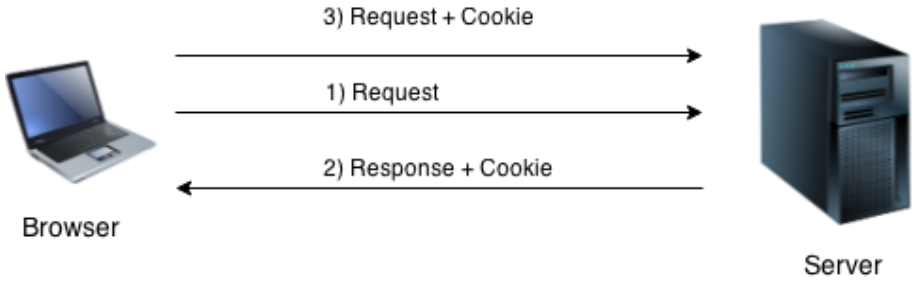
Unit Structure

8.1 INTRODUCTION

- Cookies are text files stored on the customer's computer and are saved for various information tracking purposes. Java Servlet transparently supports HTTP cookies.
- There are three steps involved in identifying recurring users:
- The server script sends a series of cookies to the browser. For example, name, age or identification number, etc.
- The browser stores this information on the local machine for future use.
- The next time the browser sends a request to the web server, it will send the cookie information to the server and the server will use that information to identify you.
- A servlet will access the cookie via the request.getCookies () request method which returns an array of Cookie objects.

Some of the common uses of cookies are:

1. Session authentication uses cookies:
2. Personalized response to the customer based on their preferences, for example, we can set the background color as a cookie in the customer's browser and then use it to customize the background color of the response, image, etc.



8.1.1 Types of cookies:

There are 2 types of cookies in servlets:

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie:

It is valid for one session only. It is removed every time the user closes the browser.

Persistent cookie:

It is valid for multiple sessions. It is not removed every time the user closes the browser. It is removed only if the user logs out or logs out.

Advantage of cookies:

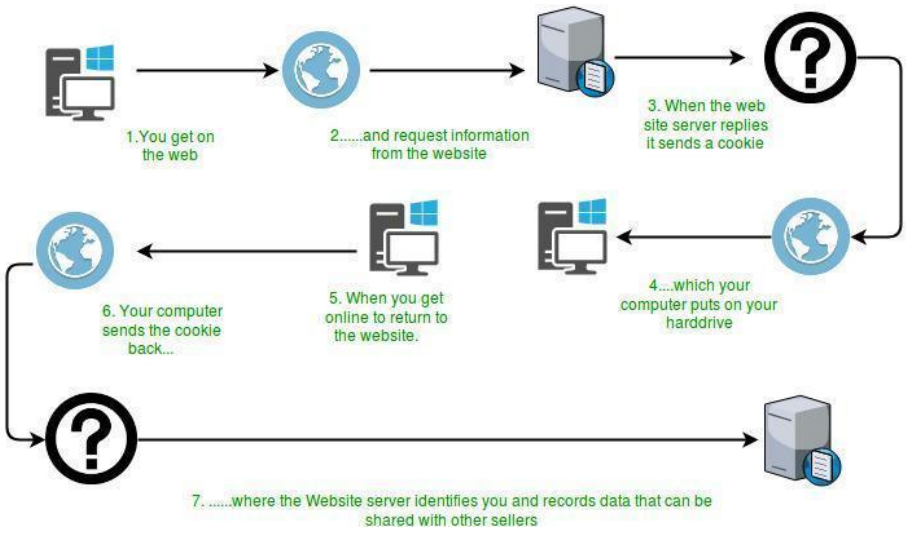
1. The simplest technique for maintaining the state.
2. Cookies are stored on the client side.

Disadvantage of cookies:

1. It will not work if cookies are disabled in the browser.
2. Only textual information can be set in the Cookie object.
3. Cookie class
4. javax.servlet.http.Cookieclass provides the functionality of using cookies. Provides many useful methods for cookies.
5. Cookie class constructor

Builder	Description
Cracker()	build a biscuit.
Cookie (string name, string value)	constructs a cookie with a specified name and value.

1.2 How do cookies work?



By default, each request is considered a new request. In the cookie technique, we add cookies with the response of the servlet. The cookies are then stored in the browser cache. Subsequently, if the user submits the request, the cookie is added with the request by default. Therefore, we recognize the user as the previous user.

8.1.3 Cookie Servlet Modes:

public void setDomain (string schema)

- This method sets the domain to which the cookie applies, for example google.com.

public String getDomain ()

- This method gets the domain to which the cookie applies, for example google.com.

public void setMaxAge (expiration int)

- This method sets the time (in seconds) that must elapse before the cookie expires.

public int getMaxAge ()

- This method returns the maximum age of the cookie, specified in seconds. By default, -1 indicates that the cookie will persist until the browser is deactivated.

public String getName ()

- This method returns the name of the cookie. The name cannot be changed after creation.

public void setValue (String newValue)

- This method sets the value associated with the cookie.

public String getValue ()

- This method gets the value associated with the cookie.

public void setPath (String uri)

- This method sets the path to which this cookie applies.

public String getPath ()

- This method gets the path to which this cookie is applied.

public void setSecure (boolean flag)

- This method sets the Boolean value that indicates whether the cookie should be sent only over encrypted connections (i.e. SSL).

public void setComment (purpose of string)

- This method specifies a comment that describes the purpose of a cookie. The comment is useful if the browser presents the cookie to the user.

public string getComment ()

- This method returns the comment describing the purpose of this cookie or null if the cookie contains no comments.

8.2 COOKIE SETTINGS WITH SERVLET

Setting up cookies with servlets involves three steps:

(1) Creation of a Cookie object:

You call the cookie constructor with a cookie name and a cookie value, both are strings.

Cookie cookie = new Cookie ("name", "value");

Note that neither the name nor the value must contain spaces or any of the following characters: [] () =, "/? @:;

(2) Set the maximum age:

Use setMaxAge to specify how long (in seconds) the cookie should be valid. The following would set a cookie for 24 hours.

cookie.setMaxAge (60 * 60 * 24);

(3) Sending the cookie to the HTTP response headers:

Use response.addCookie to add cookies in the HTTP response header as follows

response.addCookie (cookie);

8.3 WHERE ARE COOKIES USED?

- Create a temporary session where the site remembers in some way in the short term what the user was doing or had chosen from the requests of the web page, for example:
- Who the user is currently logged into.
- What the user has ordered in an online shopping cart.
- To remember low-security information more permanently, for example:
- A user's search results preferences.
- What topic did the user browse during the user's last visit?
- For advertising purposes or to improve the functionality of a site.
- Identify a user during an e-commerce session.
- To avoid entering your username and password to access the site.

8.3.1 Applications:

- Shopping cart request
- Bank online
- Generation of a visitor's profile
- Generation of a visitor's profile
- Website monitoring

8.4 WHERE ARE COOKIES PLACED?

- By default, all cookies generated are stored on the hard drive of the user's local computer.
- The locations are different but the cookie format is the same.
- Use the search function to get the cookie directory.
- It expires after a certain time.

How to create a cookie?:

Cookie ck =new one Cracker("user", "sonoo jaiswal"); // creation of a cookie object

response.addCookie (ck);// adding cookies in response

How to delete the cookie?

Cookie ck =new one Cracker("Username", ""); // removing the value of the cookie

ck.setMaxAge (0); // change the maximum age to 0 seconds

response.addCookie (ck);// adding cookies in response

How to get cookies?

Cookie ck [] = request.getCookies ();

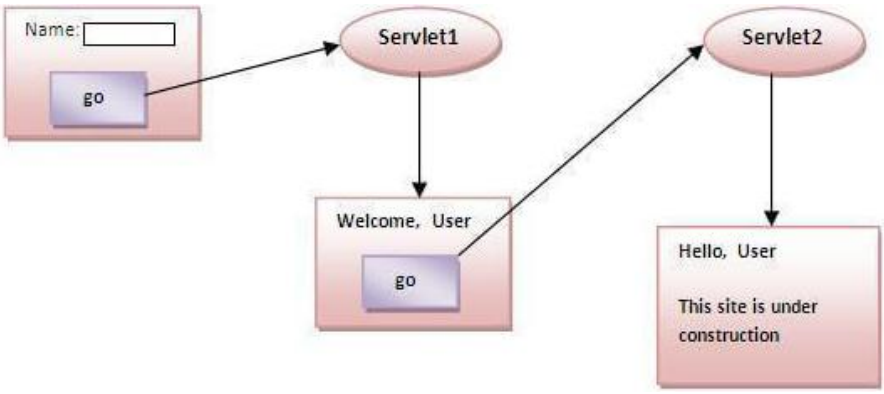
from(In t i =0; i <ck. length; i ++) {

out.print"
" + ck [i] .getName () + "" + ck [i] .getValue ()); // print the name and value of the cookie

}

8.5 SIMPLE EXAMPLE OF A SERVLET COOKIE

In this example, we are storing the user's name in the cookie object and accessing it in another servlet. As we all know, that session corresponds to the particular user. So, if you log in from too many browsers with different values, you will get a different value.



index.html

<action form =method "servlet1" = "publish">

Name: <input type ="text" name = "userName" />

<input type =value "send" = "go" />

</form>

FirstServlet.java

importjava.io. *;

importjavax.servlet. *;

importjavax.servlet.http. *;

public class FirstServlet it extends HttpServlet {

public empty doPost (HttpServletRequest request, HttpServletResponse response) {

to deal{

response.setContentType ("text / html");

PrintWriter out = response.getWriter ();

```
String n = request.getParameter ("Username");
out.print"Welcome" + n);
Cookie ck =new one Cracker("one", n); // creation of a cookie object
response.addCookie (ck);// adding cookies in response
 // create the submit button
out.print"<form action = 'servlet2'>");
out.print"<input type = 'send' value = 'go'>");
out.print"</form>");
out.close ();
}capture(Exception e) {System.out.println (e);}
}
}
```

SecondServlet.java

```
importjava.io. *;
importjavax.servlet. *;
importjavax.servlet.http. *;
public class SecondServlet it extends HttpServlet {
public empty doPost (HttpServletRequest request, HttpServletResponse response) {
 to deal{
 response.setContentType ("text / html");
 PrintWriter out = response.getWriter ();
 Cookie ck [] = request.getCookies ();
 out.print"Hello" + ck [0] .getValue ());
 out.close ();
 } capture(Exception e)
{
System.out.println (e);
}
 }
}
```

8.6 SUMMARY

A cookie is a bit of information sent by a web server to a browser that can later be read back from that browser. When a browser receives a cookie, it saves the cookie and thereafter sends the cookie back to the server each

time it accesses a page on that server, subject to certain rules. Because a cookie's value can uniquely identify a client, cookies are often used for session tracking.

8.7 REFERENCES

- https://www.javatpoint.com/java-tutorial
- https://www.tutorialspoint.com/java/index.htm
- https://www.geeksforgeeks.org/java/
- https://www.oracle.com/in/java/technologies/java-ee-glance.html
- https://developers.redhat.com/topics/enterprise-java
- https://www.javacodegeeks.com/enterprise-java-tutorials

8.8 QUESTIONS

1. Explain the cookies with your use.
2. What types of cookies are there? Provide advantages and disadvantages of cookies with their builders.
3. How do cookies work? Explain your methods.
4. Where are cookies used? Give your applications. Explain the syntax to create and delete the obtained cookies.
5. Provide an example of Cookie.

*****

9

SESSIONS

Unit Structure

9.1 INTRODUCTION

- Session simply means a particular time interval.
- Session monitoring it is a way to maintain the state (data) of a user. Also known as servlet session management.
- The HTTP protocol is stateless, so we need to maintain state through session tracking techniques. Each time the user requests the server, the server treats the request as a new request. Therefore, we need to maintain a user's status to recognize a particular user.
- HTTP is stateless, which means that each request is treated as the new request.
- All requests and responses are independent. But sometimes it is necessary to monitor customer activity on multiple requests. For example. When a user logs into your website, regardless of which web page they visit after logging in, their credentials will remain on the server until they log off. Then this is handled by creating a session.
- The session is used to store everything we can get from the client of all the requests the client makes.

9.1.1 Advantages of stateless nature:

- Keeps the protocol simple and straightforward
- Consume fewer resources on the web server.
- Can admit simultaneous visitors.

9.1.2 Disadvantages of the stateless nature:

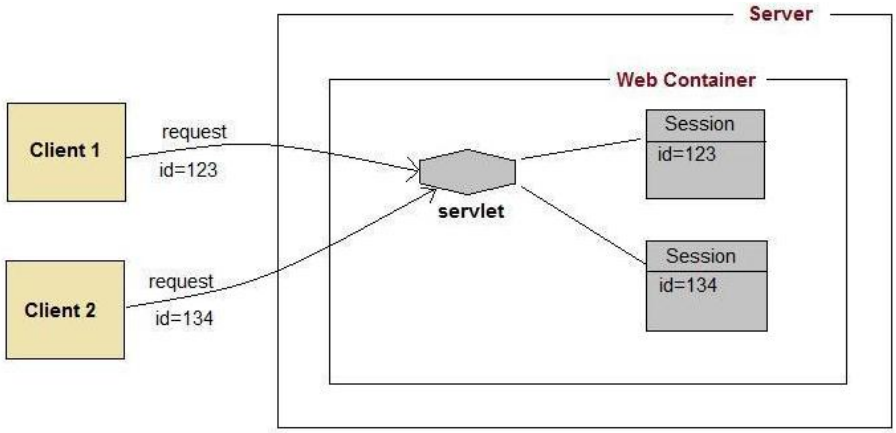
- The increased overhead required a new connection to be created with each request.
- The inability to track a single visitor crossing a website.
- The web server cannot automatically accept the browser request with a particular session.

9.1.3 Alternative solutions:

Web applications have used several techniques to circumvent HTTP stateless operations:

1. The client is identified each time it makes a request and the server stores and retrieves data related to that client - Sessions
2. The server sends the data to the client and forces the client to return them at each request made - Cookies

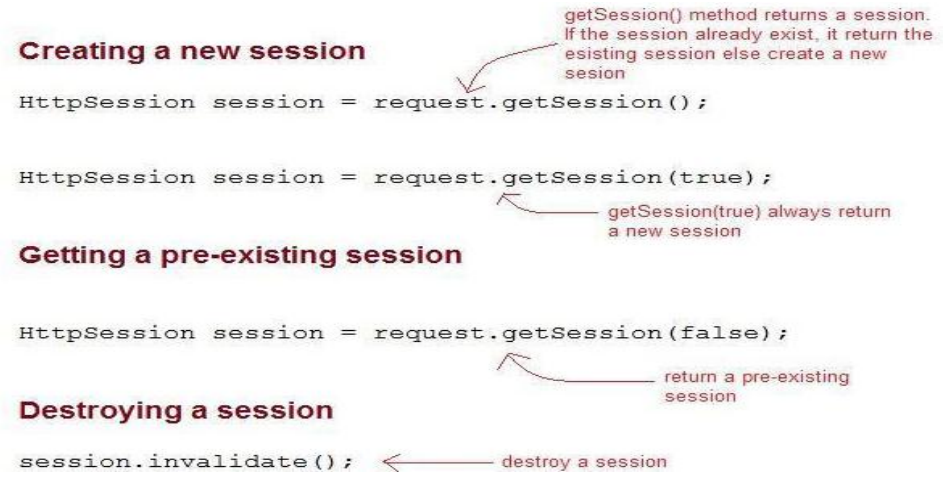
9.2 HOW THE SESSION WORKS



- The concept behind the session is that, whenever a user starts using our application, we can save unique identifying information about him, in an object available throughout the application, until it is destroyed. So wherever the user goes, we will always have his information and we can always manage which user is doing what. Whenever a user wants to exit your application, destroy the object with its information.

- On the client's first request, the web container generates a unique session ID and returns it to the client with a response. This is a temporary session created by the web container.
- The client returns the session ID with each request. It makes it easier for the web container to identify where the request is coming from.
- The web container uses this ID, finds the session that matches the ID, and associates the session with the request.

9.2.1 HttpSession interface:



9.2.2 How to get the HttpSession object?:

The HttpServletRequest interface provides two methods to get the HttpSession object:

publicHttpSessiongetSession ():

Returns the current session associated with this request or, if the request does not have a session, create one.

publicHttpSessiongetSession (boolean creation):

Returns the current HttpSession associated with this request or, if there is no current session and create is true, it returns a new session.

9.2.3 Commonly used methods of the HttpSession interface:

public string getId ():

Returns a string that contains the unique identifier value.

long public getCreationTime ():

Returns the time this session was created, measured in milliseconds since midnight on January 1, 1970 GMT.

public long getLastAccessedTime ():

Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.

public void invalidate ():

Replace this session and then unlink all objects linked to it.

In a typical stateless protocol transaction, the client:

1. Establishes a connection to the web server
2. Submit a request
3. Get an answer
4. Close connection

Because a persistent connection is not maintained between such requests, the connection from the web servers to the client is broken after the connection is closed. Disconnection between a client and the web server entails the following limitations:

- If the web server requires client authentication, for example a client needs to log in, the client needs to log in again on each request. The web server does not realize that it has already authenticated this client because the connection between the two has been lost.
- The web server cannot distinguish one client from another.

9.2.4 Solution:

- Establish a permanent connection between the client and the web server.
- Visitor's web browser requests.
- The web server assigns session IDs to each request.
- The web server identifies visitors through virtual connections called sessions.

9.3 SESSION ID

A session ID is a unique number that a website server assigns to a specific user during that user's visit (session). The session ID can be stored as a cookie, form field or URL (Uniform Resource Locator). Some web servers generate session IDs simply by incrementing static numbers. However, most servers use algorithms that involve more complex methods, such as taking into account the date and time of the visit along with other variables defined by the server administrator.

9.3.1 Session life cycle:

- A visitor, using a web browser, requests a resource from the web server.
- The web server offers the authentication form which causes the visitor's web browser to display a login form.
- The web browser returns the username and password, which are then returned to the web server.
- The web server returns a valid session ID to uniquely identify this visitor.
- The visitor's web browser sends any number of requests to the web server. The web server identifies users based on session IDs.
- The visitor closes the browser without explicitly logging out.

9.3.2 Session Monitoring API:

- The session monitoring API is based on the first four methods. This is to help the developer minimize the overhead of session monitoring. This type of session tracking is provided by the underlying technology. Let's take the example of the Java servlet. The servlet container handles session tracking activity and the user does not need to explicitly do this using Java servlets. This is the best of all methods, because all handling and errors related to session monitoring will be handled by the container itself.
- Each server client will be mapped to a javax.servlet.http.HttpSession object. Java servlets can use the session object to store and retrieve Java objects during the session. Session monitoring is best when implemented using the session monitoring API.
- A session is a collection of HTTP requests, over a period of time. A session is user specific and a new session is created for each user to track all user requests. In the servlet session, tracing can be used to track user state. Session monitoring is also known as session management, it is a mechanism used to maintain a user's state within a set of requests for a period of time. We can say that session monitoring is a means of keeping track of session data. This data represents the data that is transferred in a session.

9.3.3 Session Monitoring Methods:

- **User authentication:** This is the very common way that the user can provide authentication credentials from the login page and then we can pass the authentication information between the server and the client to maintain the session. This is not a very effective method because it will not work if the same user is logged in from different browsers.

- **HTML hidden field:** We can create a unique hidden field in the HTML and when the user starts browsing we can set its unique value for the user and keep track of the session. This method cannot be used with bindings because it requires the form to be submitted each time a client-to-server request is made with the field hidden. Also, it's not safe because we can get the hidden field value from the HTML source and use it to hack the session.
- **URL rewrite:** We can add a session identifier parameter with each request and response to keep track of the session. This is very tedious because we have to keep track of this parameter in every response and make sure it doesn't collide with other parameters.
- **Cookies:** Cookies are small pieces of information sent by the web server in the response header and stored in the browser's cookies. When the client makes an additional request, it adds the cookie to the request header and we can use it to track the session. We can maintain a session with cookies, but if the client disables cookies, it will not work.

9.3.4 Session Management API:

The Session Management API builds on the previous methods for session monitoring. Some of the major disadvantages of all of the above methods are:

1. Most of the time we don't just want to keep track of the session, we need to store some data in the session that we can use in future requests. This will require a lot of effort if we try to implement it.
2. All of the above methods are not complete by themselves, they will not all work in a particular scenario. So we need a solution that can use these session tracking methods to provide session management in all cases.

9.3.5 Example of using HttpSession:

In this example, we're setting the session-scope attribute in one servlet and getting that value from the session-scope in another servlet. To set the attribute within the session, we used the setAttribute () method of the HttpSession interface and to get the attribute we used the getAttribute method.

index.html:

<form method = "post" action = "Validate">

User: <input type = "text" name = "user" />

Password: <input type = "text" name = "pass">

<input type = "submit" value = "submit"> </form>

Validate.java:

import java.io. *;

importjavax.servlet. *;

importjavax.servlet.http. *;

Public class ValidateextendsHttpServlet

{

protected void doPost (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

response.setContentType ("text / html; character set = UTF-8");

String name = request.getParameter ("user");

String pass = request.getParameter ("pass");

if (pass.equal ("1234"))

{

HttpSessionsession = request.getSession (); session.setAttribute ("user", name); response.sendRedirect ("Welcome");

}}}

Welcome.java:

import java.io. *;

importjavax.servlet. *;

importjavax.servlet.http. *;

public class WelcomeextendsHttpServlet

{

protected void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

response.setContentType ("text / html; character set = UTF-8");

PrintWriterout = response.getWriter ();

HttpSessionsession = request.getSession ();

String user = (String) session.getAttribute ("user"); out.println ("Hello" + user);

}}

9.4 SUMMARY

The session object is used to record the access status of each client within the session scope, so that it is easy to track the operation status of each client, and the information stored in the session can obtain the valid data of these sessions when the browser makes subsequent requests.

In the jsp page, you can use the session object directly (built-in object of jsp), or you can go back to the session object through pageContext.

getSession () or request. getSession. Session can save user information and implement shopping cart and other functions.

9.5 REFERENCES

- https://www.javatpoint.com/java-tutorial
- https://www.tutorialspoint.com/java/index.htm
- https://www.geeksforgeeks.org/java/
- https://www.oracle.com/in/java/technologies/java-ee-glance.html
- https://developers.redhat.com/topics/enterprise-java
- https://www.javacodegeeks.com/enterprise-java-tutorials

9.6 QUESTIONS

1. Define the sessions. Explain session monitoring with the advantages and disadvantages of the stateless nature of the HTTP protocol.
2. How does the session work? Explain with the diagram.
3. How does the HTTP session work? Explain with the diagram.
4. Explain the steps for the stateless protocol transaction with its limitations and solutions.
5. Define the session ID. Explain the session life cycle.
6. What is the Session Monitoring API? Provide methods for session monitoring.
7. Provide an example of an HTTP session to validate the user's password.

*****

10

WORK WITH FILES

Unit Structure

10.1 FILE UPLOAD

You can use a servlet with an HTML form tag to allow users to upload files to the server. An uploaded file can be a text file or an image file or any document.

Creating a file upload form

The following HTM code below creates an upload form. The following are the important points to note:

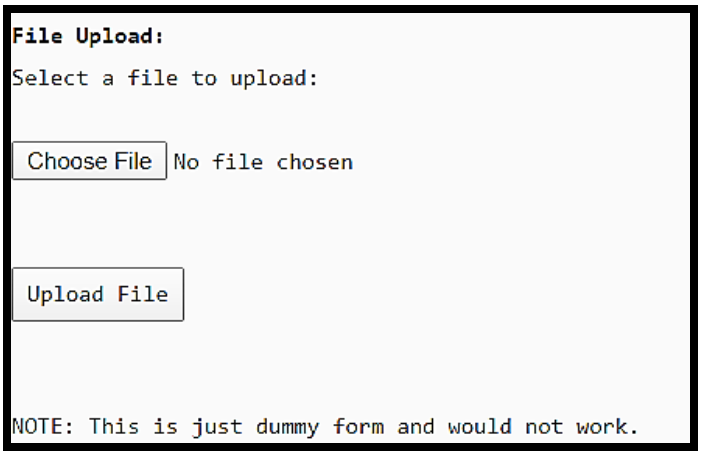
- The form method attribute must be set to POSTmethod and the GET method cannot be used
- The form enctype attribute must be set to multipart / form-data.
- The module action attribute should be set to a servlet file that would handle uploading files to the backend server. The following example uses the UploadServlet servlet to upload the file.
- To upload a single file, you need to use a single <input ... /> tag with attribute type = "file". To allow multiple file uploads, include more than one input tag with different values for the name attribute. The browser associates each of them with a Browse button.

```
<html> <head>
<title> File upload module </title>
</head> <body>
<h3> Upload file: </h3>
Select a file to upload: <br />
<form action = "UploadServlet" method = "post" enctype = "multipart / form-data">
<input type = "file" name = "file" size = "50" />
<br />
```

<input type = "send" value = "Upload file" />

</form> </body> </html>

This will display the following output which will allow you to select a file from the local PC and when the user clicks "Upload File" the form will be sent along with the selected file.



Write backend servlet

Below is the UploadServlet servlet that would take care of accepting the uploaded file and storing it in the <Tomcat-installation-directory> / webapps / data directory. This directory name can also be added using an external setting such as a context-param element in web.xml as follows:

<web-app>

....

<context-parameter>

<description> Location to store the uploaded file </description>

<param-name> upload file </param-name>

<value-param>

c: \ apache-tomcat-5.5.29 \ webapps \ data \

</param-value>

</context-param>

....

</web-app>

Below is the UploadServlet source code, which can handle uploading multiple files at the same time. Before proceeding, you should make sure of the following:

- The following example depends on FileUpload, so make sure you have the latest version of the commons-fileupload.xxjar file in your classpath. You can download it fromhttps://commons.apache.org/fileupload/.

- FileUpload depends on Commons IO, so make sure you have the latest version of the commons-io-xxjar file in your classpath. You can download it fromhttps://commons.apache.org/io/.
- When testing the following example, a file smaller than maxFileSize must be loaded; otherwise the file will not be loaded.
- Make sure you have created the c: \ temp and c: \ apache-tomcat8.0.28 \ webapps \ data directories well in advance.

```
// Import the necessary Java libraries
import java.io. *;
import java.util. *;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;
import org.apache.commons.io.output. *;
The UploadServlet public class extends HttpServlet {
 isMultipart private boolean;
 FilePath private string;
 private int maxFileSize = 50 * 1024;
 private int maxMemSize = 4 * 1024;
 private archive file;
 public void init () {
 // Get the location of the file where it would be stored.
filePath = getServletContext (). getInitParameter ("file upload");
 }
 public void doPost (HttpServletRequest request, HttpServletResponse response)
 throw ServletException, java.io.IOException {
 // Check that we have a request to upload the file
 isMultipart = ServletFileUpload.isMultipartContent (required);
 response.setContentType ("text / html");
```

```
java.io.PrintWriter out = response.getWriter ();
if (! isMultipart) {
out.println ("<html>");
out.println ("<head>");
out.println ("<title> Loading servlet </title>");
out.println ("</head>");
out.println ("<body>");
out.println ("<p> No files uploaded </p>");
out.println ("</body>");
out.println ("</html>");
Return;
}
 DiskFileItemFactory factory = new DiskFileItemFactory ();
factory.setSizeThreshold (maxMemSize); // maximum size to be stored in memory factory.setRepository (new File ("c: \\ temp")); // Location to save data larger than maxMemSize.
ServletFileUpload upload = new ServletFileUpload (factory); // Create a new file upload controller
upload.setSizeMax (maxFileSize); // maximum size of the file to upload.
 to deal {
List fileItems = upload.parseRequest (required); // Parse the request to get the elements of the file.
Iterator i = fileItems.iterator (); // Process the elements of the uploaded file
out.println ("<html>");
out.println ("<head>");
out.println ("<title> Loading servlet </title>");
out.println ("</head>");
out.println ("<body>");
 while (i.hasNext ()) {
FileItem fi = (FileItem) i.next ();
if (! fi.isFormField ()) {
// Get the parameters of the uploaded file
String fieldName = fi.getFieldName ();
String filename = fi.getName ();
String contentType = fi.getContentType ();
isInMemory = fi.isInMemory ();
```

```
long sizeInBytes = fi.getSize ();
 // Write the file
if (fileName.lastIndexOf ("\\")> = 0) {
file = new file (file path + file name.substring (file name.lastIndexOf ("\\")));
} the rest {
file = new file (file path + file name.substring (file name.lastIndexOf ("\\") + 1));
}
fi.write (file);
out.println ("Uploaded file name:" + file name + "<br>");
}
}
out.println ("</body>");
out.println ("</html>");
} catch (ex exception) {
System.out.println (es);
}}
public void doGet (HttpServletRequest request, HttpServletResponse response)
throw ServletException, java.io.IOException {
throw a new ServletException ("GET method used with" +
getClass () .getName () + ": POST method is required.");
}}}
```

10.2 DOWNLOADING FILES

In this example, we are creating three files:

- index.html
- DownloadServlet.java
- web.xml

index.html

This file provides a link to download the file.

```
<a href="servlet/DownloadServlet"> download the jsp file </a>
```

DownloadServlet.java

```
importjava.io. *;
import javax.servlet.ServletException;
```

```
importjavax.servlet.http. *;
public class DownloadServlet extends HttpServlet {
public void doGet (HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
response.setContentType ("text / html");
PrintWriter out = response.getWriter ();
String filename = "home.jsp";
String file path = "e: \\";
response.setContentType ("APPLICATION / OCTETO-STREAM");
response.setHeader ("Content-Disposition", "attachment; filename = \" "+ filename +" \ "");
FileInputStream fileInputStream = new FileInputStream (file path + file name);
In t I;
weather ((i = fileInputStream.read ())! = -1) {
out.write (i);
}
fileInputStream.close ();
out.close ();
}}
```

web.xml file

```
<web-app>
<servlet>
<servlet-name> DownloadServlet </servlet-name>
<servlet-class> DownloadServlet </servlet-class>
</servlet>
<mapping-servlet>
<servlet-name> DownloadServlet </servlet-name>
<url-pattern> / servlet / DownloadServlet </url-pattern>
</servlet-mapping>
</web-app>
```

10.3 SERVLET ANNOTATIONS

- Servlet uses the deployment descriptor (web.xml file) to deploy the application to a web server. Servlet API 3.0 introduced a new package called javax.servlet.annotation. Provides annotation types that can be used to annotate a servlet class. If you use annotation, the deployment

descriptor (web.xml) is not required. But you should be using Tomcat7 or any later version of Tomcat.

- Annotations can override the equivalent XML configuration in the web deployment descriptor file (web.xml), such as the servlet declaration and servlet mapping. Servlet containers will process the classes noted at the time of deployment. The types of annotations introduced in Servlet 3.0 are:

Sr.No.	Annotation and description
1	**@WebServlet:** Declare a servlet.
2	**@WebInitParam:** To specify an initialization parameter.
3	**@WebFilter:** Declare a servlet filter.
4	**@WebListener:** To declare a WebListener
5	**@HandlesTypes:** Declare the types of classes that a ServletContainerInitializer can handle.
6	**@HttpConstraint:** This annotation is used within the ServletSecurity annotation to represent security restrictions that will apply to all HTTP protocol methods for which a corresponding HttpMethodConstraint element is NOT produced within the ServletSecurity annotation.
7	**@HttpMethodConstraint:** This annotation is used within the ServletSecurity annotation to represent security restrictions on specific HTTP protocol messages.
8	**@MultipartConfig:** An annotation that can be specified in a Servlet class, indicating that Servlet instances expect requests that conform to the multipart / form-data MIME type.
9	**@ServletSecurity:** This annotation is used in a Servlet implementation class to specify the security restrictions that a Servlet container must apply on HTTP protocol messages.

10.4 REFERENCES

- https://www.javatpoint.com/java-tutorial
- https://www.tutorialspoint.com/java/index.htm
- https://www.geeksforgeeks.org/java/
- https://www.oracle.com/in/java/technologies/java-ee-glance.html
- https://developers.redhat.com/topics/enterprise-java
- https://www.javacodegeeks.com/enterprise-java-tutorials

10.6 SUMMARY

10.5 QUESTIONS

1. Take an example to upload a file.
2. Take an example to download a file.
3. Explain the different servlet annotations

*****

11

NON-BLOCKING

Unit Structure

11.1 INTRODUCTION TO I / O

- I / O, or input / output, refers to the interface between a computer and the rest of the world, or between a single program and the rest of the computer. It is such a crucial element of any computer system that most of any I / O is built into the operating system. Individual programs generally do most of the work for them.

- In Java programming, I / O has until recently been done using a flow metaphor. All I / O is considered to be moving individual bytes, one at a time, through an object called Stream. Stream I / O is used to contact the outside world. It is also used internally, to convert objects to bytes and then back to objects.

- NIO has the same function and purpose as the original I / O, but uses a different metaphor: block I / O. As you will learn in this tutorial, block I / O can be much more efficient than streaming I / O.

11.1.1 Why NIO?:

NIO was created to allow Java programmers to implement high-speed I / O without having to write custom native code. NIO shifts more time-consuming I / O tasks (that is, buffering and flushing) to the operating system, allowing for a noticeable increase in speed.

11.2 STREAM AGAINST BLOCKS

- The most important distinction between the original I / O library (found in java.io. *) and NIO has to do with how data is packaged and transmitted. As mentioned earlier, the original I / O takes care of the data in streams, while the NIO takes care of the data in chunks.
- A flow-oriented I / O system handles data one at a time. An input stream produces one byte of data and an output stream consumes one byte of data. It is very easy to create filters for the transmitted data. It is also relatively easy to chain multiple filters together so that each plays its part in a unique and sophisticated processing mechanism. On the other hand, streaming-oriented I / O is usually quite slow.
- A block-oriented I / O system manages the data in blocks. Each operation produces or consumes a block of data in a single step. The data processing by the block can be much faster than the byte processing (transmitted). But block-oriented I / O lacks the elegance and simplicity of flow-oriented I / O.

11.3 INTEGRATED I / O INTEGRATED

- The original I / O package and NIO have been well integrated into JDK 1.4. java.io. * was redeployed using NIO as a base, so you can now take advantage of some NIO features. For example, some of the classes in the java.io package. * contain methods for reading and writing data in blocks, leading to faster processing even on more flow-oriented systems.
- You can also use the NIO library to implement standard I / O functions. For example, you can easily use I / O blocks to move data one byte at a time. But as you will see, NIO also offers many benefits that are not available in the original I / O package.

11.4 IO BLOCKING AND NON-BLOCKING

- Various Java I / O streams are blocked. This means that when a thread calls alight() or write(), that thread is blocked until there is no data to read or until the data is completely written. The thread can't do anything else in the meantime.
- Java NIO's non-blocking mode allows a thread to request to read data from a channel and get only what is currently available, or nothing, if no data is currently available. Instead of getting stuck until the data is available for reading, the thread can continue with something else.
- The same goes for non-blocking writing. A thread may request some data to be written to a channel, but not wait for it to be fully written. The thread can continue and do something else in the meantime.
- Whereas threads spend their idle time when they aren't blocked during I / O calls, they are generally doing I / O on other channels in the

meantime. That is, a single thread can now handle multiple input and output channels.

11.5 READ FROM A FILE

- If we were using the original I / O, we would simply create a FileInputStreamand read it. In NIO, however, things work a little differently: first we get aChannel object of FileInputStreamand then use that channel to read the data.
- Whenever you perform a read operation on a NIO system, you are reading from a channel, but not reading directly from a channel. Since all data ultimately resides in the buffer, it is read from a channel to a buffer.
- So, reading from a file involves three steps: (1) getting the Channel from FileInputStream; (2) create theBumper; and (3) reading theCannel to Bumper. Now, let's see how it works.

Three simple steps:

Our first step is to get a channel. We take the channel fromFileInputStream:

```
FileInputStream fin = new FileInputStream ("readandshow.txt");

FileChannel fc = fin.getChannel ();
```

The next step is to create a buffer:

```
ByteBuffer buffer = ByteBuffer.allocate (1024);
```

And finally, we need to read from channel to buffer, as shown here:

```
fc.read (buffer);
```

11.5.1 WRITE TO FILE

Writing to a file in NIO is similar to reading from one. Let's start by getting a channel from toFileOutputStream:

```
FileOutputStream fout = new FileOutputStream ("writesomebytes.txt");

FileChannel fc = fout.getChannel ();
```

Our next step is to create a buffer and insert data into it; in this case the data will be taken from a named arrayMessage A containing the ASCII bytes for the string "Some bytes".

```
ByteBuffer buffer = ByteBuffer.allocate (1024);
 for (int i = 0; i <message.length; ++ i) {
 buffer.put (message [i]);
}
buffer.flip ();
```

Our last step is to write to the buffer:

```
fc.write (buffer);
```

11.6 LISTENERS

Java EE provides non-blocking I / O support for servlets and filters when processing requests in asynchronous mode. The following steps summarize how to use non-blocking I / O to process requests and write responses within the service's methods.

- Put the request in asynchronous mode as described in Asynchronous Processing.
- Get an inflow and / or outflow from the request and response objects in the service method.
- Assign a read listener to the input stream and / or a write listener to the output stream.
- Process the request and response within the listener's callback methods

Non-blocking I / O support in javax.servlet.ServletInputStream

Method	Description
void setReadListener (ReadListener rl)	Associate this input stream with a listener that contains callback methods to read the data asynchronously. Provide the listener as an anonymous class or use another mechanism to pass the input stream to the read listener.
boolean isReady ()	Returns true if the data can be read without blocking.

boolean isFinished ()	Returns true when all data has been read.

Non-blocking I / O support in javax.servlet.ServletOutputStream

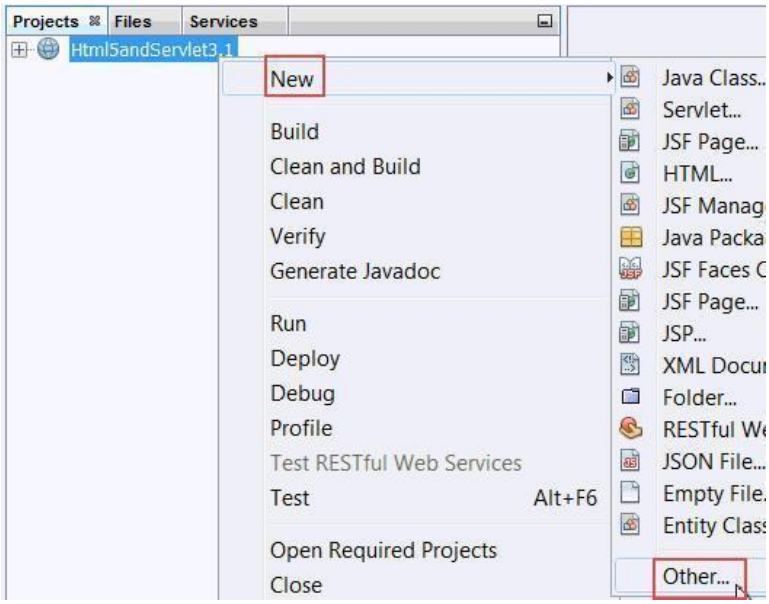
Method	Description
void setWriteListener (WriteListener wl)	Associate this output stream with a listener that contains callback methods for writing data asynchronously. Provide the write listener as an anonymous class or use another mechanism to pass the output stream to the write listener
boolean isReady ()	Returns true if the data can be written without locks.

Listening interfaces for non-blocking I / O support

Interface	methods	Description
Read Listener	void onDataAvailable () void suAllDataRead () void onError (Throwable t)	TO ServletInputStream The instance calls these methods on its listener when data is available to read, when all data has been read, or when an error occurs.
Write Listener	void onWritePossible () void onError (Throwable t)	TO ServletOutputStream The instance calls these methods in its listener when it can write data without blocking or when an error occurs.

11.7 STEPS TO IMPLEMENT WRITELISTENER

1. In the Projects tab, right-click on your project and say Html5 and Servlet 3.1 and select New> Other.

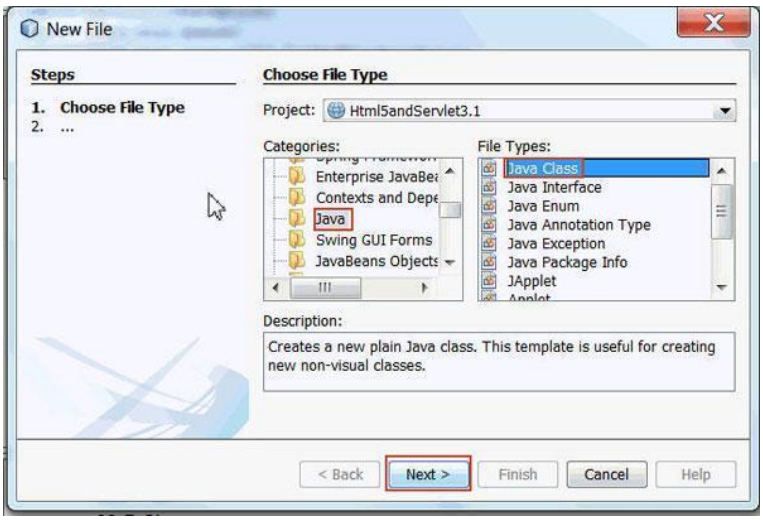


2. In the New File dialog box, complete the following steps on the Choose File Type page:

a. Select Java in Categories.

b. Select Java Class in File Types.

c. Click Next.

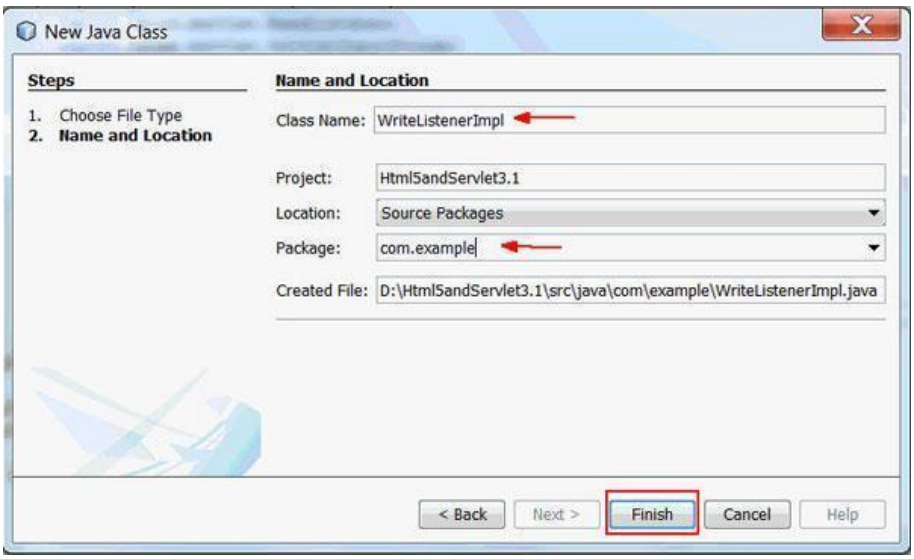


3. On the New Java Class page, do the following:

a. Pay in Write ListenerImpl as the name of the class.

b. Pay in com.example as the package name.

c. Click Finish.



4. Import the following packages:

Importjava.io.IOException;

importjava.util.Queue;

importjavax.servlet.AsyncContext;

importjavax.servlet.ServletOutputStream;

import javax.servlet.WriteListener;

5. Modify the class to implement the Write Listener Interface.

The public class WriteListenerImpl implements WriteListener {

6. Declare the following variables:

ServletOutputStreamprivateoutput=null;

privatequeue=null;

private AsyncContext context = null;

```
public class WriteListenerImpl implements WriteListener {

    private ServletOutputStream output = null;
    private Queue queue = null;
    private AsyncContext context = null;

    }
```

7. Add a constructor to the class:

WriteListenerImpl (ServletOutputStream sos, Queue q, AsyncContext c) {

output=sos;

tail=q;

context = c; }

```
public class WriteListenerImpl implements WriteListener {

    private ServletOutputStream output = null;
    private Queue queue = null;
    private AsyncContext context = null;

    WriteListenerImpl(ServletOutputStream sos, Queue q, AsyncContext c) {
        output = sos;
        queue = q;
        context = c;
    }
}
```

8. Add the onWritePossible ()method:

publicvoidonWritePossible()throwsIOException{

while (queue.peek ()! = null && output.isReady ())

Datastring= (String)queue.poll ();

output.print (data);

}

if (queue.peek () == null) {

context.com complete ();

}}

```
public class WriteListenerImpl implements WriteListener {

    private ServletOutputStream output = null;
    private Queue queue = null;
    private AsyncContext context = null;

     WriteListenerImpl(ServletOutputStream sos, Queue q, AsyncContext c) {...}
    public void onWritePossible() throws IOException {
        // write while there is data and is ready to write
        while (queue.peek() != null && output.isReady()) {
            String data = (String) queue.poll();
            output.print(data);
        }
        // complete the async process when there is no more data to write
        if (queue.peek() == null) {
            context.complete();
        }
    }

}
```

9. Add the onError method.

public void onError (final Throwable t) {

context.complete ();

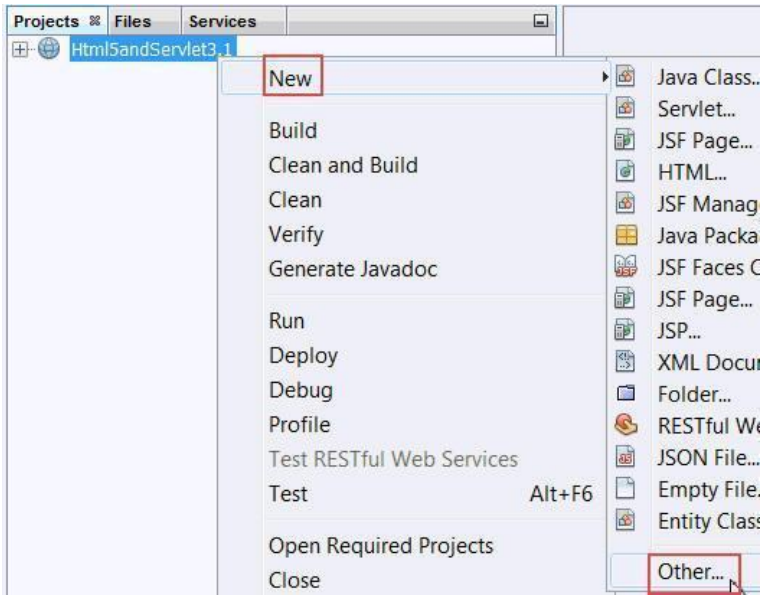
t.printStackTrace (); }

```
public class WriteListenerImpl implements WriteListener {

    private ServletOutputStream output = null;
    private Queue queue = null;
    private AsyncContext context = null;

    WriteListenerImpl(ServletOutputStream sos, Queue q, AsyncContext c) {...}

    public void onWritePossible() throws IOException {...}

    public void onError(final Throwable t) {
        context.complete();
        t.printStackTrace();
    }
}
```

11.8 STEPS TO IMPLEMENT READLISTENER

Perform the following steps to implement the ReadListenerImpl class:

1. On the Projects tab, right-click Html5 and Servlet3.1 and select New> Other.



2. In the New File dialog box, complete the following steps on the Choose File Type page:

a. Select Java in Categories.

b. Select Java Class in File Types.

c. Click Next.

3. On the New Java Class page, do the following:

a. Pay in ReadListenerImpl as the name of the class.

b. Pay in com.example as the package name.

c. Click Finish.

4. Import the following packages:

import java.io.IOException; import java.util.Queue;

import java.util.concurrent.LinkedBlockingQueue; import

javax.servlet.AsyncContext; import javax.servlet.ReadListener; import

javax.servlet.ServletInputStream; import

javax.servlet.ServletOutputStream; import

javax.Lxistepnervlet.Write.Write .HttpServletResponse;

5. Modify the class to implement the Read Listener Interface:

The public ReadListenerImpl class implements ReadListener

6. Declare the following variables:

private entry ServletInputStream = null; Private HttpServletResponse res = null; Private AsyncContext ac = null; private queue queue = new LinkedBlockingQueue**();**

```
public class ReadListenerImpl implements ReadListener {

    private ServletInputStream input = null;
    private HttpServletResponse res = null;
    private AsyncContext ac = null;
    private Queue queue = new LinkedBlockingQueue();

}
```

7. Add a constructor to the class:

ReadListenerImpl (ServletInputStreamin, ttpServletResponser, AsyncContext c) {input = in; re = r; ac = c; }

```
public class ReadListenerImpl implements ReadListener {

    private ServletInputStream input = null;
    private HttpServletResponse res = null;
    private AsyncContext ac = null;
    private Queue queue = new LinkedBlockingQueue();

    ReadListenerImpl(ServletInputStream in, HttpServletResponse r, AsyncContext c) {
        input = in;
        res = r;
        ac = c;
    }
}
```

8. Add the onDataAvailable () method:

public void onDataAvailable () throws IOException {System.out.println ("Dataisavailable");

StringBuilder sb = new StringBuilder (); int len = -1; byte b [] = new byte [1024]; while (input.isReady () && (len = input.read (b))! = -1) {String data = new String (b, 0, len); sb.append (data); } queue.add (sb.toString ()); }

```
public class ReadListenerImpl implements ReadListener {

    private ServletInputStream input = null;
    private HttpServletResponse res = null;
    private AsyncContext ac = null;
    private Queue queue = new LinkedBlockingQueue();

    ReadListenerImpl(ServletInputStream in, HttpServletResponse r, AsyncContext c) {...}

    public void onDataAvailable() throws IOException {
        System.out.println("Data is available");

        StringBuilder sb = new StringBuilder();
        int len = -1;
        byte b[] = new byte[1024];
        while (input.isReady() && (len = input.read(b)) != -1) {
            String data = new String(b, 0, len);
            sb.append(data);
        }
        queue.add(sb.toString());
    }
}
```

9. Añade el onAllDataRead ()method:

public void onAllDataRead () throws IOException {System.out.println ("All data is read"); // now all data has been read, set a WriteListener to write the output ServletOutputStream = res.getOutputStream (); WriteListener writeListener = new WriteListenerImpl (output, queue, ac); output.setWriteListener (writeListener);}

```
public class ReadListenerImpl implements ReadListener {

    private ServletInputStream input = null;
    private HttpServletResponse res = null;
    private AsyncContext ac = null;
    private Queue queue = new LinkedBlockingQueue();

    ReadListenerImpl(ServletInputStream in, HttpServletResponse r, AsyncContext c) {...}

    public void onDataAvailable() throws IOException {...}

    public void onAllDataRead() throws IOException {
        System.out.println("Data is all read");

        // now all data are read, set up a WriteListener to write
        ServletOutputStream output = res.getOutputStream();
        WriteListener writeListener = new WriteListenerImpl(output, queue, ac);
        output.setWriteListener(writeListener);
    }
}
```

10. Add the onError method.:

public void onError (final Throwable t) {ac.complete (); t.printStackTrace (); }

```
public class ReadListenerImpl implements ReadListener {

    private ServletInputStream input = null;
    private HttpServletResponse res = null;
    private AsyncContext ac = null;
    private Queue queue = new LinkedBlockingQueue();

    ReadListenerImpl(ServletInputStream in, HttpServletResponse r, AsyncContext c) {...}

    public void onDataAvailable() throws IOException {...}

     public void onAllDataRead() throws IOException {...}
       public void onError(final Throwable t) {
         ac.complete();
         t.printStackTrace();
       }
}
```

After ServletInputStream.setReadListener is called, Read Listener in DataAvailablecalled immediately if there is data to read. Otherwise, it is invoked when the data is ready. When all the data has been read,Read Listener establishes a Write Listener to write data in non-blocking mode.

11.9 SUMMARY

Java NIO's non-blocking mode enables a thread to request reading data from a channel, and only get what is currently available, or nothing at all, if no data is currently available. Rather than remain blocked until data becomes available for reading, the thread can go on with something else.

11.10 REFERENCES

- https://www.javatpoint.com/java-tutorial
- https://www.tutorialspoint.com/java/index.htm
- https://www.geeksforgeeks.org/java/
- https://www.oracle.com/in/java/technologies/java-ee-glance.html
- https://developers.redhat.com/topics/enterprise-java
- https://www.javacodegeeks.com/enterprise-java-tutorials

11.11 Questions

1. Explain I / O and NIO. Differentiate between flows and blocks.
2. Explain embedded I / O with the difference between blocking and non-blocking I / O.
3. How to read and write files? Give an example.
4. Tell listeners how to read and write a file.
5. Explain the steps to implement WriteListener.
6. Explain the steps for ReadListener.

*****

UNIT III

12

INTRODUCTION TO JAVA SERVER PAGES

Unit Structure

12.0 OBJECTIVES

After going through this chapter, you will:

- Understand the basics of Java Server Pages technology
- Learn the advantages and disadvantages of JSP
- Differentiate between JSP and Servlets
- Understand lifecycle of a JSP program
- Learn how to execute a JSP program

12.1 INTRODUCTION TO JAVA SERVER PAGES

Java Server Pages (JSP), a server side technology, which is a part of Java EE framework, is used for creating dynamic web applications in a very simple yet powerful way. JSP technology allows you to develop JSP pages which are text based documents that processes a request and generates a response. It has access to the powerful enterprise Java API including the JDBC API and hence can be used to access enterprise databases.

A JSP Page consists of both static and dynamic content. The static content can be written using plain HTML or XML and dynamic content can be written either using the regular java style of programming embedded in specific elements or using the tag style approach. This eliminates the need of writing multiple println statements that generate HTML like in Servlets.

JSP provides developers with the ability to cleanly separate the application logic from the presentation logic. In other words, JSP can easily differentiate the coding in the view layer and application logic layer. Hence any requirement changes that is needed to be done in the look and feel of the web application can very easily be done as it would require changes only in the HTML part of code and not the logical part written in java, thus eliminating the need to recompile the entire web application again.

It also supports unified expression language (UEL) to access server objects without having explicit need to declare them.

The file extension for JSP pages is .jsp

12.2 WHY USE JAVA SERVER PAGES

JSP has the following benefits:

1. Nobody can borrow the code:

Since JSP page are written, runs and remains on the server, nobody can copy the logic written in a jsp page even if they wanted to. Thus, security of the code is maintained.

2. Faster loading of pages:

Response page customization as requested by the user is done at the server side itself thus no extra code or content is sent to the client side resulting in faster loading of pages.

3. No Browser compatibility issues:

Since JSP runs on the server side, the developer ends up sending standard HTML to the user browser. This largely eliminates cross browser compatibility issues.

4. JSP Support:

JSP is supported by a number of Web Servers. Built-in Support for JSP is available in Java Web Server from Oracle.

5. Compilation:

In JSP technology, each JSP page is compiled into executable code the first time it is requested and invokes the resulting code directly on all subsequent requests. When coupled with a persistent JVM, this allows the server to process request to JSP pages much faster.

6. Similarity to HTML:

A JSP page looks a lot like a HTML or XML page except for the business logic written either in scripting elements or JSP tags or both. Writing the business logic in JSP tags brings consistency to the coding style used on the entire JSP page.

7. Separation of logic from view

It enables to separate presentation layer with the business logic layer in the web application.

12.3 DISADVANTAGES OF JSP

The disadvantages of JSP are:

1. Attractive Java code:

Putting Java code within a webpage is really bad design, but JSP makes it tempting to do just that. Avoid this as far as possible.

2. Java Code Required:

To do relatively simple things in JSP can actually demand putting Java code in a page. Assume a page needs to determine the context root of the current web application, perhaps to create a link to the web applications,home page. This is done using Java code in JSP.

<a href='<%=request.getContextPath()%> /index.html'>Home Page

Java code can be avoided by using <jsp:getProperty> but that makes the code spec even more complex.

<a href='<jsp:getProperty name="request" property="contextPath"/>/index.html'>Home Page

3. Simple Tasks are Hard to Code:

Even including page headers and footers is a bit difficult with JSP. In JSP the best way to do this is as follows:

<% @include file="/header.jsp";%>

/*Some content here*/

<% @include file="/footer.jsp";%>

4. Occupies a lot of space:

JSP pages require about double the disk space to hold the page. Because JSP pages are translated into class files, the server has to store the resultant class files with the JSP pages.

5. Debugging not easy:

It is hard to trace JSP pages error because JSP pages are translated to servlet before the compilation process.

6. Difficult Looping in JSP:

Looping in JSP is a bit complicated.

7. Database Connection not easy:

Database connectivity is not as easy as it should be. Most of the servlet engine vendors do not support connection pooling natively, as of this day. Consequently, one has to write a lot of custom code to do the job.

12.4 JSP V/S SERVLETS

Servlets provide the ability to build dynamic content for websites using Java and is supported by all Web Servers.

Servlet	JSP
Servlets run faster than JSP.	JSP runs slower than servlet as it takes time to compile the program and convert into servlets.
It is hard to write code in servlet.	It's easier to code in JSP compared to servlets.
In MVC architecture, servlet works as a controller.	In MVC architecture, JSP works as a view for displaying output.
It should be use when there is more data processing involved.	JSP is generally used when there is no involvement of much data processing.
There is no custom tag writing facility in servlets.	You can easily build custom tags that can directly call Java beans.
Servlet is a java code.	JSP is a HTML-based code.
It can accept all protocol requests, including HTTP.	It can only accept HTTP requests.
You can override the service() method.	In JSP, you can't override the service() method.
In Servlet, by default, session management is not enabled, user has to enable it explicitly.	In JSP, session management is automatically enabled.
In Servlet, you have to implement both business logic and presentation logic in the single file.	In JSP, business logic is split from presentation logic.
Modification in Servlet file is a time consuming due to reloading, recompiling, and restarting the server.	JSP modification is fast, as you just need to click one refresh button.

12.5 LIFECYCLE OF A JSP PAGE

A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

When the browser asks for a JSP, JSP engine first checks whether it needs to compile the page. If the JSP is last compiled or the recent modification is done in JSP, then the JSP engine compiles the page.

Compilation process of JSP page involves three steps:

- Parsing of JSP
- Turning JSP into servlet
- Compiling the servlet

JSP Lifecycle follows the following steps:

1. Translation of JSP page
2. Compilation of JSP page(Compilation of JSP page into _jsp.java)
3. Classloading (_jsp.java is converted to class file _jsp.class)
4. Instantiation(Object of generated servlet is created)
5. Initialisation(_jspinit() method is invoked by container)
6. Request Processing(_jspservice() method is invoked by the container)
7. Destroy (_jspDestroy() method invoked by the container)

1. Translation of the JSP Page:

A Java servlet file is generated from a JSP source file. This is the first step of JSP life cycle. In translation phase, container validates the syntactic correctness of JSP page and tag files.

The JSP container interprets the standard directives and actions, and the custom actions referencing tag libraries used in this JSP page.

For example, a program named demo.jsp as shown below:

demo.jsp

```
<html>
<head>
<title>Demo JSP</title>
</head>
<%
int demovar=0;%>
```

<body>

Count is:

<% out.println(demovar++); %>

<body>

</html>

will get translated into demo_jsp.java as below:

```
1  Public class demp_jsp extends HttpServlet{
2     Public void _jspservice(HttpServletRequest request, HttpServletResponse response)
3           Throws IOException, ServletException
4        {
5  PrintWriter out = response.getWriter();
6  response.setContentType("text/html");
7  out.write("<html><body>");
8  int demovar=0;
9  out.write("Count is:");
10 out.print(demovar++);
11 out.write("</body></html>");
12 }
13 }
14
```

In the above example,

- demo.jsp, is a JSP where one variable is initialized and incremented. This JSP is converted to the servlet (demo_jsp.class) wherein the JSP engine loads the JSP Page and converts to servlet content.
- When the conversion happens all template text is converted to println() statements and all JSP elements are converted to Java code.

2. Compilation of the JSP Page:

- The generated java servlet file is compiled into java servlet class
- The translation of java source page to its implementation class can happen at any time between the deployment of JSP page into the container and processing of the JSP page.
- In the above pictorial description demo_jsp.java is compiled to a class file demo_jsp.class

3. Class loading:

- Servlet class that has been loaded from JSP source is now loaded into the container

4. Instantiation:

- In this step the object i.e. the instance of the class is generated.
- The container manages one or more instances of this class in the response to requests and other events. Typically, a JSP container is built using a servlet container. A JSP container is an extension of servlet container as both the container support JSP and servlet.

- A JSPPage interface which is provided by container provides init() and destroy () methods.
- There is an interface HttpJSPPage which serves HTTP requests, and it also contains the service method.

5. Initialization:

```
public void jspInit()
{
//initializing the code
}
```

- _jspinit() method will initiate the servlet instance which was generated from JSP and will be invoked by the container in this phase.
- Once the instance gets created, init method will be invoked immediately after that
- It is only called once during a JSP life cycle, the method for initialization is declared as shown above

6. Request processing:

```
void _jspservice(HttpServletRequest request HttpServletResponse response)
{
//handling all request and responses
}
```

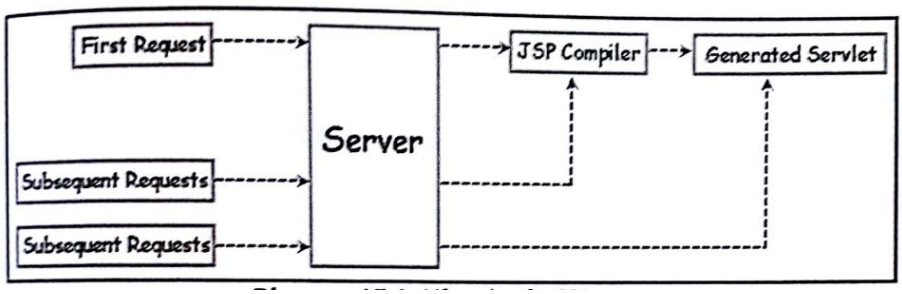
- _jspservice() method is invoked by the container for all the requests raised by the JSP page during its life cycle
- For this phase, it has to go through all the above phases and then only service method can be invoked.
- It passes request and response objects
- This method cannot be overridden
- The method is shown above: It is responsible for generating of all HTTP methods i.eGET, POST, etc.

7. Destroy:

```
public void _jspdestroy()
{
//all clean up code
}
```

_jspdestroy() method is also invoked by the container

- This method is called when container decides it no longer needs the servlet instance to service requests.
- When the call to destroy method is made then, the servlet is ready for a garbage collection
- This is the end of the life cycle.
- We can override jspdestroy() method when we perform any cleanup such as releasing database connections or closing open files.



Whenever the JSP file changes ,the Web Server automatically detects the change and rebuilds the corresponding servlet. The JSP to Servlet compilation phase imposes a slight delay the first time a page is retrieved. Many web servers permit pre-compilation of JSPs to get around this problem.

12.6 HOW DOES JSP FUNCTION:

JSP code spec can be broken into two categories:

- Elements that are processed by the JSP Engine on the Web Server
- Template data or everything other than such elements that the JSP engine ignores.

A JSP page is executed by a web server that either has a built-in JSP engine or accesses a third party JSP engine,which it is configured to use. When a client asks for a JSP page , the Web Server sends that request and delivers it to the JSP engine along with a response object.

The JSP engine then processes the client's request and delivers the output back to the Web Server for further delivery to the client.

Let's look into a simple example to understand the functioning of a JSP page. In this example, we are creating a html page to accept a name from the user and submit it to the Server. The Web Server will process the request which will consist the name entered by the user and in return print a "Hello " name on the client's browser.

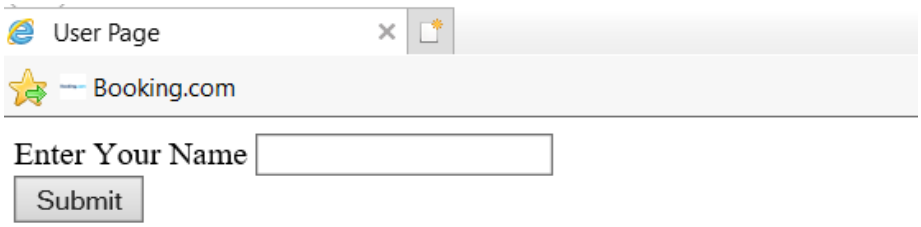
Example: Program to understand the basic functioning of a JSP page.

Index.html

<html>

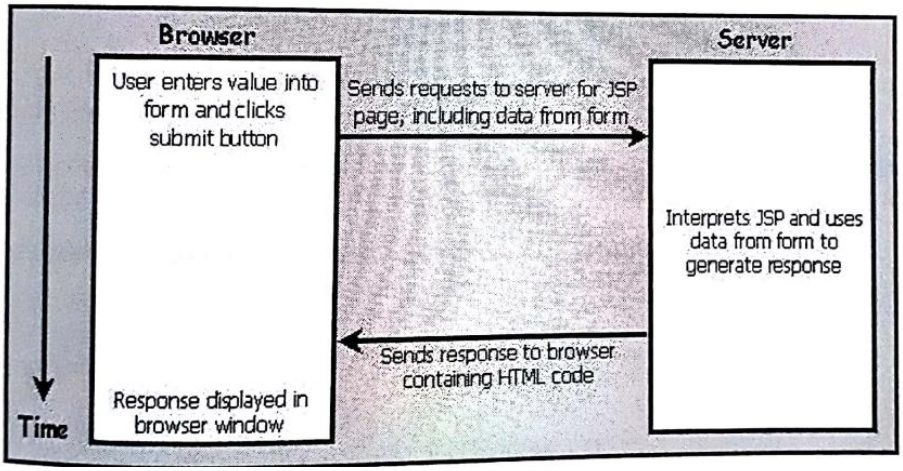
```
<head>
<title>User Page</title>
</head>
<body>
<form action="Hello.jsp">
        Enter Your Name <input type="text" name="name" ><br>
        <input type="submit" value="Submit">
</form>
</body>
</html>
```

When viewed in a browser,the HTML page looks like:



When the user clicks Submit ,the data entered on the form will be sent as a request to the JSP page on the Web Server for further processing.

The Web Server accepts the data returned from the browser and passes it as a parameter to hello.jsp which creates the response HTML page which will be sent back to the client's browser.



Hello.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
   <head>
        <title>JSP Page</title>
```

```
    </head>
    <body>
        Hello
        <%=request.getParameter("name")%>
    </body>
</html>
```

In the above Hello.jsp page, the first line:

<%@page contentType="text/html" pageEncoding="UTF-8"%>

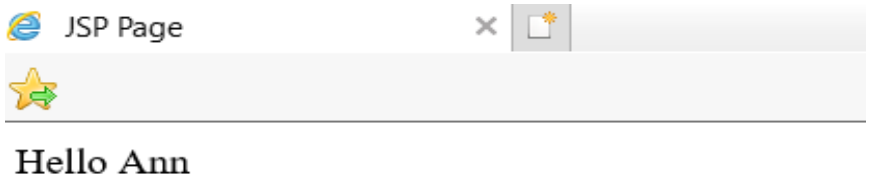
is a JSP directive (denoted by <%@). Hence the JSP engine recognizes that:

- The MIME type is text/html for JSP Style JSP tag.
- The character encoding is UTF-8 for XML style tags.

The next part of code <%=request.getParameter("name")%> is a block of expression tag which is used to evaluate the expression after = and return the value .

Thus the output of the above code will be Hello followed by the name entered by the user on index.html.

The response from the web server displayed in the user browser is :



12.7 HOW DOES JSP PROGRAM EXECUTE?

The following happens when a user browser requests index.jsp.

The browser sends its request to the Web Server as

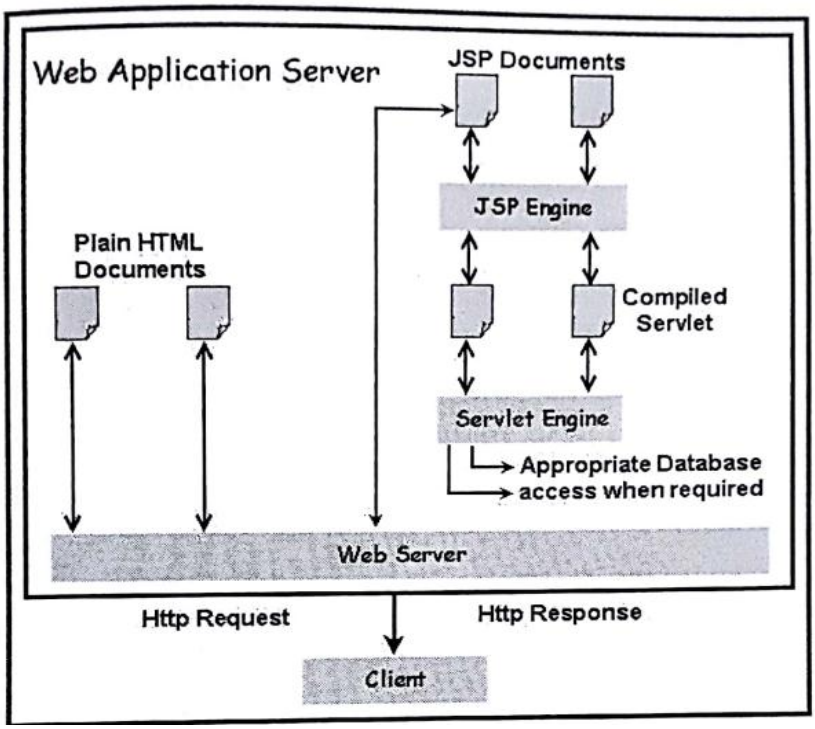
http://localhost:8080/Sample/index.jsp?numtimes=5

This specifies the value of numtimes as a GET parameter.

Web Server recognizes index.jsp in the URL sent in by the browser. Web Server recognizes index.jsp as a jsp page by its extension and that the information delivered by the browser encoded in the URL must be passed onto index.jsp.

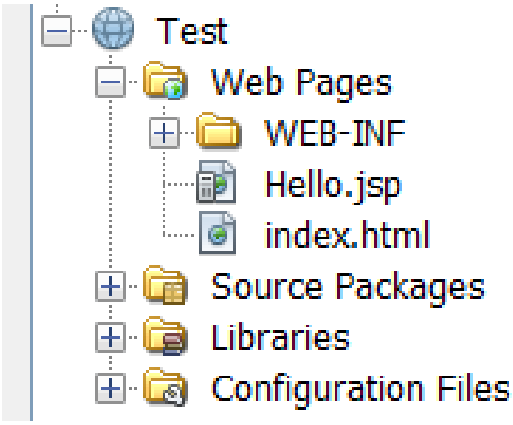
Index.jsp is then translated into a java class by the JSP engine, This translation and compilation phase occurs only when the JSP is first called. (or it is subsequently changed)Hence there will be slight delay the first time when index.jsp is run.

For every subsequent request for that JSP page thereafter, there is no delay because the request is forwarded directly to the servlet already in the memory.



12.8 DIRECTORY STRUCTURE OF JSP

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



12.9 SUMMARY

- Java Server Pages is a server side technology used for creating dynamic web application.
- Tags are used to insert JAVA code into HTML pages.
- JSP is first converted into servlet by JSP container before processing the client's request.

- JSP easily separates the presentation logic from the business logic
- JSP code can be easily modified to incorporate any look and feel changes to the application

12.10 LIST OF REFERENCES

Java EE 7 for Beginners, Sharanam Shah, Vaishali Shah, First Edition, SPD

Web References:

1. https://www.guru99.com

12.11 QUESTIONS

Q1. How does JSP differ from Servlets?

Q2. Write a note on lifecycle of a JSP.

Q3. What are the advantages and disadvantages of using JSP for developing web applications?

Q4. How are JSP pages executed?

*****

13

GETTING STARTED WITH JAVA SERVER PAGES, ACTION ELEMENTS

Unit Structure

13.0 OBJECTIVES

After going through this chapter, you will:

- Understand Java Server Pages documents
- Learn the various elements that can be used in a JSP page
- Lean how to include and forward JSP pages
- Understand what are Java Beans and why it is used?

13.1 INTRODUCTION

A JSP Page looks very similar to a HTML or XML page. It consists of both static and dynamic content. The static content can be written using plain HTML or XML and dynamic content can be written either using the regular java style of programming embedded in specific elements or using the tag style approach.

JSP tags are nothing but holders of Java code spec in an HTML page. This code spec is then executed by the web server whenever the page is requested.

13.2 COMMENTS IN JSP

Comments are text that is written for maintaining JSP pages. These are ignored by the JSP engine when it translates the JSP page into a Servlet. The comment will therefore not be sent to the user (Web Browser) in the response and thus will not be visible using the browser's View Source option.

There are 2 syntax of writing comments in a JSP page:

1) <%-- This is JSP comment --%>

This comment will be ignored by the JSP engine:

2) <!--This is HTML comment -->

This is a HTML comment and will be ignored by the browser.

Example: Program showing how to write JSP comments

```
<html>
  <head>
    <title>JSP Program to show comments</title>
  </head>
  <body>
      Hello World!
    <%-- This is a JSP comment and will not be processed by JSP Engine
--%>
  </body>
</html>
```

13.3 JSP DOCUMENTS

A JSP document can use either the traditional JSP style syntax or XML style JSP syntax within its source file.

JSP pages uses the traditional or short-hand syntax, whereas JSP documents are completely XML-compliant.

JSP documents are also referred to as JSP pages using XML syntax.

Following are the advantages of using JSP documents:

1. JSP documents can be easily verified as well-formed XML/HTML.
2. JSP documents can be validated against an XML-Schema.
3. JSP documents can be readily written and parsed using standard XML tools.
4. JSP uses XML compliant include and forward actions as well as custom tags.
5. JSP documents require slightly more developer discipline than JSP pages. This makes the code spec more readable and maintainable especially to those to whom JSP is new.

13.4 JSP ELEMENTS

JSP page usually provide dynamic behaviour. This means they are supposed to change the response as per specific client requests.

JSP pages can be given dynamic behaviour by embedding Java code in them. To clearly separate JSP elements are used. It helps to inform the JSP translator which part of code is java and which part is HTML.

JSP Elements enclose the Java code in a JSP page and are categorized as follows:

1. Directives
2. Scripting Elements
3. Action Elements

13.5 JSP DIRECTIVES

JSP directives serve special processing information about the page to the JSP Server.

A JSP directive affects the overall structure of the servlet class. They do not produce any output that is visible to the client.

It usually has the following form:

<%@ directive attribute = "value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag:

Sr.No.	Directive & Description
1	<%@ page ... %> Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	<%@ include ... %> Includes a file during the translation phase.
3	<%@ taglib ... %> Declares a tag library, containing custom actions, used in the page

13.5.1 Jsp Page Directive:

The page directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive:

<%@ page attribute = "value" %>

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.page attribute = "value" />

The following are the most common attributes associated with the page directive:

Sr.No.	Attribute and Example
1	language Defines the programming language used in the JSP page. Eg: <%@ page language="java"%>
2	import Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes. Eg: <%@ page import="java.util.Date" %>
3	contentType Defines the character encoding scheme. Eg: <%@ page contentType=application/msword %>
4	extends Specifies a superclass that the generated servlet must extend. Eg: <%@ page extends = "somePackage.SomeClass" %>

5	isErrorPage Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute. Eg: <%@ page isErrorPage="true" %>
6	session Specifies whether or not the JSP page participates in HTTP sessions. Eg: <%@ page session = "true" %>
7	info The info attribute lets you provide a description of the JSP. <%@ page info = "This is a JSP Page" %>

Example:

```
<%@ page contentType="text/html" %>
<html>
   <body>
      Today is: <%= new java.util.Date()%>
   </body>
</html>
```

13.5.2 JSP Include Directive:

The include directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the include directives anywhere in your JSP page.

The general usage form of this directive is as follows:

<%@ include file = "relative url" >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows:

<jsp:directive.include file = "relative url" />

The file attribute:

A page-relative or context-relative URI path to the file that will be included at the current position in the file. This attribute includes a static file ,merging its content with the including page before the combined result is converted to a JSP page implementation class. A page can contain multiple include directives.

Example:

```
<html>
<body>
  <%@ include file="header.html" %>
  Today is: <%= java.util.Calendar.getInstance().getTime() %>
  </body>
</html>
```

13.5.3 JSP Taglib Directive:

The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page. Custom Tags allow developers to hide complex server side code spec from web designers.

A taglib directive in a JSP is a link to an XML document that describes a set of custom tag. This XML document also determine which Tag Handler class implements the action of each tag. The XML document names the tag library which holds the custom tags.The JSP engine uses this tag library to determine what to do when it comes across custom tags

The taglib directive follows the syntax given below:

<%@ taglib uri = "uri" prefix = "prefixOfTag" >

You can write the XML equivalent of the above syntax as follows:

<jsp:directive.taglib uri = "uri" prefix = "prefixOfTag" />

The uri attribute:

A Uniform Resource Identifier (URI) that identifies the Tag Library Descriptor ,which is used to uniquely name the set of custom tags and inform the server what to do with the specified tags.

The prefix attribute:

It defines the prefix string in <prefix>:<tagname> pair and informs the JSP container which bits of markup are custom tags.

Example:

```
<%@ taglib uri = "http://www.abc.com/mylib" prefix = "mytag" %>
<html>
 <body>
  <mytag:hello/>
```

```
</body>
</html>
```

13.6 SCRIPTING ELEMENTS

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- Declaration tag
- Expression tag
- Scriptlet tag

These elements allow declaring variables and methods,including scripting code and evaluating an expression.

13.6.1 Declarations Tag:

The JSP declaration tag is used to declare fields and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

The Syntax of JSP declaration tag is as follows:

```
<%!
 Java variable and  method declaration(s)
 %>
```

You can write the XML equivalent of the above syntax as follows:

```
 <jsp:declaration>
 Java variable and  method declaration(s)
 </jsp:declaration>
```

Example:

```
<%!
        int num=0;
        public void count(){
        {
                int num=10;
}
%>
```

13.6.2 Scriptlets Tag:

A scriptlet is a block of Java code spec that is executed at runtime. Scriptlets also known as JSP code fragments are embedded within <% …. %> tags.

A Scriptlet can produce output passed through an output stream back to the client.

The Syntax of JSP Scriptlets tag is as follows:

<%

Scriptlet code Spec

%>

You can write the XML equivalent of the above syntax as follows:

<jsp:scriptlet>

Scriptlet code Spec

</jsp:scriptlet>

The following example program prints “Welcome to JSP” on the page.

Example:

```
<html>
<body>
<% out.print("Welcome to jsp"); %>
</body>
</html>
```

13.6.3 Expressions Tag:

It is mainly used to print the values of variable or method. The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data.

After an expression is evaluated, the result is converted to a string and displayed.

The Syntax of JSP Expression tag is as follows:

<%= statement %>

You can write the XML equivalent of the above syntax as follows:

<jsp:expression>

Statements:

</jsp:expression>

Example:

<html>

<body>

Current Time: <%= java.util.Calendar.getInstance().getTime() %>

</body>

</html>

13.7 ACTION ELEMENTS

JSP Action tags are used to control the flow between pages and to use Java Bean.

JSP Action Elements are processed during the request processing phase as opposed to JSP directives which are processed during translation. Actions use construct in XML syntax.It looks like a regular HTML tag and does not follow the <% … %> syntax.

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

13.7.1 JSP: forward Action Tag:

The forward action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or

a Java Servlet. It is same as forwarding to resources using RequestDispatcher interface in Servlets.

Syntax:

<jsp:forward page=”<url>”>

<jsp:param name=”<ParameterName>” value=”<ParameterValue>”/>

</jsp:forward>

page attribute:

It is a string or an expression representing the relative URL of the component to which the request is forwarded.

<jsp:param> tag

It is used to send one or more name=value pairs as parameters to a dynamic resource such as JSP,Servlets or other resources.

<jsp:forward > Example without parameter

Index.jsp

<html>

<head>

<title>The forward JSP example</title>

</head>

<body>

<jsp:forward page = "date.jsp" />

</body>

</html>

<jsp:forward > Example with parameter

Index.jsp

<html>

<body>

<h2>Forwarding with Parameters</h2>

<jsp:forward page="date.jsp" >

<jsp:param name="name" value="Saturday" />

</jsp:forward>

```
  </body>
</html>
```

date.jsp

```
<html>
   <body>
      <%out.print("Today is:" +
java.util.Calendar.getInstance().getTime());%>
      <%= request.getParameter("name")%>
   </body>
</html>
```

13.7.2 JSP: Include Action Tag:

The include action terminates the action of the current page and forwards the request to another resource such as a static page, another JSP page, or a Java Servlet. It is same as forwarding to resources using RequestDispatcher interface in Servlets.

Syntax:

```
<jsp:include page="<url>">
<jsp:param name="<ParameterName>" value="<ParameterValue>"/>
</jsp: include>
```

page attribute:

It is a string or an expression representing the relative URL of the component to which the request is forwarded.

<jsp:param> tag:

It is used to send one or more name=value pairs as parameters to a dynamic resource such as JSP,Servlets or other resources.

<jsp:include > Example without parameter

Index.jsp

```
<html>
<head>
 <title>The include JSP example</title>
</head>
<body>
```

```
<jsp:include page = "footer.jsp" />
</body>
</html>
```

<jsp:include > Example with parameter

Index.jsp

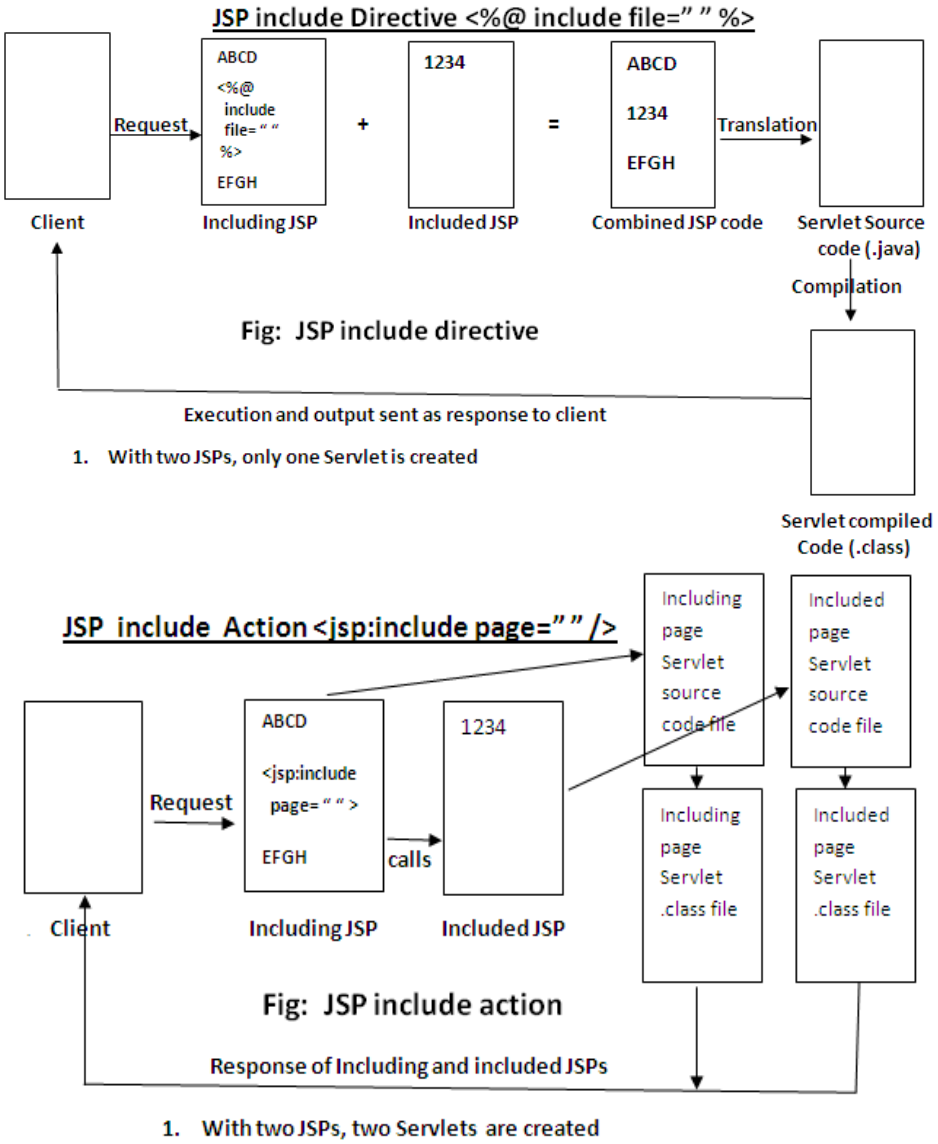
```
<html>
<body>
<h2>Including with Parameters</h2>
<jsp:include page="date.jsp" >
<jsp:param name="name" value="Saturday" />
</jsp:include>
 </body>
</html>
```

date.jsp

```
<html>
<body>
<% out.print("Today is:" + java.util.Calendar.getInstance().getTime());%>
<%= request.getParameter("name")%>
</body>
</html>
```

Note:

- The difference between jsp include directive and include action tag is Include action includes response of a resource into the response of the JSP page
- Include directive includes resources in a JSP page at translation time.



13.7.3 jsp: useBean Action Tag:

Before understanding what is <jsp:useBean> ,we need to first understand JavaBean.

JavaBean

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Example of JavaBean class:

//Employee.java

```
package mypack;

public class Employee implements java.io.Serializable{

private int id;

private String name;

public Employee(){}

public void setId(int id){this.id=id;}

public int getId(){return id;}

public void setName(String name){this.name=name;}

public String getName(){return name;}

}
```

To access the JavaBean class, we should use getter and setter methods.

```
package mypack;
 public class Test{
 public static void main(String args[]){
 Employee e=new Employee();//object is created
  e.setName("Steve");//setting value to the object
 System.out.println(e.getName());
 }
}
```

<jsp:usebean> Action Tag:

In JSP, <jsp:useBean> is used to access the bean. <jsp:useBean> instantiates an object of the class specified by the class and binds it to a variable with the name specified by ID.A new object is instantiated only if there is no existing one with the same ID and scope. Once a bean exists, its properties can be modified using <jsp:setProperty> or by using a scriptlet and calling a method explicitly. Existing properties can be read in a JSP or scriptlet by using <jsp:getProperty>.

<jsp:useBean> makes a JavaBean available to a JSP Page and ensures that the bean object is available for an appropriate scope specified in the element.

Syntax of jsp:useBean action tag:

```
<jsp:useBean id= "instanceName" scope= "page | request | session | applic
ation"

class= "packageName.className" type= "packageName.className"
```

beanName="packageName.className | <%= expression >" >

</jsp:useBean>

Attributes and Usage of jsp:useBean action tag:

1. **id:** is used to identify the bean in the specified scope.
2. **scope:** represents the scope of the bean. It may be page, request, session or application. The default scope is page.
   - **page:** specifies that you can use this bean within the JSP page. The default scope is page.
   - **request:** specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.
   - **session:** specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
   - **application:** specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.
3. **class:** instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. **type:** provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. **beanName:** instantiates the bean using the java.beans.Beans.instantiate() method.

Example:

Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator{
  public int cube(int n){return n*n*n;}
  }
```

index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>
   <%
int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

13.7.4 <jsp:setProperty> Action Tag:

<jsp:setProperty> is used in conjunction with <jsp:useBean> and sets the value of simple and indexed properties in a bean.

The properties in a bean can be set either:

- At request time from parameters in the request object or
- At request time from an evaluated expression or
- From a specified string(or hard coded in the page)

Syntax of jsp:setProperty Action Tag

```
<jsp:setProperty name="instanceOfBean" property= "*"   |
  property="propertyName" param="parameterName"  |
  property="propertyName" value="{ string | <%= expression %>}"
 />
```

Example:

```
<jsp:setProperty name="bean" property="*" />
<jsp:setProperty name="bean" property="username" />
<jsp:setProperty name="bean" property="username" value="Chris" />
```

13.7.5 jsp:getProperty action tag:

The jsp:getProperty action tag accesses the value of a bean property,converts it to a String and prints it.

Syntax of jsp:getProperty action tag

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Example:

```
<jsp:useBean id="obj" scope="page" class="mypack.Student"/>
<jsp:getProperty name="obj" property="name" />
```

Here, <jsp:getProperty> invokes getName() available in the Student class.

13.8 JSP GUI EXAMPLE

Create a registration and login JSP application to register and authenticate the user based on username and password using JDBC.

Initial.html:

```
<a href="index.html">Sign up for New User</a> <br><br>
```

```
<a href="login.html">Login for Existing User</a>
```

Index.html

```
<html>
   <head>
      <title>Registration Page</title>
   </head>
   <body>
      <form action="Register.jsp" >
         <h1> New User Registration Page</h1>
         Enter UserName <input type="text" name="txtName" ><br>
         Enter Password <input type="password" name="txtPass1" ><br>
         Re-Enter Password<input type="password" name="txtPass2"
><br>
         Enter Email<input type="text" name="txtEmail" ><br>
         Enter Country Name <input type="text" name="txtCon" ><br>
         <input type="reset" >
         <input type="submit" value="REGISTER" >
      </form>
   </body>
</html>
```

Register.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8"
import="java.sql.*"%>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-
8">
      <title>Registration JSP Page</title>
   </head>
   <body>
      <%
         String uname = request.getParameter("txtName");
         String pass1 = request.getParameter("txtPass1");
         String pass2 = request.getParameter("txtPass2");
         String email = request.getParameter("txtEmail");
         String ctry = request.getParameter("txtCon");
```

```
            if (pass1.equals(pass2)) {
                try {
                    Class.forName("org.apache.derby.jdbc.ClientDriver");
                    Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/mydb", "root", "root");
                    PreparedStatement stmt = con.prepareStatement("insert into UserDetails values(?,?,?,?)");
                    stmt.setString(1, uname);
                    stmt.setString(2, pass1);
                    stmt.setString(3, email);
                    stmt.setString(4, ctry);
                    int row = stmt.executeUpdate();
                    if (row == 1) {
                        out.println("<h1>Registration Successful!!!</h1>");
                         out.println("<br><a href='login.html'>Login here</a>");
                    } else {
                        out.println("<h1>Registration Failed!!!</h1>");
                            %>
        <jsp:include page="index.html"/>
        <%
                }
                } catch (Exception e) {
                    out.println(e);
                }
            } else {
                out.println("<h1>Password Mismatch!!!</h1>");
            %>
            <jsp:include page="index.html"/>
            <%   }
            %>
        </body>
    </html>
```

Login.html

```
<html>
    <head>
```

```
    <title>Login Application</title>
    </head>
  <body>
    <form method="post" action="login.jsp">
      <h1> Login Application</h1>
      Enter Username <input type="text" name="t1"><br><br>
      Enter Password <input type="password" name="t2"><br><br>
      <input type="reset">
      <input type="submit" value="Login">
    </form>
  </body>
</html>
```

login.jsp

```
<%@page contentType="text/html" pageEncoding="UTF-8" import="java.sql.*"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Login JSP Page</title>
  </head>
  <body>
    <%
      String uname = request.getParameter("t1");
      String pass = request.getParameter("t2");
          try {
        Class.forName("org.apache.derby.jdbc.ClientDriver");
        Connection con = DriverManager.getConnection("jdbc:derby://localhost:1527/mydb", "root", "root");
        Statement stmt = con.createStatement();
       ResultSet rs = stmt.executeQuery("select password from UserDetails where username='" + uname + "'");
        rs.next();
        if (pass.equals(rs.getString(1))) {
          out.println("<h1>Welcome"+uname+"</h1>");
```

```
            } else {
                out.println("<h1>Login Failed!!!</h1>");
        %>
        <jsp:include page="login.html"/>
        <%
            }
        } catch (Exception e) {
    out.println("<h1>User does not exist!!!!!</h1>");
        %>
        <jsp:include page="login.html"/>
            <%
            }
        %>
    </body>
</html>
```

13.9 SUMMARY

- JSP comments are ignored by the JSP engine when it translates the JSP page into a Servlet.
- A JSP document can use either the traditional JSP style syntax or XML style JSP syntax within its source file.
- JSP Elements enclose the Java code in a JSP page and are categorized as follows:

  1. Directives
  2. Scripting Elements
  3. Action Elements

- JSP directives serve special processing information about the page to the JSP Server.
- The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

  1. Declaration tag
  2. Expression tag
  3. Scriptlet tag

- JSP Action tags are used to control the flow between pages and to use Java Bean.

13.10 LIST OF REFERENCES

1. Java EE 7 for Beginners, Sharanam Shah, Vaishali Shah, First Edition, SPD

Web References:

1. https://www.javatpoint.com
2. https://www.tutorialspoint.com
3. https://www.guru99.com

13.11 QUESTIONS

Q1. What are the different ways of writing comments in Java?

Q2. What are the various attributes used in the page directive?Explain with an example.

Q3. What is the benefit of using taglib directive?

Q4. What are the various Scripting elements available in JSP? Why are they used?

Q5. What is the difference between include directive and include action tag?

Q6. What is a JavaBean?Why is it used?Explain with an example.

Q7. How is <jsp:useBean> action tag used to set and access properties of a JavaBean?

*****

14

IMPLICIT OBJECTS, SCOPE AND EL EXPRESSIONS

Unit Structure

14.0 OBJECTIVES

After going through this chapter, you will:

- You will understand what are the various implicit objects in JSP
- Learn the various scope of objects in JSP
- Know what are character quoting conventions and how to write them
- Learn what is Unified Expression Language and the benefits of using them

14.1 IMPLICIT OBJECTS

Java Scripting Elements provide a great deal of power and flexibility to the developer to achieve dynamic website content delivery.To achieve this, JSP engine exposes a number of internal Java objects to the developer. These objects do not need to be declared or instantiated by the developer but are provided by the JSP engine in its implementation and its execution.

All implicit objects are available only to scriptlets or expressions.They are not available in declarations.

There are 9 implicit objects in JSP as follows:

Object	Type
out	JspWriter
request	HttpServletRequest
response	HttpServletResponse
config	ServletConfig
application	ServletContext
session	HttpSession
pageContext	PageContext
page	Object
exception	Throwable

1. out:

This is the JspWriter object associated with the output stream of the response.For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

PrintWriter out=response.getWriter();

But in JSP, you don't need to write this code as out is pre-defined.

Example

<html>

<body>

<% out.print("This is Enterprise Java"); %>

</body>

</html>

Here, we are simply printing the line "This is Enterprise Java" as the response.

2. Request:

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.

It can also be used to set, get and remove attributes from the jsp request scope.

Example

index.html

<form action="welcome.jsp">

<input type="text" name="uname">

```
<input type="submit" value="go"><br/>
</form>
welcome.jsp
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

3. response:

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example of response implicit object

```
index.html
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
welcome.jsp
<%
response.sendRedirect("http://www.google.com");
%>
```

4) config:

In JSP, config is an implicit object of type ServletConfig. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page.

Generally, it is used to get initialization parameter from the web.xml file.

Example:

```
index.html
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
  <init-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</init-param>
  </servlet>
  <servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
  </web-app>
```

welcome.jsp:

```
<%
out.print("Welcome "+request.getParameter("uname"));
  String driver=config.getInitParameter("dname");
out.print("driver name is="+driver);
%>
```

5) application:

In JSP, application is an implicit object of type ServletContext.

The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope.

t("driver namExample:

index.html

```
<form action="welcome">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<jsp-file>/welcome.jsp</jsp-file>
</servlet>
  <servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
  <context-param>
<param-name>dname</param-name>
<param-value>sun.jdbc.odbc.JdbcOdbcDriver</param-value>
</context-param>
  </web-app>
```

welcome.jsp

```
<%
  out.print("Welcome "+request.getParameter("uname"));
  String driver=application.getInitParameter("dname");
out.prine is="+driver);
%>
```

6. session:

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

Example

index.html

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
  String name=request.getParameter("uname");
out.print("Welcome "+name);
  session.setAttribute("user",name);
  <a href="second.jsp">second jsp page</a>
  %>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)session.getAttribute("user");
out.print("Hello "+name);
%>
</body>
</html>
```

7. pageContext:

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, page scope is the default scope.

Example

index.html

```
<html>
<body>
<form action="welcome.jsp">
```

```
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
</body>
</html>
```

welcome.jsp

```
<html>
<body>
<%
String name=request.getParameter("uname");
out.print("Welcome "+name);
pageContext.setAttribute("user",name,PageContext.SESSION_SCOPE);
<a href="second.jsp">second jsp page</a>
%>
</body>
</html>
```

second.jsp

```
<html>
<body>
<%
String name=(String)pageContext.getAttribute("user",PageContext.SESSI
ON_SCOPE);
out.print("Hello "+name);
%>
</body>
</html>
```

8) page:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type.For example:

<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

<% this.log("message"); %>

9) exception:

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive. Let's see a simple example:

Example

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>
  Sorry following exception occured:<%= exception %>
  </body>
</html>
```

14.2 SCOPE OF JSP OBJECTS

The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object. Every object created in a JSP page will have a scope. Object scope in JSP is segregated into four parts and they are page, request, session and application.

Page Scope:

Objects with page scope are accessible only within the page in which they're created. The data is valid only during the processing of the current response; once the response is sent back to the browser, the data is no longer valid. If the request is forwarded to another page or the browser makes another request as a result of a redirect, the data is also lost.

//Example of JSP Page Scope

```
<jsp:useBean id="employee" class="EmployeeBean" scope="page" />
```

Request Scope:

Objects with request scope are accessible from pages processing the same request in which they were created. Once the container has processed the request, the data is released. Even if the request is forwarded to another page, the data is still available though not if a redirect is required.

//Example of JSP Request Scope

```
<jsp:useBean id="employee" class="EmployeeBean" scope="request" />
```

Session Scope:

Objects with session scope are accessible from pages processing requests that are in the same session as the one in which they were created. A session is the time users spend using the application, which ends when

they close their browser, when they go to another Web site, or when the application designer wants (after a logout, for instance). So, for example, when users log in, their username could be stored in the session and displayed on every page they access. This data lasts until they leave the Web site or log out.

//Example of JSP Session Scope

<jsp:useBean id="employee" class="EmployeeBean" scope="session" />

Application Scope:

Objects with application scope are accessible from JSP pages that reside in the same application. This creates a global object that's available to all pages.

Application scope uses a single namespace, which means all your pages should be careful not to duplicate the names of application scope objects or change the values when they're likely to be read by another page (this is called thread safety). Application scope variables are typically created and populated when an application starts and then used as read-only for the rest of the application.

//Example of JSP Application Scope

<jsp:useBean id="employee" class="EmployeeBean" scope="application" />

14.3 CHARACTER QUOTING CONVENTIONS

Because certain character sequences are used to represent start and stop tags, the developer sometimes needs to escape a character so the JSP engine does not interpret it as part of a special character sequence.

In a scripting element, if the character needs %> needs to be used,escape the greater than sign with a backslash.

<%String message=" This is the %/> message";%>

The backslash before the expression acts as an escape character and informs the JSP engine to not evaluate it.

There are a number of cases where backslash needs to be used otherwise characters will be treated specially by the JSP engine.

Escape Characters	Description
\'	A single quote in an attribute that uses single quote
\"	A double quote in an attribute that uses double quote
\\	A backslash in an attribute that uses backslash

%\>	Escaping the scriptlet end tag with a backslash
<\%	Escaping the scriptlet start tag with a backslash
\$	Escaping the $ sign with a backslash

14.4 UNIFIED EXPRESSION LANGUAGE (UEL)

JSP Expression Language provides a way to simplify expressions. It is a simple language used for accessing implicit objects. Java classes and for manipulating collections in an elegant manner. It is the newly added feature in JSP technology version 2.0.

The expression language also allows page authors to use simple expressions to dynamically read data from JavaBean components.

Unified Expression Language allows usage of simple expressions to perform the following tasks:

- Dynamically read application data stored in JavaBeans components, various data structures and implicit objects.
- Dynamically write data such user input into forms to JavaBeans components.
- Dynamically perform arithmetic operations.

14.4.1 Types of Evaluation Expressions:

Unified EL supports two types of evaluation expressions: Immediate and Deferred evaluation

Immediate Evaluation:

Immediate evaluation means that the expression is evaluated and the result returned as soon as the page is first rendered.

Syntax:

${<Expression>}

Here, Expression stands for valid expression.

The following example shows a tag whose value attribute references an immediate evaluation expression that updates the quantity of books retrieved from the backing bean named catalog:

<h:outputText value="${catalog.bookQuantity}" />

Deferred Evaluation:

Deferred evaluation means that the technology using the expression language can use its own machinery to evaluate the expression sometime later during the page's lifecycle, whenever it is appropriate to do so.

Syntax:

#{<Expression>}

Here, Expression stands for valid expression.

Because of its multiphase lifecycle, JavaServer Faces technology uses mostly deferred evaluation expressions. During the lifecycle, component events are handled, data is validated, and other tasks are performed in a particular order. Therefore, a JavaServer Faces implementation must defer evaluation of expressions until the appropriate point in the lifecycle.

Other technologies using the EL might have different reasons for using deferred expressions.

The following example shows a JavaServer Faces h:inputText tag, which represents a field component into which a user enters a value. The h:inputText tag's value attribute references a deferred evaluation expression that points to the name property of the customer bean:

<h:inputText id="name" value="#{customer.name}" />

14.4.2 Value Expressions:

The unified EL provides two types of value expressions:

- **Rvalue Expressions:**

Can only read data, but cannot write data. Expressions that use deferred valuation syntax are always rvalue expressions.

- **Lvalue Expressions:**

Can read and write data. Expressions that uses deferred evaluation syntax can act as both Rvalue and Lvalue expressions.

Consider the following two value expressions:

${customer.name}

#{customer.name}

The former uses immediate evaluation syntax, whereas the latter uses deferred evaluation syntax. The first expression accesses the name property, gets its value, and passes the value to the tag handler. With the second expression, the tag handler can defer the expression evaluation to a later time in the page lifecycle if the technology using this tag allows.

In the case of JavaServer Faces technology, the latter tag's expression is evaluated immediately during an initial request for the page. During a postback request, this expression can be used to set the value of the name property with user input.

14.4.3 Method Expressions:

EL also supports deferred method expressions. A method expression is used to refer to a public method of a bean and has the same syntax as an lvalue expression.

A JSF component element usesmethod expressions, which in turn invokes method that do some process on behalf of the component element. For standard components, these methods are necessary for handling events that the components generates as well as validating component data.

Example

Solution:

<h:inputTextid=”firstnameid” value=”#{customer.firstname}”

Validator=”#{customer.validateFirstname}”/>

Explanation:

The validator attribute of <h:inputText> references validateFirstnameid() owned by a bean called customer.

<h:inputText> specifies that validateFirstName() should be invoked during the validation process phase of the JSF lifecycle.

Because a method can be invoked during different phases of the lifecycle, method expressions must always use the deferred evaluation syntax.

14.4.4 Operators:

EL the following operators, most of which are usual operators available in Java:

1. Arithmetic Operators:

The following are the arithmetic operators:

- +Addition
- [Binary]:(subtraction)
- :Multiplication
- / or div:Division
- % or mod:modulo[remainder]
- -[unary]:Negation of a value

Example

1. ${5*5+4}
2. ${1.2E4+1.4}

3. ${10 mod 4}
4. ${3 div 4}

Output:

1. 29
2. 12001.4
3. 2
4. 0.75

2. Logical Operators:

The following are the logical operators:

- && or AND: Test for logical AND
- || or OR : Test for logical OR
- ! or NOT :Unary Boolean complement

Example

<%-- Evaluates if variable is not empty --%>

${!empty<VariableAName>}

3. Relational Operators:

The following are the relational operators:

== or eq :Test for equality

!= or ne: :Test for inequality

< or lt: Test for less than

> or gt : Test for greater than

<= or le : Test for less than or equal

>= or ge : Test for greater than or equal

Example:

${10>3}

${1>8}

${10 le 3}

4. Conditional Operators:

The following is the syntax for conditional operators:

Condition ? If true :If false

Solution

A?B:C

Here B is evaluated if A is true else is evaluated if A is false

5. The [Dot] Operator:

It is a shorthand for calling a JavaBeans property accessor for the property whose name is on the right side of the operator.

Solution

${pageContext.servletContext.servletContextName}

6. The [] Operator:

Is is used for polymorphic indexing, which can be used for indexing collections including Maps,Lists and Arrays. The value inside the brackets is used as a key into a map or as a List or array index.

Example:

${colors[5]}

${colors[1]>colors[6]}

7. The empty operator:

It is a prefix operator that is used to determine if a value is null or empty.

Example

${empty Name}

This expression returns true if Name refers a null value.

14.4.5 JSP EL IMPLICIT OBJECTS:

The JSP expression language supports the following implicit objects :

Sr. No.	Implicit object & Description
1	pageScope Scoped variables from page scope
2	requestScope Scoped variables from request scope
3	sessionScope Scoped variables from session scope

4	applicationScope Scoped variables from application scope
5	param Request parameters as strings
6	paramValues Request parameters as collections of strings
7	header HTTP request headers as strings
8	headerValues HTTP request headers as collections of strings
9	initParam Context-initialization parameters
10	cookie Cookie values
11	pageContext The JSP PageContext object for the current page

14.5 SUMMARY

- JSP provides a number of implicit objects which need not be declared or instantiated by the developer.
- Implicit objects are available only to scriptlets or expressions.They are not available in declarations.
- JSP objects have 4 scopes: page,request,session and application
- The escape character backslash can be used to inform the JSP engine not to evaluate certain expressions.
- Unified Expression Language provides a way to simplify expressions and can be used for accessing implicit objects.
- UEL supports both immediate and deferred evaluation.
- UEL also supports various operators and can be used to call methods.

14.6 LIST OF REFERENCES

Java EE 7 for Beginners, Sharanam Shah, Vaishali Shah, First Edition, SPD

Web References:

1. https://www.javatpoint.com
2. https://www.java-samples.com

14.7 QUESTIONS

Q1. List the various implicit objects in JSP.

Q2. Explain the scope of JSP objects.

Q3. What is Immediate and Deferred Evaluation? Explain with an example.

Q4. Write a note on character quoting conventions.

Q5. Explain Method Expressions in short.

Q6. What are the various operators supported by JSP EL?

*****

15

JSP STANDARD TAG LIBRARIES

Unit Structure

15.0 OBJECTIVES

After going through this chapter, you will:

- Understand what is Java Server Pages Tag Libraries
- Understand the advantages and disadvantages of JSTL
- Learn what are the issues of using Scriptlet Tags
- Learn the various types of Tag Libraries and available tags

15.1 INTRODUCTION TO JSP STANDARD TAG LIBRARIES

JSTL was introduced to allow JSP programming developers to create web applications using tags rather than scriptlets (Java code). It is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications. JSTL does nearly everything that a regular scriptlet does.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

15.2 DISADVANTAGES OF JSP SCRIPTLET TAGS

1. The Java code embedded within the Scriptlet looks ugly and inconsistent with the HTML tags.
2. The developer who does not know Java actually cannot modify the embedded Java code. Thus, this disadvantage nullifies the major benefit of JSP, which is the empowerment of designers and business people to update page content.
3. The Java code embedded within the scriptlets cannot be re-used by another JSP Pages. So, the common logic code gets duplicated in multiple JSP pages.
4. Accessing values from HTTP request and sessions needs to be specifically typecasted to the object's class, which should be known to the JSP by importing or fully qualifying the class name.

15.3 ADVANTAGES OF JSTL

1. JSTL tags are XML based tags, they cleanly and consistently blend into a page's HTML markup tags.
2. JSTL tags are easier to use effectively as they do not require any knowledge of Java programming.
3. JSTL tags can be reused in various pages unlike scriptlets which needs to be repeated everywhere.
4. JSTL tags can reference objects in Request and Session objects without knowing the object's type with no typecasting required.
5. JSTL makes use of UEL which makes it easier to call the getter and setter methods on Java objects.

15.4 DISADVANTAGES OF JSTL

1. JSTL increases the processing burden on the server. Java scriptlet and the tag libraries both are compiled into a servlet, which is then executed by the Servlet engine. Java code in scriptlets is pretty much just copied into Servlets but on the other hand, JSTL tags casue much more code to be added to the Servlet.
2. JSTL provides a powerful set of reusable libraries to JSP developers but JSTL cannot do everything that the Java code spec can do.

15.5 HOW IS JSTL DIFFERENT FROM SCRIPTLETS

An example of scriptlet-based programming, which counts to 10, is shown here:

<html>

```
 <head>
  <title>Count to 10 in JSP scriptlet</title>
 </head>
 <body>
<%
for(int i=1;i<=10;i++)
{
%>
<%=i%><br/>
<%
}
%>
 </body>
</html>
```

As you can see from the preceding example, using scriptlet code produces page source code that contains a mix of HTML tags and Java statements making the code looking non-consistent.

Consider the following example, which shows how to count from 1 to 10 using JSTL rather than scriptlet code.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
 <head>
  <title>Count to 10 Example (using JSTL)</title>
 </head>
 <body>
  <c:forEach var="i" begin="1" end="10" step="1">
   <c:out value="${i}" />
   <br />
  </c:forEach>
 </body>
</html>
```

When you examine the preceding source code, you can see that the JSP page consists entirely of tags thus bringing consistency and uniformity to the code.

15.6 TYPES OF JSTL TAG LIBRARIES

JSTL Tag Libraries can be broken down into specific functional areas belonging to an application. JSTL is composed of five tag libraries:

- Core Tag Library
- Functions Tag Library
- Database/SQL Tag Library
- Formatting Tag Library
- XML Tag Library

Let’s understand the various tags under each of the above Tag Libraries.

15.6.1 Core Tag Library:

The Core Tag Library contains tags that are essential to nearly any Web application. Examples of core tag libraries are looping, evaluation of expression and basic input and output.

The URI of the Core Tag Library is “http://java.sun.com/jsp/jstl/core” and prefix is c.

The syntax used for including JSTL Core tags library in your JSP is:

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

The Core Tag Library consists of four distinct functional sections:

A) General - Purpose Actions:

These actions allow adding and removing variables ,displaying variable values and enclosing a group of tags within a try-catch block.

1. <c:out>

The <c:out> tag displays the result of an expression. This is almost similar to the way <%= %> works.

Example:

<c:out value = "${'Hello World’}"/>

This will print Hello World as a response.

2. <c:set>

The <c:set> tag sets the result of an expression evaluation in a 'scope'.

Example

<c:set var = "salary" scope = "session" value = "${2000}"/>

This will set a variable names session with the value 2000.

3. <c:remove >

The <c:remove > tag removes a scoped variable (from a particular scope, if specified).

Example:

<c:remove var = "salary"/>

This will remove the variable named salary.

4. <c:catch>

The <c:catch> tag catches any Throwable that occurs in its body and optionally exposes it.

Example:

<c:catch var ="catchException">

<% int x = 5/0;%>

</c:catch>

This code block will catch ArithmeticException.

B) Conditional Actions Or Flow Control Statements:

Conditional Actions are used for conditional processing within a JSP page.

1. <c:if>

The <c:if> tag evaluates an expression and displays its body content only if the expression evaluates to true.

Example:

<c:if test = "${salary > 20000}">

<p>My salary is: <c:out value = "${salary}"/><p>

</c:if>

This will print the statement within <c:if > tag if the condition mentioned in test evaluates to true.

2. <c:choose>,<c:when>,<c:otherwise>:

The <c:choose> works like a Java switch statement in that it lets you choose between a number of alternatives. Where the switch statement has case statements, the <c:choose> tag has <c:when> tags. Just as a switch statement has the default clause to specify a default action, <c:choose> has <c:otherwise> as the default clause.

Example:

<c:set var="number1" value="${222}"/>

<c:set var="number2" value="${12}"/>

<c:set var="number3" value="${10}"/>

<c:choose>

```
 <c:when test="${number1 < number2}">
    ${"number1 is less than number2"}
 </c:when>
 <c:when test="${number1 <= number3}">
    ${"number1 is less than equal to number2"}
 </c:when>
 <c:otherwise>
   <c:out value=" ${'number1 is largest number!'}"/>
 </c:otherwise>
</c:choose>Example
<c:set var="number1" value="${222}"/>
<c:set var="number2" value="${12}"/>
<c:set var="number3" value="${10}"/>
<c:choose>
 <c:when test="${number1 < number2}">
    ${"number1 is less than number2"}
 </c:when>
 <c:when test="${number1 <= number3}">
    ${"number1 is less than equal to number2"}
 </c:when>
 <c:otherwise>
   <c:out value=" ${'number1 is largest number!'}"/>
 </c:otherwise>
</c:choose>
```

C) Iterator Actions:

Iterator Actions simplify iteration through collection of objects.

1. <c:forEach >

The <c:forEach> tag is a commonly used tag because it repeats the nested body content for fixed number of times or over collection.

Example

```
  <c:forEach var = "i" begin = "1" end = "5">
        Item <c:out value = "${i}"/><p>
   </c:forEach>
```

This will print the values from 1 to 5.

2. <c:forTokens>

The <c:forTokens> tag iterates over tokens which is separated by the supplied delimeters. It is used for break a string into tokens and iterate through each of the tokens to generate output.

This tag has similar attributes as <c:forEach> tag except one additional attributes delims which is used for specifying the characters to be used as delimiters.

Example

<c:forTokens items=”Chris-Steve-Liza" delims="-" var="name">

<c:out value="${name}"/><p>

</c:forTokens>

This will print the names as separate tokens.

D.) URL RELATED ACTIONS:

These actions are used to import resources, redirect HTTP responses ,create URLs or encode a request of parameters.

1. <c:redirect>

The < c:redirect > tag redirects the browser to a new URL.

Example:

<c:redirect url="http://abc.com"/>

This will redirect to abc.com

2. <c:url>

The < c:url > tag creates a URL with optional query parameter. It is used for url encoding or url formatting. This tag automatically performs the URL rewriting operation.

Example:

<c:url value="/Register.jsp"/>

3. <c:param>

The < c:param > tag add the parameter in a containing 'import' tag's URL.

Example:

<c:url value="/index.jsp" var="completeURL"/>

<c:param name="user" value="Ann"/>

15.6.2 Functions Tag Library:

The Functions Tag Library provides a number of standard functions, most of these functions are common string manipulation functions.

The URI of the Functions Tag Library is "http://java.sun.com/jsp/jstl/functions" and prefix is fn.

The syntax used for including JSTL Functions tags library in your JSP is:

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>

Tag	Explanation	Example	Output
fn:contains()	It is used to test if an input string containing the specified substring in a program.	${fn:contains('Java','av')}	True
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.	${fn:containsIgnoreCase('Java', 'AV')}	True
fn:endsWith()	It is used to test if an input string ends with the specified suffix.	${fn:endsWith('JSP Program', 'Program')}	True
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup.	${fn:escapeXml('It is <xyz>second String.</xyz>')}	It is <xyz>second String.</xyz>
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.	${fn:indexOf('Hello World',' 'Hello') }	0
fn:trim()	It removes the blank spaces from both the ends of a string.	${fn:trim(Welcome to JSP programming)}	Welcome to JSP programming
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.	${fn:endsWith('JSP Program', 'JSP')}	True
fn:split()	It splits the string into an array of substrings.	${fn:split(Welcome-to-JSP-Programming, '-')}	Welcome To JSP Programming
fn:toLowerCase()	It converts all the characters of a string to lower case.	${fn:toLowerCase("HELLO")}	hello

fn:toUpperCase()	It converts all the characters of a string to upper case.	${fn:toLowerCase("hello")}	HELLO
fn:substring()	It returns the subset of a string according to the given start and end position.	${fn:substring("This is the first string.", 5, 17)}	is the first
fn:substringAfter()	It returns the subset of string after a specific substring.	${fn:substringAfter(“Chris Kevin”, "Chris")}	Kevin
fn:substringBefore()	It returns the subset of string before a specific substring.	${fn:substringBefore(“Chris Kevin”, "Kevin")}	Chris
fn:length()	It returns the number of characters inside a string, or the number of items in a collection.	${fn:length(“Hello”)}	5
fn:replace()	It replaces all the occurrence of a string with another string sequence.	${fn:replace(“Chris Kevin”, "Chris", "Steve")}	Steve Kevin

15.6.3 Database/ SQL Tag Library:

The SQL tag library allows the tag to interact with RDBMSs (Relational Databases) such as Microsoft SQL Server, mySQL, or Oracle.

The URI of the SQL Tag Library is “http://java.sun.com/jsp/jstl/sql” and prefix is sql.

The syntax used for including JSTL Database tags library in your JSP is:

<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

1) <sql:setDataSource>

The <sql:setDataSource> tag is used to create the data source variable directly from JSP and it is stored inside a scoped variable. It can be used as input for other database actions.

Example:

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost/test"

user="root" password="1234"/>

This code is used for setting the connection with database server.

2) <sql:query>

The <sql:query> tag is used for executing the SQL query defined in its sql attribute or the body. It is used to execute an SQL SELECT statement and saves the result in scoped variable.

Example:

<sql:query dataSource="${db}" var="rs">

SELECT * from Students;

</sql:query>

3) <sql:update>

The <sql:update> tag is used for executing the SQL DML query defined in its sql attribute or in the tag body. It may be SQL UPDATE, INSERT or DELETE statements.

Example:

<sql:update dataSource="${db}" var="count">

INSERT INTO Students VALUES (154,’Chris’, 'Kevin', 25);

</sql:update>

4. <sql:param>

The <sql:param> tag sets the parameter value in SQL statement.

It is used as nested tag for <sql:update> and <sql:query> to provide the value in SQL query parameter. If null value is provided, the value is set at SQL NULL for value attribute.

Example:

<c:set var="StudentId" value="152"/>

<sql:update dataSource="${db}" var="count">

DELETE FROM Students WHERE Id = ?

<sql:param value="${StudentId}" />

</sql:update>

5. <sql:dateParam>

The <sql:dateParam> is used to set the specified date for SQL query parameter.

It is used as nested tag for <sql:update> and <sql:query> to provide the date and time value for SQL query parameter.

If null value is provided, the value is set at SQL NULL.

Example:

```
<%
Date DoB = new Date("2000/10/16");
int studentId = 151;
%>
<sql:update dataSource="${db}" var="count">
  UPDATE Student SET dob = ? WHERE Id = ?
  <sql:dateParam value="<%=DoB%>" type="DATE" />
  <sql:param value="<%=studentId%>" />
</sql:update>
```

6. <sql:transaction>

The <sql:transaction> tag is used for transaction management. It is used to group multiple <sql:update> into common transaction. If you group multiple SQL queries in a single transaction, database is hit only once.

It is used for ensuring that the database modifications are performed by the nested actions which can be either rolled back or committed.

Example:

```
<%
Date DoB = new Date("2000/10/16");
int studentId = 151;
%>
<sql:transaction dataSource="${db}">
  <sql:update var="count">
    UPDATE Student SET First_Name = ‘Ann’ WHERE Id = 150
  </sql:update>
  <sql:update var="count">
    UPDATE Student SET Last_Name= ‘Seema’ WHERE Id = 153
  </sql:update>
  <sql:update var="count">
   INSERT INTO Student VALUES (101,'Kate', 'David', '2021/10/7');
  </sql:update>
</sql:transaction>
```

15.6.4 Formatting Tag Library:

The formatting tags provide support for message formatting, number and date formatting etc.

The url for the formatting tags is http://java.sun.com/jsp/jstl/fmt and prefix is fmt.

The syntax used for including JSTL FORMATTING tags library in your JSP is:

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

1. <fmt:parseNumber>

The <fmt:parseNumber> tag is used to Parses the string representation of a currency, percentage, or number. It is based on the customized formatting pattern.

Example:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<c:set var="Amount" value="786.970" />
<fmt:parseNumber var="j" type="number" value="${Amount}" />
<p><i>Amount is:</i>  <c:out value="${j}" /></p>
  <fmt:parseNumber var="j" integerOnly="true" type="number" value="$
{Amount}" />
<p><i>Amount is:</i>  <c:out value="${j}" /></p>
```

2. <fmt:formatNumber>

The <fmt:formatNumber> tag is used to format the numerical value using the specific format or precision. It is used to format percentages, currencies, and numbers according to the customized formatting pattern.

Example:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<body>
<h3>Formatting of Number:</h3>
<c:set var="Amount" value="9850.14115" />
<p> Formatted Number-1:
<fmt:formatNumber value="${Amount}" type="currency" /></p>
<p>Formatted Number-2:
<fmt:formatNumber type="number" groupingUsed="true" value="${Amo
unt}" /></p>
<p>Formatted Number-3:
<fmt:formatNumber type="number" maxIntegerDigits="3" value="${Am
ount}" /></p>
```

```
<p>Formatted Number-4:
<fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /></p>
<p>Formatted Number-5:
<fmt:formatNumber type="percent" maxIntegerDigits="4" value="${Amount}" /></p>
<p>Formatted Number-6:
<fmt:formatNumber type="number" pattern="###.###$" value="${Amount}" /></p>
</body>
```

3. <fmt:parseDate>

The <fmt:parseDate> tag parses the string representation of a time and date. It is used to format the time and date according to a customized formatting pattern.

Example:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
<head>
<title>fmt:parseDate Tag</title>
</head>
<body>
<h3>Parsed Date:</h3>
<c:set var="date" value="13-09-2021" />
<fmt:parseDate value="${date}" var="parsedDate"  pattern="dd-MM-yyyy" />
<p><c:out value="${parsedDate}" /></p>
</body>
</html>
```

4. <fmt:bundle>

The <fmt:bundle> tag loads the resource bundle which is used by its tag body. This tag will make the specified bundle available for all <fmt:message> tags that occurs between the boundary of <fmt:bundle> and </fmt:bundle> tags.

It is used to create the ResourceBundle objects which will be used by their tag body.

Let us define the default resource bundle Simple.java as follows:

```
package com.javatpoint;
import java.util.ListResourceBundle;
public class Simple extends ListResourceBundle {
 public Object[][] getContents() {
return contents;
}
 static final Object[][] contents = { { "colour.Violet", "Violet" },
{ "colour.Indigo", "Indigo" }, { "colour.Blue", "Blue" }, };
}
```

Now you can use the below JSTL tags to display the three colors as follows:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
<head>
<title>fmt:bundle Tag</title>
</head>
<body>
  <fmt:bundle basename="com.javatpoint.Simple" prefix="colour.">
   <fmt:message key="Violet"/><br/>
   <fmt:message key="Indigo"/><br/>
  <fmt:message key="Blue"/><br/>
  </fmt:bundle>
  </body>
</html>
```

Output:

1. Violet
2. Indigo
3. Blue

5. <fmt:setTimeZone>

The <fmt:setTimeZone> tag store the time zone inside a time zone configuration variable. It is used for copy a time zone object inside a specified scope variable.

Let's see the simple example to understand the formatting <fmt:setTimeZone> tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
<head>
<title>fmt:setTimeZone Tag</title>
</head>
<body>
<c:set var="date" value="<%=new java.util.Date()%>" />
<p><b>Date and Time in Indian Standard Time(IST) Zone:</b>
 <fmt:formatDate value="${date}" type="both" timeStyle="long" dateStyl
e="long" /></p>
<fmt:setTimeZone value="GMT-10" />
<p><b>Date and Time in GMT-
10 time Zone: </b><fmt:formatDate value="${date}"
 type="both" timeStyle="long" dateStyle="long" /></p>
</body>
</html>
```

6. <fmt:setBundle> and <fmt:message>

The <fmt:setBundle> tag is used to load the resource bundle and store their value in the bundle configuration variable or the name scope variable.

It is used for creating the ResourceBundle object which will be used by tag body.

The <fmt:message> tag is used for displaying an internationalized message. It maps the key of localized message to return the value using a resource bundle specified in the bundle attribute.

Let us define the default resource bundle Main.java as follows:

```
package com.javatpoint;
import java.util.ListResourceBundle;
public class Main extends ListResourceBundle {
public Object[][] getContents() {
 return contents;
 }
static final Object[][] contents = { { "vegetable.Potato", "Potato" },
 { "vegetable.Tomato", "Tomato" }, { "vegetable.Carrot", "Carrot" }, };
}
```

Now, compile the above class as Main.class and make it available in CLASSPATH of your Web application folder. Now you can use the below JSTL tags to display the three vegetables as follows:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<html>
<head>
<title>fmt:setBundle Tag</title>
</head>
<body>
<fmt:setBundle basename="com.javatpoint.Main" var="lang"/>
<fmt:message key="vegetable.Potato" bundle="${lang}"/><br/>
  <fmt:message key="vegetable.Tomato" bundle="${lang}"/><br/>
 <fmt:message key="vegetable.Carrot" bundle="${lang}"/><br/>
</body>
</html>
```

15.6.5 XML TAG LIBRARY:

The JSTL XML tags are used for providing a JSP-centric way of manipulating and creating XML documents. The xml tags provide flow control, transformation etc.

The url for the xml tags is http://java.sun.com/jsp/jstl/xml and prefix is x.

The JSTL XML tag library has custom tags used for interacting with XML data. The syntax used for including JSTL XML tags library in your JSP is:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/xml" prefix="x" %>
```

1. <x:out> and <x:parse> tag

The <x:out> tag is used for displaying the result of an xml Path expression and writes the result to JSP writer object.

The <x:parse> tag is used for parse the XML data specified either in the tag body or an attribute. It is used for parse the xml content and the result will stored inside specified variable.

Let's see the simple example to understand the xml <x:out> and <x:parse> tag:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
 <%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
  <html>
<head>
```

```
 <title>XML Tags</title>
</head>
<body>
<h2>Vegetable Information:</h2>
<c:set var="vegetable">
<vegetables>
   <vegetable>
     <name>onion</name>
     <price>40/kg</price>
   </vegetable>
 <vegetable>
     <name>Potato</name>
     <price>30/kg</price>
   </vegetable>
 <vegetable>
<name>Tomato</name>
<price>90/kg</price>
</vegetable>
</vegetables>
</c:set>
<x:parse xml="${vegetable}" var="output"/>
<b>Name of the vegetable is</b>:
<x:out select="$output/vegetables/vegetable[1]/name" /><br>
<b>Price of the Potato is</b>:
<x:out select="$output/vegetables/vegetable[2]/price" />
</body>
</html>
```

2. <x:set>

The <x:set> tag is used to set a variable with the value of an XPath expression. It is used to store the result of xml path expression in a scoped variable.

Example

```
<x:set var="fragment" select="$output/vegetables/vegetable[1]/name "/>
```

3. <x:choose>, <x:when>, <x:otherwise>

The <x:choose> tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java switch statement in which we choose between a numbers of alternatives.

The <x:when> is subtag of <x:choose> that will include its body if the condition evaluated be 'true'.

The <x:otherwise> is also subtag of <x:choose> it follows <x:when> tags and runs only if all the prior condition evaluated is 'false'.

The <x:when> and <x:otherwise> works like if-else statement. But it must be placed inside <x:choose> tag.

Example:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
 <html>
<head>
 <title>x:choose Tag</title>
</head>
<body>
<h3>Books Information:</h3>
 <c:set var="xmltext">
<books>
<book>
 <name>Three mistakes of my life</name>
 <author>Chetan Bhagat</author>
 <price>200</price>
</book>
<book>
 <name>Tomorrow land</name>
 <author>Brad Bird</author>
 <price>2000</price>
</book>
</books>
</c:set>
 <x:parse xml="${xmltext}" var="output"/>
<x:choose>
```

```
<x:when select="$output//book/author = 'Chetan bhagat'">
  Book is written by Chetan bhagat
</x:when>
<x:when select="$output//book/author = 'Brad Bird'">
  Book is written by Brad Bird
</x:when>
<x:otherwise>
  The author is unknown...
</x:otherwise>
</x:choose>
</body>
</html>
```

Output:

Books Information:

Book is written by Brad Bird

4. <x:if>

The <x:if> tag is used for evaluating the test XPath expression. It is a simple conditional tag which is used for evaluating its body if the supplied condition is true.

Example:

```
<x:if select="$output/vegetables/vegetable/price < 100">
Vegetables prices are very low.
</x:if>
```

15.7 SUMMARY

JSTL allows JSP programming developers to create web applications using tags rather than scriptlets (Java code).

1. JSTL tags are XML based tags, they cleanly and consistently blend into a page's HTML markup tags.
2. JSTL makes use of UEL which makes it easier to call the getter and setter methods on Java objects.
3. JSTL increases the processing burden on the server.
4. JSTL is composed of five tag libraries:
   a. Core Tag Library

b. Functions Tag Library

c. Database/SQL Tag Library

d. Formatting Tag Library

e. XML Tag Library

15.8 LIST OF REFERENCES

1. Java EE 7 for Beginners, Sharanam Shah, Vaishali Shah, First Edition, SPD

Web References:

1. https://www.tutorialspoint.com

2. https://www.javatpoint.com

15.9 QUESTIONS

Q1. What are the various advantages of using JSTL over scriptlets?

Q2. What are the various disadvantages of using JSTL?

Q3. Explain <xml:parse>,<xml:set> and <x:out> tags with example.

Q4. Explain how database connection can be established and queries can be executed using Database Tag Library.

Q5. Explain any 5 tags of Formatting Tag Library.

Q6. What are the various conditional and iteration tags in Core Tag Library?

*****

UNIT IV

16

INTRODUCTION TO ENTERPRISE JAVABEANS

Unit Structure

16.0 OBJECTIVES

After going through this chapter, you will be able to:

- Understand use, advantage, disadvantage of EJB
- Analysis architecture of EJB.
- Type of Java Beans
- How to access and use beans in clients

16.1 INTRODUCTION

What is EJB?:

EJB stands for Enterprise Java Bean.

An Enterprise Java Bean is in its basic form any POJO (Plain Old Java Object) that is registered with the container in which it is deployed. Enterprise Java Beans are deployed into an EJB container. The EJB container is governed by the EJB specification.

EJB (Enterprise Java Bean) is used to develop scalable, robust and secured enterprise applications in java.

EJB is a server-side software element that summarizes business logic of an application.

Enterprise Java Beans (EJB) is a development architecture for building highly scalable and robust enterprise level applications to be deployed on J2EE compliant Application Server such as JBOSS, Web Logic etc.

EJB 3.0 is being a great shift from EJB 2.0 and makes development of EJB based applications quite easy.

EJB stands for Enterprise Java Beans. EJB is an essential part of a J2EE platform. J2EE platform has component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.

EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability, and high performance. An EJB application can be deployed on any of the application server compliant with the J2EE 1.3 standard specification.

16.1.1 When to use Enterprise Java Beans?:

- **Application needs Remote Access:** In other words, it is distributed.
- **Application needs to be scalable:** EJB applications supports load balancing, clustering and fail-over.
- **Application needs encapsulated business logic:** EJB application is differentiated from demonstration and persistent layer.

16.1.2 Advantages of Enterprise Java Beans:

- **Interoperability:**

EJB architecture is mapped to standard CORBA.EJB make it work with components developed in different language like VC++ and CORBA.

The EJB client view interface serves as well-defined integration point between components built using different programming languages.

- **One business logic having many presentation logic:**

EJB performs a separation between business logic and presentation logic.

This separation makes it possible to develop multiple presentation logic for the same business process.

- **Complete Focus only on Business Logic:**

This allows the server vendor to concentrate on system level functionalities, while the developer can concentrate more on only the business logic for the domain specific applications.

Developer need not code for these hardcore services. The results of application get more quickly.

- **Server-Side Write Once, Run Anywhere:**

EJB uses java language which is portable across multiple platforms. They can be developed once and then deployed multiple platforms without recompilation or source code modification.

- **EJB provides Distributed Transaction support:**

EJB provides transparency for distributed transactions. This means that a client can begin a transaction and then invoke methods on Beans present within two different servers, running on different machines, platforms or JVM.

- **It provides of vendor specific enhancements:**

Since the EJB specification provides a lot of flexibility for the vendors to create their own enhancements, the EJB environment may end being feature rich.

16.1.3 Disadvantages of Enterprise Java Beans:

- The EJB specification is an inconvenient tool because of its vast documentation and complex nature. A good developer must take the time to read and study the EJB specification - even if some information is irrelevant to EJB code writing and deployment.

- EJB requires more development and debugging resources than basic Java coding, as it is difficult to determine whether a bug is inside the code or EJB container.

- EJB implementation is complex. For example, a developer may write 10 or more files (versus one) for a simple application, such as printing simple text like "hello world."

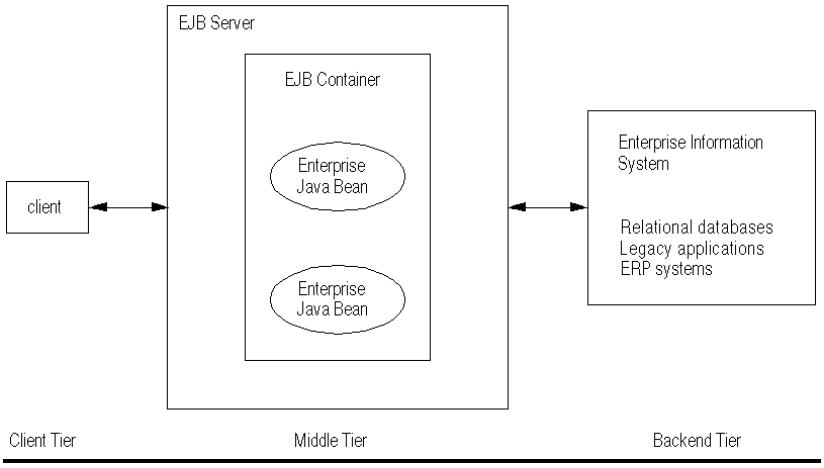
- EJB specification changes result in obsolete code. Thus, making code compatible with a new EJB container requires extra effort and higher costs.

16.2 ARCHITECTURE OF EJB

EJB Architecture:

The Enterprise JavaBeans (EJB) component architecture is designed to enable enterprises to build scalable, secure, multiplatform, business-critical applications as reusable, server-side components.

EJB architecture is at the heart of the Java 2 platform, Enterprise Edition (J2EE). With the growth of the Web and the Internet, more and more enterprise applications are now Web based, including both intranet and extranet applications.



Together, the J2EE and EJB architectures provide superior support for Web-based enterprise applications.

EJB architecture is composed of:

17 Enterprise bean server

18 Enterprise bean container

19 Enterprise bean

20 Enterprise bean clients

16.2.1 Enterprise Bean Server:

An EJB server is a component transaction server. It supports the EJB server side component model for developing and deploying distributed enterprise level applications in multi-tiered environment.

The key responsibilities of an Application Server are:

- Management API
- Process and thread management
- Database connection pooling and caching
- System Resources management

16.2.2 Enterprise Bean Container:

The EJB container is one of the logical constructs which makes up the Full Java EE profile. An EJB container manages the enterprise beans contained within it. EJB server provides one or more containers. From our architecture diagram, we saw that the EJB container construct is the second outmost construct of the architecture. Furthermore, its key responsibilities are:

- It provides a runtime environment for Enterprise Java Beans
- It provides persistence management
- It is responsible for the Lifecycle management of EJBs
- It is in charge of ensuring that all EJBs are secured

16.2.3 Enterprise Bean:

Enterprise Beans are reusable modules code that combine related tasks into well-defined interface.

These enterprise bean EJB components contain the methods that execute business login and access data sources. Business component developed using EJB architecture are called as Enterprise Beans.

EJBS are server-side components for encapsulating application's business logic. An EJB can offer specific enterprise service either alone or in conjunction with other EJBs.

16.2.4 Enterprise Bean Clients:

There are two types of client view:

- Remote Client View
- Local Client View

1. Remote Client View:

The remote client view specification became available beginning with EJB 1.1. The remote client view of an enterprise bean is location independent. A client running in the same JVM as a bean instance uses the same API to access the bean as a client running in a different JVM on the same or different machine.

Remote interface: The remote interface specifies the remote business methods that a client can call on an enterprise bean.

Remote home interface: The remote home interface specifies the methods used by remote clients for locating, creating, and removing instances of enterprise bean classes.

2. Local Client View:

Unlike the remote client view, the local client view of a bean is location dependent. Local client view access to an enterprise bean requires both the local client and the enterprise bean that provides the local client view to be in the same JVM.

The local client view therefore does not provide the location transparency provided by the remote client view.

Local interfaces and local home interfaces provide support for lightweight access from enterprise bean that are local clients.

Session and entity beans can be tightly couple with their clients, allowing access without the overhead typically associated with remote method calls.

The local client view specification is available in EJB 2.0 or later.

Local interface:

The local interface is a lightweight version of the remote interface, but for local clients. It includes business logic methods that can be called by a local client.

Local home interface:

The local home interface specifies the methods used by local clients for locating, creating, and removing instances of enterprise bean classes.

16.3 CONTAINER AND ITS TYPES

EJB Containers:

Enterprise beans (EJB components) are Java programming language server components that contain business logic. The EJB container provides local and remote access to enterprise beans.

The container is responsible for creating the enterprise bean, binding the enterprise bean to the naming service so other application components can access the enterprise bean, ensuring only authorized clients have access to the enterprise bean's methods, saving the bean's state to persistent storage, caching the state of the bean, and activating or passivating the bean when necessary.

It is responsible for all the operations of the EJB applications.

The container acts as an intermediary action between the business logic of the bean and the rest of the world of the enterprise application.

- One or more EJB modules can be installed within a single EJB container.
- The role of EJB container is to perform transactional actions such as,

1. Starting a transaction.
2. Rollback a transaction or commit a transaction.
3. Managing various connection pools for the database resources.
4. Bean's instance variables with corresponding data items which are stored in a database will be synchronized.

The four types of container that J2EE supports:

1. EJB container
2. Web Container
3. Application client container
4. Applet client container

1) EJB Container:

- An EJB container will provide the runtime environment for EJB applications within the application server.
- It is responsible for all the operations of the EJB applications.
- The container acts as an intermediary action between the business logic of the bean and the rest of the world of the enterprise application.
- One or more EJB modules can be installed within a single EJB container. The role of EJB container is to perform transactional actions such as,

1. Starting a transaction.
2. Rollback a transaction or commit a transaction.
3. Managing various connection pools for the database resources.
4. Bean's instance variables with corresponding data items which are stored in a database will be synchronized.

2) Web Container:

- A web container implements a web component such as servlet container.
- A servlet container supports the operations of a servlet.
- It supports the web server operations and the client java operations such as JRE,maps the URL specific requests into servlet requests.
- The servlet cotainers have the ability to dynamically add or remove servlets from the system.

- Individual servlets will get registered by the servlet container.
- The servlet API is provided by different vendors for a specific servlet standard.

3) Application Client Container:

- An application client container includes set of java classes, libraries and the set of files that are needed and distributed among various java client applications which executes on their own JVMs.
- The ACC is responsible for managing the applications execution by providing all the system services that are needed for the execution of java client programs.
- It is light-weighted and communicates with different application servers.

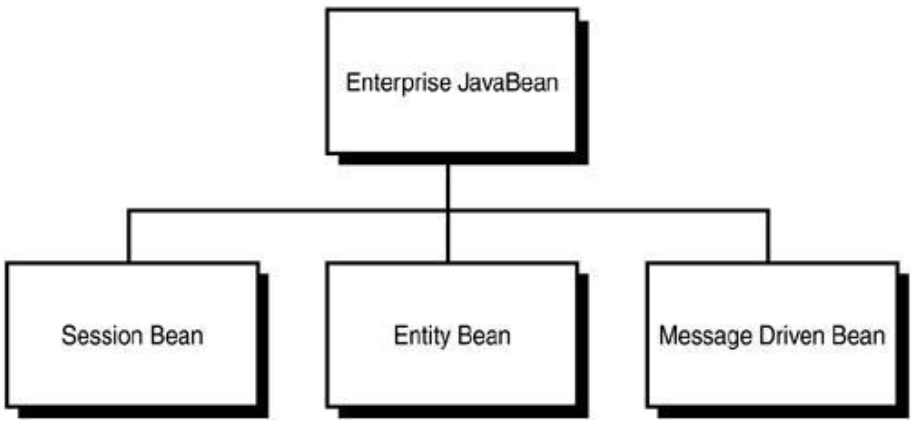
4) Applet client container:

- Like application client container, applet container executes the client applications.
- The difference is an applet executes the application in a separate browser.
- They execute on their own JVMs.
- It supports the applet programming model.
- JEE client may use java plug-in to provide the required environment that executes the applet.

Developer classes vs. container classes:

- The developer classes are the classes that are authored by the developers
- The container classes are the classes which supports the container to manage the container specific functionality.

16.4 TYPES OF ENTERPRISE JAVA BEANS



The EJB 2.0 specification defines three types of Enterprise JavaBeans: the session bean, the entity bean, and the message-driven bean.

Session beans contain business-processing logic. Entity beans contain data-processing logic. Message-driven beans allow clients to asynchronously invoke business logic.

16.4.1 Session Bean:

- Session beans are Java beans which encapsulate the business logic in a centralized and reusable manner such that it can be used by a number of clients.
- A session bean objects are short-lived.
- Are not persistent in a database.
- They can be stateful or stateless.
- Execute for a single client.
- Can be transaction aware.

As its name suggests, session beans implement a conversation between a client and the server side. Session beans execute a particular business task on behalf of a single client during a single session. They implement business logic such as workflow, algorithms, and business rules.

Session beans are analogous to interactive sessions. Just as an interactive session isn't shared among users, a session bean is not shared among clients. Like an interactive session, a session bean isn't persistent (that is, its data isn't saved to a database). Session beans are removed when the EJB container is shut down or crashes.

You can think of a session bean object as an extension of the client on the server side. It works for its client, sparing the client from complexity by executing business tasks inside the server.

Session beans typically contain business process logic and workflow, such as sending an email, looking up a stock price from a database, and implementing compression and encryption algorithms.

There are 3 types of Session beans:

- Stateless
- Stateful
- Singleton

I. Stateless Session Beans:

A stateless session bean, by comparison, does not maintain any conversational state. Stateless session beans are pooled by their container to handle multiple requests from multiple clients.

II. Stateful Session Beans:

A stateful session bean acts on behalf of a single client and maintains client-specific session information(called conversational state) across multiple method calls and transactions. It exists for the duration of a single client/server session.

III. Singleton Session Beans:

Provide shared data to client and components within an access application and are instantiated only once per application.

16.4.2 Entity Beans:

If you've worked with databases, you're familiar with persistent data. The data in a database is persistent; that is, it exists even after the database server is shut down.

Entity beans are persistent objects. They typically represent business entities, such as customers, products, accounts, and orders. Typically, each entity bean has an underlying table in a relational database, and each instance of the bean corresponds to a row in that table.

The state of an entity bean is persistent, transactional, and shared among different clients. It hides complexity behind the bean and container common services. Because the clients might want to change the same data, it's important that entity beans work within transactions. Entity beans typically contain data-related logic, such as inserting, updating, and removing a customer record in the database.

Two types of entity beans are relevant to persistence:

- container-managed persistence (CMP)
- bean-managed persistence (BMP).

In a CMP entity bean, the EJB container manages the bean's persistence according to the data-object mapping in the deployment descriptor. Any change in the entity bean's state will be automatically saved to the database by the container. No code is required in the bean to reflect these changes or to manage the database connection. On the other hand, a BMP entity bean has to manage both the database connections and all the changes to the bean's state.(Entity bean that manage their own persistence are called BMP entity bean.)

16.4.3 Message Driven Beans:

Message-driven beans are enterprise beans that receive and process JMS messages. Unlike session or entity heans, message-driven be ans have no interfaces. They can be accessed only through messaging and they do not maintain any conversational state.

Message-driven beans allow asynchronous communication between the queue and the listener, and provide separation between message processing and business logic.

Message driven beans are:

- Do not have home and component interface.
- Do not have business methods but define message listener method which the EJB container invokes to deliver messages.
- Do not hold any state between calls of the message listener method.
- Are relatively short-lived.
- Can be Transaction aware.
- Do not represent directly shared data in the database, but they can access and update this data.

In synchronous communication, the client blocks until the server-side object completes processing. In asynchronous communication, the client sends its message and does not need to wait for the receiver to receive or process the message. Session and entity beans process messages synchronously.

Message-driven beans, on the other hand, are **stateless components** that are asynchronously invoked by the container as a result of the arrival of a Java Message Service (JMS) message. A message-driven bean receives a message from a JMS destination, such as a queue or topic, and performs business logic based on the message contents, such as logic to receive and process a client notification.

An example of a message-driven bean is when a shopper makes an online purchase order; an order bean could notify a credit verification bean. A credit verification bean could check the shopper's credit card in the background and send a notification message for approval. Because this notification is asynchronous, the shopper doesn't have to wait for the background processing to complete.

16.5 ACCESSING ENTERPRISE BEANS

Enterprise beans are accessed by the client in two ways either through a no-interface view or through a business interface.

- **No-interface view:**

A no-interface view of an enterprise bean exposes the public methods of the enterprise bean implementation class to clients.

Clients using the no-interface view of an enterprise bean may invoke any public methods in the enterprise bean implementation class or any superclasses of the implementation class.

- **Business interface:**

A business interface is a standard Java programming language interface that contains the business methods of the enterprise bean.

To use a session bean client will have to required either bean's business's interface methods or enterprise bean's public methods which has a no-interface view.

Session beans can have more than one business interface. Session beans should, but are not required to, implement their business interface or interfaces.

16.6 HOW TO USE ENTERPRISE BEANS IN CLIENTS

The client of an enterprise bean obtains a reference to an instance of an enterprise bean through either dependency injection, using Java programming language annotations, or JNDI lookup, using the Java Naming and Directory Interface syntax to find the enterprise bean instance.

Dependency injection is the simplest way of obtaining an enterprise bean reference. Clients that run within a Java EE server-managed environment, Java Server Faces web applications, JAX-RS web services, other enterprise beans, or Java EE application clients, support dependency injection using the javax.ejb.EJB annotation.

Applications that run outside a Java EE server-managed environment, such as Java SE applications, explicit lookup. JNDI supports a must perform an global syntax for identifying Java EE components to simplify this explicit lookup.

16.6.1 Remote or Local Access:

1. Local Clients:

A local client has these characteristics.

- It must run in the same application as the enterprise bean it accesses.
- It can be a web component or another enterprise bean.
- To the local client, the location of the enterprise bean it accesses is not transparent.

Accessing Local Enterprise Beans Using the No-Interface View:

Client access to an enterprise bean that exposes a local, no-interface view is accomplished through either dependency injection or JNDI lookup.

To obtain a reference to the no-interface view of an enterprise bean through dependency injection, use the **javax.ejb.EJB** annotation and specify the enterprise bean's implementation class

@EJB

ExampleBean exampleBean;

To obtain a reference to the no-interface view of an enterprise bean through JNDI lookup, use the javax.naming.InitialContext interface's lookup method:

ExampleBean exampleBean=(ExampleBean)

InitialContext.lookup("java:module/ExampleBean");

Clients do not use the new operator to obtain a new instance of an enterprise bean that uses a no-interface view.

16.6.2 Remote Clients:

A remote client of an enterprise bean has the following characteristics.

It can run on a different machine and a different JVM from the enterprise bean it accesses. (It is not required to run on a different JVM.)

It can be a web component, an application client, or another enterprise bean.

To a remote client, the location of the enterprise bean is transparent.

The enterprise bean must implement a business interface. That is, remote clients may not access an enterprise bean through a no-interface view.

Accessing Remote Enterprise Beans Using the business Interface View:

Client access to an enterprise bean that implements a remote business interface is

accomplished through either dependency injection or JNDI lookup.

To obtain a reference to the remote business interface of an enterprise bean through

dependency injection, use the javax.ejb.EJB annotation and specify the enterprise bean's remote business interface name:

@EJB

Example:

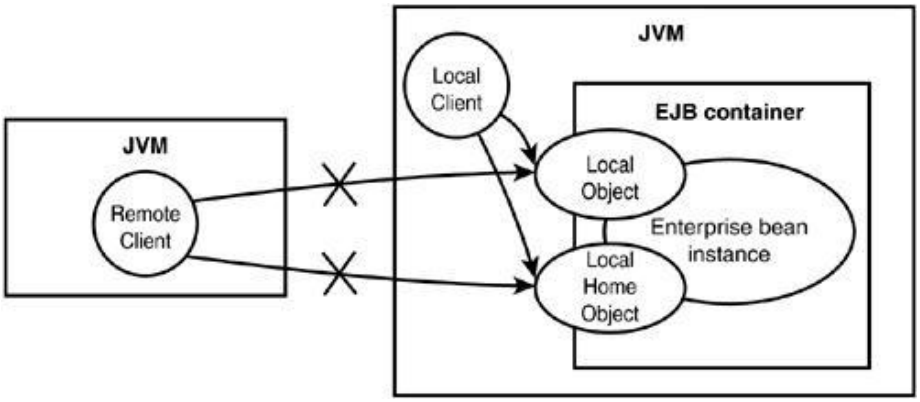
To obtain a reference to a remote business interface of an enterprise bean through JNDI lookup, use the javax.naming.InitialContext interface's lookup method:

ExampleRemote example=(ExampleRemote)

InitialContext.lookup("java:global/myApp/ExampleRemote");

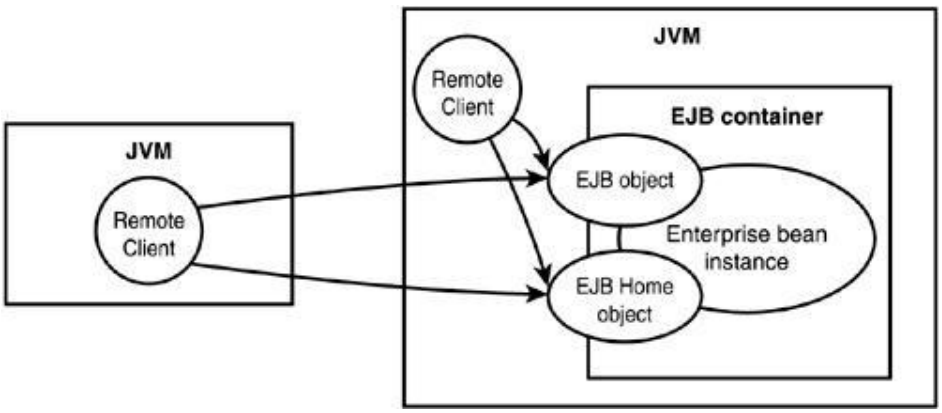
16.6.3 Characteristics of Local clients:

- They must run in the same application.
- Clients can be a web component or other enterprise bean.
- To the local client the location of the enterprise bean it accesses is not transparent.



16.6.4 Characteristics of Remote clients:

- An enterprise bean which will access the client, called remote client if it run on an other machine or JVM however, running on different JVM is not necessary.
- Clients can be a web component, an application client, or other enterprise bean.
- To the Remote client the location of the enterprise bean it accesses is transparent
- Remote clients which will use an enterprise bean that, enterprise bean will must have to implement the business interface.



EnterpriseBeanApplication/PackagingEnterpriseBeans

16.7 SUMMARY

EJB stands for **Enterprise Java Beans**. EJB is an essential part of a J2EE platform. J2EE platform has component based architecture to provide multi-tiered, distributed and highly transactional features to enterprise level applications.

EJB provides an architecture to develop and deploy component based enterprise applications considering robustness, high scalability, and high performance. An EJB application can be deployed on any of the application server compliant with the J2EE 1.3 standard specification

16.8 REFERENCES

1. Java EE for beginners by-Sharanam Shah
2. Advanced Java Programming by-Uttan Kumar Roy
3. Java EE 8 by-Elder Moreas
4. www.javatutorial.com

16.9 UNIT END QUESTIONS

1. Explain EJB with advantages and disadvantages?
2. Explain different types of EJB?
3. How to use Enterprise Beans in clients?
4. Explain characteristics of local clients?

*****

17

WORKING WITH SESSION BEANS AND MESSAGE DRIVEN BEAN

Unit Structure

17.0 OBJECTIVES

After going through this chapter, you will be able to:

- Understand use, advantage, disadvantage of Session Beans
- When to use session beans.
- Type of Session Beans
- How to access and beans in different interfaces
- Understand what is message Driven Beans
- Life Cycle of Message Driven Beans
- Use of Message Driven Beans

17.1 INTRODUCTION

17.1.1 What is Session Bean?:

A session bean is an EJB 3.0 or EJB 2.1 enterprise bean component created by a client for the duration of a single client/server session. The session bean performs work for its client, shielding it from complexity by executing business tasks inside the server.

A session bean is not persistent. (That is, its data is not saved to a database.)

A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views.

17.1.2 When to Use Session Beans:

Stateful session beans are appropriate if any of the following conditions are true.

- The bean's state represents the interaction between the bean and a specific client.
- The bean needs to hold information about the client across method invocations.
- The bean mediates between the client and the other components of the application, presenting a simplified view to the client.
- Behind the scenes, the bean manages the work flow of several enterprise beans.

To improve performance, you might choose a stateless session bean if it has any of these traits.

- The bean's state has no data for a specific client.
- In a single method invocation, the bean performs a generic task for all clients. For example, you might use a stateless session bean to send an email that confirms an online order.
- The bean implements a web service.

Singleton session beans are appropriate in the following circumstances.

- State needs to be shared across the application.
- A single enterprise bean needs to be accessed by multiple threads concurrently.
- The application needs an enterprise bean to perform tasks upon application startup and shutdown.
- The bean implements a web service.

17.2 TYPES OF SESSION BEANS

Session beans are of three types: stateful, stateless, and singleton.

Stateful Session Beans:

The state of an object consists of the values of its instance variables. In a stateful session bean, the instance variables represent the state of a unique

client/bean session. Because the client interacts ("talks") with its bean, this state is often called the **conversational state**.

As its name suggests, a session bean is similar to an interactive session. A session bean is not shared; it can have only one client, in the same way that an interactive session can have only one user. When the client terminates, its session bean appears to terminate and is no longer associated with the client.

The state is retained for the duration of the client/bean session. If the client removes the bean, the session ends and the state disappears. This transient nature of the state is not a problem, however, because when the conversation between the client and the bean ends, there is no need to retain the state.

Stateless Session Beans:

A **stateless session bean** does not maintain a conversational state with the client. When a client invokes the methods of a stateless bean, the bean's instance variables may contain a state specific to that client but only for the duration of the invocation. When the method is finished, the client-specific state should not be retained. Clients may, however, change the state of instance variables in pooled stateless beans, and this state is held over to the next invocation of the pooled stateless bean. Except during method invocation, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client. That is, the state of a stateless session bean should apply across all clients.

Because they can support multiple clients, stateless session beans can offer better scalability for applications that require large numbers of clients. Typically, an application requires fewer stateless session beans than stateful session beans to support the same number of clients.

A stateless session bean can implement a web service, but a stateful session bean cannot.

Singleton Session Beans:

A **singleton session bean** is instantiated once per application and exists for the lifecycle of the application. Singleton session beans are designed for circumstances in which a single enterprise bean instance is shared across and concurrently accessed by clients.

Singleton session beans offer similar functionality to stateless session beans but differ from them in that there is only one singleton session bean per application, as opposed to a pool of stateless session beans, any of which may respond to a client request. Like stateless session beans, singleton session beans can implement web service endpoints.

Singleton session beans maintain their state between client invocations but are not required to maintain their state across server crashes or shutdowns.

Applications that use a singleton session bean may specify that the singleton should be instantiated upon application startup, which allows the singleton to perform initialization tasks for the application. The singleton may perform cleanup tasks on application shutdown as well, because the singleton will operate throughout the lifecycle of the application.

17.3 REMOTE AND LOCAL INTERFACES

1. Local interface:

The local interface is used for Local client. Local interface are the type of interface that are used for making local connections to EJB. **@Local** annotation is used for declaring interface as Local. **Javax.ejb.Local** package is used for creating Local interface.

Syntax:

```
@Local
public interface InterfaceName { ... }
```

Example:

```
Package ejb;
import javax.ejb.Local;
@Local
Public interface SessionLocal
{
}
```

2. Remote interface:

The Remote interface is used for Remote client. Remote interface are the interface that has the methods that relate to a particular bean instance.

@Remote annotation is used for declaring interface as Remote. **Javax.ejb.Remote** package is used for creating Remote interface.

Syntax:

```
@Remote
public interface InterfaceName { ... }
```

Example:

```
Package ejb;
import javax.ejb.Remote;
@Remote
Public interface SessionRemote
```

```
{
    String getMessage();
    String getAddress();
}
```

17.4 ACCESSING INTERFACES

Accessing Local Enterprise Beans Using the No-Interface View:

Client access to an enterprise bean that exposes a local, no-interface view is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the no-interface view of an enterprise bean through dependency injection, use the javax.ejb.EJB annotation and specify the enterprise bean's implementation class:
- @EJB

  ExampleBean exampleBean;
- To obtain a reference to the no-interface view of an enterprise bean through JNDI lookup, use the javax.naming.InitialContext interface's lookup method:
- ExampleBean exampleBean = (ExampleBean)

  InitialContext.lookup("java:module/ExampleBean");

Clients do not use the new operator to obtain a new instance of an enterprise bean that uses a no-interface view.

17.5 ACCESSING LOCAL ENTERPRISE BEANS THAT IMPLEMENT BUSINESS INTERFACES

Client access to enterprise beans that implement local business interfaces is accomplished through either dependency injection or JNDI lookup.

- To obtain a reference to the local business interface of an enterprise bean through dependency injection, use the javax.ejb.EJB annotation and specify the enterprise bean's local business interface name:
- @EJB

  Example example;
- To obtain a reference to a local business interface of an enterprise bean through JNDI lookup, use the javax.naming.InitialContext interface's lookup method:
- ExampleLocal example = (ExampleLocal)

  InitialContext.lookup("java:module/ExampleLocal");

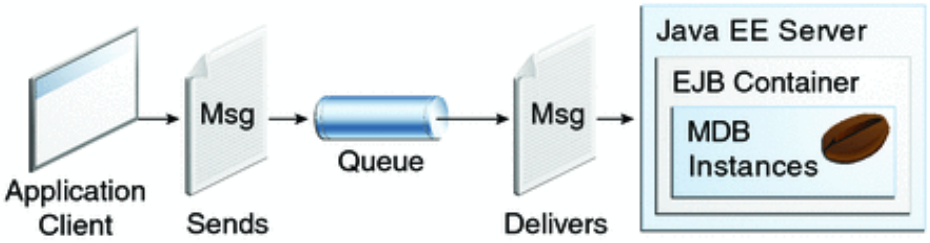
17.6 WHEN TO USE MESSAGE-DRIVEN BEANS

A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. So, it is like JMS Receiver.

MDB asynchronously receives the message and processes it.

A message driven bean receives message from queue or topic, so you must have the knowledge of JMS API.

A **message-driven bean** is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

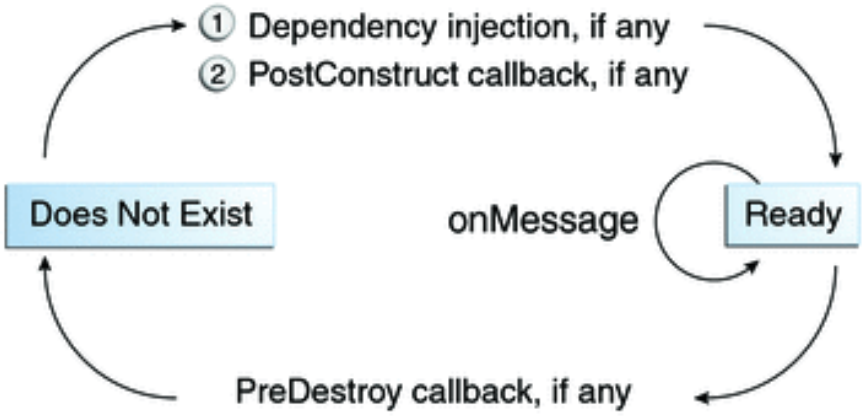


Session beans allow you to send JMS messages and to receive them synchronously but not asynchronously. To avoid tying up server resources, do not to use blocking synchronous receives in a server-side component; in general, JMS messages should not be sent or received synchronously. To receive messages asynchronously, use a message-driven bean.

17.7 THE LIFECYCLE OF A MESSAGE-DRIVEN BEAN

Figure illustrates the stages in the lifecycle of a message-driven bean.

Figure illustrates Lifecycle of a Message-Driven Bean:



The EJB container usually creates a pool of message-driven bean instances. For each instance, the EJB container performs these tasks.

1. If the message-driven bean uses dependency injection, the container injects these references before instantiating the instance.
2. The container calls the method annotated @PostConstruct, if any.

Like a stateless session bean, a message-driven bean is never passivated and has only two states: nonexistent and ready to receive messages.

At the end of the lifecycle, the container calls the method annotated @PreDestroy, if any. The bean's instance is then ready for garbage collection

17.8 MESSAGE DRIVEN BEAN EXAMPLE

To create the message driven bean, you need to declare @MessageDriven annotation and implement MessageListener interface.

In eclipse ide, create **EJB Project** then create a class as given below:

File: MyListener.java

```
1.  package com.javatpoint;
2.  import javax.ejb.MessageDriven;
3.  import javax.jms.*;
4.
5.  @MessageDriven(mappedName="myTopic")
6.  public class MyListener implements MessageListener{
7.      @Override
8.      public void onMessage(Message msg) {
9.          TextMessage m=(TextMessage)msg;
10.         try{
11.         System.out.println("message received: "+m.getText());
12.         }catch(Exception e){System.out.println(e);}
13.     }
14. }
```

17.9 SUMMARY

A **message-driven bean** is an enterprise bean that allows Java EE applications to process messages asynchronously. This type of bean normally acts as a JMS message listener, which is similar to an event listener but receives JMS messages instead of events. The messages can be sent by any Java EE component (an application client, another enterprise

bean, or a web component) or by a JMS application or system that does not use Java EE technology. Message-driven beans can process JMS messages or other kinds of messages.

The most visible difference between message-driven beans and session beans is that clients do not access message-driven beans through interfaces. Unlike a session bean, a message-driven bean has only a bean class. A **session bean** encapsulates business logic that can be invoked programmatically by a client over local, remote, or web service client views. To access an application that is deployed on the server, the client invokes the session bean's methods.

17.10 REFERENCES

1. Java EE for beginners by-Sharanam Shah
2. Advanced Java Programming by-Uttan Kumar Roy
3. Java EE 8 by-Elder Moreas
4. www.javatutorial.com

17.11 UNIT END QUESTIONS

1. What is session beans? Explain use of session beans.
2. What are the types of session beans?
3. Explain local and remote interfaces
4. What is Message Driven Beans with its uses?
5. Explain Lifecycle of Message Driven Beans?
6. Write a program to show implementation of Message Driven Beans?

*****

18

INTERCEPTORS

Unit Structure

18.0 OBJECTIVES

After going through this chapter, you will be able to:

- Understand what is interceptors
- LifeCycle of interceptors
- Different annotations of interceptors
- Understand what is naming directory
- Different packages of directory
- Its Architecture

18.1 INTRODUCTION

Interceptors are used in conjunction with Java EE managed classes to allow developers to invoke interceptor methods on an associated **target class**, in conjunction with method invocations or lifecycle events. Common uses of interceptors are logging, auditing, and profiling. An interceptor can be defined within a target class as an **interceptor method**, or in an associated class called an **interceptor class**. Interceptor classes

contain methods that are invoked in conjunction with the methods or lifecycle events of the target class.

Interceptor classes and methods are defined using metadata annotations, or in the deployment descriptor of the application containing the interceptors and target classes.

18.2 INTERCEPTOR METADATA ANNOTATIONS

Interceptor Metadata Annotation	Description
javax.interceptor.AroundInvoke	Designates the method as an interceptor method.
javax.interceptor.AroundTimeout	Designates the method as a timeout interceptor, for interposing on timeout methods for enterprise bean timers.
javax.annotation.PostConstruct	Designates the method as an interceptor method for post-construct lifecycle events.
javax.annotation.PreDestroy	Designates the method as an interceptor method for pre-destroy lifecycle events.

18.3 INTERCEPTOR CLASSES

Interceptor classes may be designated with the optional javax.interceptor.Interceptor annotation, but interceptor classes aren't required to be so annotated. An interceptor class must have a public, no-argument constructor.

The target class can have any number of interceptor classes associated with it. The order in which the interceptor classes are invoked is determined by the order in which the interceptor classes are defined in the javax.interceptor.Interceptors annotation. However, this order can be overridden in the deployment descriptor.

Interceptor classes may be targets of dependency injection. Dependency injection occurs when the interceptor class instance is created, using the naming context of the associated target class, and before any @PostConstruct callbacks are invoked.

18.4 INTERCEPTOR LIFECYCLE

Interceptor classes have the same lifecycle as their associated target class. When a target class instance is created, an interceptor class instance is also created for each declared interceptor class in the target class. That is, if the target class declares multiple interceptor classes, an instance of each class is created when the target class instance is created. The target class instance and all interceptor class instances are fully instantiated before any @PostConstruct callbacks are invoked, and any @PreDestroy callbacks

are invoked before the target class and interceptor class instances are destroyed.

18.5 THE INTERCEPTOR APPLICATION

The interceptor example demonstrates how to use an interceptor class, containing an @AroundInvoke interceptor method, with a stateless session bean.

The HelloBean stateless session bean is a simple enterprise bean with two business methods, getName and setName, to retrieve and modify a string. The setName business method has an @Interceptors annotation that specifies an interceptor class, HelloInterceptor, for that method.

```
@Interceptors(HelloInterceptor.class)
public void setName(String name) {
   this.name = name;
}
```

The HelloInterceptor class defines an @AroundInvoke interceptor method, modifyGreeting, that converts the string passed to HelloBean.setName to lowercase.

```
@AroundInvoke
public Object modifyGreeting(InvocationContext ctx) throws Exception {
   Object[] parameters = ctx.getParameters();
   String param = (String) parameters[0];
   param = param.toLowerCase();
   parameters[0] = param;
   ctx.setParameters(parameters);
   try {
      return ctx.proceed();
   } catch (Exception e) {
      logger.warning("Error calling ctx.proceed in modifyGreeting()");
      return null;
   }
}
```

The parameters to HelloBean.setName are retrieved and stored in an Object array by calling the InvocationContext.getParameters method. Because setName has only one parameter, it is the first and only element in the array. The string is set to lowercase and stored in the parameters array, then passed to InvocationContext.setParameters. To return control to the session bean, InvocationContext.proceed is called.

The user interface of interceptor is a JavaServer Faces web application that consists of two Facelets views: index.xhtml, which contains a form for entering the name, and response.xhtml, which displays the final name.

18.6 RUNNING THE INTERCEPTOR EXAMPLE

You can use either NetBeans IDE or Ant to build, package, deploy, and run the interceptor example.

To Run the interceptor Example Using NetBeans IDE:

1. From the File menu, choose Open Project.
2. In the Open Project dialog, navigate to tut-install/examples/ejb/.
3. Select the interceptor folder and click Open Project.
4. In the Projects tab, right-click the interceptor project and select Run.

   This will compile, deploy, and run the interceptor example, opening a web browser page to http://localhost:8080/interceptor/.
5. Type a name into the form and select Submit.

   The name will be converted to lowercase by the method interceptor defined in the HelloInterceptor class.

To Run the interceptor Example Using Ant:

1. **Go to the following directory:**

   tut-install/examples/ejb/interceptor/
2. **To compile the source files and package the application, use the following command:**

   ant

   This command calls the default target, which builds and packages the application into a WAR file, interceptor.war, located in the dist directory.
3. **To deploy and run the application using Ant, use the following command:**

   ant run

   This command deploys and runs the interceptor example, opening a web browser page to http://localhost:8080/interceptor/.
4. **Type a name into the form and select Submit:**

   The name will be converted to lowercase by the method interceptor defined in the Hello Interceptor class.

18.7 SERVICES

The Java Naming and Directory Interface™ (JNDI) is an application programming interface (API) that provides naming and directory functionality to applications written using the Java™ programming language. It is defined to be independent of any specific directory service implementation. Thus a variety of directories -new, emerging, and already deployed can be accessed in a common way.

18.7.1 Naming Service:

The Java Naming and Directory Interface (JNDI) is an application programming interface (API) for accessing different kinds of naming and directory services. JNDI is not specific to a particular naming or directory service, it can be used to access many different kinds of systems including file systems; distributed objects systems like CORBA, Java RMI, and EJB; and directory services like LDAP, Novell NetWare, and NIS+.

JNDI is similar to JDBC in that they are both Object-Oriented Java APIs that provide a common abstraction for accessing services from different vendors. While JDBC can be used to access a variety of relational databases, JNDI can be used to access a variety of of naming and directory services. Using one API to access many different brands of a service is possible because both JDBC and JNDI subscribe to the same architectural tenet: Define a common abstraction that most vendors can implement. The common abstraction is the API. It provides an objectified view of a service while hiding the details specific to any brand of service. The implementation is provided by the vendor, it plugs into the API and implements code specific to accessing that vendor's product.

JNDI provides two APIs and one SPI. JNDI has a naming API that allows Java applications to access naming systems like CORBA's Naming services and a directory API that extends the naming service to provide access to directory services like LDAP. JNDI also has a SPI (Service-Provider Interface) which is a programming model that vendors use to write JNDI plug-ins or implementations for their specific product. Each vendor's plug-in is called a service-provider. A service-provider implements the JNDI APIs so that a Java application can access that vendor's product. For the most part, JNDI hides the implementation details of the a service-provider so that Java developer that uses JNDI can use the same objects and method regardless of the brand of naming or directory service accessed. This is the real power behind APIs like JDBC and JNDI: They provide one programming model for accessing many different products; there is no need to learn a different programming model every time a different product is used.

18.7.2 Directory Service:

Directory services are an essential part of today’s network-centric computing infrastructure. Directory-enabled applications now power almost all the mission critical processes of an enterprise, including

resource planning, value chain management, security and firewalls, and resource provisioning. Directory services also provide the foundation for deployment of e-business and extranet applications. So what exactly is a Directory Service?

A directory service is the collection of software and processes that store information about your enterprise, subscribers, or both. An example of a directory service is the Domain Name System (DNS), which is provided by DNS servers. A DNS server stores the mappings of computer host names and other forms of domain name to IP addresses. A DNS client sends questions to a DNS server about these mappings (e.g. what is the IP address of test.example.com?). Thus, all of the computing resources (hosts) become clients of the DNS server. The mapping of host names enables users of the computing resources to locate computers on a network, using host names rather than complex numerical IP addresses.

Whereas the DNS server stores only two types of information: names and IP addresses, an LDAP directory service can store information on many other kinds of real-world and conceptual objects. Sun Java System Directory Server stores all of these types of information in a single, network-accessible repository. You may for example want to store physical device information, employee information (name, E-mail address), contract or account information (name, delivery dates, contract numbers, etc.), authentication information, manufactured production information. It is worth noting that although a directory service can be considered an extension of a

18.8 CHARACTERISTICS OF DIRECTORY SERVICES

- **Hierarchical naming model:**

A hierarchical naming model uses the concept of containment to reduce ambiguity between names and simplify administration. The name for most objects in the directory is relative to the name of some other object which conceptually contains it. For example, the name of an object representing an employee of a particular company contains the name of the object representing the company, and the name of the company might contain the name of the objects representing the country where the company operates, e.g. cn=John Smith, o=Example Corporation, c=US. Together the names of all objects in the directory service form a tree, and each Directory Server holds a branch of that tree, which in the Sun Java System Directory Server documentation is also referred to as a suffix.

- **Extended search capability:**

Directory services provide robust search capabilities, allowing searches on individual attributes of entries.

- **Distributed information model:**

A directory service enables directory data to be distributed across multiple servers within a network.

- **Shared network access:**

While databases are defined in terms of APIs, directories are defined in terms of protocols. Directory access implies network access by definition. Directories are designed specifically for shared access among applications. This is achieved through the object-oriented schema model. By contrast, most databases are designed for use only by particular applications and do not encourage data sharing.

- **Replicated data:**

Directories support replication (copies of directory data on more than one server) which make information systems more accessible and more resistant to failure.

- **Datastore optimized for reads:**

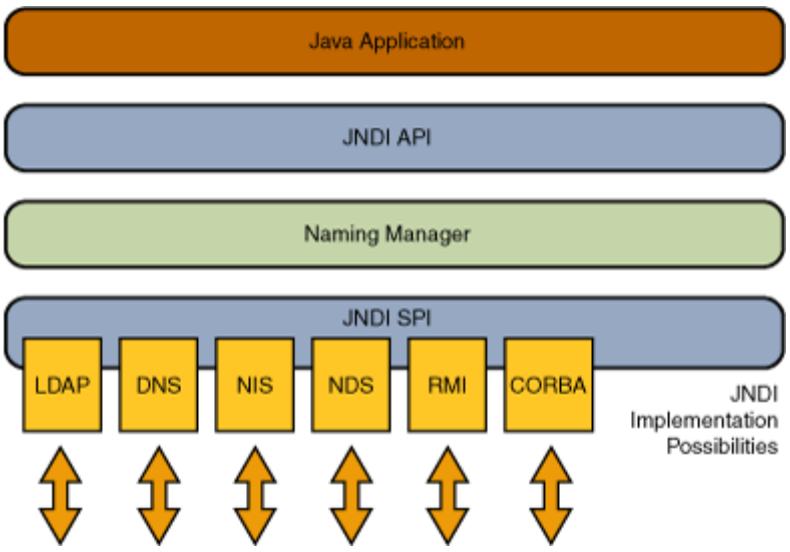
The storage mechanism in a directory service is generally designed to support a high ratio of reads to writes.

- **Extensible schema:**

The schema describes the type of data stored in the directory. Directory services generally support the extension of schema, meaning that new data types can be added to the directory.

18.9 ARCHITECTURE

The JNDI architecture consists of an API and a service provider interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services. The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services.



Packaging:

JNDI is included in the Java SE Platform. To use the JNDI, you must have the JNDI classes and one or more service providers. The JDK includes service providers for the following naming/directory services:

- Lightweight Directory Access Protocol (LDAP)
- Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
- Java Remote Method Invocation (RMI) Registry
- Domain Name Service (DNS)

Other service providers can be downloaded from the JNDI page or obtained from other vendors.

18.10 DIFFERENT PACKAGES

The JNDI is divided into five packages:

- javax.naming
- javax.naming.directory
- javax.naming.ldap
- javax.naming.event
- javax.naming.spi

Naming Package:

The javax.naming package contains classes and interfaces for accessing naming services.

Context:

The javax.naming package defines a Context interface, which is the core interface for looking up, binding/unbinding, renaming objects and creating and destroying subcontexts.

Lookup:

The most commonly used operation is lookup(). You supply lookup() the name of the object you want to look up, and it returns the object bound to that name.

Bindings:

listBindings() returns an enumeration of name-to-object bindings. A binding is a tuple containing the name of the bound object, the name of the object's class, and the object itself.

List:

list() is similar to listBindings(), except that it returns an enumeration of names containing an object's name and the name of the object's class. list() is useful for applications such as browsers that want to discover information about the objects bound within a context but that don't need all of the actual objects. Although listBindings() provides all of the same information, it is potentially a much more expensive operation.

Name:

Name is an interface that represents a generic name—an ordered sequence of zero or more components. The Naming Systems use this interface to define the names that follow its conventions as described in the Naming and Directory Concepts lesson.

References:

Objects are stored in naming and directory services in different ways. A reference might be a very compact representation of an object.

The JNDI defines the Reference class to represent reference. A reference contains information on how to construct a copy of the object. The JNDI will attempt to turn references looked up from the directory into the Java objects that they represent so that JNDI clients have the illusion that what is stored in the directory are Java objects.

18.11 RESOURCES AND JNDI NAMING

In a distributed application, components need to access other components and resources, such as databases. For example, a servlet might invoke remote methods on an enterprise bean that retrieves information from a database. In the Java EE platform, the Java Naming and Directory Interface (JNDI) naming service enables components to locate other components and resources.

A **resource** is a program object that provides connections to systems, such as database servers and messaging systems. (A Java Database Connectivity resource is sometimes referred to as a data source.) Each resource object is identified by a unique, people-friendly name, called the JNDI name. For example, the JNDI name of the JDBC resource for the Java DB database that is shipped with the GlassFish Server is jdbc/__default.

An administrator creates resources in a JNDI namespace. In the GlassFish Server, you can use either the Administration Console or the asadmin command to create resources. Applications then use annotations to inject the resources. If an application uses resource injection, the GlassFish Server invokes the JNDI API, and the application is not required to do so. However, it is also possible for an application to locate resources by making direct calls to the JNDI API.

A resource object and its JNDI name are bound together by the naming and directory service. To create a new resource, a new name/object binding is entered into the JNDI namespace. You inject resources by using the @Resource annotation in an application.

You can use a deployment descriptor to override the resource mapping that you specify in an annotation. Using a deployment descriptor allows you to change an application by repackaging it rather than by both recompiling the source files and repackaging. However, for most applications, a deployment descriptor is not necessary.

Data source resource definition in Javav EE:

DataSource resources are used to define a set of properties required to identify and access a database through the JDBC API. These properties include information such as the URL of the database server, the name of the database, and the network protocol to use to communicate with the server. DataSource objects are registered with the Java Naming and Directory Interface (JNDI) naming service so that applications can use the JNDI API to access a DataSource object to make a connection with a database.

Prior to Java EE 7, DataSource resources were created administratively as described in Configuring WebLogic JDBC Resources in Administering JDBC Data Sources for Oracle WebLogic Server. Java EE 7 provides the option to programmatically define DataSource resources for a more flexible and portable method of database connectivity.

The name element uniquely id entifies a DataSource and is registered with JNDI. The value specified in the name element begins with a namespace scope. Java EE 7 includes the following scopes:

- java:comp: Names in this namespace have per-component visibility.
- java:module: Names in this namespace are shared by all components in a module, for example, the EJB components defined in an a ejb-jar.xml file.
- java:app: Names in this namespace are shared by all components and modules in an application, for example, the application-client, web, and EJB components in an .ear file.
- java:global: Names in this namespace are shared by all the applications in the server.

18.12 SUMMARY

Interceptors are used in conjunction with Java EE managed classes to allow developers to invoke interceptor methods on an associated target class, in conjunction with method invocations or lifecycle events. Common uses of interceptors are logging, auditing, and profiling.

Clients use the naming service to locate objects by name. The Java Naming and Directory Interface (JNDI) is designed by Sun Microsystems Ltd. to simplify access to the directory infrastructure, which advanced network applications are being built on, by providing an unified set of interfaces

18.13 REFERENCES

1. Java EE for beginners by-Sharanam Shah
2. Advanced Java Programming by-Uttan Kumar Roy
3. Java EE 8 by-Elder Moreas
4. www.javatutorial.com

18.14 UNIT END QUESTIONS

1. What is interceptors with different annotations?
2. Explain interceptors lifecycle?
3. Explain the steps for running interceptors application?
4. Explain interceptors with example?
5. What is naming service and directory service ?
6. Explain Characteristics of Directory Service?
7. Explain different packages of java naming directory?

*****

UNIT V

19

PERSISTENCE, OBJECT/RELATIONAL MAPPING AND JPA

Unit Structure

19.1 OBJECTIVES

This chapter will introduce Data Persistence it's means for an application to persist and retrieve information from a non-volatile storage system. Persistence is vital to enterprise applications because of the required access to relational databases.

19.2 PERSISTENCE, OBJECT/RELATIONAL MAPPING AND JPA

Data is an important asset to any computer application. All computer applications require that a person or another computer access their data. This data is used in different ways.

Data can be:

- Read-only
- Read-write
- Read for update over multiple requests
- Modified through batch updates
- Used in bulk data retrieval

The attention span of a computer is only as long as its cord is connected to a power supply. The precious data is within the confines of electronic

memory. If the application does not preserve data when it was powered off, the application is of little or rather no practical use. Hence, it is required to make the precious data live longer than the application. This is where Persistence comes in.

19.3 WHAT IS PERSISTENCE?

Most business applications require that data must be persistent. Data can be labeled as persistent only when it manages to survive day to day problems such as system crashes and network failures. Often multiple users request for data simultaneously. Here, there is a definite possibility of data getting corrupt, if mid-request, system failure, occurs. Maintaining the persistence of data in an enterprise wide application is quite a challenging job.

In enterprise application architecture, data persistence is implemented as:

- Having data stored outside an application's active memory, known as **persistent data store**, typically, a relational database or an object database or a flat file system

- Having a **rollback system**, where, in case of system failure, the state of the data is rolled back to its last known valid data state

Persistence is one of the fundamental concepts of **application development**. It allows DATA to outlive the execution of an application that created it. It is one of the most vital pieces of an application without which all the data is simply lost. Majority of applications use persistent data. For example, GUI applications need to store user preferences across program invocations, Web applications track user movements and orders over long periods of time.

Hence, it is imperative to choose an appropriate persistence data store. Often when choosing the persistence data store the following fundamental qualifiers are considered:

1. The length of time data must be persisted

2. The volume of data

Application may consider an HTTP session when the life of a piece of data is limited to the user's session. However, persistence over several sessions or several users requires a larger data store. Large amounts of data should not be stored in an HTTP session, instead a database should be considered. The type of database that is chosen also plays an important influence on the architecture and design.

19.4 PERSISTENCE IN JAVA

Most of enterprise systems and applications save data into a relational database of some kind. This is why persistence has been a major application development concern for many decades. Persistence in Java

usually means storing data in a relational database using SQL. In Java, persistence is accomplished by storing data in a Relational Database Management System [RDBMS]. SQL is used to get data in and out of the relational database. Java Database Connectivity [JDBC]: The Java API IS used to connect to the RDBMS and fire SQL statements.

Persistence of Object-Oriented Models:

Today most of the development is carried out in an object-oriented manner using languages such as Smalltalk, C++ and Java. Object [Domain] modelling is a concept always linked with Persistence. In fact, it is often the domain model that is persisted.

Object Oriented Programming is based on OBJECTS that represent the business model [the real world]. Objects are easily traversed through relationship graphs using inheritance, associations. When thinking in terms of Java as the programming language of choice, the business logic of an application works with Objects of different class types.

However, when dealing with the data store, it's important to note that the tables of a database are not Objects, which becomes an issue. This is where the concept of Object Persistence comes in. Object Persistence deals with persistence in object-oriented programs such as Java. It means determining how objects and their relationships are persisted in a relational database.

Object persistence is about:

- Mapping object state
- Determining how an object's state [data stored in member variables of an object] is stored in database table columns
- Dealing with the fact that object state types may not align with relational database types
- Mapping object relationships
- Determining how associations between objects are stored as relational database keys or in relational database intersection tables

Why Object-Oriented Models?

- Business logic can be implemented in Java as opposed to stored procedures
- Design patterns and sophisticated object-oriented concepts such as inheritance and polymorphism can be used
- Provides code reusability and maintainability

In most of the applications, storing and retrieving information usually involves some kind of interaction with a relational database.

Why Relational Database?:

Today, computer applications that involve storing data, involve accessing a relational database. Relational databases are the persistence store for most applications. A relational database is a choice because of the following:

- It is a proven data persistence technology
- Provides flexible and robust approach to data management
- It is the De-facto standard in software development

19.5 CURRENT PERSISTENCE STANDARDS IN JAVA

The Java platform was always supported for managing persistence to relational databases. It

provides programming interfaces that in turn provide gateway to the relational databases. There already exist the following options that allow the Java developers to store and retrieve persistent data:

- Serialization
- JDBC
- EJB 2 Entity Beans
- Java Data Objects

Serialization:

Serialization allows transforming an object graph into a series of bytes, which can then be sent over the network or stored in a file. Serialization seems to be quite simple, but it has a few limitations:

1. **Unsuitable for Large Amount of Data:** It must store and retrieve the entire object graph at once. This makes it unsuitable for dealing with large amounts of data.
2. **Lacks strict data integrity:** Changes made to objects cannot be undone, if an error occurs while updating information. This makes it unsuitable for applications that require strict data integrity.
3. **No Concurrent Access to Information:** Multiple threads or programs cannot read and write the same serialized data concurrently without conflicting with each other.
4. **No Querying Capability:** It provides no query capabilities.

JDBC:

JDBC overcomes a lot of limitations that serialization has such as:

1. It can handle large amounts of data

2. Ensures data integrity
3. Supports concurrent access to information
4. Provides a sophisticated query language in SQL

Unfortunately, **JDBC does not provide ease of use.** JDBC was not designed for storing objects. JDBC allows the Java programs to fully interact with the database. However, this interaction is heavily reliant on SQL. It requires a considerable amount of code spec that deals with taking tabular row and column data and converting it back and forth into objects.

EJB 2 Entity Beans:

The Enterprise Edition of the lava platform introduced entity Enterprise JavaBeans [EJBs]

EJB 2.x entities are components that represent persistent information in a data store.

EJB 2.x entities:

- Provide an object-oriented view of persistent data
- Use a strict standard, making them portable across vendors

Unfortunately, EJB 2.x entities:

- Are slow
- Are difficult to code
- Are not serializable
- Require a one-to-one mapping to database tables
- Require expensive application servers to run, as they have to reside within a J2EE application server environment
- Require developers to determine which bean field maps to which table column
- Do not offer features such as inheritance, polymorphism and complex relations

Java Data Objects:

Java Data Objects [JDO] was initiated to provide another persistence specification effort due to the failures of the EJB persistence model.

JDO:

- Provides an object-oriented query language, which was not well accepted by the relational database users

- Supports non-relational databases. In fact, it was driven by members of the object-oriented database community and is now being adopted by object-oriented database products as the primary API

JDO never became an integrated part of the enterprise Java platform. It had many good features in it and was adopted by a small community of devoted and loyal users who stuck by it and tried to promote it. Unfortunately, the major commercial vendors did not share the same view of how a persistence framework should be implemented.

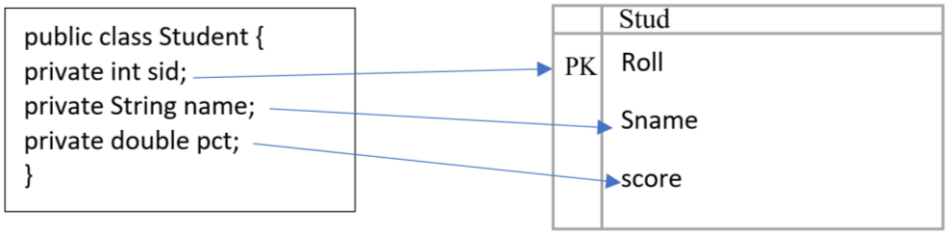
19.6 WHY ANOTHER PERSISTENCE STANDARDS?

The answer to this question is that, each of the above-mentioned persistence solutions have severe limitations.

Object/ Relational Mapping:

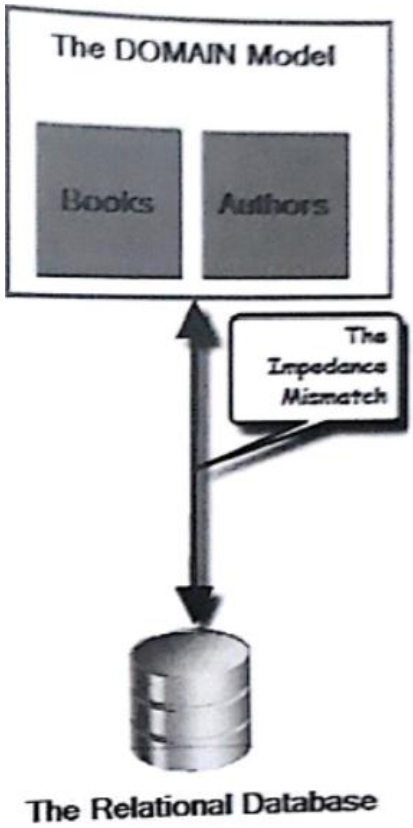
The object oriented [domain] model use classes, whereas the relational databases use tables. This creates a gap **[The Impedance Mismatch]**. Getting the data and associations from objects into relational table structure and vice versa requires a lot of tedious programming due to the difference between the two. This difference is called **The Impedance Mismatch**.

Developers need something simple to convert from one to the other automatically. Bridging the gap between the object model and the relational model is known as Object-Relational Mapping [O-R mapping or ORM].



The Impedance Mismatch:

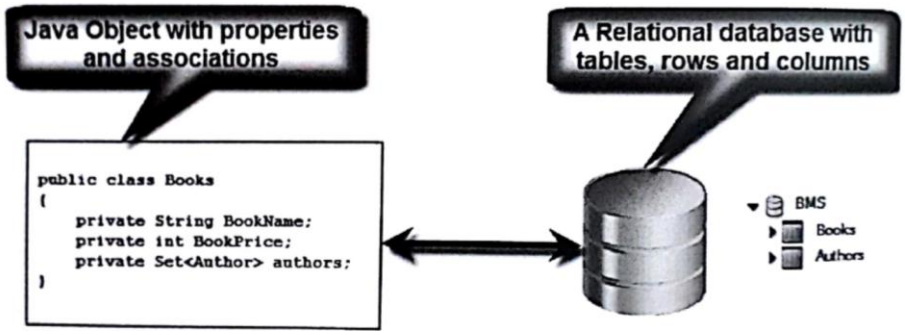
This issue arises as the design of relational data and object-oriented instances share a very different relationship structure within their respective environments. Relational databases are structured in **a tabular** manner and the object-oriented instances are structured in a **hierarchica**l manner. This means that in this object-oriented world, data is represented as **OBJECTS** [often Called **DOMAIN** model]. However, the storage medium is based on a **RELATIONAL** paradigm. Hence, there exists an inevitable mismatch, the so-called Object/Relational **Impedance Mismatch** which creates a **vacuum** between the Object-Oriented Model of a well-designed application [the **DOMAIN** model] and the relational model in a database schema. This vacuum is surprisingly wide.



How to Map One to the Other?:

The most native approach that is usually taken is a simple mapping between each class the

database table. This approach requires writing a lot of code spec that maps one to the other. This code spec is often complex, tedious and costly to develop.



The Solution to The Impedance Mismatch:

This impedance mismatch has led to the development of several different object persistence

technologies attempting to bridge the gap between the relational world and the object-oriented world. Hence the solution is **using an Object Relational Mapping Tool**. An Object Relational Mapping Tool provides

a simple, API for **storing** and **retrieving** Java objects directly to and from the relational database.

Object/Relational Mapping [ORM] is a technique that allows an application written in an

object oriented language to deal with the information [it manipulates] as objects, rather than using database specific concepts such as ROWS, COLUMNS and TABLES which is facilitated by a software called **Object/Relational Mapper**.

An **Object/Relational Mapper** is a piece of software that is used to transform: **An OBJECT**

view of the data INTO A RELATIONAL view Object/Relational Mapper also offers persistence services [CRUD] such as:

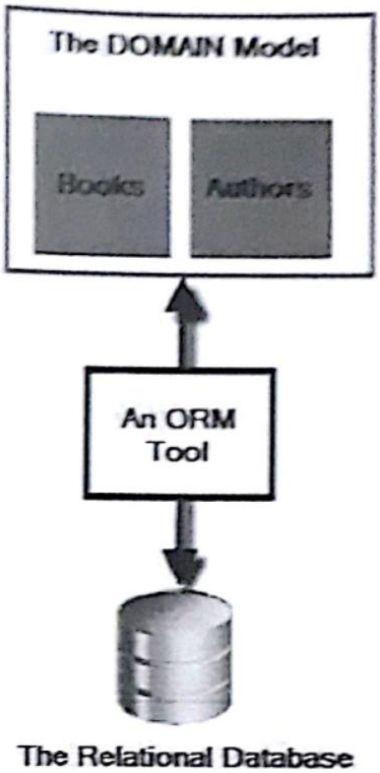
1. CREATE
2. READ
3. UPDATE
4. DELETE

O/R mapping is performed by a persistence framework. This framework knows how to:

- **Query** the database to retrieve objects
- **Persist** those objects back to their representation in the database's tables and columns All this is known with the help of Mappings. Mappings are defined in **metadata**, typically annotations.

ORM has several benefits. In particular:

- Eliminates writing SQL to load and persist object state, leaving the developer free to concentrate on the business logic
- Enables creating an appropriate DOMAIN model, after which, the developer only needs to think in terms of OBJECIS, rather than TABLES, ROWS and COLUMNS
- Reduces dependence on database specific SQL and thus provides Portability across databases
- Reduces more than 30% of the amount of Java code spec that needs to be written by adopting an ORM



19.7 OBJECT/RELATIONAL MAPPING

An ORM provides the following advantages:

1. Better System Architecture:

Most of the times all the application functionality and the database access code spec is held together. This brings in some severe disadvantages. It becomes really difficult to reuse code spec. Hence code repetitions occur at several different places. Changing anything becomes quite difficult, as each and every place that holds the repetitive code spec needs to be located and changed accordingly. If the application functionality [business logic] and the database access code spec [persistence mechanism] are separated, applying changes become very easy. Changes can be made to one part without influencing the other parts.

2. Reduce Coding Time:

Most of the time the database access code spec is simple inserts, updates or deletes. These are SQL statements which sometimes are quite tedious to code. ORM tool helps here, by generating them on the fly and thereby saves a lot of time.

3. Caching and Transactions:

Most ORM tools such as Hibernate come with features such as Caching and Transactions. These features, if chosen to hand code are not so easy to implement. And it definitely does not make sense to develop them when they already exist.

Where Does Java Persistence API Fit In?:

Today, there are several good ORM tools available that perform the mapping between objects←→ Relational Database Tables and thereby solve the Impedance Mismatch:

1. Hibernate
2. OpenJPA
3. TopLink
4. EclipseLink

These Object-Relational Mapping tools/frameworks allow the developers to focus on the object model instead of dealing with the mismatch between the object-oriented and relational paradigms. Unfortunately, each of these tools has its own set of APIs. This ties the application code spec to the proprietary interfaces of a specific vendor [ORM tool]. Since the code spec is tied to a specific vendor [referred to as Vendor Lock-In], switching to another tool is not possible without rewriting all the persistence code spec.

Java Persistence API:

The Java Enterprise Edition's [Java EE 5] Enterprise JavaBeans 3 has introduced a new way of communicating with databases called the lava Persistence API [JPA], a section of EJB 3.0.

JPA defines a persistence framework which provides a way of automatically mapping normal Java objects to an SQL database. In other words, it helps load, search and save the data model objects. JPA combines the best features from most of the available persistence standards:

- Is extremely easy to use, entities can be created as simple as creating serializable classes
- Supports large data sets
- Provides data consistency
- Allows concurrent use of information
- Provides querying capabilities of JDBC
- Allows using advanced object-oriented concepts such as inheritance

Java Persistence API As A Specification:

JPA is a specification from Sun, which is released under Java EE 5 specification.

It is not:

- An implementation
- A product

Hence, it cannot be used as it is for persistence. It needs an ORM implementation to work with and persist Java Objects. Technically, JPA is just a set of interfaces [a Specification] and thus requires an implementation. All specifications require vendors or open source projects to implement them. Using JPA therefore requires picking up an implementation [ORM tool such as Hibernate, Toplink, OpenJPA or any other ORM that implements JPA].

JPA defines the interface that an implementation has to implement. The whole point of having a standard interface is that users can, in principle, switch between implementations of JPA without changing their code. This way, JPA helps prevent **Vendor Lock-In** by relying on a strict specification such as JDO and EJB 2.x entities. Currently most of the persistence vendors [ORM tool providers] have released implementations of JPA. Since, the Java Persistence code spec covers several persistence frameworks into a single API, an application written using JPA will work across several implementations [Hibernate Toplink and so on]. This is very useful, especially when, development begins using the free 'open source ORM implementations and later on when the need arises, the open source' implementation is swapped with a commercial ORM implementation.

JPA 1.0 as a part of EJB 3:

After the failure of EIB 2, BIB 3.0 came into existence to make Enterprise JavaBeans easier and more productive to use. EJB 3.0 introduced a new model for persistence called The Java Persistence API 1.0. Developers from the leading vendors of object-relational mapping solutions such as Hibernate, Toplink and JDO joined the Java group and helped shape the new EJB specification. This resulted into a new specification of EJB 3.0 [Java EE 5] with the following distinct pieces:

1. Existing EJB 2.1 APIs and the traditional contracts from the perspectives of the container, the bean provider and the client with new features such as Java EE injection, EJB 3.0 interceptor Specifications and lifecycle call-back changes

2. A simplified API for developing new session and message-driven components

3. The Java Persistence API

Even though, Java Persistence API is defined as part of the EJB 3.0 specification, it is not needed to have an EJB container or a Java EE application server in order to run applications that use persistence.

19.8 SUMMARY

Data Persistence is a means for an application to persist and retrieve information from a non-volatile storage system. Persistence is vital to enterprise applications because of the required access to relational databases.

19.9 PRACTICE QUESTIONS:

MCQ:

Q.1) Which one of the following best illustrates the concept of Object Persistence.

a) Determining how an object's state[data stored in member variable of an object]is stored in database table columns.

b) Provides an object – oriented view of persistent data.

c) Design patterns and sophisticated object oriented concepts such as inheritance and polymorphism can be used.

d) Provide one to one mapping to database table.

Answer: A

Q.2) Which of the following is not a correct explanation of JDO.

a) JDO is a standard way to access persistent data in databases, using plain old Java objects (POJO) to represent persistent data.

b) JDO is an object relational mapping tool.

c) Provides object relational query language and support non-relational databases.

d) JDO was popular and an integrated part of enterprise java platform.

Answer: D

Q.3) What is ORM.

a) Object Relation Map

b) Object Rate Mapping

c) Object Relational Mapping

d) Object Relational Mapper

Answer: C

Q.4) Which method is used to remove a persistent instance from the datastore.

a) Session.remove()

b) Session.delete()

c) Session.del()

d) Session.rm()

Answer: B

Q.5) Which tool provides a set of persistent annotations to define mapping metadata.

a) JPA

b) JSR

c) XML

d) JRE

Answer: A

Q.6) Which of the following simplifies Object Relational Mapping tool.

a) Data Creation ,Data Isolation, Data Access

b) Data Manipulation, Data Creation, Data Extraction

c) Data Creation, Data Manipulation, Data Access

d) Data Isolation, Data Extraction, Data Manipulation

Answer: C

Q.7) The problem which arises because of the difference between model of programming language and model of database is classified as.

a) modelling mismatch

b) referential mismatch

c) dependence mismatch

d) impedance mismatch

Answer: D

Q.8) An ORM Framework persist your objects according to the mapping metadata you provide.

a) False

b) True

c) May be

d) Can't say

Answer: B

Q.9) Which technique is used by Hibernate to persist collections of embeddable types.

a) ElementCollection

b) ManyToMany

c) OneToMany

d) CollectionElement

Answer: A

Q.10) What is the JPA equivalent of hibernate.cfg.xml file.

a) configuration.xml

b) persistence.xml

c) jpa.configuration.xml

d) jpa.persistence.xml

Answer: B

Descriptive:

1. Write a note on Java Persistence API.
2. Explain the architecture of JPA 2.0.
3. What is Persistence?
4. What are the various options that allow the Java developers to store and retrieve persistent data?
5. What is Object/Relational Mapping?
6. List and explain various advantages provided by Object Relational Mapping.

19.10 REFERENCES

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Java EE 7 For Beginners	Sharanam Shah, Vaishali	SPD	First	2017

		Shah			
2.	Java EE 8 Cookbook: Build reliable applications with the most robust and mature technology for enterprise development	Elder Moraes	Packt	First	2018
3.	Advanced Java Programming	Uttam Kumar Roy	Oxford Press	NA	2015

*****

20

JAVA PERSISTENT API

Unit Structure

20.1 OBJECTIVES

In this chapter we will learn the Java Persistence API (JPA) specification of Java. Also learn how it is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems. As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

20.2 INTRODUCTION TO JAVA PERSISTENCE API

20.2.1 Introduction to Java Persistence API:

Conventionally when a programmer is dealing with applications, he/she probably imagines that an application has some specific functions [business logic / application logic] and all that it has to do is process and then finally save data in a database. When thinking in terms of Java as the programming language of choice for most programmers, the business logic of an application works with OBJECTS of different CLASS types. However, when dealing with the storage medium, it’s important to note that the tables of a database are not OBJECT. This becomes a little difficult to handle.

Today, there are several Object/Relational mapping tools that have become popular because they help bridge up the gap between the Objects and the relational database which in turn allows the developers concentrate on the business logic:

- Hibernate
- TopLink
- EclipseLink
- OpenJPA

And many others are available, unfortunately, each of these tools has its own set of APIs. This ties the application code spec to the proprietary interfaces of a specific vendor [ORM tool]. In spite of having so many Object/Relational mapping tools, there is no single persistence standard for the Java platform that can be used in both the Java EE and Java SE environments. This is where Java Persistence API comes in.

JPA helps standardize the persistence API for the Java platform. A lot of these Object/ Relational tool vendors [such as TopLink and Hibernate, as well as other leading application server vendors and JDO vendors] have widely accepted the JSR-220 specification and most of them have also released their implementation of JPA.

20.2.2 The Java Persistence API:

The Java Persistence API is a standard API that allows accessing, persisting and managing data between Java objects / classes and the relational database in Java EE 5 platform. JPA was defined as a part of the EJB 3.0 specification as a replacement to the EJB 2 CMP (Container-Managed Persistence) Entity Beans specification. It is now considered the standard industry approach for Object → Relational Mapping in the Java Industry.

With the help of a standard API, JPA provides standard mechanisms to using Object Relational Mapping tools.

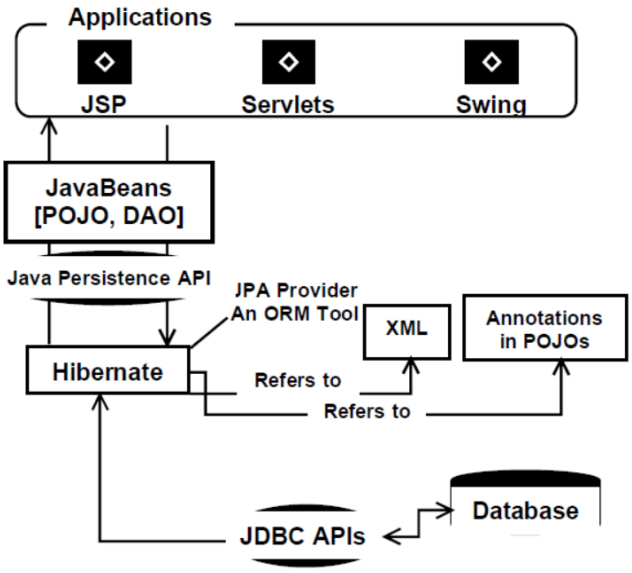
Java Persistence consists of three areas:

- The Java Persistence API
- Object/ Relational mapping metadata
- The Query language

Object-relational mapping with the Java Persistence API is entirely metadata driven. It can be done by adding annotations to the code spec OR using externally defined XML 0R using annotations as well as externally defined XML.

JPA uses Annotations or XML to map Objects to a Relational Database. These Objects are called Entities. Entities are nothing but POJOs that do not extend any class nor implement any interface. JPA also allows querying and retrieving data using its own query language called Java Persistence Query Language. It generates all the necessary SQL calls to achieve this and thereby, relieves the developers from manual result-set handling and object conversion.

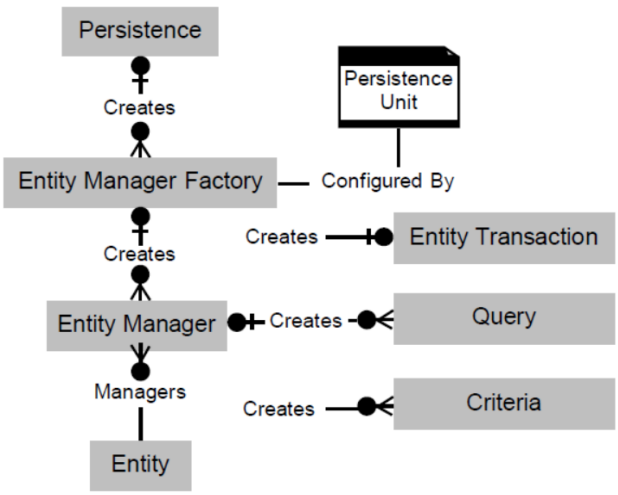
20.2.3 JPA, ORM, Database and the Application:



JPA is made up of a few classes and interfaces. The application communicates with the configured IPA provider [in this case EclipseLink] to access the underlying data. Typically, applications invoke the appropriate methods of the Java Persistence API. These methods are passed the persistence objects and instructed to operate upon them. The information about the mapping [metadata] between the instance variables of classes and the columns of the tables in the database is available either in XML and/ or POJOs with the help of Annotations.

POJOs are Java classes that represent the tables in the database. Data Access Object [DAO] is the design pattern that can be used [if required] to deal with database operations. EclipseLink uses the database [using JDBC API internally] and refers to the metadata to provide persistence services [and Persistent Objects] to the application. The application talks to EclipseLink via the JPA to perform the SELECT, INSERT, UPDATE and DELETE operations on the database tables. The ORM tool automatically creates the required SQL queries and fires them using the JDBC APIs.

20.2.4 Architecture of JPA:



Persistence:

The javax.persistence.Persistence class contains static helper methods to obtain EntityManagerFactoryinstances in a vendor-neutral fashion.

EntityManagerFactory:

The EntityManagerFactory is created with the help of a Persistence Unit during the application start up. It serves as a factory for spawning EntityManager objects when required. Typically, it is created once [one EntityManagerFactory object per database] and for later use kept alive for later use.The javax.persistence.EntityManagerFactory class is a factory for EntityManagers.

EntityManager:

The EntityManager object [javax.persistence.EntityManager] is lightweight and inexpensive to create. It provides the main interface to perform actual database operations. All the POJOs i. e. persistent objects are saved and retrieved with the help of EntityManager object. Typically, EntityManager objects are created as needed and destroyed when not required. Each EntityManager manages a set of persistent objects and has APIs to insert new Objects and delete existing ones. EntityManagers also act as factories for Query instances and CriteriaQuery instances.

Entity:

Entites are persistent objects that represent datastore records.

Entity Transaction:

A Transaction represents a unit of work with the database. Any kind of modifications initiated via the EntityManager object are placed within a transaction. An EntityManager object helps creating an EntityTransaction object. Transaction Objects are typically used for a short time and are closed by either committing or rejecting.

Query:

Persistent objects are retrieved using a Query object. Query objects [javax.persistence.Query] allows using SQL or Java Persistence Query Language [JPQL] queries to retrieve the actual data from the database and create objects.

Criteria:

Criteria API IS a non-string-based API for the dynamic construction of object-based queries [javax.persistence.criteria]. Just like JPQL static and dynamic queries, criteria query objects are passed to the EntityManager'screateQuery() method to create Query objects and then executed using the methods of the Query API. A CriteriaQuery object can be thought of as a set of nodes corresponding to the semantic constructs of the query:

- Domain objects, which correspond to the range variables and other identification variables of the JPQL FROM clause
- WHERE clause predicates, which comprise one or more conditional expression objects
- SELECT clauses, which comprise one or more select item objects
- ORDER-BY and GROUP-BY items
- Subqueries

20.2.5 How JPA works?:

An XML is created or annotations are added to POJOS, which inform the JPA provider [such as Eclipselink] about:

1. The classes needed to Store the data
2. How the classes are related to the tables and columns in the database

This way all the necessary information is provided to the JPA provider.

During the runtime, the IPA provider reads the XML and/or annotations and dynamically builds Java classes to manage the translation between the database and the Java objects. An EntityManagerFactory is created from the compiled collection of mapping metadata. The EntityManagerFactory provides the mechanism for managing persistent classes and the EntityManagerinterface. The EntityManager class provides the interface between the persistent data store and the application. The EntityManager interface wraps a JDBC connection, which can be user managed or controlled by the JPA provider and is only intended to be used by a single application thread, then closed and discarded.

All the database interaction is done via a simple, intuitive API that JPA provides. This allows performing queries against the objects represented by the database. This API informs the JPA provider:

- To save the changes whenever the objects are changed
- To store the objects in the database whenever new objects are created Based on all the above discussion, the following is what will be required to build an application that persists data.

An ORM tool:

- To avoid low level JDBC and SQL code
- To leverage object-oriented programming and object model usage
- To provide database and schema independence
- Since it’s free [Most ORMs are free and open source]

- To use high end performance features such as caching and sophisticated database and query optimizations

JPA:

- To gain portability across application sewers and persistence providers [ORMs]
- Since it's a standard and part of EJB 3 and Java EE
- Since it provides a usable and functional specification
- Since it supports both Java EE and Java SE

20.2.6 JPA Specification:

Specification is part of Java EE 6 where IPA has been officially separated from distinct API.

JPA 2.0 brings in the following enhancements:

1. ORM mapping enhancements such as:

(a) Ability to model collections, maps and lists using @ElementCollection annotation

(b) Ability to map unidirectional one-to-many relationships as JPA 1.0 only allowed Bidirectional one-to-many relationships

2. EntityManager and the Query APIs now support:

(a) Retrieving the first result

(b) Accessing the underlying vendor-specific entity manager/ query objects

(c) Pessimistic locking

3. JPQL has been enhanced with SQL-like CASE, NULLIF, COALESCE and like capabilities

4. Criteria API similar to the one that Hibernate provides

5. Standardization of:

(a) Second level caching

(b) Hints for Query configuration and for EntityManager configuration

(c) Metadata to support DDL generation and Java2DB mapping

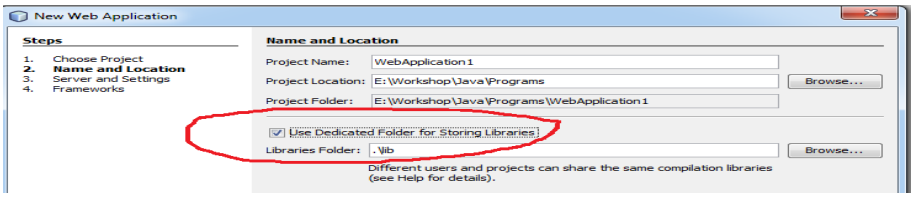
6. Support for validation

20.3 WRITING JPA APPLICATION

JPA Practical using GuestBook

Steps:

1. Create Web Application with dedicated folder for Library



2. Add Simple java class or Persitent Entity class from Database (code below GuestBook.java)
3. Add SQL Connector Jar file to Library
4. Create Persistence Unit using jdbc connection to MySQL database
5. Create the JSP files (codes given below)
6. Run the Application.

GuestBook.java

~~~~~~~~~~~~~~

```
package tyit;
import javax.persistence.*;


@Entity
@Table(name="GuestBook")
public class GuestBook {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name="VisitorNo", unique=true, updatable=false)
   private Integer visitorNo;
   @Column(name="VisitorName")
   private String visitorName;
   @Column(name="Message")
   private String message;
   @Column(name="MessageDate")
```
~~~~~~~~~~~~~~

```
private String messageDate;

public GuestBook() {
}

public Integer getVisitorNo() {
   return visitorNo;
}
public void setVisitorNo(Integer visitorNo) {
   this.visitorNo = visitorNo;
}

public String getVisitorName() {
   return visitorName;
}
public void setVisitorName(String visitorName) {
   this.visitorName = visitorName;
}

public String getMessage() {
   return message;
}
public void setMessage(String message) {
   this.message = message;
}

public String getMessageDate() {
   return messageDate;
}
public void setMessageDate(String messageDate) {
   this.messageDate = messageDate;
}
```

```
}
```

index.jsp

~~~~~~~~~~

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <body style="background-color: pink;">
        Sign the Guest Book
         <form action="GuestBookView.jsp" method="post">
                    Visitor Name: <input name="guest" maxlength="25" size="50" />
                                  Message: <textarea rows="5" cols="36" name="message"></textarea>
              <input type="submit" name="btnSubmit" value="Submit" />
         </form>
   </body>
</html>
```

GuestBookView.jsp

~~~~~~~~~~~~~~~~~

```
<%@page import="java.util.*,javax.persistence.*,tyit.GuestBook" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<%!
   private EntityManagerFactory entityManagerFactory;
   private EntityManager entityManager;
   private EntityTransaction entityTransaction;
   List<GuestBook> guestbook;
%>
<%
                                    entityManagerFactory = Persistence.createEntityManagerFactory("JPAApplicationPU");
   entityManager = entityManagerFactory.createEntityManager();
   String submit = request.getParameter("btnSubmit");
```

```
    if(submit != null && ("Submit").equals(submit)) {
      try {
        String guest = request.getParameter("guest");
        String message = request.getParameter("message");
        String messageDate = new java.util.Date().toString();

        GuestBook gb = new GuestBook();
        gb.setVisitorName(guest);
        gb.setMessage(message);
        gb.setMessageDate(messageDate);
        entityTransaction = entityManager.getTransaction();
        entityTransaction.begin();
        entityManager.persist(gb);
        entityTransaction.commit();
      } catch (RuntimeException e) {
        if(entityTransaction != null) entityTransaction.rollback();
        throw e;
      }
      response.sendRedirect("GuestBookView.jsp");
    }

    try {
          guestbook = entityManager.createQuery("SELECT * from
GuestBook").getResultList();
    } catch (RuntimeException e) { }
   entityManager.close();
%>
<html>
  <body>
    View the Guest Book  <b>Click <a href="index.jsp"> here</a> to
sign the guestbook.</b>

    <hr />
```

```
<%
 Iterator iterator = guestbook.iterator();
 while (iterator.hasNext()) {
    GuestBook obj = (GuestBook) iterator.next();
%>
    On <%= obj.getMessageDate() %>,<br />
    <b><%= obj.getVisitorName() %>:</b>
    <%= obj.getMessage() %>
    <br /><br />
<%
 }
%>
</body>
</html>
```

20.4 SUMMARY

The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

As JPA is just a specification, it doesn't perform any operation by itself. It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

20.5 SAMPLE QUESTIONS

MCQ:

Q.1 What is the full form of JPQL

a) Java Persistence Query Language

b) Java Provider Query Language

c) Java POJO Query Language

d) Java performance query language

Ans: a) Java Persistence Query Language

Q.2 Which tool automatically creates the required SQL queries

a) XML

b) JPQL

c) ORM

d) JPA

Ans: c) ORM

Q.3 Which one is the simple java class that represents a row in a database table.

a) Attribute

b) Primary key

c) foreign key

d) Entity

Ans: d) Entity

Q.4 An object is called persistent if it is stored in the database and can be accessed anytime. This type of entity property is called as

a) Persistability

b) Persistent Identity

c) Transactionality

d) Granularity.

Ans: a) Persistability

Q.5 Which keyword makes possible to filter results after evry join, leading to smaller results after each successive join.

a) ON

b) delete.

c) enter

d) remove.

Ans: a) ON

Q.6 Which annotation is used to link two tables through a relation table?

a. @RelationTable

b. @JoinTable

c. @LinkTable

d. @GroupTable

Ans: b) @JoinTable

Q.7 Which annotation is used to create Pk-Fk relation b/w two tables?

a. @JoinColumn

b. @ForeignKey

c. @JoinedKey

d. @PrimaryKey

Ans: a) @JoinColumn

Q.8 Which statement(S) is/are incorrect

a. Stored procedure may return a value and function must return a value.

b. Function has only IN parameter.

c. Try and Catch can be used with both stored procedure and function.

d. Stored procedure has IN and OUT parameter.

Ans: c)Try and Catch can be used with both stored procedure and function.

Q.9 Which API is used to define queries for entities and their persistent state by creating query-defining objects

a) Criteria API.

b) Query API

c) Entity API

d) Transaction API

Ans: a) criteria API

Q.10 JPA 2.1 introduced Which method to call databse funtions which are not directly supported by the standard

a) delete()

b) insert()

c) call()

d) function()

Ans: d) function()

Descriptive:

1. Write a note on Java Persistence API.
2. Explain the architecture of JPA 2.0.
3. Using suitable example explain the various components of JPA.
4. Create simple JPA application to store and retrieve Book details. << similar to above example >>
5. Develop a JPA Application to demonstrate use of ORM associations.

20.6 REFERENCES

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Java EE 7 For Beginners	Sharanam Shah, Vaishali Shah	SPD	First	2017
2.	Java EE 8 Cookbook: Build reliable applications with the most robust and mature technology for enterprise development	Elder Moraes	Packt	First	2018
3.	Advanced Java Programming	Uttam Kumar Roy	Oxford Press	NA	2015

*****

21

HIBERNATE

Unit Structure

21.1 OBJECTIVES

In this we will explain What is hibernate and how to install Hibernate & other associated packages to prepare a develop environment for the Hibernate applications. We will work with MySQL database to experiment with Hibernate examples, so make sure you already have setup for MySQL database.

21.2 INTRODUCTION TO HIBERNATE

21.2.1 What is Hibernate?:

Any project that requires database interaction have started looking at ORM tools than considering the traditional approach i.e. JDBC. Hibernate ease the job of programmer in working with traditional database using concrete SQL queries in Java by using java object mapped with data base and allow programmer to interact with database just like other java class or object. The objective of Hibernate is to free the programmer from tedious database interaction and focus on working with java objects and features of application instead of worrying about how to work with data from database. Hibernate does this by copying data from database table to java class and saving state of an object to database table.

Hibernate is a free, open source, high performance persistent ORM and query tool.

Gavin King, founded the Hibernate Project in 2001 at JBoss Inc. [now part of RedHat Inc.]

Hibernate provides:

- Mapping of java classes to Database table.
    - e.g.: Student.java class → Student table in Oracle
- Mapping of java data type to SQL data type.

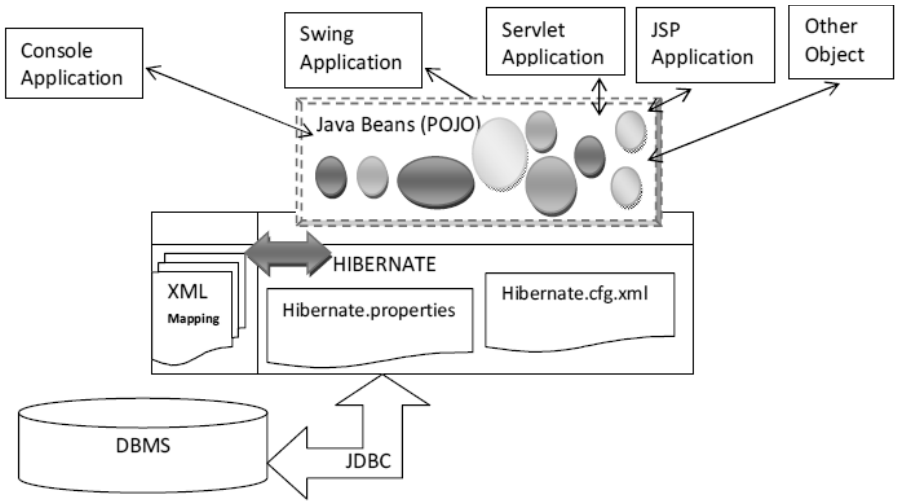
- e.g.: int → number, String → varchar, java.sql.Date → DateTime, etc.

- Flexibility in Querying and retrieving data from any database.
- Freedom to switch to any data base without changing the application logic/presentation logic.

21.2.2 Why Hibernate?:

- Hibernate is a high-performance Object/Relational persistence and query service, available free under the open source GNU Lesser General Public License (LGPL).
- Takes care of all SQL operations in a java program.
- Make feel like working with Objects rather than SQL in performing Create, Read, Update and Delete SQL operations.
- Complete portability across database.
- Supports IDE like Netbeans by providing plug-in.
- Cuts down development time by using Object oriented technology like inheritance, composition and java collection framework.
- Can have multiple primary key generations through multiple identity column mapping.
- Hibernate has two cache layers for handling thread safety, non-blocking concurrent data access, connection pooling etc.
- Allows working with any database like Oracle, MySql, DB2, Sybase, PostgresSQL, Apache Derby, MS SQL Server, MS Access, etc.

21.2.3 Hibernate, Database and The Application



Application program uses persistent objects that represent data from database. Configuration is stored in configuration files like

Hibernate.properties and hibernate.cfg.xml to map the objects to corresponding database and hibernate dilect to choose appropriate SQL statement for database. A mapping file is used to map instance variables of class to database columns. Hibernate uses JDBC API with JTA to perform database operations like Create, Insert, Update, Delete, Select etc. to automatically fire queries bases on operations performed by application program on java objects.

21.2.4 Components of Hibernate:

The main components of Hibernate are:

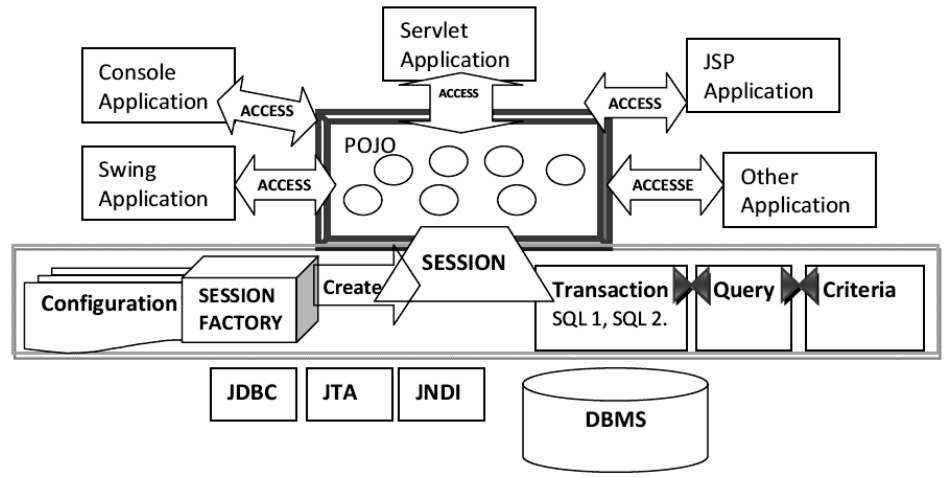
1. Connection Management: Hibernate solves the problems which arise when a relational database is connected by an application written in object oriented programming language style, due to data type differences, manipulative differences, transactional differences, structural and integrity differences. Connection Management provides efficient connection management and removes the overhead of database interaction from application program.

2. Transaction Management: Transaction in hibernate is managed by JTA and JDBC. It allow to fire more than one SQL query at a time.

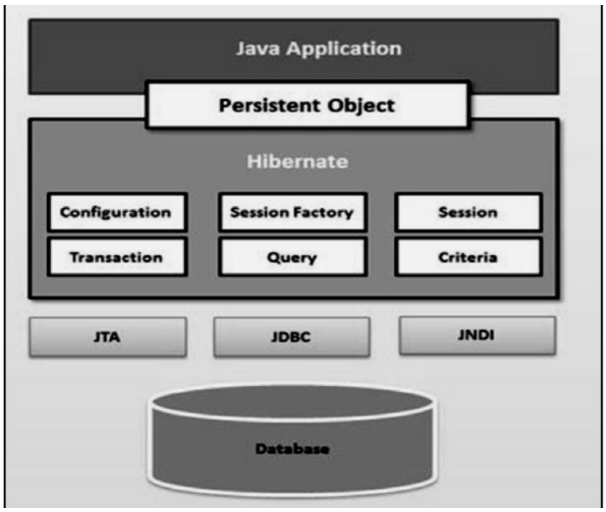
3. Object Relation Management: It is used to map java objects to database tables. Hibernate stores the persistent objects in session and reads the state of an object to execute appropriate database query.

21.1.5 Architecture of Hibernate:

Hibernate is a layered architecture. The main components are Configuration, Session Factory, Session, Transaction, Query and Criteria. Hibernate uses existing Java APIs, like JDBC for database connectivity, Java Transaction API(JTA) for transaction and Java Naming and Directory Interface (JNDI) for easy integration with other enterprise applications.



Following is a detailed view of the Hibernate Application Architecture with few important core classes.



Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction offunctionality common to relational databases, allowing almost any database with a JDBC driver to supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Configuration: It represents properties/configuration of a hibernate application. It the first object created in a hibernate application and created once at the time of application execution. This object reads the configuration file to establish database connection and mapping. This object helps in creating session factory.

It represents a configuration or properties file required by the Hibernate.

The Configuration object provides two keys components:

1. Database Connection: This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.

2. Class Mapping Setup This component creates the connection between the Java classes and database tables.

Session Factory: It is created using configuration object at the time of application startup to serve as a base for creating light weight sessions conveniently during client's request. One session factory is created for one database for multiple database multiple session factory objects are created.

Session Object: Sessions are single threaded, lightweight objects to communicate with database represented by session class from org.hinernate package. Persistent object are created, saved and retrieved using session object during client interaction. It wraps the Connection class from java.sql package and serves as factory for Transaction. The session objects should not be kept open for a long time because they are

not usually thread safe and they should be created and destroyed them as needed.

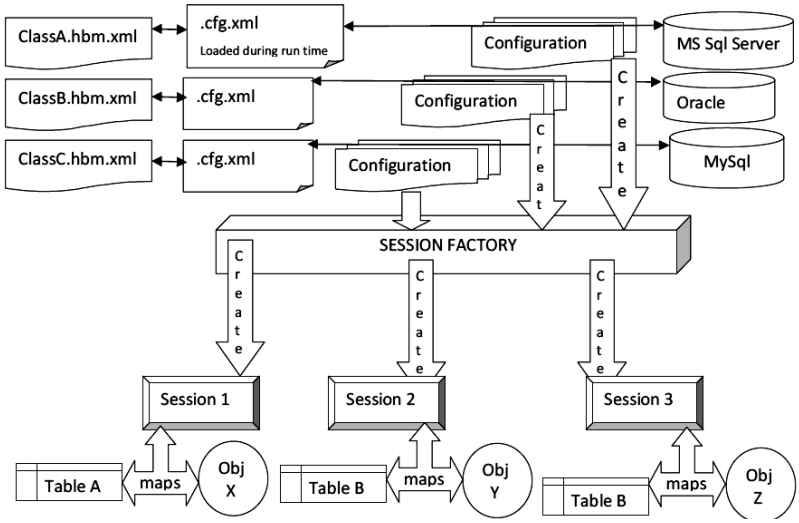
Transaction: Transaction is a single threaded object used by application to represent group of SQL queries to form a unit of work called transaction. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA). All changes during a session are placed within transaction. Transactions are either completed using commit or canceled using rollback. The org.hibernate.Transaction interface provides methods for transaction management.

Query: It uses either conventional SQL or Hibernate Query Language (HQL) to communicate with database. It associates the query parameters, restricts the results coming from database and executes queries. Persistent objects are retrieved using query object.

Criteria: Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.

21.2.6 How Hibernate Works?:

- All configuration files hibernate.cfg.xml are created to describe about the java classes and there mapping with database tables.
- At the time of application startup these files are compiled to provide hibernate framework with necessary information.
- This dynamically builds java class objects by mapping them to appropriate database table.
- A session factory object is created from compiled collections of mapping documents.
- Session Factory spawns a lightweight session to provide interface between java objects and applications.
- Database communication is performed by this session using hibernate API used to map the changes from java object to database table and vice versa.



21.3 WRITING HIBERNATE APPLICATION

In this section we will develop a Hibernate application to store Feedback of Website Visitor in MySQL Database. The application to be built is called Guestbook Feedback Entry using Hibernate. This application should be capable of accepting and displaying employee details using database. To achieve this, it should provide a user interface that accepts Guest's name, message and date.

From the application development perspective, the following software will be required on the

development machine:

1. Java Development Kit
2. NetBeans IDE
3. MySQL community Server [The database server]
4. JDBC driver for MySQL
5. Hibernate 4.XXX or Higher (ORM Tool) [Available on www.hibernate.org/downloads]

create database feedbackdb;

create table GuestBook(

vno int PRIMARY KEY AUTO_INCREMENT,

vname varchar(50),

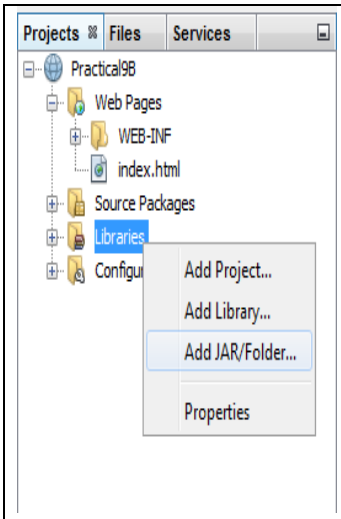
msg varchar(100),

mdate varchar(50)

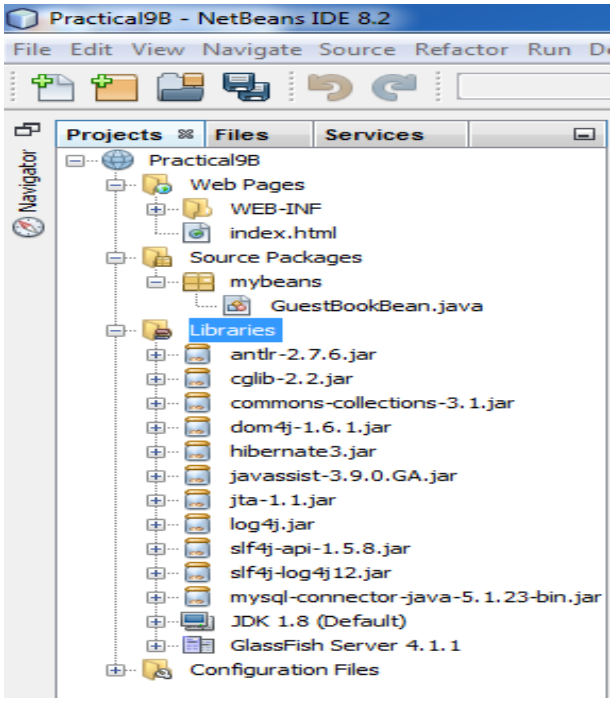
)

Library Files: The Java library [.JAR] i.e. JDBC driver is required. This will be specific to a relational database to be used. In this case MySQL is used as the database of choice, hence, the database specific JDBC driver file will be **MySQL Connector/J 5.1.10**.

The following is the list of jar files required



hibernate-release-5.3.0.Final\lib\required
-->antlr.jar
-->hibernate-common-annotations
-->hibernate-core
-->hibernate-jpa-2.0-api
-->javaassist
-->jboss-logging
-->jboss-transaction-api-1.1-spec
hibernate-release-5.3.0.Final\lib\jpa
-->hibernate-entitymanager.jar
download from dom4j.sourceforge.net
-->dom4j-2.1.1.jar



The Application Development Approach:

The application will be built using JSP.

The **data entry form** that captures the data will be called **index.html** and the page that will fetch and display the entries will be called **Fetch.jsp**. The captured data will be stored in a table called **GuestBook** under the feedbackdb **MySQL database server**.

In the Java application, the POJO that will represent the GuestBook database table will be called **mypack.GuestBookBean.java**.

The following steps are required to build this application:

1. Create the database schema

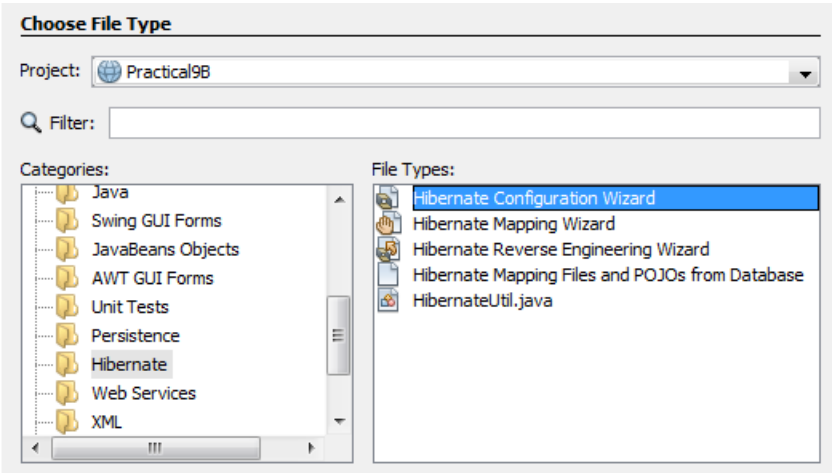
2. Create the Web application
3. Add the Java libraries to the application
4. Create a POJO to represent the table in the database schema
5. Generate a hibernate configuration file.
6. Annotate the POJO to indicate the mapping between the JavaBean properties and the columns in the table
7. Create JSPs with code spec:
   (a) To build a **Configuration** object
   (b) To build a **SessionFactory** object by referencing the Configuration object.
   (c) To obtain an HiberanteSession object from the SessionFactory
   (d) To perform the required database operations.

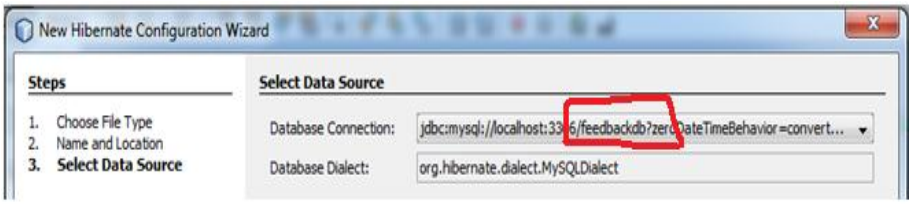
GuestBookBean.java

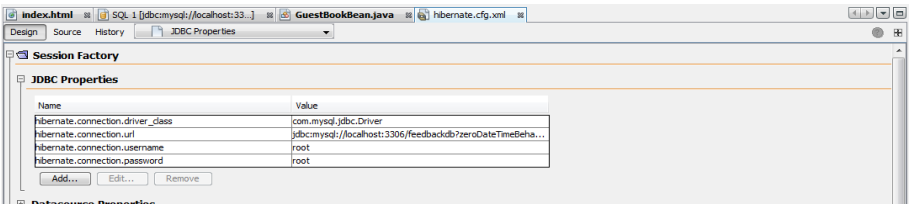
```
package mypack;
import javax.persistence.*;
@Entity
@Table(name="guestbook")
public class GuestBookBean implements java.io.Serializable {
   @Id
   @GeneratedValue
   @Column(name="vno")
   private Integer visitorNo;
   @Column(name="vname")
   private String visitorName;
   @Column(name="msg")
   private String msg;
   @Column(name="mdate")
   private String msgDate;
  public GuestBookBean() {  }
  public Integer getVisitorNo()          {   return visitorNo;  }
  public String getVisitorName()   {   return visitorName;  }
  public String getMsg()           {   return msg;  }
  public String getMsgDate()       {   return msgDate;  }
```

```
  public void setVisitorNo(Integer vn)          {    visitorNo = vn ;  }
   public void setVisitorName(String vn)     {    visitorName = vn;  }
   public void setMsg(String m)                {    msg = m;  }
   public void setMsgDate(String md)          {    msgDate=md;  }
}
```

Source packages →new → others→select category Hibernate →Hibernate Configuration Wizard








```
<hibernate-configuration>
  <session-factory>
    <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property
>
    <property
name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</prope
rty>
    <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/feedbackdb
?zeroDateTimeBehavior=convertToNull</property>
```

```
    <property name="hibernate.connection.username">root</property>
    <property name="hibernate.connection.password">root</property>

    <mapping class="mypack.GuestBookBean" />
  </session-factory>
</hibernate-configuration>
```

index.html:

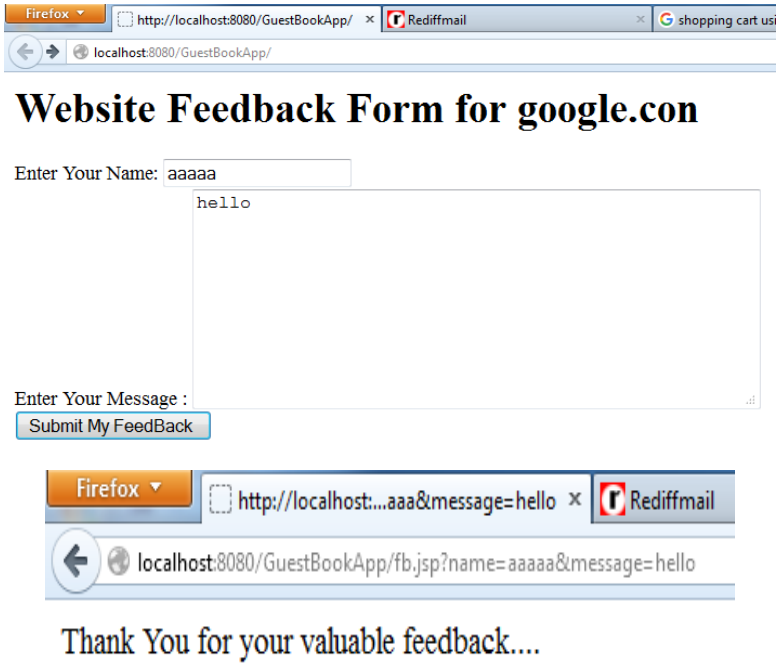
```
<h1>Website Feedback Form for google.con </h1>
<form action="fb.jsp" >
   Enter Your Name: <input type="text" name="name" ><br>
   Enter Your Message : <textarea rows="10" cols="50" name="message"
></textarea><br>
   <input type="submit" value="Submit My FeedBack " >
</form>
```

fb.jsp:

```
<%@page  import="org.hibernate.*, org.hibernate.cfg.*, mypack.*"  %>
<%!          SessionFactory sf;
org.hibernate.Session hibSession;
%>
<%
sf = new Configuration().configure().buildSessionFactory();
hibSession = sf.openSession();
Transaction tx = null;
GuestBookBean gb = new GuestBookBean();
try{
tx = hibSession.beginTransaction();
String username = request.getParameter("name");
String usermsg = request.getParameter("message");
String nowtime = ""+new java.util.Date();
gb.setVisitorName(username);
gb.setMsg(usermsg);
gb.setMsgDate(nowtime);
```

```
hibSession.save(gb);
tx.commit();
out.println("Thank You for your valuable feedback....");
}catch(Exception e){out.println(e);}
hibSession.close();
%>
```

Output:



21.4 SUMMARY

In this Chapter we studied hibernate and steps to install Hibernate & other associated packages to prepare a develop environment for the Hibernate applications. We also worked with MySQL database to

experiment with Hibernate examples, so make sure you already have setup for MySQL database.

21.5 PRACTICE QUESTIONS

MCQ:

1) What is hibernate?

   a) CRM

   b) Programming Tool

   c) ORM

   d) SQL tool.

2) Hibernate framework simplifies the development of java application to interact with the database

   a) True

   b) False

3) Which of the following is true about SessionFactory object in hibernate?

   a) SessionFactory object configures Hibernate for the application using the supplied configuration file.

   b) SessionFactory object allows for a Session object to be instantiated.

   c) The SessionFactory is a thread safe object.

   d) All options mentioned for this question.

4) Which method is used to update the state of the given instance from the underlying database?

   a) Session.store()

   b) Session.keep()

   c) Session.update()

   d) Session.load()

5) HOL stands for

   a) Hibernate Queue Language

   b) Hibernate Query Language

   c) Hypertext Query Language

   d) HighSpeed Query Language

6) Hibernate uses PersisterClassProvider by default.

   a) True

   b) Fales

7) _____ object is used to create SessionFactory object in Hibernate.

   a) Session

   b) Configuration

   c) Transaction

   d) TransactionFactory

8) In hibernate, QBC stands for

   a) Query By Criteria

   b) Query By Call

   c) Query By Code

   d) Query By Column

9) Which method is easy for Java Programmer to add criterion?

   a) SQL

   b) HCQL

   c) HQL

   d) AQL

10) Which of the following simplifies an Object Relational Mapping Tool?

   a) Data creation

   b) Data manipulation

   c) Data access

   d) All options mentioned for this question.

11) _____ is not a core interface of hibernate.

   a) Criteria

   b) Session

   c) SessionManagement

   d) Configuration

12) Is SessionFactory a ThreadSafe object

   a) Yes

   b) No

13) Is Session created per thread in hibernate?

   a) Yes

   b) No

14) All POJO must implement non-argument constructor in hibernate.

   a) True

   b) False

15) When several entities point to the target entity, that is achieved by

a) @OneToOne

b) @OneToMany

c) @ManyToOne

d) @ManyToMany

16) If entity is not annotated with @Table, what will happen?

a) Throws error because no table name is assigned

b) No error and class name will be mapped with table name.

17) A _____ is used to get a physical connection with a database.

a) SessionFactory

b) Session

c) Transaction

d) ConnectionProvider

18) A _____ represents a unit of work with the database and the Java object.

a) SessionFactory

b) ConnectionProvoder

c) Transaction

d) Session

19) _____ is a factory of JDBC connections.

a) SessionFactory

b) ConnectionProvoder

c) Transaction

d) Session

20) Mapping in hibernate can be given to an ORM tool either in the form of an _____or in the form of the annotations

a) XHTML

b) JSON

c) HTML

d) XML

Answers:

1) c
2) a
3) d
4) c
5) b
6) a
7) b
8) a
9) d
10) d
11) c
12) a
13) a
14) a
15) c
16) b
17) b
18) c
19) b
20) d

Descriptive:

1. Explain software development approach of Hibernate?
2. Develop a Hibernate application to store and retrieve employee details in MySQL Database.
3. Develop a Hibernate application to store Feedback of Website Visitor in MySQL Database.
4. Develop an application to demonstrate Hibernate One- To -One Mapping Using Annotation.

21.6 REFERENCES

Books and References:					
Sr. No.	**Title**	**Author/s**	**Publisher**	**Edition**	**Year**
1.	Java EE 7 For Beginners	Sharanam Shah, Vaishali Shah	SPD	First	2017
2.	Java EE 8 Cookbook: Build reliable applications with the most robust and mature technology for enterprise development	Elder Moraes	Packt	First	2018
3.	Advanced Java Programming	Uttam Kumar Roy	Oxford Press	NA	2015

*****