Q.P.CODE 19832 ANSWER KEY

Semester III CORE JAVA

[Total Marks: 75]

Q. 1 **Attempt All (Each of 5Marks)**

(a) **Multiple Choice Questions:**

1. Which of these is supported by method overriding in Java?
   C) **Polymorphism**
2. Which of these have highest precedence?
   A) ()
3. Which of these keywords is used to manually throw an exception?
   C) **throw**
4. Which of these access specifiers can be used for a class so that it's members can be accessed by a different class in the different package?
   A) **Public**
5. Which of these class object can be used to form a dynamic array?
   D) **ArrayList & Vector**

(b) **Fill in the blanks:**

1. JVM stands for **Java Virtual Machine.**
2. **equals()** method of class String is used to compare two String objects for their equality.
3. When we close window **WindowEvent** events will be generated.
4. **this** keyword is used by method to refer to the object that invoked it.
5. Is it not possible to instantiate the **abstract** class.

(c) **Answer in ONE-TWO sentences:**

1. **What is a use of throws keyword?**
   The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
2. **What do you mean by operator precedence?**
   It's a rules for specifying the order in which the operators in an expression are evaluated when the expression has several operators. For example, multiplication and division have a higher precedence than addition and subtraction.
3. State TRUE or FALSE-"void is a default return type of constructor"
   FALSE
4. **What is the default value of the local variables?**
   The local variables are not initialized to any default value, neither primitives nor object references.
5. **What is constructor?**
   Constructor in java is a special type of method/block that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Q. 2 **Attempt the following (Any THREE)**

(a) **Explain different features of Java in detail.**
(any five points-1 marks each)

1. Compiled and interpreted
2. Platform independent and portable
3. Object-oriented
4. Robust and secure
5. Distributed
6. Familiar, simple and small
7. Multithreaded and interactive
8. High performance
9. Dynamic and Extensible

(b) **Write a java program to reverse the digit of a number. Number is passed using the method.**

```
import java.util.Scanner;
class RevNum
{
static int myfun(int num)
{
int x=0;
while( num != 0 )
    {
      x = x * 10;
      x = x + num%10;
      num = num/10;
    }
return x;
}
  public static void main(String args[])
  {
    int num=0,ans=0;

    System.out.println("Input your number and press enter: ");

    Scanner in = new Scanner(System.in);
    num = in.nextInt();
    ans=myfun(num);
    System.out.println("Reverse of input number is: "+ans);
  }
}
```

(c) **What is interface? Explain its significance in java.**

(Definition: 1marks,3marks significance)

Definition An interface in java is a blueprint of a class. It has static constants and abstract methods. The interface in java is a mechanism to achieve abstraction. There can be only abstract methods in the java interface not method body.

Significance:

It is used to achieve abstraction and multiple inheritance in Java. Java Interface also represents IS-A relationship. It cannot be instantiated just like abstract class. There are mainly three importance reasons to use interface. They are given below.

1. It is used to achieve abstraction.
2. By interface, we can support the functionality of multiple inheritance.
3. It can be used to achieve loose coupling.

(d) **Difference between method Overloading and Overriding.**

(1 marks for each difference)

Method Overloading	**Method Overriding**
In Method Overloading, Methods of the same class shares the same name but each method must have different number of parameters or parameters having different types and order.	In Method Overriding, sub class have the same method with same name and exactly the same number and type of parameters and same return type as a super class.
Method Overloading means more than one method shares the same name in the class but having different signature.	Method Overriding means method of base class is re-defined in the derived class having same signature.
Method Overloading is to "add" or "extend" more to method's behavior.	Method Overriding is to "Change" existing behavior of method.
It is a **compile time polymorphism**.	It is a **run time polymorphism**.
It may or may not need **inheritance** in Method Overloading.	It always requires inheritance in Method Overriding.
In Method Overloading, methods must have **different signature**.	In Method Overriding, methods must have **same signature**.
In Method Overloading, methods have same name different signatures but in the same class.	In Method Overriding, methods have same name and same signature but in the different class.
Method Overloading does not require more than one class for overloading.	Method Overriding requires at least two classes for overriding.

(e) **What do you mean by final variables, final methods, and final classes? Explain it.**

(1 mark for each definition with example)

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be**: variable, method, class.** The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only.

final variable: If you make any variable as final, you cannot change the value of final variable(It will be constant).

```
final int speedlimit=90;
```

final method: If you make any method as final, you cannot override it.

```
 final void run()

{System.out.println("running");}
```

final class: If you make any class as final, you cannot extend it

```
1. final class Bike{}
```

(f) **'Subclassing an interface" Explain it with suitable program.**

Definiton: Subclassing an interface means using one interface as parent and extending its properties to developed a child interface. (1mark)

Example: (4mark)

Consider the following scenario. You have an interface A and several implementing classes. It defines 2 methods.

```
interface A{
int doThis();
int doThat();
}
```

Now suppose you want to add another method to the interface A: we create another interface and make it extend the previous interface.

```
interface A_new extends A{
int doThisAndThat();
}
```

Now your users have the option to either use the old interface or upgrade to the new interface.

Any class that implements an interface must implement the methods declared in that interface plus all the methods that are present in the super interface.

If the implementing class is abstract it may choose to implement all, some or none of the methods declared in the interface. But a concrete subclass of the abstract class must implement all the non implemented methods.

Q. Attempt the following (Any THREE)

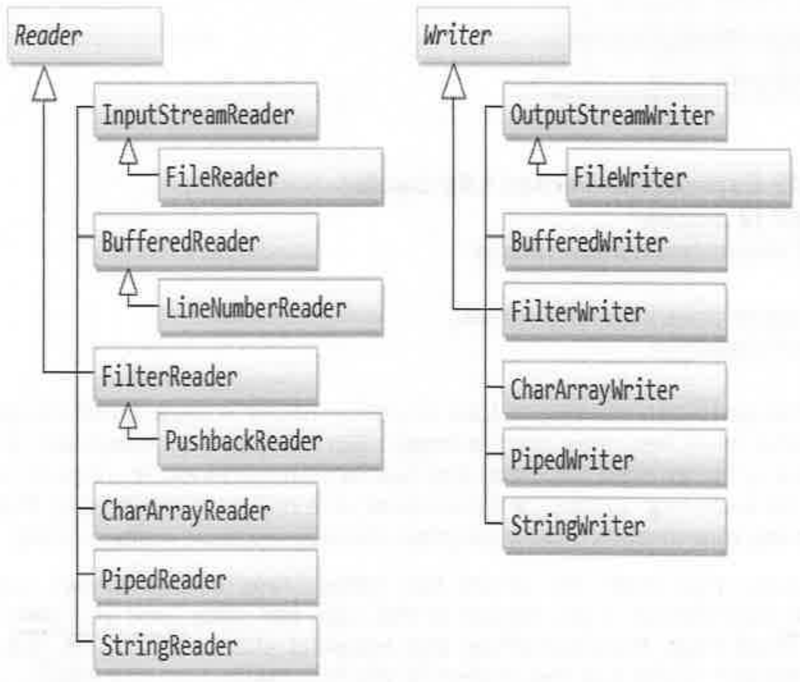
3

(a) **Explain the difference between byte stream classes and character stream classes. (1mark for each difference)**

A stream is a method to sequentially access a file. I/O Stream means an input source or output destination representing different types of sources e.g. disk files.The java.io package provides classes that allow you to convert between Unicode character streams and byte streams of non-Unicode text.

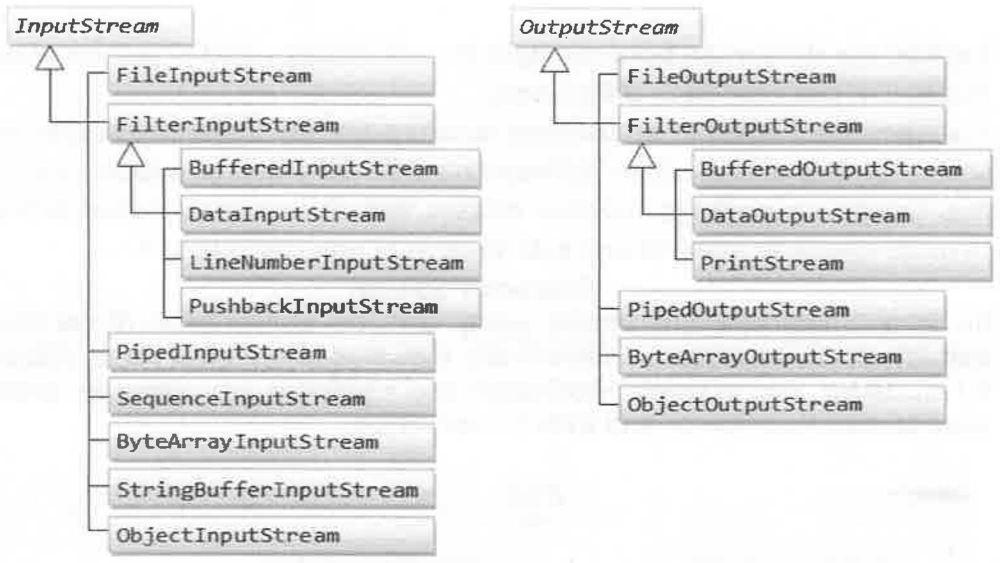
Character Stream

In Java, characters are stored using Unicode conventions (Refer this for details). Character stream automatically allows us to read/write data character by character. For example FileReader and FileWriter are character streams used to read from source and write to destination.



Byte Stream

Byte streams process data byte by byte (8 bits). For example FileInputStream is used to read from source and FileOutputStream to write to the destination.



(b) **What is thread? Explain the Thread Life Cycle.**
What is thread? (2 marks)
In Java, "thread" means two different things:

- An instance of class java.lang.Thread.
- A thread of execution.

An instance of Thread is just…an object. Like any other object in Java, it has variables and methods, and lives and dies on the heap. But a thread of execution is an individual process (a "lightweight" process) that has its own call stack. In Java, there is one thread per call stack—or, to think of it in reverse, one call stack per thread. Even if you don't create any new threads in your program, threads are back there running.

The main() method, that starts the whole ball rolling, runs in one thread, called (surprisingly) the main thread. If you looked at the main call stack (and you can, any time you get a stack trace from something that happens after main begins, but not within another thread), you'd see that main() is the first method on the stack— the method at the bottom. But as soon as you create a new thread, a new stack materializes and methods called from that thread run in a call stack that's separate from the main() call stack.

Life cycle of a Thread (3marks):
A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state. But for better understanding the threads, we are explaining it in the 5 states. The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. **New**: The thread is in new state if you create an instance of Thread class but before the invocation of start() method.

2. **Runnable**: The thread is in runnable state after invocation of start() method, but the thread scheduler has not selected it to be the running thread.
3. **Running**: The thread is in running state if the thread scheduler has selected it.
4. **Non-Runnable (Blocked):** This is the state when the thread is still alive, but is currently not eligible to run.
5. **Terminated**: A thread is in terminated or dead state when its run() method exits.

(c) **Explain how multiple catch can be used in exception handling mechanism? (program 4 marks ,output 1marks)**

Program:
If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block. Let's see a simple example of java multi-catch block.

```
public class TestMultipleCatchBlock{
 public static void main(String args[]){
  try{
   int a[]=new int[5];
   a[5]=30/0;
  }
  catch(ArithmeticException e){System.out.println("task1 is completed");}

  catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
  catch(Exception e){System.out.println("common task completed");}
 System.out.println("rest of the code...");
} }
```

Output:

```
task1 completed
rest of the code...
```

(d) **Discuss the importance of network programming. (any five points 1mark each)**

The term *network programming* refers to writing programs that execute across multiple devices (computers), in which the devices are all connected to each other using a network.

The java.net package of the J2SE APIs contains a collection of classes and interfaces that provide the low-level communication details, allowing you to write programs that focus on solving the problem at hand.

The java.net package provides support for the two common network protocols –

- **TCP** – TCP stands for Transmission Control Protocol, which allows for reliable communication between two applications. TCP is typically used over the Internet Protocol, which is referred to as TCP/IP.
- **UDP** – UDP stands for User Datagram Protocol, a connection-less protocol that allows for packets of data to be transmitted between applications.

Sockets provide the communication mechanism between two computers using TCP. A client program creates a socket on its end of the communication and attempts to connect that socket to a server.

When the connection is made, the server creates a socket object on its end of the communication. The client and the server can now communicate by writing to and reading from the socket.

The java.net.Socket class represents a socket, and the java.net.ServerSocket class provides a mechanism for the server program to listen for clients and establish connections with them.

(e) **Write a program for copying chars from one file to another.**

```
import java.io.*;

class CopyFile
{
public static void main(String args[])
{
File FI =new File("input.txt");
File FO=new File("output.txt");

FileReader fr=null;
FileWriter fw=null;

try
{
fr=new FileReader(FI);
fw=new FileWriter(FO);
int ch;
while((ch=fr.read())!=-1)
{
fw.write(ch);
System.out.print(" "+ch);
}
}
catch(IOException e)
{
```

```
System.out.println(e);
System.exit(-1);
}
finally
{
try
{
fr.close();
fw.close();
System.out.println("\n\n............................Its Done...............................\n");
System.out.println("................See input.txt and output.txt files..................\n\n");

}
catch(IOException e)
{}
}}}
```

(f) **Discuss various methods of Socket class.**

(Any five methods 1 mark each)

Socket class:

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

public void connect(SocketAddress host, int timeout) throws IOException
This method connects the socket to the specified host. This method is needed only when you instantiate the Socket using the no-argument constructor.

public InetAddress getInetAddress()
This method returns the address of the other computer that this socket is connected to.

public int getPort()
Returns the port the socket is bound to on the remote machine.

public int getLocalPort()
Returns the port the socket is bound to on the local machine.

public SocketAddress getRemoteSocketAddress()
Returns the address of the remote socket.

public InputStream getInputStream() throws IOException
Returns the input stream of the socket. The input stream is connected to the output stream of the remote socket.

public OutputStream getOutputStream() throws IOException
Returns the output stream of the socket. The output stream is connected to the input stream of the remote socket.

public void close() throws IOException
Closes the socket, which makes this Socket object no longer capable of connecting again to any server.

Q. 4 **Attempt the following (Any THREE)**

(a) **Write a short note on GridLayout Layout manager. (meaning 1mark,constructor 3marks,diagram 1marks)**

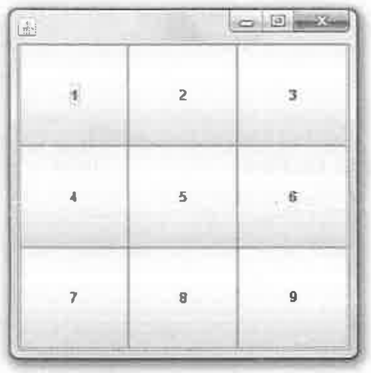
The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.

2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class:



(b) **What are inner classes? Discuss its types. (Definition 1marks,1marks each type)**

Definition: The class written within is called the **nested class**, and the class that holds the inner class is called the **outer class**.

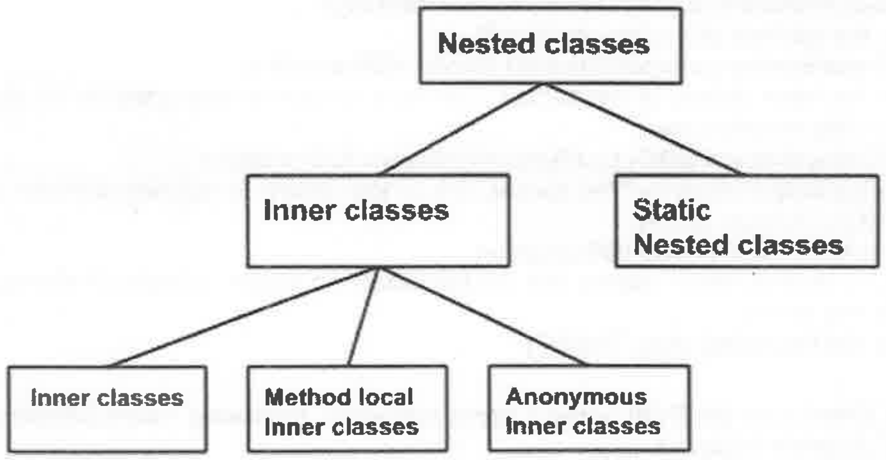
Syntax

Following is the syntax to write a nested class. Here, the class **Outer_Demo**is the outer class and the class **Inner_Demo** is the nested class.

```
class Outer_Demo {
  class Nested_Demo {
  }
}
```

Nested classes are divided into two types −

- **Non-static nested classes** − These are the non-static members of a class.
- **Static nested classes** − These are the static members of a class.



Inner Classes (Non-static Nested Classes)

Inner classes are a security mechanism in Java. We know a class cannot be associated with the access modifier **private**, but if we have the class as a member of

other class, then the inner class can be made private. And this is also used to access the private members of a class.
Inner classes are of three types depending on how and where you define them. They are –

- Inner Class
- Method-local Inner Class
- Anonymous Inner Class

Inner Class
Creating an inner class is quite simple. You just need to write a class within a class. Unlike a class, an inner class can be private and once you declare an inner class private, it cannot be accessed from an object outside the class.
Following is the program to create an inner class and access it. In the given example, we make the inner class private and access the class through a method.
Method-local Inner Class
In Java, we can write a class within a method and this will be a local type. Like local variables, the scope of the inner class is restricted within the method.
A method-local inner class can be instantiated only within the method where the inner class is defined.
Anonymous Inner Class
An inner class declared without a class name is known as an anonymous inner class. In case of anonymous inner classes, we declare and instantiate them at the same time. Generally, they are used whenever you need to override the method of a class or an interface. The syntax of an anonymous inner class is as follows –
Syntax

```
AnonymousInner an_inner = new AnonymousInner() {
   public void my_method() {
      ........
      ........
   }
};
```

Static Nested Class
A static inner class is a nested class which is a static member of the outer class. It can be accessed without instantiating the outer class, using other static members. Just like static members, a static nested class does not have access to the instance variables and methods of the outer class. The syntax of static nested class is as follows –
Syntax

```
class MyOuter {
   static class Nested_Demo {
   }
}
```

(c) **Explain List interfaces and its classes. (1mark for each method)**
The List interface extends Collection and declares the behavior of a collection that stores a sequence of elements.

- Elements can be inserted or accessed by their position in the list, using a zero-based index.
- A list may contain duplicate elements.
- In addition to the methods defined by Collection, List defines some of its own,

which are summarized in the following table.

- Several of the list methods will throw an UnsupportedOperationException if the collection cannot be modified, and a ClassCastException is generated when one object is incompatible with another.

1	**void add(int index, Object obj)** Inserts obj into the invoking list at the index passed in the index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten.
2	**boolean addAll(int index, Collection c)** Inserts all elements of **c** into the invoking list at the index passed in the index. Any pre-existing elements at or beyond the point of insertion are shifted up. Thus, no elements are overwritten. Returns true if the invoking list changes and returns false otherwise.
3	**Object get(int index)** Returns the object stored at the specified index within the invoking collection.
4	**int indexOf(Object obj)** Returns the index of the first instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
5	**int lastIndexOf(Object obj)** Returns the index of the last instance of obj in the invoking list. If obj is not an element of the list, .1 is returned.
6	**ListIterator listIterator( )** Returns an iterator to the start of the invoking list.
7	**ListIterator listIterator(int index)** Returns an iterator to the invoking list that begins at the specified index.
8	**Object remove(int index)** Removes the element at position index from the invoking list and returns the deleted element. The resulting list is compacted. That is, the indexes of subsequent elements are decremented by one.
9	**Object set(int index, Object obj)** Assigns obj to the location specified by index within the invoking list.
10	**List subList(int start, int end)** Returns a list that includes elements from start to end.1 in the invoking list. Elements in the returned list are also referenced by the invoking object.

(d) **What is radio button? Discuss its usage in java. (1mark for each step)**

In java.awt we are not having a special class for radio buttons but we can create radio button from Checkbox class.

java.awt we have a predefined class called CheckboxGroup through which we can add all the checkboxes to the object of CheckboxGroup class.

Steps for converting checkboxes into radiobutton:

1. Create objects of CheckboxGroup class.

```
CheckboxGroup cbg1=new CheckboxGroup ();
```

CheckboxGroup object allows to select a single checkbox among 'n' number of check boxes.

2. Create 'n' number of objects of Checkbox class.

For example:

```
Checkbox b1, b2, b3;

b1=new Checkbox ("COBOL");
b2=new Checkbox ("Cpp");
b3=new Checkbox ("Java");
```

3. Decide which checkboxes are adding to which CheckboxGroup object. In order to add the checkboxes to the CheckboxGroup object, in Check box class we have the following method:

```
public void setCheckboxGroup (CheckboxGroup);
```

For example:

```
b1.setCheckboxGroup (cbg1);
b2.setCheckboxGroup (cbg1);
```

4. Register the events of checkbox with ItemListener by using the following method:

public void addItemListener (ItemListener);

Instead of using a setCheckboxGroup method we can also used the following Constructor which is present in Check box class.

```
Checkbox (String, CheckboxGroup, boolean);
Checkbox (String, boolean, CheckboxGroup);


Checkbox cb1=new Checkbox ("COBOL", cbg1, false);
Checkbox cb2=new Checkbox ("Cpp", cbg1, false);
```

(e) **What is role of map interface? Explain its working with suitable program. (map interface usage 2marks,program 3marks)**

The Map interface maps unique keys to values. A key is an object that you use to retrieve a value at a later date.

- Given a key and a value, you can store the value in a Map object. After the value is stored, you can retrieve it by using its key.
- Several methods throw a NoSuchElementException when no items exist in the invoking map.
- A ClassCastException is thrown when an object is incompatible with the elements in a map.
- A NullPointerException is thrown if an attempt is made to use a null object and null is not allowed in the map.
- An UnsupportedOperationException is thrown when an attempt is made to change an unmodifiable map.

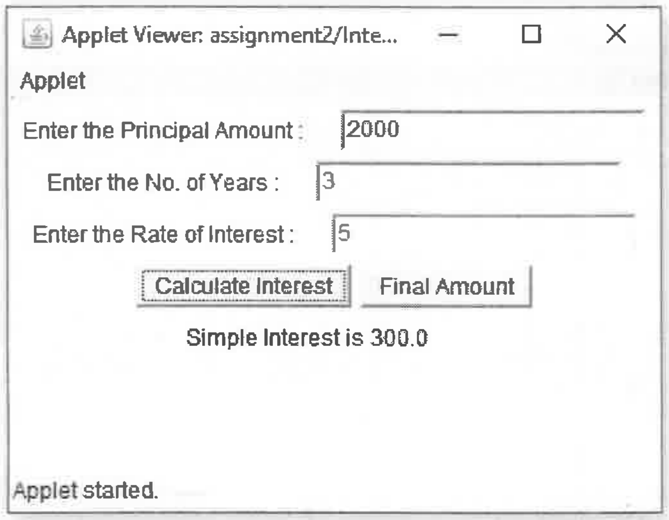
```
import java.util.*;
public class CollectionsDemo {

  public static void main(String[] args) {
    Map m1 = new HashMap();
    m1.put("keta", "82");
    m1.put("Mayank", "31");
    m1.put("Abhishek", "12");
    m1.put("Daisy", "14");
    System.out.println();
    System.out.println(" Map Elements");
    System.out.print("\t" + m1);
  }
}
```

(f) **Design a GUI application that accepts Principle Amount, No. of Years &amp; Rate of Interest from 3 text fields, when you click "Calculate Interest" button, the data is sent to a function that returns the simple interest. When you click on "Final Amount" button, the final amount by adding principle amount and interest should be displayed.**

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class SimpleInterest extends Applet implements ActionListener
{
Label l1,l2,l3,print;
TextField t1,t2,t3;
Button b1,b2;
float
Simple_Interest,Principal_Amount,No_of_Years,Final_Amount,Rate_of_Interest,SI;
public void init()
{
l1=new Label("Enter the Principal Amount : ");
l2=new Label("Enter the No. of Years : ");
l3=new Label("Enter the Rate of Interest : ");
t1=new TextField(20);
t2=new TextField(20);
t3=new TextField(20);
b1=new Button(" Calculate Interest ");
b2=new Button(" Final Amount ");
print=new Label(" ");
add(l1);
add(t1);
add(l2);
add(t2);
add(l3);
add(t3);
add(b1);
add(b2);
add(print);
b1.addActionListener(this);
b2.addActionListener(this);
}
public void actionPerformed(ActionEvent ae)
{
Principal_Amount=Float.parseFloat(t1.getText());
No_of_Years=Float.parseFloat(t2.getText());
Rate_of_Interest=Float.parseFloat(t3.getText());
if(b1==ae.getSource())
{
Simple_Interest=simple_interest(Principal_Amount,No_of_Years,Rate_of_Interest);
print.setText("Simple Interest is "+Simple_Interest);
}
if(b2==ae.getSource())
{
Final_Amount=Principal_Amount+simple_interest(Principal_Amount,No_of_Years,Rat
e_of_Int erest);
print.setText("Final Amount is "+Final_Amount);
}}
public float simple_interest(float P,float R,float T)
{
SI=(P*R*T)/100;
return SI;
}}
```

Output:


Applet Viewer: assignment2/Inte...

Applet

Enter the Principal Amount : 2000

Enter the No. of Years : 3

Enter the Rate of Interest : 5

Simple Interest is 300.0

Applet started.

Q. 5 **Attempt the following (Any THREE)**

(a) **Describe abstract class called Shape which has three subclasses say Triangle, Rectangle, Circle. Define one method area( ) in the abstract class and override it in its three subclasses to calculate area for specific object.**

```
import java.io.;
        abstract class Shape
        {               abstract void area( );   }
        class Triangle extends Shape
        { public void area( )
                {
                        float b,h,area;
                        DataInputStream dis = new DataInputStream(System.in);
                        System.out.println("Calculating area of Triangle");
                        System.out.println("Enter Base : ");
                       b=Float.parseFloat(dis.readLine( ));
                        System.out.println("Enter vertical height : ");
                        h=Float.parseFloat(dis.readLine( ));
                        area=(0.5)*b*h;
                        System.out.println("Area of Triangle is : "+area);
                }}

        class Rectangle extends Shape
        {
                public void area( )
                {       float w,h,area;
                        DataInputStream dis = new DataInputStream(System.in);
                System.out.println("Calculating area of Rectangle");
                System.out.println("Enter width : ");
                w=Float.parseFloat(dis.readLine( ));
                System.out.println("Enter height : ");
                h=Float.parseFloat(dis.readLine( ));
                area=w*h;
                System.out.println("Area of Rectangle is : "+area);
        }}

class Circle extends Shape
{
        public void area( )
        {       float r,area;
                DataInputStream dis = new DataInputStream(System.in);
                System.out.println("Calculating area of Circle");
                System.out.println("Enter Radius");
                r=Float.parseFloat(dis.readLine( ));
                area=3.14*r;
                System.out.println("Area of circle is : "+area);
        }}
public class CallingAll
{
        public static void main(String ar[ ])
        {
                Triangle t1 = new Triangle( );
                Rectangle r1 = new Rectangle( );
                Circle c1 = new Circle( );
                t1.area( );
                r1.area( );
                c1.area( );
        }       }
```

(b) **List and explain the methods used in inter-thread communication.**

- We have three final methods of *class Object* to perform inter thread communication
  1. wait( )
  2. notify( )
  3. notifyAll( )
- Inter thread/process communication is an important topic and very much helpful in case of synchronized methods and threads. As the name suggests, the inter thread communication allows the threads to communicate and pass the signals to

each other and manually control the synchronized threads.

- We can also say that, to avoid pooling of threads into a single method Java allows to control any thread by passing signals to them. This is done by using three **final methods of class Object :**
  1. wait( )
  2. notify( )
  3. notifyAll( )
- These methods are final methods of class Object and therefore any class can never override them. Remember that, the application of these methods is only seen for the synchronized threads and methods. Now, let's discuss about these methods in brief.

1. **wait( ) method**
   - This method causes the invoking thread to wait unless another thread invokes the notify( ) or notifyAll( ) method. The current thread must be in concern this object's monitoring. And to avoid any exceptions it must be called from synchronized method only.
   - This is an overloaded method that has two signatures :

     public final void wait( ) throws InterruptedException

     public final void wait(long timeout) throws InterruptedException
   - The first signature causes the invoking thread to wait unless it is notified. The second signature causes the invoking thread to wait unless its *timeout* time is elapsed.
2. **notify( ) method**

   This method *wakes up a single thread* that is waiting under invoking object's monitoring system.Signature of this method is :

   public final void notify( )
3. **notifyAll( ) method**
   - This method *wakes up all the threads* that are waiting under invoking object's monitoring system. This method is defined as :

     public final void notifyAll( )
   - In case of this method, the thread that has highest priority will be unlocked first and if multiple threads are on equal priority then the thread which is paused/locked first, will be resumed/unlocked first.
   - Many developers do not recognize the difference between sleep( ) and wait( ) methods. By functionalities and signatures the wait( ) and sleep( ) methods looks similar but there are some differences between them.

(c) **Write a short note on MouseListener interface.**
(I marks each method)

The Java MouseListener is notified whenever you change the state of mouse. It is notified against MouseEvent. The MouseListener interface is found in java.awt.event package. It has five methods.

Methods of MouseListener interface

The signature of 5 methods found in MouseListener interface are given below:

1. **public abstract void mouseClicked(MouseEvent e);** Invoked when the mouse button has been clicked (pressed and released) on a component.
2. **public abstract void mouseEntered(MouseEvent e);** Invoked when the mouse enters a component.
3. **public abstract void mouseExited(MouseEvent e);** Invoked when the mouse exits a component
4. **public abstract void mousePressed(MouseEvent e);** Invoked when a mouse button has been pressed on a component.
5. **public abstract void mouseReleased(MouseEvent e);** Invoked when a mouse button has been released on a component.

(d) **Write an application that generates custom exception if any value from its command line arguments is negative.**

```
class CustomExceptionDemo
        {
                public static void main(String ar[ ])
                {
                        int n = Integer.parseInt(ar[0]);
                        System.out.println("main starts");
                        try
                        {
                        System.out.println("In try block");
                        if(n<0)
                        {
                        throw new CustomException("Number is negative");
                        }
                        }catch(CustomException ce)
                        {
                        System.out.println("Custom Exception occurs");
                        }
                        System.out.println("main ends");
                } //end of main
        } //end of class

        public class CustomException extends Exception
        {
                public CustomException(String m)
                {       super(m);               }
        }
```

(e) **What are the major difference between an interface and a class? (1mark for each difference)**

Following are the major differences between an interface and a class.

Class	Interface
In a class, we can only define member functions, we cannot declare them.	In an interface, we can only declare member functions, we cannot define them.
In a class, the data members are by default variable and non-static. (without applying any keyword)	In an interface, the data members are by default final and static. (without applying any keyword)
In a class, static member functions are allowed to define.	Interface does not allow static member functions.
We can create objects of a class.	We cannot create objects of any interface. We can create its references only.
We can define constructor/s in class.	We cannot define any constructor in an interface.
A class cannot extend multiple classes.	An interface can extend multiple interfaces.