T.Y. Diploma : Sem. V
[CO/CM/IF]
Java Programming

Time : 3 Hrs.] Prelim Question Paper Solution **[Marks : 100**

Q.1(a) Attempt any THREE of the following: [12]

Q.1(a) (i) Describe any two relational and any two logical operators in Java with simple example. [4]

Ans.: Relational Operators: When comparison of two quantities is performed depending on their relation, certain decisions are made. Java supports six relational operators as shown in table.

Operators	Meaning
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
= =	Equal to
! =	Not equal to

Program demonstrating Relational Operators

```
class relational
{
public static void main(String args[])
{
    int i=37;
    int j=42;
    System.out.println("i>j"+(i>j)): //false
    System.out.println("I<j"+(i<j)); //true
    System.out.println("i>=j"+(i>=j)); //false
    System.out.println("i<=j"+(I<=j)); // true
    System.out.println("i==j"+(i==j)); // false
    System.out.println("i!=j"+(i!=j)); //true
    }
}
```

Logical Operators: Logical operators are used when we want to form compound conditions by combining two or more relations. Java has three logical operators as shown in table:

Operators	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

Program demonstrating logical Operators

```
class Log
{
    public static void main(String args[])
    {
    boolean A=true;
    boolean B=false;
    System.out.println("A\B"+(a|B)); //true
    System.out.println("A&B"+(A&B)); // false
```

```
        System.out.println("!A"+(!A)); // false
        System.out.println("A^B"+(A^B)); // true
        System.out.println("(A|B)&A"+((A|B)&A)); // true
        }
    }
```

Q.1(a) (ii)What is scope of variable? Give example of class variable, instance variable and local variable. **[4]**

Ans.: The scope of a variable defines the section of the code in which the variable is visible. The variables can be class variable which is associated with the class and is shared with all the instances of the class or an instance variable which is declared in a class and each instance has a separate copy or a local variable which is defined in a block or a method. As a general rule, variables that are defined within a block are not accessible outside that block. The lifetime of a variable refers to how long the variable exists before it is destroyed.

Example:

```
class VariableTypes
{
static int a = 0;       //class variable
int b = 0;              // instance variable
VariableTypes()
{
a++;
b++;
int c = 0;    //local variable
c++;
System.out.println("In constructor printing a: "+a);//will be accessed
System.out.println("In constructor printing b: "+b);//will be accessed
System.out.println("In constructor printing c: "+c);//will be accessed
}
public static void main(String ar[])
{
VariableTypes s = new VariableTypes();
VariableTypes s1 = new VariableTypes();
VariableTypes s2 = new VariableTypes();
System.out.println("in main printing a: "+VariableTypes.a);//will be accessed
System.out.println("in main printing b: "+s.b);//will be accessed
System.out.println("in main printing c "+s.c);//will not be accessed
because this is a local variable declared in constructor
}
}
```

Q.1(a) (iii) Write any two methods of array list class with their syntax. **[4]**

Ans.: **booleanadd (E e):** Appends the specified element to the end of this list

Void add (int index, E element) Inserts the specified element at the specified position in this list.

void clear(): Removes all of the elements from this list

Objectclone():Returns a shallow copy of this ArrayList instance

booleancontains(Object o): Returns true if this list contains the specified element.

Eget(int index): Returns the element at the specified position in this list.

intindexOf(Object o): Returns the index position of the element in the list

booleanisEmpty() :Returns true if the list is empty.

intlastIndexOf(Object o): Returns the index of the last occurrence of the object specified.

booleanisEmpty() :Returns true if the list is empty.

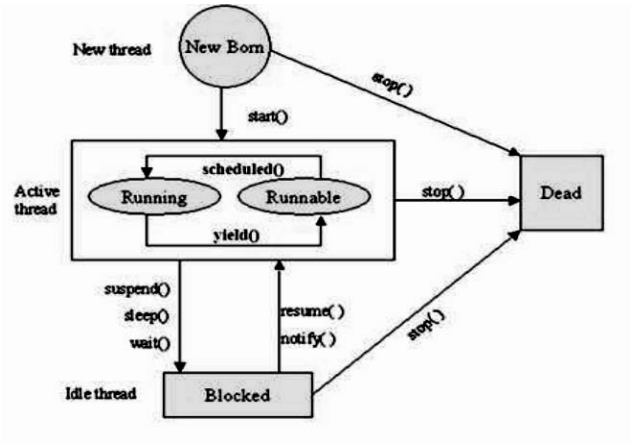
intlastIndexOf(Object o): Returns the index of the last occurrence of the object specified.

boolean remove(Object o): Removes the first occurrence of the object from the list if it is present.

int size( ): returns the number of elements in the list.

Q.1(a)(iv) What is thread? Draw thread life cycle diagram in Java. [4]

Ans.: A thread is a single sequential flow of control within a program. They are lightweight processes that exist within a process. They share the process's resource, including memory and are more efficient. JVM allows an application to have multiple threads of execution running concurrently. Thread has a priority. Threads with higher priority are executed in preference to threads with lower priority.



Q.1(b) Attempt any ONE of the following: [6]

Q.1(b) (i) Create a class 'Rectangle' that contains 'length' and width' as data members. From this class derive class box which has additional data member 'depth'. Class 'Rectangle' consists of a constructor and an area ( ) function. The derived 'Box' class have a constructor and override function named area ( ) which returns surface area of 'Box' and a volume ( ) function. Write a java program calling all the member function. [6]

Ans.:

```
import java.io.*;
class Rectangle
{
int length, width;
Rectangle(int length, int width)
{
this.length = length;
this.width = width;
}
int area()
{
return length*width;
}
}
class Box extends Rectangle
{
int depth;
Box(int length, int width, int depth)
{
super(length, width);
this.depth = depth;
```

```
}
int area()
{
return 2*(length*width+width*depth+depth*length);
}
int volume(int l, int w, int d)
{
return l*w*d;
}
public static void main(String args[])
{
BufferedReader bin = new BufferedReader(new
InputStreamReader(System.in));
Try
{
System.out.println("Enter the length, width and depth");
int l = Integer.parseInt(bin.readLine());
int w = Integer.parseInt(bin.readLine());
int h = Integer.parseInt(bin.readLine());
Box b = new Box(l,w,h);
Rectangle r = new Rectangle(l,w);
System.out.println("Area of the Rectangle is :"+r.area());
System.out.println("Area of the Box is :"+b.area());
System.out.println("volume of the Rectangle is
:"+b.volume(l,w,h));
} catch(Exception e)
{
System.out.println("Exception caught"+e);
}
}
}
```

Q.1(b) (ii) Design an applet which display equals size three rectangle one below the other and fill them with orange, white and green color respectively. **[6]**

Ans.:

```
import java.awt.*;
Importjava.applet.*;
/*
<applet code = DisplayRectangle.class height = 300 width = 300></applet>
*/
public class Display Rectangle extends Applet {
public void init( ) {
setBackground(Color.PINK);
}
Public void paint(Graphics g) {

g.setColor(Color.ORANGE);
g.fillRect(40, 40, 40, 30);
g.setColor(Color.WHITE);
g.fillRect(40, 90, 40, 30);
g.setColor(Color.GREEN);
g.fillRect(40, 140, 40, 30);
}
}
```

OR

```
import java.awt.*;
importjava.applet.*;
/*
<applet code = DisplayRectangle.class height = 300 width = 300></applet>
*/
public class DisplayRectangle extends Applet {
public void paint(Graphics g) {
g.setColor(Color.ORANGE);
g.fillRect(40, 40, 40, 30);
g.setColor(Color.BLACK);
g.drawRect(40, 90, 40, 30);
g.setColor(Color.GREEN);
g.fillRect(40, 140, 40, 30);
}
}
```

Q.2Attempt any TWO of the following: **[16]**

Q.2(a) Write a program to implement a vector class and its method for adding and removing elements. After remove display remaining list. **[8]**

Ans.:

```
import java.io.*;
import java.lang.*;
import java.util.*;
class vector2
{
public static void main(String args[])
{
vector v=new vector();
Integer s1=new Integer(1);
Integer s2=new Integer(2);
String s3=new String("fy");
String s4=new String("sy");
Character s5=new Character('a');
Character s6=new Character('b');
Float s7=new Float(1.1f);
Float s8=new Float(1.2f);
v.addElement(s1);

v.addElement(s2);
v.addElement(s3);
v.addElement(s4);
v.addElement(s5);
v.addElement(s6);
v.addElement(s7);
v.addElement(s8);
System.out.println(v);
    v.removeElement(s2);
    v.removeElementAt(4);
    System.out.println(v);
}
}
```

Q.2(b)What is meant by interface? State its need and write syntax and features of interface. [8]

Ans.: **Interface** is the mechanism by which multiple inheritance is possible in java. It is a reference type in Java. An interface has all the methods undefined. For a java class to inherit the properties of an interface, the interface should be implemented by the child class using the keyword "implements". All the methods of the interface should be defined in the child class.

Example:

```
interface MyInterface{
    int strength=60;
    void method1();
    void method2();
}
public class MyClass implements MyInterface {
    int total;
    MyClass(int t) {
            total = t;
    }
    public void method1() {
            int avg = total/strength;
            System.out.println("Avg is "+avg);
    }
    public void method2() {

    }
    public static void main(String a[]) {
            MyClass c = new MyClass(3600);
            c.method1();
    }
}
```

Need: A java class can only have one super class. Therefore for achieving multiple inheritance, that is in order for a java class to get the properties of two parents, interface is used. Interface defines a set of common behaviours. The classes implement the interface, agree to these behaviours and provide their own implementation to the behaviours.

Syntax:

```
interface InterfaceName {
        int var 1 = value;
        int var2 = value;
    public return_type methodname 1 (parameter_list);
    public return_type methodname 2 (parameter_list);
}
```

Features: Interface is defined using the keyword "interface". Interface is implicitly abstract. All the variables in the interface are by default final and static. All the methods of the interface are implicitly public and are undefined (or implicitly abstract). It is compulsory for the subclass to define all the methods of an interface. If all the methods are not defined then the subclass should be declared as an abstract class.

Q.2 (c)Write an applet program that accepts two input, strings using <Param> tag and concatenate the strings and display it in status window. [8]

Ans.:

```
import java.applet.*;
importjava.awt.*;
/*<applet code = AppletProgram.class height = 400 width = 400>
<param name = "string1" value = "Hello">
<param name = "string2" value = "Applet">
</applet>*/
public class AppletProgram extends Applet
{
String str1;
public void init()
{
str1 = getParameter("string1").concat(getParameter("string2"));
}
public void paint(Graphics g)
{
showStatus(str1);
}
}
```

Q.3Attempt any FOUR of the following: [16]

Q.3(a) Describe the following attributes of applet : [4]

(i) Codebase (ii) Alt (iii) Width (iv) Code

Ans.: **(i) Codebase:** Codebase is an optional attribute that specifies the base URL of the applet code or the directory that will be searched for the applet's executable class file.

(ii) Alt: Alternate Text. The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser cannot run java applets.

(iii)Width: Width is required attributes that give the width (in pixels) of the applet display area.

(iv)Code: Code is a required attribute that give the name of the file containing your applet's compiled class file which will be run by web browser or appletviewer.

Q.3(b) How garbage collection is done in Java? Which methods are used for it? [4]

Ans.:

- Garbage collection is a process in which the memory allocated to objects, which are no longer in use can be freed for further use.
- Garbage collector runs either synchronously when system is out of memory or asynchronously when system is idle.
- In Java it is performed automatically. So it provides better memory management.

Method used for Garbage Collection:

The java.lang.Object.finalize() is called by garbage collector on an object when garbage collection determines that there are no more reference to the object.

A subclass override the finalize method to dispose of system resources or to perform other cleanup.

General Form :

```
protected void finalize()
{    // finalization code here
}
```

Q.3(c) Describe following string class method with example : **[4]**

(i) compareTo( )

(ii) equalsIgnoreCase( )

Ans.: (i) compareTo( )

Syntax: intcompareTo(Object o)

Or

intcompareTo(String anotherString)

There are two variants of this method. First method compares this String to another Object and second method compares two strings lexicographically.

Eg.
```
String str1 = "Strings are immutable";
String str2 = "Strings are immutable";
String str3 = "Integers are not immutable";
    int result = str1.compareTo( str2 );
    System.out.println(result);
    result = str2.compareTo( str3 );
    System.out.println(result);
```

(ii) equalsIgnoreCase( ):

public boolean equalsIgnoreCase(String str)

This method compares the two given strings on the basis of content of the string irrespective of case of the string.

Example:

```
String s1="javatpoint";
String s2="javatpoint";
String s3="JAVATPOINT";
String s4="python";
System.out.println(s1.equalsIgnoreCase(s2));//true because content and d case both are same.
System.out.println(s1.equalsIgnoreCase(s3));//true because case is ign ored.
System.out.println(s1.equalsIgnoreCase(s4));//false because content is not same.
```

Q.3(d) Write any two methods of File and FileInputStream class each. **[4]**

Ans.: File Class Methods

1. **String getName( )** - returns the name of the file.
2. **String getParent( )** - returns the name of the parent directory.
3. **boolean exists( )** - returns true if the file exists, false if it does not.
4. **void deleteOnExit( )** -Removes the file associated with the invoking object when the Java Virtual Machine terminates.
5. **boolean isHidden( )**-Returns true if the invoking file is hidden. Returns false otherwise.

FileInputStream Class Methods:

1. **int available( )** - Returns the number of bytes of input currently available for reading.
2. **void close( )** - Closes the input source. Further read attempts will generate an IOException.
3. **void mark(int numBytes)** -Places a mark at the current point in the inputstream that will remain valid until numBytes bytes are read.
4. **boolean markSupported( )** -Returns true if mark( )/reset( ) are supported by the invoking stream.
5. **int read( )**- Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
6. **int read(byte buffer[ ])**- Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.

Q.3(e) Describe use of 'super' and 'this' with respect to inheritance. [4]

Ans.: Using inheritance, you can create a general class that defines traits common to a set of related items. This class can then be inherited by other, more specific classes, each adding those things that are unique to it. In the terminology of Java, a class that is inherited is called a superclass. The class that does the inheriting is called a subclass. Therefore, a subclass is a specialized version of a superclass.

. Whenever a subclass needs to refer to its immediate superclass, it can do so by use of the keyword**super**.

Superhas two general forms. The first calls the super class constructor. The second is used to access a member of the superclass that has been hidden by a member of a subclass.

super() is used to call base class constructer in derived class.

Super is used to call overridden method of base class or overridden data or evoked the overridden data in derived class.

e.g. use of super( )

```
class BoxWeightextends Box
{
BowWeight(int a ,intb,int c ,int d)
    {
super(a,b,c) // will call base class constructer Box(int a, int b, int c)
weight=d // will assign value to derived class member weight.
    }
```

e.g. use of super.

```
Class Box
{
Box()
    {
    }
void show()
{
    //definition of show
}
} //end of Box class
Class BoxWeight extends Box
{
 BoxWeight()
    {
     }
void show() // method is overridden in derived
{
Super.show() // will call base class method
}
}
```

The this Keyword

Sometimes a method will need to refer to the object that invoked it. To allow this, Java defines the this keyword. this can be used inside any method to refer to the current object. That is, this is always a reference to the object on which the method was invoked. You can use this anywhere a reference to an object of the current class' type is permitted. To better understand what this refersto, consider the following version of Box( ): // A redundant use of this. Box(double w, double h, double d) { this.width = w; this.height = h; this.depth = d; }

Instance Variable Hiding

when a local variable has the same name as an instance variable, the local variable hides the instance variable. This is why width, height, and depth were not used as the names of the parameters to the Box( ) constructor inside the Box class. If they had been, then width would have referred to the formal parameter, hiding the instance variable width.

```
// Use this to resolve name-space collisions.
Box(double width, double height, double depth)
{
this.width = width;
this.height = height;
this.depth = depth;
}
```

Q.4(a) Attempt any THREE of the following: [12]

Q.4(a) (i) What is the use of try catch and finally statement give example. [4]

Ans.: (i) try : Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the **try** block, it is thrown.

Syntax:

```
try
{
// block of code to monitor for errors
    }
```

For eg. Try

```
{
for(inti = 1;i <= 10;i+=2)
{
System.out.println("Odd Thread : "+i);
Thread.sleep(1500);
}
}
```

(ii) catch: Your code can catch this exception (using **catch**) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java runtime system. A catch block immediately follows the try block. The catch block too can have one or more statements that are necessary to process the exception.

Syntax:

```
catch (ExceptionType1 exOb)
 {
// exception handler for ExceptionType1
}
```

For eg.

```
catch (InterruptedExceptione){ }
```

(iii) finally: It can be used to handle an exception which is not caught by any of the previous catch statements. finally block can be used to handle any statement generated by try block. It may be added immediately after try or after last catch block.

Syntax:

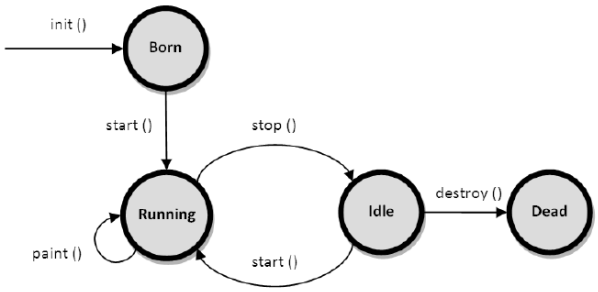
```
    finally
    {
       // block of code to be executed before try block ends
    }
```

For eg.

```
finally{System.out.println("finally block is always executed");}
```

Q.4(a) (ii)Explain applet life cycle with suitable diagram. [4]

Ans.:



Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. The applet states include:

- Born or initialization state
- Running state
- Idle state
- Dead or destroyed state

Initialization state: Applet enters the initialization state when it is first loaded. This is done by calling the init() method of Applet class. At this stage the following can be done:

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

Initialization happens only once in the life time of an applet.

```
public void init()
{
//implementation
}
```

Running state: applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. start() can also be called if the applet is already in idle state. start() may be called more than once. start() method may be overridden to create a thread to control the applet.

```
public void start()
{
  //implementation
}
```

Idle or stopped state: an applet becomes idle when it is stopped from running. Stopping occurs automatically when the user leaves the page containing the currently running applet. stop() method may be overridden to terminate the thread used to run the applet.

```
public void stop()
{
//implementation
}
```

Dead state: an applet is dead when it is removed from memory. This occurs automatically by invoking the destroy method when we quit the browser. Destroying stage occurs only once in the lifetime of an applet. destroy() method may be overridden to clean up resources like threads.

```
public void destroy()
```

```
{
//implementation
}
```

Display state: applet is in the display state when it has to perform some output operations on the screen. This happens after the applet enters the running state. paint() method is called for this. If anything is to be displayed the paint() method is to be overridden. public void paint(Graphics g)

```
{
//implementation
}
```

Q.4(a)(iii) Write a program to generate Fibonacci series 1 1 2 3 5 8 13 21 34 55 89. [4]

Ans.:

```
class FibonocciSeries
{
    public static void main(String args[])
    {
        int num1 = 1,num2=1,ans;
        System.out.println(num1);
        while (num2< 100)
        {
        System.out.println(num2);
        ans = num1+num2;
        num1 = num2;
        num2=ans;
}
    }
}
```

Q.4(a) (iv) State syntax and describe any two methods of map class. [4]

Ans.: The Map Classes Several classes provide implementations of the map interfaces. A map is an object that stores associations between keys and values, or key/value pairs. Given a key, you can find its value. Both keys and values are objects. The keys must be unique, but the values may be duplicated. Some maps can accept a null key and null values, others cannot.

Methods:

void clear // removes all of the mapping from map

booleancontainsKey(Object key) //Returns true if this map contains a mapping for the specified key.

Boolean conainsValue(Object value)// Returns true if this map maps one or more keys to the specified value

Boolean equals(Object o) //Compares the specified object with this map for equality.

Q.4(b) Attempt any ONE of the following: [6]

Q.4(b) (i) What are the access control parameters? Explain the concept with suitable example. [6]

Ans.: Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors.

1) **Default Access Modifier - No keyword:**

   A variable or method declared without any access control modifier is available to any other class in the same package. Visible to all class in its package.

   Example:

   Variables and methods can be declared without any modifiers, as in the following

   Examples:

   String version = "1.5.1";

```
boolean processOrder()
{
    return true;
}
```

2) **Private Access Modifier - private:**
Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.
Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.
Examples:

```
private String format;
private void get() { }
```

3) **Public Access Modifier - public:**
A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe.
However if the public class we are trying to access is in a different package, then the public class still need to be imported.
Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.
Examples:

```
public double pi = 3.14;
public static void main(String[] arguments)
{
    //…
}
```

4) **Protected Access Modifier - protected:**
Variables, methods and constructors which are declared protected in a super class can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.
E.g
The following parent class uses protected access control, to allow its child class override

```
protected void show( )
{ }
```

5) **Private protected:** Variables, methods which are declared protected in a super class can be accessed only by the subclasses in same package. It means visible to class and its subclasses.
Example:

```
Private protected void show( )
{   }
```

Q.4(b) (ii)What is garbage collection in Java? Explain finalize method in Java. [6]

Ans.: Garbage collection is a process in which the memory allocated to objects, which are no longer in use can be freed for further use.

- Garbage collector runs either synchronously when system is out of memory or asynchronously when system is idle.
- In Java it is performed automatically. So it provides better memory management.
- A garbage collector can be invoked explicitly by writing statement System.gc(); //will call garbage collector.

Example:

```
public class A
{
    int p;
    A ( )
    {
    p = 0;
    }
 }
 class Test
 {
    public static void main(String args[])
    {
    A a1= new A();
    A a2= new A();
    a1=a2; // it deallocates the memory of object a1
    }
}
```

Method used for Garbage Collection finalize:

- The java.lang.Object.finalize() is called by garbage collector on an object when garbage collection determines that there are no more reference to the object.
- A subclass override the finalize method to dispose of system resources or to perform other cleanup.
- Inside the finalize() method, the actions that are to be performed before an object is to be destroyed, can be defined. Before an object is freed, the java run-time calls the finalize() method on the object.

General Form :

```
protected void finalize()
{   // finalization code here
}
```

Q.5Attempt any TWO of the following: [16]

Q.5(a) Define an exception called 'No match Exception' that is thrown when a string is not equal to "MSBTE". Write program. [8]

Ans.:

```
//program to create user defined Exception No Match Exception import java.io.*;
class NoMatchException extends Exception
{
NoMatchException(String s)
{
super(s);
}
}
class test1
{
public static void main(String args[]) throws IOException
```

```
{
BufferedReader br= new BufferedReader(new
InputStreamReader(System.in) );
System.out.println("Enter a word:");
String str= br.readLine();
Try
{
if (str.compareTo("MSBTE")!=0) // can be done with equals()
throw new NoMatchException("Strings are not equal");
else
System.out.println("Strings are equal");
}
catch(NoMatchException e)
{
System.out.println(e.getMessage());
}
}
}
```

Q.5(b) Write a program to create two threads, one to print numbers in original order and other in reverse order from 1 to 10. **[8]**

Ans.:

```
class thread1 extends Thread
{
Public void run( )
{
for(int i = 1; i < = 10; i ++)
{
System.out.println("Thread 1;"+ i);
}
}
}
Class thread2 extends Thread
{
Public void run( )
{
for (int i = 10; i > = 1; i --)
{
System.out.println("Thread 2 :" + i);
}
}
}
Class test
{
Public static void main(String args[ ])
{
thread1 t1 = new thread1 ( );
thread2 t2 = new thread2 ( );
t1 .start( );
t2 .start( );
}
}
```

Q.5(c) Explain <applet> tag with its major attributes only. State purpose of get AvailableFontFamilyName( ) method of graphics environment class. [8]

Ans.: The HTML APPLET Tag and attributes

The APPLET tag is used to start an applet from both an HTML document and from an applet viewer.

The syntax for the standard APPLET tag:

<APPLET
[**CODEBASE** = *codebaseURL*]
CODE = *appletFile*
[**ALT** = *alternateText*]
[**NAME** = *appletInstanceName*]
WIDTH = *pixels* **HEIGHT** = *pixels*
[**ALIGN** = *alignment*]
[**VSPACE** = *pixels*] [**HSPACE** = *pixels*] >
[< **PARAM NAME** = *AttributeName***VALUE** = *AttributeValue*>]
[< **PARAM NAME** = *AttributeName2* **VALUE** = *AttributeValue*>]
. . .
</APPLET>

1. **CODEBASE** is an optional attribute that specifies the base URL of the applet code or the directory that will be searched for the applet's executable class file.
2. **CODE** is a required attribute that give the name of the file containing your applet's compiled class file which will be run by web browser or appletviewer.
3. **ALT:** Alternate Text. The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser cannot run java applets.
4. **NAME** is an optional attribute used to specifies a name for the applet instance.
5. **WIDTH AND HEIGHT** are required attributes that give the size(in pixels) of the applet display area.
6. **ALIGN** is an optional attribute that specifies the alignment of the applet.
   The possible value is: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.
7. **VSPACE AND HSPACE** attributes are optional, VSPACE specifies the space, in pixels, about and below the applet. HSPACE VSPACE specifies the space, in pixels, on each side of the applet
8. **PARAM NAME AND VALUE:**
   The PARAM tag allows you to specifies applet- specific arguments in an HTML page applets access there attributes with the get Parameter()method.

Purpose of getAvailableFontFamilyName() method:

It returns an array of String containing the names of all font families in this Graphics Environment localized for the specified locale

Syntax:

public abstract String[] getAvailableFontFamilyNames(Locale l)

Parameters:

1 - a Localeobject that represents a particular geographical, political, or cultural region. Specifying null is equivalent to specifying Locale.getDefault().

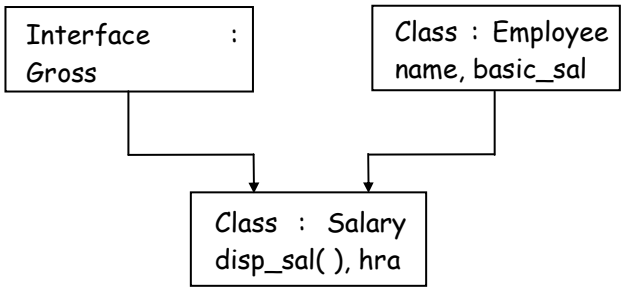
Or

String[] getAvailableFontFamilyNames()

It will return an array of strings that contains the names of the available font families

Q.6Attempt any FOUR of the following: **[16]**

Q.6(a) Write a program to implement following inheritance : **[4]**



Ans.:

```
interface gross
{
int ta=1000;
int da=4000;
public void gross_sal();
}
class employee
{
String name="Abc";
int basic_sal=8000;
}
class salary extends employee implements gross
{
int hra;
int total=0;
salary(int h)
{
hra=h;
}
public void gross_sal()
{
total=basic_sal+ta+da+hra;
}
void disp_sal()
{
gross_sal();
System.out.println("Name :"+name);
System.out.println("Total salary :"+total);
}
```

Q.6(b) What is use of Array list class? State any two methods with their use from ArrayList. **[4]**

Ans.: **Use of ArrayList class:**

1. ArrayListsupports dynamic arrays that can grow as needed.
2. ArrayListis a variable-length array of object references. That is, an ArrayListcan dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

Methods of ArrayList class :

1. **void add(int index, Object element)**
   Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >size()).

2. **boolean add(Object o)**
   Appends the specified element to the end of this list.
3. **booleanaddAll(Collection c)**
   Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.
4. **booleanaddAll(int index, Collection c)**
   Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5. **void clear()**
   Removes all of the elements from this list.
6. **Object clone()**
   Returns a shallow copy of this ArrayList.
7. **boolean contains(Object o)**
   Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
8. **void ensureCapacity(intminCapacity)**
   Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
9. **Object get(int index)**
   Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
10. **intindexOf(Object o)**
    Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
11. **intlastIndexOf(Object o)**
    Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
12. **Object remove(int index)**
    Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size()).
13. **protected void removeRange(intfromIndex, inttoIndex)**
    Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
14. **Object set(int index, Object element)**
    Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
15. **int size()**
    Returns the number of elements in this list.
16. **Object[] toArray()**
    Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.
17. **Object[] toArray(Object[] a)**
    Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
18. **void trimToSize()**
    Trims the capacity of this ArrayList instance to be the list's current size.

Q.6(c) Explain method overloading with example. [4]

Ans.: Method Overloading means to define different methods with the same name but different parameters lists and different definitions. It is used when objects are required to perform

similar task but using different input parameters that may vary either in number or type of arguments. Overloaded methods may have different return types. It is a way of achieving polymorphism in java.

```
int add( int a, int b)              // prototype 1
int add( int a , int b , int c)     // prototype 2
double add( double a, double b)     // prototype 3
```

Example :

```
class Sample
{
    int addition(int i, int j)
    {
        return i + j ;
    }
    String addition(String s1, String s2)
    {
        return s1 + s2;
    }
    double addition(double d1, double d2)
    {
        return d1 + d2;
    }
}
class AddOperation
{
    public static void main(String args[])
    {
        Sample sObj = new Sample();
        System.out.println(sObj.addition(1,2));
        System.out.println(sObj.addition("Hello ","World"));
        System.out.println(sObj.addition(1.5,2.2));
    }
}
```

Q.6(d) Explain serialization in relation with stream classes. [4]

Ans:

- Serialization in java is a mechanism of writing the state of an object into a byte stream.
- Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.
- After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.
- Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.
- The ObjectOutputStream class contains many write methods for writing various data types such as writeObject() method. This method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains method for deserializing an object as readObject(). This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.
- For a class to be serialized successfully, two conditions must be met:
  The class must implement the java.io.Serializable interface.
  All of the fields in the class must be serializable. If a field is not serializable, it must be marked transient.

Q.6(e) What is byte code? Explain any two tools available in JDK. [4]

Ans.: **Byte code:** Bytecode in Java is an intermediate code generated by the compiler such as Sun's javac, that is executed by JVM. Bytecode is compiled format of Java programs it has a .class extension.

Tools	Brief Description
javac	The compiler for the Java programming language.
java	The launcher for Java applications. In this release, a single launcher is used both for development and deployment. The old deployment launcher, jre, is no longer provided.
javadoc	API documentation generator.
appletviewer	Run and debug applets without a web browser.
jar	Manage Java Archive (JAR) files.
jdb	The Java Debugger.
javah	C header and stub generator. Used to write native methods.
javap	Class file disassemble

Q.6(f) Write a java program to copy contents of one file to another file. [4]

Ans.:

```
import java.io.*;
class filecopy
{
public static void main(String args[]) throws IOException
{
FileInputStream in= new FileInputStream("input.txt"); //FileReader
class can be used
FileOutputStream out= new FileOutputStream("output.txt");
//FileWriter class can be used
int c=0;
try
{
while(c!=-1)
{
c=in.read();
out.write(c);
}
System.out.println("File copied to output.txt....");
}
Finally
{
if(in!=null)
in.close();
if(out!=null)
out.close();
}
}
}
```

❑ ❑ ❑ ❑ ❑