Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills).
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgment on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

1. A) Attempt any 3: **Marks 12**

a) State any four features of Java.

(Any four feature- 1 Mark each)

Ans: **1.Compiled & 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. 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

b) What is Constructor? Explain constructor overloading with example.

(Constructor Definition. -1 Mark, Constructor overloading- 1 Mark, Example- 2 Marks)

Ans: **Constructor:**

Constructor is a special type of method that is used to initialize the object. Constructor is invoked at the time of object creation. It provides data for the object that is why it is known as constructor. It has same name as class name.

Constructor overloading:

Overloading is having more than one form. Overloading refers to the use of the same thing for different purpose. (**Constructor overloading means to define a various constructor but with different signatures**)

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

Example:

```
Class Sample
{
        int x;
        Sample()
        {
                X=0;
        }
        Sample(int a)
        {
                X=a;
        }
        Sample(Sample s)
        {
                X=s.x;
        }
}
```

c) State how to create user defined package and accessed in Java.

(Create package- 1 Mark, Access- 1 Mark, any one Example- 2 Marks)

Ans: **To create** a package is quite easy: simply include a **package** command as the first statement in a Java source file. Any classes declared within that file will belong to the specified package. The **package** statement defines a name space in which classes are stored. If you omit the **package** statement, the class names are put into the default package, which has no name. Java is uses file system directories to store packages. This directory name must match with the package name exactly.

Syntax:

package *pkg*;

Here, *pkg* is the name of the package.

To access package In a Java source file, **import** statements occur immediately following the **package** statement (if it exists) and before any class definitions.

Syntax:

import *pkg1*[.*pkg2*].(*classname*|*);

Example:

```
package MyPack;
public class Balance
{
         String name;
        double bal;
        public Balance(String n, double b)
        {
        name = n;
        bal = b;
         }
        public void show()
        {
```

```
        if(bal<0)
        System.out.print("--> ");
        System.out.println(name + ": $" + bal);
        }
}
import MyPack.*;
class TestBalance
{
        public static void main(String args[])
        {
         Balance test = new Balance("ABCDE", 199.88);
        test.show(); // you may also call show()
        }
}
```

d) What is thread? Describe the complete life cycle of 'thread'.

(Thread Definition- 1 Mark, Thread Life cycle Explanation- 3 Marks)

(Note: Diagram not necessary)

Ans: **Thread:** thread is the smallest unit of dispatch able code. It similar to a program that has a single flow of control. A thread is known as an execution context or a lightweight process. Every thread has a beginning, a body and an end. Thread itself is not a program, but runs within a program.

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 andcan 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.

B) Attempt any one: **Marks 06**

a) Write an applet to accept a user name in the form of parameter and print ‘Hello<User Name>’.

(Correct logic-4 Marks, for syntax-2 Mark)

Ans:
```
import java.lang.*;
import java.awt.*;
import java.applet.*;
public class Hello extends Applet
```

```
{
String message;
        public void init()
        {
                message=getParameter("username");
                message="Hello" + message;
        }
        public void paint(Graphics g)
        {
                g.drawString(message, 100,100);
        }
}
/*<applet code=Hello.class height=300 width=300>
<param name="username" value="Abc">
</applet>
*/
```

b) Write a program to implement the Fibonacci series using loop control.

(Correct logic -4 Marks, for syntax -2 Marks)

Ans:

```
class Fibonacci
{
public static void main(String args[])
{
        int prev, next, sum, n;
        prev=next=1;
        for(n=1;n<=10;n++)
        {
                System.out.println(prev);
                sum=prev+next;
                prev=next;
                next=sum;
        }
}
}
```

(Note: Any relevant logic can be considered)

2. Attempt any two: **Marks 16**

a) Write a program to create two thread one thread to print numbers in original order and other in reverse order from 1 to 50.

(Correct logic-5 Marks, for syntax- 3 Marks)

Ans:

```
class FirstThread extends Thread
{
 public void run()
        {
                for (int i=1; i<=50; i++)
                 {
                System.out.println("Message from First Thread : " +i);
                }
        }
  }
  class SecondThread extends Thread
        {
                public void run()
                {
                        for (int i=50; i>0; i--)
                        {
                        System.out.println( "Message from Second Thread : " +i);
                        }
                }
        }
        public class ThreadDemo
         {
        public static void main(String args[])
        {
        FirstThread firstThread = new FirstThread();
        SecondThread secondThread = new SecondThread();
        firstThread.start();
        secondThread.start();
        }
        }
```

b) Define a class employee with data members as emp_id, name and salary accept data for 5 objects and print it.
(Class declaration-2 Marks, Accept data- 2 Marks, Array of object- 2 Marks, Display data- 2 Marks)

Ans:

```
import java.io.*;
class employee
{
int empid;
String name;
double salary;
void getdata()
{
        BufferedReader obj = new BufferedReader (new InputStreamReader(System.in));
        System.out.print("Enter Emp number : ");
        empid=Integer.parseInt(obj.readLine());
        System.out.print("Enter Emp Name : ");
        name=obj.readLine();
        System.out.print("Enter Emp Salary : ");
        salary=Double.parseDouble(obj.readLine());
}
void show()
{
        System.out.println("Emp ID :  " + empid);
           System.out.println(“Name :  " + name);
           System.out.println(“Salary :  " + salary);
 }
}
classEmpDetails
{
           public static void main(String args[])
           {
                employee e[] = new employee[5];
                for(int i=0; i<5; i++)
                {
                e[i] = new employee();
                e[i].getdata();
                }
                System.out.println(" Employee Details are : ");
                for(int i=0; i<5; i++)
                e[i].show();
            }
    }
```

(Note: Any relevant logic can be considered)

c) Write syntax and example of

1) drawString( ) 2) drawRect( ) 3) drawOval( ) 4) drawArc( )

(Each Method syntax-1 Mark, Example-1 Mark)

Ans: 1) drawString( )

Displaying String:

drawString() method is used to display the string in an applet window

Syntax:

void drawString(String message, int x, int y);

where message is the string to be displayed beginning at x, y

Example:

g.drawString("WELCOME", 10, 10);

2) drawRect( )

Drawing Rectangle:

The drawRect() method displays an outlined rectangle. The general forms of this method is

void drawRect(int top,int left,int width,int height)

The upper-left corner of the Rectangle is at top and left. The dimension of the Rectangle is specified by width and height.

Example:

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

3) drawOval( )

Drawing Ellipses and circles:

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

Syntax:

void drawOval(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 specify the same width and height.

Example:

g.drawOval(10,10,70,50); // draws ellipse
g.drawOval(100,10,50,50); // draws circle

4) drawArc( )

Drawing Arc:

It is used to draw arc.

Syntax:

void drawArc(int x, int y, int w, int h, int start_angle, int sweep_angle);
where x, y starting point, w& h are width and height of arc, and
start_angle is starting angle of arc
sweep_angle is degree around the arc.

Example:

g.drawArc(10, 10, 30, 40, 40, 90);

3. Attempt any 4: **Marks 16**

a) Explain Bit-wise operator with example.

(Any 4 (Each Bitwise operator explanation- 1/2 Mark, Example-1/2 Mark))

Ans: **1) Bitwise NOT (~):** called bitwise complement, the unary NOT operator, inverts all of the bits of its operand.

e.g ~ 0111 (decimal 7)
= 1000 (decimal 8)

2) Bitwise AND (&): the AND operator, &, produce a 1 bit if both operands are also 1, A zero is produced in all the cases.

e.g 0101 (decimal 5)
&0011 (decimal 3)
= 0001 (decimal 1)

3) Bitwise OR(|) : the OR operator, | , combines bits such that if either of the bits in the operand is a 1, then the resultant bit is a 1

```
e.g        0101 (decimal 5)
          |0011 (decimal 3)
         = 0111 (decimal 7)
```

4) Bitwise XOR(^) : the XOR operator , ^ , combines bits such that if exactly one operand is 1,then the result is 1. Otherwise result is zero.

```
e.g        0101 (decimal 5)
         ^ 0011 (decimal 3)
        = 0110 (decimal 6)
```

5) The Left Shift (<<): the left shift operator, <<, shifts all of the bits in a 'value' to the left a specified number of times specified by 'num'

General form : value <<num

```
e.g.       x << 2    (x=12)
           0000 1100  << 2
         = 0011 0000 (decimal 48)
```

6) The Right Shift (>>): the right shift operator, >>, shifts all of the bits in a 'value' to the right a specified number of times specified by 'num' General form: value >>num.

```
e.g.       x>> 2      (x=32)
           0010 0000 >> 2
         = 0000 1000 (decimal 8)
```

7) Unsigned Right Shift (>>>) : >>> always shifts zeros into high order bit.

```
e.g.       int a= -1
           a=a>>>24
11111111    11111111    11111111    11111111 (-1 in binary as int) >>> 24
00000000    00000000    00000000    11111111(255 in binary as an int)
```

8) Bitwise Operators Compound Assignments: All of the above binary bitwise operators have a compound form similar to that of the algebraic operators, which combines the assignment with the bitwise operation. These two statements are equivalent.

```
e.g.       a = a >> 4 ;
           a >> = 4;
```

b) Write a program to copy contents of one file to another file using character stream class.

(Correct logic -3 Marks, for syntax -1 Mark)

Ans:

```
import java.io.*;
public class CopyChar {
public static void main(String args[])
  {
        //Declare and Create input and output file
        File inFile=new File("input.txt");
        File outFile=new File("output.txt");
        FileReader ins = null;
        FileWriter outs = null;
        try
        {
        ins = new FileReader(inFile); //opens input file
        outs = new FileWriter(outFile); //opens output file
        int c;
        while ((c = ins.read()) != -1)
        {                 //Read and Write till end
              outs.write(c);
        }
        }
        catch(FileNotFoundException e)
        {
                System.out.println("File not found");
        }
        catch(IOException e)
        {
                System.out.println(e);
                System.exit(-1);
        }
        finally {  //closes File
         try {
           ins.close();
           outs.close();
        }
        catch(IOException e)
        {
        }
    }
  }}
```

c) Write a program to accept a password from user and throw "Authentication failure" exception if the password is incorrect.

(Correct logic-3 Marks, for syntax-1 Mark)

Ans:

```
import java.io.*;
class PasswordException extends Exception
{
        PasswordException(String msg)
        {
        super(msg);
        }
}
class  PassCheck
{
        public static void main(String args[])
        {
        BufferedReader bin=new BufferedReader(new InputStreamReader(System.in));
        try
        {
                System.out.println("Enter Password : ");
                String pass= bin.readLine();
                if(pass.equals("MSBTE"))
                {
                        System.out.println("Authenticated ");
                }
                else
                {
                        throw new PasswordException("Authentication failure");
                }
        }
        catch(PasswordException e)
        {
                System.out.println(e);
        }
        catch(IOException e)
        {
                System.out.println(e);
        }
 } }
```

d) Explain method overloading with example.

(Define Method overloading-1 Marks, Any relevant Example-3 Marks)

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));
    }
}
```

e) List any 4 built in packages from Java AP1 along with their use.

(Any 4 four, for each listing and use- 1 Mark)

Ans:

1. **java.lang** - language support classes. These are classes that java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
2. **java.util** – language utility classes such as vectors, hash tables, random numbers, date etc.
3. **java.io** – input/output support classes. They provide facilities for the input and output of data
4. **java.awt** – set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
5. **java.net** – classes for networking. They include classes for communicating with local computers as well as with internet servers.
6. **java.applet** – classes for creating and implementing applets.

4. A) Attempt any 3: Marks 12

a) Explain following method of vector class

1) elementAt ( ) 2) removeElement ( )

3) insertElementAt ( ) 4) addElement ( )

(Explanation of each method with syntax - 1 Mark)

Ans: 1) elementAt( ) : Returns the element at the location specified by index.

Syntax: Object elementAt(int index)

Example: Vector v = new Vector( );

v.elementAt(3); //return 3rd element from vector

2) removeElement( ) : Removes element from the vector. If more than one instance of the specified object exists in the vector, and then it is the first one that is removed. Returns **true** if successful and **false** if the object is not found.

Syntax: boolean removeElement(Object element)

Example: Vector v = new Vector( );

v.removeElement(2.45); //remove object from vector

3) insertElementAt( ) : Adds element to the vector at the location specified by the index.

Syntax : void insertElementAt(Object element, int index)

Example: Vector v = new Vector( );

v.insertElementAt("J", 2); //insert character object in vector at 2nd position.

4) **addElement ( ) :** The object specified by element is added to the vector.

Syntax: void addElement(Object element)

Example: Vector v = new Vector();

v.addElement(new Integer(5)); //add integer object 5 to vector

v.addElement("JAVA"); //add string object to vector

b) Explain type casing with suitable example.

(Type casting each type -2 Marks (Explanation- 1 Mark, Example-1Mark))

Ans: The process of converting one data type to another is called casting. Casting is necessary when method return a type different than the one we require. A boolean value cannot be assigned to any other data type.

There are 2 types of type casting:

1) **Implicit Conversion (Widening or Automatic type conversion):** Conversion from one primitive datatype to another datatype that has more bits. Widening type conversion will occur only if following two conditions are met.

- The two types are compatible.
- The destination type is larger than the source type.

e.g int x = 10; // occupies 4 bytes

double y = x; // occupies 8 bytes, no type casting required

System.out.println(y); // prints 10.0

2) **Explicit Conversion (Narrowing conversion):** Conversion from one primitive data type to another datatype that has fewer bits. Can result into loss of data. floating-point values are truncated when they are cast to integers.

double x = 10.5; // 8 bytes

int y = x; // 4 bytes ; raises compilation error "loss of precision".

Syntax :

Type variable1 = (type) variable2 ;

double x=10.5;

int y=(int)x;

System.out.println(x); //10.5

System.out.println(y): //10

c) What is thread priority? Write default priority values and methods to change them.

(Thread Priority explanation- 1 Mark, Default priority values- 1Mark, Each method- 1 Mark)

Ans: **Thread Priority:** 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

NORM_PRIORITY = 5 (Default Priority)

MAX_PRIORITY = 10

1. **setPriority:**

   Syntax: public void setPriority(int number);

   This method is used to assign new priority to the thread.

2. **getPriority:**

   Syntax: public int getPriority();

   It obtain the priority of the thread and returns integer value.

d) Design an applet which display a triangle filled with red colour and a message as "The triangle" in blue below it.

(Correct logic- 3 Marks, for syntax-1 Mark)

Ans:

```
import java.awt.*;
import java.applet.*;
public class DrawTriangle extends Applet
{
public void paint(Graphics g)
{
   int xPoints[] = { 10, 170, 80, 10 };
   int yPoints[] = { 20, 40, 140, 20 };
   int nPoints = xPoints.length;
   g.set Color(Color.red);
   g.fillPolygon(xPoints, yPoints, nPoints);
   g.setColor(Color.blue);
   g.drawString("The triangle",50,160);
}
}
/*
<applet code="DrawTriangle.class" width="350" height="300">
</applet> */
```

(Note: Any relevant logic can be considered)

B) Attempt any one: **Marks 06**

a) Write all primitive data types available in Java with their storage class.

(*List all data types- 2 Marks, 1/2 mark to each data type storage size - 8 × ½=4 Marks)*

Ans: Java defines eight primitive datatypes as:

byte	float
short	double
long	boolean
char	int

These can be classified into four groups as:

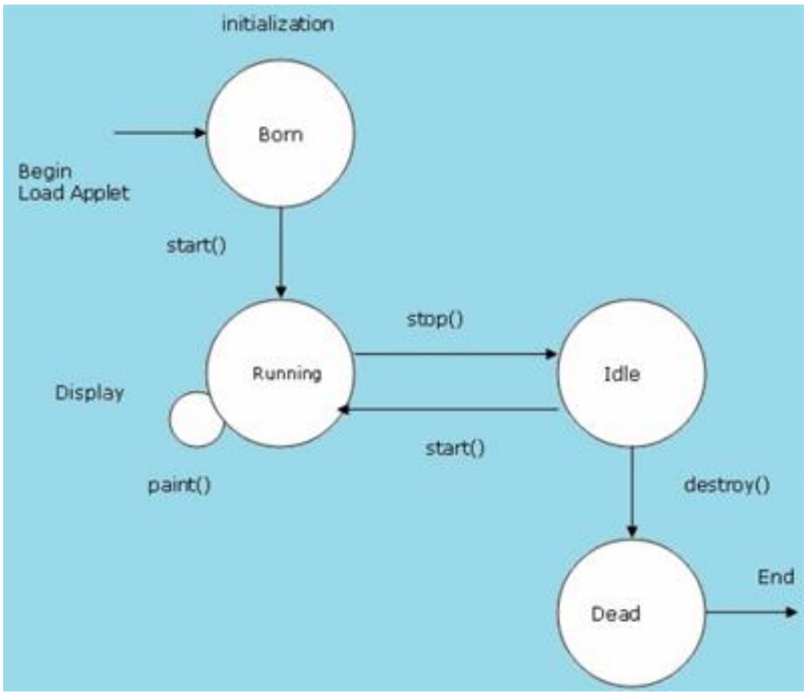
Storage sizes

Sr. No	Type	Keyword	Size (bit)
1	long integer short integer Integer Byte	long short int byte	64 16 32 8
2	Double Float	double float	64 32
3	Character	char	16
4	Boolean	boolean	8

b) Explain life cycle of applet with diagram.

(Any Relevant Correct Block Diagram- 2 Mark, Explanation- 4 Marks)

Ans:



Applet Life cycle has following states:

Initialization State

Applet enters the initialization state when it is first loaded. This achieved by calling **init( )** method of applet. Now the applet is born. This is where you should initialize variables, create object needed by applet, load images and font. This method is called only once during the run time of your applet.

Running State( )

Applet enters the running state when the system calls the **start( )** method of applet. This occurs automatically after applet initialized. It is also called to restart an applet after it has been stopped. Whereas **init( )** is called once—the first time an applet is load **start( )** is called each time an applet's HTML document is displayed onscreen. So, if a user leaves a web page and comes back, the applet resumes execution at **start( )**.

Idle State

Applet becomes idle when it is stopped from running. The **stop( )** method is called when a web browser leaves the HTML document containing the applet—when it goes to another page, for example. When **stop( )** is called, the applet is probably running. You should use **stop( )** to suspend threads that don't need to run when the applet is not visible. You can restart them when **start( )** is called if the user returns to the page.

Dead State

The applet is said to be dead when it is removed from memory. This occurs automatically by invoking **destroy( )** method when you quit the browser.

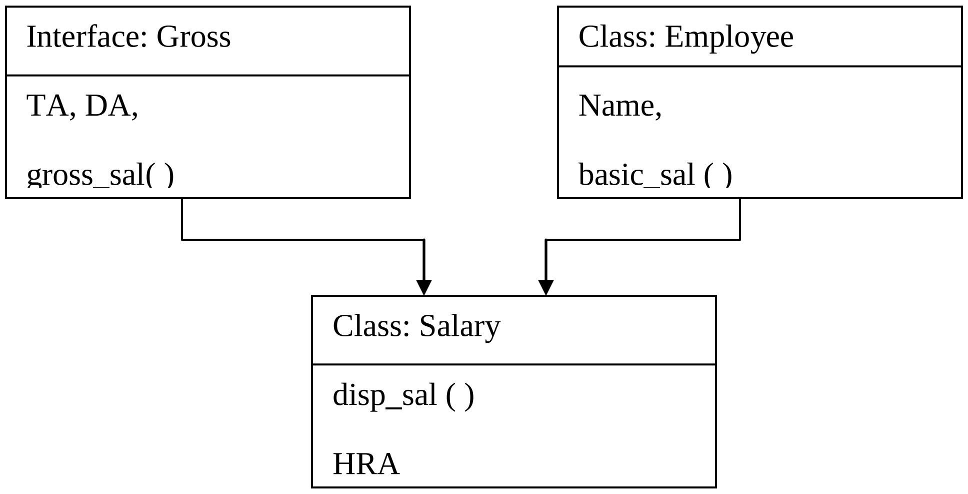
At this point, you should free up any resources the applet may be using. The **stop( )** method is always called before **destroy( )**. Like init() destroying stage occurs only once in life cycle.

Display State

The **paint( )** method is called each time your applet's output must be redrawn. This situation can occur for several reasons. For example, the window in which the applet is running may be overwritten by another window and then uncovered. Or the applet window may be minimized and then restored. **paint( )** is also called when the applet begins execution. Whatever the cause, whenever the applet must redraw its output, **paint( )** is called. The **paint( )** method has one parameter of type **Graphics**. This parameter will contain the graphics context, which describes the graphics environment in which the applet is running. This context is used whenever output to the applet is required.

5. Attempt any two: Marks 16

1) Write a program to make following inheritance:



(Correct logic- 5 Marks, for syntax-3 Marks)

Ans:

```
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();
                }
 }
```

(Note: Any relevant logic can be considered)

2) Explain following terms with respect to exception handling

1) try 2) catch 3) throw 4) throws 5) finally

(try, catch, finally – Explanation with syntax – 2 Marks each ;

throw, throws – Explanation – 1 Mark each)

Ans: Java exception handling is managed via five keywords: **try**, **catch**, **throw**, **throws**, and **finally**.

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.

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.

finally - Any code that absolutely must be executed before a method returns is put in a **finally** block. A try block must have at least one catch block or a finally block.\

Syntax:

```
try {
// block of code to monitor for errors
}
catch (ExceptionType1 exOb) {
// exception handler for ExceptionType1
}
catch (ExceptionType2 exOb) {
// exception handler for ExceptionType2
}
// ...
finally {
// block of code to be executed before try block ends
}
```

throw

It is possible for a program to throw an exception explicitly, using the throw statement. The general form of throw is :

throw new *ThrowableInstance*;

or

throw Throwableinstance;

throw statement explicitly throws an built-in /user- defined exception.

When throw statement is executed, the flow of execution stops immediately after throw statement, and any subsequent statements are not executed.

throws

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 method's declaration. A throws clause lists the types of exceptions that a method might throw. This is necessary for all exceptions, except those of type Error or Runtime Exception, or any of their subclasses. All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result.

This is the general form of a method declaration that includes a throws clause:

type method-name(parameter-list)* throws *exception-list

{

// body of method

}

Here, *exception-list* is a comma-separated list of the exceptions that a method can throw.

3) **Write a program to accept as many integers as user wants in vector. Delete a specific element and display remaining list.**

(Accept vector elements - 2 Marks, remove element – 1 Mark, Display element – 2 Marks , for other logic & syntax-3 Marks)

Ans:

```
import java.util.*;
import java.io.*;
public class VectorDemo
{
  public static void main(String args[]) throws IOException
  {
    Vector v = new Vector();
    BufferedReader obj = new BufferedReader (new InputStreamReader(System.in));
     System.out.println("Enter number of elements u want to add :");
      int n= Integer.parseInt(obj.readLine());
```

```
        for( int i=0; i< n;i++)
           {
           System.out.println("Enter Integer:");
           Integer p= new Integer(Integer.parseInt(obj.readLine());
            v.addElement(p);
           }
          System.out.println("Enter index of element to be deleted :");
           int n1= Integer.parseInt(obj.readLine());
           v.removeElementAt(n1);
           System.out.println("\nElements in vector after removing specific element:");
                for( i=0 ;i<v.size(); i++)
                     {
                      System.out.println(v.elementAt(i));
                     }
      }
    }
```

(Any relevant logic can be considered)

6. Attempt any 4: **Marks 16**

a) Explain any 4 components of JDK.

(Any four components- 1 Marks each)

Ans: component of JDK.

1. **Java Compiler** – Java Compiler is used to compile java files. Java Compiler component of JDK (Java Development Kit) is accessed using "javac" command.
2. **Java Interpreter** – Java Interpreter is used to interpret the java files that are compiled by Java Compiler. Java Interpreter component of JDK (Java Development Kit) is accessed using "java" command.
3. **Java Disassembler**-Java Disassembler is used to dissemble Java class file. Java Dissembler component of JDK (Java Development Kit) is accessed using "javap" command.

4. **Java Header File Generator** - Java Header File Generator is used to generate C language header files and source files to implement the native methods. Java Header File Generator component of JDK (Java Development Kit) is accessed using "javah" command.
5. **Java Documentation** – Java Documentation is required for easy maintenance of code. Java Documentation component of JDK (Java Development Kit) is accessed using "javadoc" command.
6. **Java Debugger** – Java Debugger is used to debug the java files. Java Debugger component of JDK (Java Development Kit) is accessed using "jdb" command.
7. **Java Applet Viewer** – Java Applet Viewer is used to view the Java Applets. Java Applet Viewer component of JDK (Java Development Kit) is accessed using "appletviewer" command.

b) Explain the functionality of file class.

(Use of File Class – 1 Mark, Any one constructor-1 Mark, Any two methods- 2 Marks)

Ans: **File Class:** A File object is used to obtain or manipulate information associated with a disk file, such as the permissions, time, date and directory path, and to navigate subdirectories hierarchies. File class does not operate on streams. It describes properties of File itself.

File class Constructor

1. **File(File parent, String child)**
   This method creates a new File instance from a parent abstract pathname and a child pathname string.
2. **File(String pathname)**
   This method creates a new File instance by converting the given pathname string into an abstract pathname.
3. **File(String parent, String child)**
   This method creates a new File instance from a parent pathname string and a child pathname string.
4. **File(URI uri)**
   This method Creates a new File instance by converting the given file: URI into an abstract pathname.

Methods

1. **public String getName()**
   Returns the name of the file or directory denoted by this abstract pathname.
2. **public String getParent()**
   Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory.
3. **public File getParentFile()**
   Returns the abstract pathname of this abstract pathname's parent, or null if this pathname does not name a parent directory.
4. **public String getPath()**
   Converts this abstract pathname into a pathname string.
5. **public boolean isAbsolute()**
   Tests whether this abstract pathname is absolute. Returns true if this abstract pathname is absolute, false otherwise
6. **public String getAbsolutePath()**
   Returns the absolute pathname string of this abstract pathname.
7. **public boolean canRead()**
   Tests whether the application can read the file denoted by this abstract pathname. Returns true if and only if the file specified by this abstract pathname exists and can be read by the application; false otherwise.
8. **public boolean canWrite()**
   Tests whether the application can modify to the file denoted by this abstract pathname. Returns true if and only if the file system actually contains a file denoted by this abstract pathname and the application is allowed to write to the file; false otherwise.
9. **public boolean exists()**
   Tests whether the file or directory denoted by this abstract pathname exists. Returns true if and only if the file or directory denoted by this abstract pathname exists; false otherwise
10. **public boolean isDirectory()**
    Tests whether the file denoted by this abstract pathname is a directory. Returns true if and only if the file denoted by this abstract pathname exists and is a directory; false otherwise.
11. **public boolean isFile()**

Tests whether the file denoted by this abstract pathname is a normal file. A file is normal if it is not a directory and, in addition, satisfies other system-dependent criteria. Any non-directory file created by a Java application is guaranteed to be a normal file. Returns true if and only if the file denoted by this abstract pathname exists and is a normal file; false otherwise.

12. public long lastModified()

Returns the time that the file denoted by this abstract pathname was last modified. Returns a long value representing the time the file was last modified, measured in milliseconds since the epoch (00:00:00 GMT, January 1, 1970), or 0L if the file does not exist or if an I/O error occurs.

13. public long length()

Returns the length of the file denoted by this abstract pathname. The return value is unspecified if this pathname denotes a directory.

c) What is interface? How do they differ from class?

(Interface definition- 1 Mark, Any three Differences -1 Mark each)

Ans: **Interface** is known as kind of a class. So interface also contains methods and variables but with major difference the interface defines only abstract method and final fields. This means that interface do not specify any code to implement those methods and data fields contains only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods.

Sr. No.	**CLASS**	**INTERFACE**
1	It has instance variable.	It has final variable.
2	It has non abstract method.	It has by default abstract method.
3	We can create object of class.	We can't create object of interface.
4	Class has the access specifiers like public, private, and protected.	Interface has only public access specifier
5	Classes are always extended.	Interfaces are always implemented.
6	The memory is allocated for the classes.	We are not allocating the memory for the interfaces.

d) Explain break and continue statement with example.

(break statement with example-2 Marks, continue statement with example- 2 Marks)

Ans: 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  x[] = {10, 20, 30, 40, 50};
     for(int i=0; i<5; i++ )
   {
       if( x[i] = = 30 )
             break;
       System.out.print( x[i] );
       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  x[] = {10, 20, 30, 40, 50};
    for(int i=0; i<5; i++ )
   {
```

```
            if( x[i] = = 30 )
                  continue;
            System.out.print( x[i] );
            System.out.print("\n");
          }
        }
      }
```

(Any other similar type of examples can be considered)

e) Write a program to accept number from user and convert it into binary by using wrapper class method.

***(Correct logic- 3 Marks, for syntax-1 Mark)*)**

Ans:

```
import java.io.*;
public class MyIntegerToBinary
{
   public static void main(String args[])throws IOException
  {
   BufferedReader obj = new BufferedReader (new InputStreamReader(System.in));
   int i;
   System.out.print("Enter number that you want to convert to binary: ");
   i=Integer.parseInt(obj.readLine());
   String binary = Integer.toBinaryString(i);    //Converts integer to binary
   System.out.println("Binary value: "+binary);
  }
}
```