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]

(A) 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 Relation 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 operations 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
    Systeam.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 are stream classes? List any two input stream classes from character stream. **[4]**

(A) **Definition:** The java. Io package contain a large number of stream classes that provide capabilities for processing all types of data. These classes may be categorized into two groups based on the data type on which they operate.

1. Byte stream classes that provide support for handling I/O operations on bytes.
2. Character stream classes that provide support for managing I/O operations on characters.

Character Stream Class can be used to read and write 16–bit Unicode characters. There are two kinds of character stream classes, namely, reader stream classes and writer stream classes.

Reader stream classes: It is used to read characters from files. These classes are functionally similar to the input stream classes, except input streams use bytes as their fundamental unit of information while reader streams use characters.

Input Stream Classes

1. BufferedReader
2. CharArrayReader
3. InputStreamReader
4. FileReader
5. PushbackReader
6. FilterReader
7. PipeReader
8. StringReader

Q.1(a) (iii) Explain any four features of Java. **[4]**

(A) **(1) Compile & Interpreted**

Java is a two staged system. It combines both approaches. First java compiler translates source code into byte code instruction. Byte codes are not machine instructions. In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compile and interpreted language.

(2) Platform independent and portable

Java programs are portable i.e. it can be easily moved from one computer system to another. Changes in OS, Processor, system resources won't force any change in java programs. Java compiler generates byte code instructions that can be implemented on any machine as well as the size of primitive data type is machine independent.

(3) Object Oriented

Almost everything in java is in the form of object. All program codes and data reside within objects and classes. Similar to other OOP languages java also has basic OOP properties such as encapsulation, polymorphism, data abstraction, inheritance etc. Java comes with an extensive set of classes (default) in packages.

(4) Robust & Secure

Java is a robust in the sense that it provides many safeguards to ensure reliable codes. Java incorporates concept of exception handling which captures errors and eliminates any risk of crashing the system. Java system not only verify all memory access but also ensure that no viruses are communicated with an applet. It does not use pointers by which you can gain access to memory locations without proper authorization.

(5) Distributed
It is designed as a distributed language for creating applications on network. It has ability to share both data and program. Java application can open and access remote object on internet as easily as they can do in local system.

(6) Multithreaded
It can handle multiple tasks simultaneously. Java makes this possible with the feature of multithreading. This means that we need not wait for the application to finish one task before beginning other.

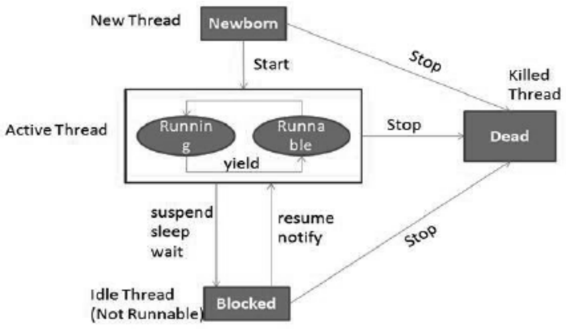
(7) Dynamic and Extensible
Java is capable of dynamically linking new class library's method and object. Java program supports function written in other languages such as C, C++ which are called as native methods. Native methods are linked dynamically at run time.

Q.1(a) (iv) Describe life cycle of thread. **[4]**

(A) Thread Life Cycle Thread has five different states throughout its life.

1. Newborn State
2. Runnable State
3. Running State
4. Blocked State
5. Dead State

Thread should be in any one state of above and it can be move from one state to another by different methods and ways.



1. **Newborn state:** When a thread object is created it is said to be in a new born state. When the thread is in a new born state it is not scheduled running from this state it can be scheduled for running by start() or killed by stop(). If put in a queue it moves to runnable state.
2. **Runnable State:** It means that thread is ready for execution and is waiting for the availability of the processor i.e. the thread has joined the queue and is waiting for execution. If all threads have equal priority then they are given time slots for execution in round robin fashion. The thread that relinquishes control joins the queue at the end and again waits for its turn. A thread can relinquish the control to another before its turn comes by yield().
3. **Running State:** It means that the processor has given its time to the thread for execution. The thread runs until it relinquishes control on its own or it is pre-empted by a higher priority thread.
4. **Blocked state:** A thread can be temporarily suspended or blocked from entering into the runnable and running state by using either of the following thread method.
   suspend() : Thread can be suspended by this method. It can be rescheduled by resume().
   wait(): If a thread requires to wait until some event occurs, it can be done using wait method and can be scheduled to run again by notify().
   sleep(): We can put a thread to sleep for a specified time period using sleep(time) where time is in ms. It reenters the runnable state as soon as period has elapsed /over

5. **Dead State:** Whenever we want to stop a thread form running further we can call its stop(). The statement causes the thread to move to a dead state. A thread will also move to dead state automatically when it reaches to end of the method. The stop method may be used when the premature death is required.

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

Q.1(b) (i) Explain following methods of string class with their syntax and suitable example. [6]
(1) substring ( ) (ii) replace ( )

(A) (1) substring()

Syntax:

String substring(intstartindex)

Startindex specifies the index at which the substring will begin. It will returns a copy of the substring that begins at startindex and runs to the end of the invoking string

String substring(intstartindex,intendindex)

Here startindex specifies the beginning index,andendindex specifies the stopping point. The string returned all the characters from the beginning index, upto, but not including, the ending index.

```
String Str = new String("Welcome");
System.out.println(Str.substring(3)); //come
System.out.println(Str.substring(3,5));//co
```

(2) replace()

This method returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

Syntax:

```
String replace(char original,char replacement)
S2=S1.replace('x','y'); - Replaces all appearance of 'x' with 'y'.
```

Example:

```
String Str = new String("Welcome");
System.out.println(Str.replace('o', 'T')); // WelcTme
```

Q.1(b) (ii) Write a program to create a vector with seven elements as (10, 30, 50, 20, 40, 10, 20). Remove element at 3rd and 4th position. Insert new element at 3rd position. Display the original and current size of the vector. [6]

(A)

```
import java.util.*;
public class VectorDemo
{
    public static void main(String args[])
    {
        Vector v = new Vector();
        v.addElement(new Integer(10));
        v.addElement(new Integer(20));
        v.addElement(new Integer(30));
        v.addElement(new Integer(40));
        v.addElement(new Integer(10));
        v.addElement(new Integer(20));
        System.out println(v.size());// display original size
        v.removeElementAt(2);  // remove 3rd element
        v.removeElementAt(3);   // remove 4th element
        v.insertElementAt(11,2)  // new element inserted at 3rd position
        System.out.println("Size of vector  after  insert delete operations: " + v.size());
    }
}
```

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

Q.2(a) Describe dynamic dispatch method with example. [8]

(A) Dynamic Dispatch Method

Method overriding forms the basis for one of java's most powerful concept dynamic method dispatch. Dynamic method dispatch is the mechanism by which a call to a overridden method is resolved at run time rather than at compile time.

This shows how java implements run time polymorphism. As you know a superclass reference variable can refer to a subclass object. Java uses this fact to resolve calls to overridden method at run time. When a overridden method is called through a superclass reference java determines which method to execute based upon the type of the object being referred at the time the call is made.

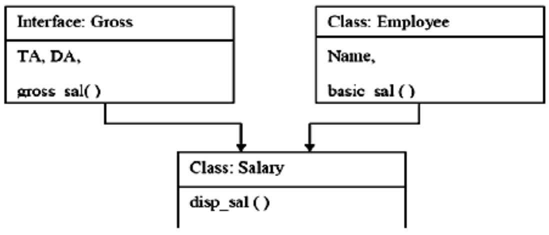
Program

```
Class A
{    Void Show ( )
     {
     System . out . println ("in show of A") ;
     } }
     Class B extends A
          {    Void Show ( )
          {    System . out . println ("in show of B") ;
     } }
     Class C extends B
     {    void show ( )
     {    System . out . println ("in show of C") ;
     } }
          Class DynaMethod
     {    public static void main (String args [ ])
     {    A   a   =   new    A   ( ) ;
          B   b   =   new    B   ( ) ;
          C   c   =   new    C   ( ) ;
          A   r ;
          r = a ;
          r . show ( )
          r = b ;
          r . show ( )
          r = c ;
          r . show ( ) ;
     }
}
```

Q.2(b) Describe with example how to achieve multiple inheritance with suitable program. [8]

(A) Multiple inheritances: It is a type of inheritance where a derived class may have more than one parent class. It is not possible in case of java as you cannot have two classes at the parent level Instead there can be one class and one interface at parent level to achieve multiple interface. Interface is similar to classes but can contain on final variables and abstract method. Interfaces can be implemented to a derived class.

Example:



Code :
```
interface Gross
{
double TA=800.0;
double DA=3500;
void gross_sal();
}
class Employee
{
String name;
double basic_sal;
Employee(String n, double b)
{
name=n;
basic_sal=b;
}
void display()
{
System.out.println("Name of Employee :"+name);
System.out.println("Basic Salary of Employee :"+basic_sal);
}
}
class Salary extends Employee implements Gross
{ double HRA;
Salary(String n, double b, double h)
{
super(n,b);
HRA=h;
}
void disp_sal()
{ display();
System.out.println("HRA of Employee :"+hra);
}
public void gross_sal()
{
double gross_sal=basic_sal + TA + DA + HRA;
System.out.println("Gross salary of Employee :"+gross_sal);
}
}
class EmpDetails
{ public static void main(String args[])
{ Salary s=new Salary("Sachin",8000,3000);
s.disp_sal();
s.gross_sal();
}
}
```

Q.2(c) Write a simple applet program which display three concentric circle. [8]

(A)
```
import java.awt.*;
import java.applet.*;
public class Shapes extends Applet
{
    public void paint (Graphics g)
        {
```

```
        g.drawString("Concentric Circle",120,20);
        g.drawOval(100,100,190,190);
        g.drawOval(115,115,160,160);
        g.drawOval(130,130,130,130);
        }
}
/*<applet code="Shapes.class" height=300 width=200>
</applet>*/
(OR)

HTML Source:
<html>
<applet code="Shapes.class" height=300 width=200>
</applet>
</html>
```

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

Q.3(a) Define constructor. Explain parameterized constructor with example. [4]

(A) Constructor

- A constructor is a special member which initializes an object immediately upon creation.
- It has the same name as class name in which it resides and it is syntactically similar to any method.
- When a constructor is not defined, java executes a default constructor which initializes all numeric members to zero and other types to null or spaces.
- Once defined, constructor is automatically called immediately after the object is created before new operator completes.

Parameterized constructor

When constructor method is defined with parameters inside it, different value sets can be provided to different constructor with the same name.

Example

```
ClassRect
{
int length, breadth;
Rect(int l, int b) // parameterized constructor
{
        length=l;
        breadth=b;
}
public static void main(String args[])
{
        Rect r = new Rect(4,5); // constructor with parameters
        Rect r1 = new Rect(6,7);
        System.out.println("Area : "+(r.length*r.breadth)); // o/p Area : 20
        System.out.println("Area : "+(r1.length*r1.breadth)); // o/p Area : 42
 }
 }
```

Q.3(b) Write a program to check whether an entered number is prime or not. [4]

(A)
```
class PrimeNo
    {
    public static void main(String args[])
            {
    intnum=Integer.parseInt(args[0]);
```

```
        int flag=0;
        for(int i=2;i<num;i++)
                {
                if(num%i==0)
                {
                        System.out.println(num + " is not a prime number");
                        flag=1;
                        break;
                }
    }
    if(flag==0)
    System.out.println(num + " is a prime number");
    }
    }
```

Q.3(c) Explain serialization with stream classes. **[4]**

(A) Serialization is the process of writing the state of an object to a byte stream. This is useful when you want to save the state of your program to a persistent storage area, such as a file. At a later time, you may restore these objects by using the process of deserialization.

Serialization is also needed to implement Remote Method Invocation (RMI). RMI allows a Java object on one machine to invoke a method of a Java object on a different machine. An object may be supplied as an argument to that remote method. The sending machine serializes the object and transmits it. The receiving machine deserializes it.

Example:
Assume that an object to be serialized has references to other objects, which, inturn, have references to still more objects. This set of objects and the relationships among them form a directed graph. There may also be circular references within this object graph. That is, object X may contain a reference to object Y, and object Y may contain a reference back to object X. Objects may also contain references to themselves.

The object serialization and deserialization facilities have been designed to work correctly in these scenarios. If you attempt to serialize an object at the top of an object graph, all of the other referenced objects are recursively located and serialized. Similarly, during the process of deserialization, all of these objects and their references are correctly restored.

Q.3(d) State syntax and explain it with parameters for : **[4]**
(i) drawRect ( ) **(ii) drawOval ( )**

(A) **(i) drawRect( )**
The drawRect( ) method display an outlined rectangle.

Syntax:

```
voiddrawRect(inttop,intleft,intwidth,int height)
```

This method takes four arguments, the first two represents the x and y co-ordinates of the top left corner of the rectangle and the remaining two represent the width and height of rectangle.
Example:

```
g.drawRect(10,10,60,50);
```

(ii) drawOval( )
To draw an Ellipses or circles used drawOval()method.

Syntax:

```
voiddrawOval(int top, int left, int width, int height)
```

The ellipse is drawn within a bounding rectangle whose upper-left corner is specified by top and left and whose width and height are specified by width and height to draw a circle or filled circle, specify the same width and height the following program draws several ellipses and circle.

Example:

```
g.drawOval(10,10,50,50);//this is circle
g.drawOval(10,10,120,50);//this is oval
```

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

(A) 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.

Super has 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 refers to, 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) Explain break and continue statements with example. [4]

(A) Break:

The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Example:

```
public class Test
{
  public static void main(String args[])
     {
  int [] numbers = {10, 20, 30, 40, 50};
  for(int x : numbers ) {
  if( x == 30 ) {
          break;
     }
  System.out.print( x );
  System.out.print("\n");
          }
     }
}
```

Continue:

The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

A labeled continue statement skips the current iteration of an outer loop marked with the given label.

Example:
```
public class Test {
public static void main(String args[]) {
int [] numbers = {10, 20, 30, 40, 50};

for(int x : numbers ) {
if( x == 30 ) {
        continue;
    }
System.out.print( x );
System.out.print("\n");
    }
  }
}
```

Q.4(a) (ii) Describe use of 'throws' with suitable example. [4]

(A) Throws clause

If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a „throws" clause in the methods declaration. A throws clause lists the types of exception that a method might throw. General form of method declaration that includes 'throws' clause.

Type method-name (parameter list) throws exception list {// body of method} Here exception list can be separated by a comma.

Eg.
```
class XYZ
    {
XYZ();
{
        // Constructer definition
    }
            void show() throws Exception
        {
            throw new Exception()
            }
            }
```

Q.4(a) (iii) State syntax and describe working of 'for each' version of for loop with one example. [4]

(A) The For-Each Alternative to Iterators

If you won't be modifying the contents of a collection or obtaining elements in reverse order, then the for-each version of the for loop is often a more convenient alternative to cycling through a collection than is using an iterator. Recall that the for can cycle through any collection of objects that implement the Iterable interface. Because all of the collection classes implement this interface, they can all be operated upon by the for.

// Use the for-each for loop to cycle through a collection.

// Use a for-each style for loop.

Syntax: for(data_type variable : array | collection)
```
          { }
```

Example
```
ClassForEach
{
public static void main(String args[])
{
```

```
            intnums[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
            int sum = 0; // use for-each style for to display and sum the values
            for(int x : nums)
            {
                System.out.println("Value is: " + x);
                sum += x;
            }
            System.out.println("Summation: " + sum);
    }
    }
```

Q.4(a) (iv) Define JDK. List the tools available in JDK explain any one in detail. [4]

(A) Definition

A Java Development Kit (JDK) is a collection of tools which are used for developing, designing, debugging, executing and running java programs.

Tools of JDK

java
javap
javah
javadoc
jdb
appletviewer
javac

1. **Java Compiler** - it is used to translate java source code to byte code files that the interpreter can understand.

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

Q.4(b) (i) Write method to set font of a text and describe its parameters. [6]

(A) The AWT supports multiple type fonts emerged from the domain of traditional type setting to become an important part of computer-generated documents and displays. The AWT provides flexibility by abstracting font-manipulation operations and allowing for dynamic selection of fonts.

Fonts have a family name, a logical font name, and a face name. The family name is the general name of the font, such as Courier. The logical name specifies a category of font, such as Monospaced. The face name specifies a specific font, such as Courier Italic

To select a new font, you must first construct a Font object that describes that font. One Font constructor has this general form:

    Font(String fontName, intfontStyle, intpointSize)

To use a font that you have created, you must select it using setFont( ), which is defined by Component.
It has this general form:
VoidsetFont(Font fontObj)
Example

```
importjava.applet.*;
importjava.awt.*;
importjava.awt.event.*;
public class SampleFonts extends Applet
 {
int next = 0;
Font f;
String msg;
```

```
public void init()
{
f = new Font("Dialog", Font.PLAIN, 12);
msg = "Dialog";
setFont(f);
public void paint(Graphics g)
{
g.drawString(msg, 4, 20);
}
}
```

Q.4(b) (ii) Describe final method and final variable with respect to inheritance. [4]

(A) **final method:** making a method final ensures that the functionality defined in this method will never be altered in any way, ie a final method cannot be overridden.

E.g.of declaring a final method:

```
        final void findAverage() {
            //implementation
        }
        EXAMPLE
class A
{       final void show()
    {
    System.out.println("in show of A");
    }
}
class B extends A
{
    void show() // can not override because it is declared with final
    {
    System.out.println("in show of B");
    }
}
```

final variable: the value of a final variable cannot be changed. final variable behaves like class variables and they do not take any space on individual objects of the class.

E.g.of declaring final variable:

final int size = 100;

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

Q.5(a) What is thread priority? How thread priority are set and changed? Explain with example. [8]

(A) Thread Priority

Threads in java are sub programs of main application program and share the same memory space. They are known as light weight threads. A java program requires at least one thread called as main thread. The main thread is actually the main method module which is designed to create and start other threads in java each thread is assigned a priority which affects the order in which it is scheduled for running. Thread priority is used to decide when to switch from one running thread to another. Threads of same priority are given equal treatment by the java scheduler. Thread priorities can take value from 1-10. Thread class defines default priority constant values as

MIN_PRIORITY = 1
NORMPRIORITY = 5 (Default Priority)
MAXPRIORITY = 10

1. setPriority:
   Syntax:public void setPriority(int number);
   This method is used to assign new priority to the thread.
2. getPriority:
   Syntax:public intgetPriority();
   It obtain the priority of the thread and returns integer value.
   Example:

```
class clicker implements Runnable
{
int click = 0;
Thread t;
private volatile boolean running = true;
public clicker(int p)
{
    t = new Thread(this);
    t.setPriority(p);
}
public void run()
{
while (running)
{
        click++;
}
}
public void stop()
{
        running = false;
}
public void start()
{
        t.start();
}
}
class HiLoPri
{
        public static void main(String args[])
{
            Thread.currentThread().setPriority(Thread.MAX_PRIORITY);
            clicker hi = new clicker(Thread.NORM_PRIORITY + 2);
            clicker lo = new clicker(Thread.NORM_PRIORITY - 2);
            lo.start();
            hi.start();
try {
        Thread.sleep(10000);
}
catch (InterruptedException e)
{
        System.out.println("Main thread interrupted.");
}
lo.stop();
hi.stop();
// Wait for child threads to terminate.
try
{
```

```
                hi.t.join();
                lo.t.join();
        }
        catch (InterruptedException e) {
                System.out.println("InterruptedException caught");
        }
                System.out.println("Low-priority thread: " + lo.click);
                System.out.println("High-priority thread: " + hi.click);
        }
        }
```

Q.5(b) Write a program to input name and age of person and throws user defined exception, if entered age is negative. [8]

(A)

```
import java.io.*;
class Negative extends Exception
{
Negative(String msg)
{
super(msg);
}
        }
class Negativedemo
{
        public static void main(String ar[])
        {
            int age=0;
            String name;
        BufferedReaderbr=new BufferedReader (newInputStreamReader(System.in));
        System.out.println("enter age and name of person");
try
{
age=Integer.parseInt(br.readLine());
name=br.readLine();
{
if(age<0)
throw new Negative("age is negative");
else
throw new Negative("age is positive");
        }
}
catch(Negative n)
{
        System.out.println(n);
}
catch(Exception e)
{
 }
```

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

(A) 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 = AttributeNameVALUE = 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:
l - 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.6 Attempt any FOUR of the following: [16]

Q.6(a) Write a program to search a number in an array. [4]

(A)
```
class Search {
    public static void main (String args [ ]) throws IOException
    {
    BufferedReader br = new BufferedReader (new InputStream Reader (System.in));
    int a [ ] = {1, 2, 3, 4, 5, 6, 7, 8, 9 ,10};
    int k Flag = 0;
```

```
        System.out.println (“Enter the number to be searched in arrary”);
        try {
            k = Integer.parse Int(br.readLine ( ));
        } Catch (Number Format Exception e) {
            System.out.println (“Invalid format”);
        }
        for (i = 0;   i < 10;   i ++)
        {
            if (a[i] = = k)
            {
                flag = 1;
                break ;
            }
        }
        if (flag = = 1)
            {
        System.out.println (“The no” + k + “is at” + i + 1 + “position”);
            }
        else
            { System.out.println (“No not Round”);
            }
        }
    }
```

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

(A) 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) Design an Applet program which displays a rectangle filled with red color and message as "Hello Third year Students" in blue color. **[4]**

(A)
```
import java.awt.*; import java.applet.*;
public class DrawRectangle extends Applet
{
    public void paint(Graphics g)
{
g.setColor(Color.red);
g.fillRect(10,60,40,30);
g.setColor(Color.blue);
g.drawString("Hello Third year Students",70,100);
}
}
/*
<applet code="DrawRectangle.class" width="350" height="300">
</applet> */
```

Q.6(d) Design a package containing a class which defines a method to find area of rectangle. Import it in Java application to calculate area of rectangle. **[4]**

(A)
```
package Area;
public class Rectangle
{
    doublelength,bredth;
    public doublerect(float l,float b)
{
        length=l;
        bredth=b;
        return(length*bredth);
}
}
import Area.Rectangle;
class RectArea
{
public static void main(String args[])
{
Rectangle r=new Rectangle( );
double area=r.rect(10,5);
System.out.println("Area of rectangle= "+area);
}
}
```

Q.6(e) Define a class having one 3-digit number as a data member. Initialize and display reverse of that number. **[4]**

(A)
```
class reverse
{
public static void main(String args[])
{
intnum = 253;
intremainder,
result=0; while(num>0)
{
remainder = num%10;
result = result * 10 + remainder;
num = num/10;
}
System.out.println("Reverse number is : "+result);
}
}
```

❑ ❑ ❑ ❑ ❑