MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)

SUMMER – 2019 EXAMINATION
MODEL ANSWER

Subject: Java Programming **Subject Code:** **22412**

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 judgement 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.

Q. No.	Sub Q.N.	Answer	Marking Scheme
1.	 **a)** **Ans.**	**Attempt any FIVE of the following:** **List any eight features of Java.** **Features of Java:** 1. Data Abstraction and Encapsulation 2. Inheritance 3. Polymorphism 4. Platform independence 5. Portability 6. Robust 7. Supports multithreading 8. Supports distributed applications 9. Secure 10. Architectural neutral 11. Dynamic	**10** **2M** ***Any eight features 2M***
	b) **Ans.**	**State use of finalize( ) method with its syntax.** **Use of finalize( ):** Sometimes an object will need to perform some action when it is	**2M**

		destroyed. Eg. If an object holding some non java resources such as file handle or window character font, then before the object is garbage collected these resources should be freed. To handle such situations java provide a mechanism called finalization. In finalization, specific actions that are to be done when an object is garbage collected can be defined. To add finalizer to a class define the finalize() method. The java run-time calls this method whenever it is about to recycle an object.
		Syntax:
		protected void finalize() {
		}
			Use 1M
			Syntax 1M

	c)
	Ans.
		Name the wrapper class methods for the following:
		(i) To convert string objects to primitive int.
		(ii) To convert primitive int to string objects.
		(i) To convert string objects to primitive int:
		String str=”5”;
		int value = Integer.parseInt(str);
		(ii) To convert primitive int to string objects:
		int value=5;
		String str=Integer.toString(value);
			2M
			1M for each method

	d) Ans.	**List the types of inheritances in Java.** ***(Note: Any four types shall be considered)*** **Types of inheritances in Java:** i. Single level inheritance ii. Multilevel inheritance iii. Hierarchical inheritance iv. Multiple inheritance v. Hybrid inheritance	2M *Any four types ½M each*

	e)
	Ans.
		Write the syntax of try-catch-finally blocks.
		try{
		//Statements to be monitored for any exception
		} catch(ThrowableInstance1 obj) {
		//Statements to execute if this type of exception occurs
		} catch(ThrowableInstance2 obj2) {
		//Statements
		}finally{
			2M
			Correct syntax 2M

		//Statements which should be executed even if any exception happens }	
	f) **Ans.**	**Give the syntax of < param > tag to pass parameters to an applet.** **Syntax:** <param name=”name” value=”value”> **Example:** <param name=”color” value=”red”>	**2M** ***Correct syntax 2M***
	g) **Ans.**	**Define stream class. List its types.** **Definition of stream class:** An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays. Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects. Java’s stream based I/O is built upon four abstract classes: InputStream, OutputStream, Reader, Writer. **Types of stream classes:** i. Byte stream classes ii. Character stream classes.	**2M** ***Definition 1M*** ***Types 1M***
2.	**a)** **Ans.**	**Attempt any <u>THREE</u> of the following:** **Explain the concept of platform independence and portability with respect to Java language.** ***(Note: Any other relevant diagram shall be considered).*** Java is a platform independent language. This is possible because when a java program is compiled, an intermediate code called the byte code is obtained rather than the machine code. Byte code is a highly optimized set of instructions designed to be executed by the JVM which is the interpreter for the byte code. Byte code is not a machine specific code. Byte code is a universal code and can be moved anywhere to any platform. Therefore java is portable, as it can be carried to any platform. JVM is a virtual machine which exists inside the computer memory and is a simulated computer within a computer which does all the functions of a computer. Only the JVM needs to be implemented for each platform. Although the details of the JVM will defer from platform to platform, all interpret the same	**12** **4M** ***Explanation 3M***

byte code.

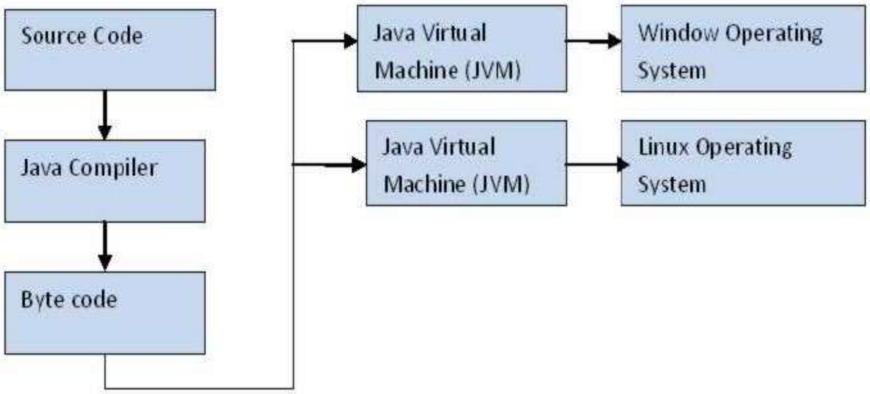


Diagram 1M

b) **Explain the types of constructors in Java with suitable example.** ***(Note: Any two types shall be considered).*** **4M**

Ans. Constructors are used to initialize an object as soon as it is created. Every time an object is created using the 'new' keyword, a constructor is invoked. If no constructor is defined in a class, java compiler creates a default constructor. Constructors are similar to methods but with to differences, constructor has the same name as that of the class and it does not return any value.

The types of constructors are:

1. Default constructor
2. Constructor with no arguments
3. Parameterized constructor
4. Copy constructor

Explanation of the two types of constructors 2M

1. Default constructor: Java automatically creates default constructor if there is no default or parameterized constructor written by user. Default constructor in Java initializes member data variable to default values (numeric values are initialized as 0, Boolean is initialized as false and references are initialized as null).

Example 2M

```
class test1 {
int i;
boolean b;
byte bt;
float ft;
String s;
public static void main(String args[]) {
test1 t = new test1(); // default constructor is called.
System.out.println(t.i);
System.out.println(t.s);
System.out.println(t.b);
System.out.println(t.bt);
System.out.println(t.ft);
   }
}
```

2.Constructor with no arguments: Such constructors does not have any parameters. All the objects created using this type of constructors has the same values for its datamembers.

Eg:

```
class Student {
int roll_no;
String name;
Student() {
roll_no = 50;
name="ABC";
}
void display() {
System.out.println("Roll no is: "+roll_no);
System.out.println("Name is : "+name);
}
public static void main(String a[]) {
Student s = new Student();
s.display();
}
}
```

3. Parametrized constructor: Such constructor consists of parameters. Such constructors can be used to create different objects with datamembers having different values.

```
class Student {
int roll_no;
String name;
Student(int r, String n) {
roll_no = r;
name=n;
}
void display() {
System.out.println("Roll no is: "+roll_no);
System.out.println("Name is : "+name);
}
public static void main(String a[]) {
Student s = new Student(20,"ABC");
s.display();
}
}
```

4. Copy Constructor : A copy constructor is a constructor that creates a new object using an existing object of the same class and initializes each instance variable of newly created object with corresponding instance variables of the existing object passed as argument. This constructor takes a single argument whose type is that of the class containing the constructor.

```
class Rectangle
{
 int length;
 int breadth;
 Rectangle(int l, int b)
{
   length = l;
   breadth= b;
   }
  //copy constructor
  Rectangle(Rectangle obj)
  {
   length = obj.length;
   breadth= obj.breadth;
   }
 public static void main(String[] args)
 {
 Rectangle r1= new Rectangle(5,6);
 Rectangle r2= new Rectangle(r1);
System.out.println("Area  of First Rectangle : "+
(r1.length*r1.breadth));
```

		System .out.println("Area of First Second Rectangle : "+
		(r1.length*r1.breadth));
		 }
		}	

	c)
	Ans.
		Explain the two ways of creating threads in Java.
		Thread is a independent path of execution within a program.
		There are two ways to create a thread:
		1. By extending the Thread class.
		Thread class provide constructors and methods to create and perform operations on a thread. This class implements the Runnable interface. When we extend the class Thread, we need to implement the method run(). Once we create an object, we can call the start() of the thread class for executing the method run().
		Eg:
		class MyThread extends Thread {
		public void run() {
		    for(int i = 1;i<=20;i++) {
		    System.out.println(i);
		 }
		}
		public static void main(String a[]) {
		MyThread t = new MyThread();
		 t.start();
		   }
		}
		a. By implementing the runnable interface.
		Runnable interface has only on one method- run().
		Eg:
		class MyThread implements Runnable {
		public void run() {
		   for(int i = 1;i<=20;i++) {
		   System.out.println(i);
		 }
		}
		  public static void main(String a[]) {
		  MyThread m = new MyThread();
		  Thread t = new Thread(m);
		  t.start();
		  }
			4M
			2M each for explaining of two types with example

```
}
```

d) **Distinguish between Input stream class and output stream class.** **4M**

Ans. Java I/O (Input and Output) is used to process the input and produce the output.

Java uses the concept of a stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations. A stream is a sequence of data. In Java, a stream is composed of bytes.

Sr. No.	Input stream class	Output stream class
1	Java application uses an input stream to read data from a source;	Java application uses an output stream to write data to a destination;.
2	It may read from a file, an array, peripheral device or socket	It may be a write to file, an array, peripheral device or socket
3	Input stream classes reads data as bytes	Output stream classes writes data as bytes
4	Super class is the abstract inputStream class	Super class is the abstract OutputStream class
5	Methods: public int read() throws IOException public int available() throws IOException public void close() throws IOException	Methods: public void write(int b) throws IOException public void write(byte[] b) throws IOException public void flush() throws IOException public void close() throws IOException
6	The different subclasses of Input Stream are: File Input stream, Byte Array Input Stream, Filter Input Stream, Piped Input Stream, Object Input Stream, DataInputStream.	The different sub classes of Output Stream class are: File Output Stream, Byte Array Output Stream , Filter output Stream, Piped Output Stream, Object Output Stream, DataOutputStream

Any four points for input stream class and output stream class 1M each

3. **Attempt any THREE of the following:** **12**

a) **Define a class student with int id and string name as data members and a method void SetData ( ). Accept and display the data for five students.** **4M**

Ans.

```
import java.io.*;
class student
{
int id;
String name;
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
void SetData()
{
 try
{
   System.out.println("enter id and name for student");
    id=Integer.parseInt(br.readLine());
    name=br.readLine();
}
   catch(Exception ex)
   {}
}
 void display()
{
     System.out.println("The id is " + id + " and the name is "+ name);
 }
public static void main(String are[])
{
     student[] arr;
     arr = new student[5];
     int i;
     for(i=0;i<5;i++)
    {
           arr[i] = new student();
    }
    for(i=0;i<5;i++)
    {
           arr[i].SetData();
     }
```

Correct logic 4M

```
          for(i=0;i<5;i++)
    {
         arr[i].display();
    }
  }
}
```

b) **Explain dynamic method dispatch in Java with suitable example.** **4M**

Ans. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.

- When an overridden method is called through a superclass reference, Java determines which version (superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

Explanation 2M

Therefore, if a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed. Here is an example that illustrates dynamic method dispatch:

```
// A Java program to illustrate Dynamic Method
// Dispatch using hierarchical inheritance
class A
{
   void m1()
   {
      System.out.println("Inside A's m1 method");
   }
}

class B extends A
{
      // overriding m1()
      void m1()
```

Example 2M

```
        {
            System.out.println("Inside B's m1 method");
         }
}

class C extends A
{
        // overriding m1()
        void m1()
       {
            System.out.println("Inside C's m1 method");
       }
}

// Driver class
class Dispatch
{
      public static void main(String args[])
     {
            // object of type A
            A a = new A();

            // object of type B
            B b = new B();

            // object of type C
            C c = new C();

            // obtain a reference of type A
            A ref;

            // ref refers to an A object
            ref = a;

            // calling A's version of m1()
            ref.m1();

            // now ref refers to a B object
            ref = b;
```

		// calling B's version of m1()
		ref.m1();
		
		// now ref refers to a C object
		ref = c;
		
		// calling C's version of m1()
		ref.m1();
		}
		}	

	c) Ans.	**Describe the use of following methods:** **(i) Drawoval ( )** **(ii) getFont ( )** **(iii) drawRect ( )** **(iv) getFamily ( )** **(i) Drawoval ( ):** Drawing Ellipses and circles: To draw an Ellipses or circles used 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 or filled circle, specify the same width and height. *Example:* g.drawOval(10,10,50,50); **(ii) getFont ( ):** It is a method of Graphics class used to get the font property Font f = g.getFont(); String fontName = f.getName(); Where g is a Graphics class object and fontName is string containing name of the current font. **(iii) drawRect ( ):** The drawRect() method display an outlined rectangle. Syntax*:* 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);	**4M** ***Each method 1M***

		(iv) getFamily ( ): The getfamily() method Returns the family of the font. String family = f.getFamily(); Where f is an object of Font class	
	d) Ans.	**Write a program to count number of words from a text file using stream classes.** ***(Note : Any other relevant logic shall be considered)***	4M ***Correct program 4M***

```
import java.io.*;
public class FileWordCount
{
public static void main(String are[]) throws IOException
{
File f1 = new File("input.txt");
int wc=0;
FileReader fr = new FileReader (f1);
int c=0;
try
{
  while(c!=-1)
{
   c=fr.read();
   if(c==(char)' ')
   wc++;
 }
 System.out.println("Number of words :"+(wc+1));
}
finally
{
   if(fr!=null)
   fr.close();
    }
  }
}
```

4.		**Attempt any <u>THREE</u> of the following:**	**12**
	a) **Ans.**	**Describe instance Of and dot (.) operators in Java with suitable example.** **Instance of operator:** The java instance of operator is used to test whether the object is an instance of the specified type (class or subclass or interface).	**4M**

The instance of in java is also known as type comparison operator because it compares the instance with type. It returns either true or false. If we apply the instance of operator with any variable that has null value, it returns false.

Example

```
class Simple1{
 public static void main(String args[]){
 Simple1 s=new Simple1();
 System.out.println(sinstanceofSimple1);//true
 }
}
```

Description and example of each operator 2M

dot (.) operator:

The dot operator, also known as separator or period used to separate a variable or method from a reference variable. Only static variables or methods can be accessed using class name. Code that is outside the object's class must use an object reference or expression, followed by the dot (.) operator, followed by a simple field name.

Example

this.name="john"; where name is a instance variable referenced by 'this' keyword

c.getdata(); where getdata() is a method invoked on object 'c'.

b) Explain the four access specifiers in Java. **4M**

Ans. There are 4 types of java access modifiers:

1. private 2. default 3. Protected 4. public

1) private access modifier: The private access modifier is accessible only within class.
2) default access specifier: If you don't specify any access control specifier, it is default, i.e. it becomes implicit public and it is accessible within the program.
3) protected access specifier: The protected access specifier is accessible within package and outside the package but through inheritance only.
4) public access specifier: The public access specifier is accessible everywhere. It has the widest scope among all other modifiers.

Each access specifiers 1M

c) **Differentiate between method overloading and method overriding.** **4M**

Ans.

Sr. No.	**Method overloading**	**Method overriding**
1	Overloading occurs when two or more methods in one class have the same method name but different parameters.	Overriding means having two methods with the same method name and parameters (i.e., method signature)
2	In contrast, reference type determines which overloaded method will be used at compile time.	The real object type in the run-time, not the reference variable's type, determines which overridden method is used at runtime
3	Polymorphism not applies to overloading	Polymorphism applies to overriding
4	overloading is a compile-time concept.	Overriding is a run-time concept

Any four points 1M each

d) **Differentiate between Java Applet and Java Application (any four points)** **4M**

Ans.

Sr. No.	**Java Applet**	**Java Application**
1	Applets run in web pages	Applications run on stand-alone systems.
2	Applets are not full featured application programs.	Applications are full featured programs.
3	Applets are the small programs.	Applications are larger programs.
4	Applet starts execution with its init().	Application starts execution with its main ().
5	Parameters to the applet are given in the HTML file.	Parameters to the application are given at the command prompt
6	Applet cannot access the local file system and resources	Application can access the local file system and resources.
7	Applets are event driven	Applications are control driven.

Any four points 1M each

e) **Write a program to copy content of one file to another file.** **4M**

Ans.

```
class fileCopy
{
  public static void main(String args[]) throws IOException
{
   FileInputStream in= new FileInputStream("input.txt");
   FileOutputStream out= new FileOutputStream("output.txt");
   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();
     }
   }
}
```

Correct logic 2M

Correct Syntax 2M

5. **Attempt any TWO of the following:** **12**

a) **Describe the use of any methods of vector class with their syntax.** **6M**

(Note: Any method other than this but in vector class shall be considered for answer).

Ans.

- boolean add(Object obj)-Appends the specified element to the end of this Vector.
- Boolean add(int index,Object obj)-Inserts the specified element at the specified position in this Vector.
- void addElement(Object obj)-Adds the specified component to the end of this vector, increasing its size by one.
- int capacity()-Returns the current capacity of this vector.
- void clear()-Removes all of the elements from this vector.
- Object clone()-Returns a clone of this vector.

Any 6 methods with their use 1M each

		• boolean contains(Object elem)-Tests if the specified object is a component in this vector. • void copyInto(Object[] anArray)-Copies the components of this vector into the specified array. • Object firstElement()-Returns the first component (the item at index 0) of this vector. • Object elementAt(int index)-Returns the component at the specified index. • int indexOf(Object elem)-Searches for the first occurence of the given argument, testing for equality using the equals method. • Object lastElement()-Returns the last component of the vector. • Object insertElementAt(Object obj,int index)-Inserts the specified object as a component in this vector at the specified index. • Object remove(int index)-Removes the element at the specified position in this vector. • void removeAllElements()-Removes all components from this vector and sets its size to zero.	
	b) Ans.	**Explain the concept of Dynamic method dispatch with suitable example.** Method overriding is one of the ways in which Java supports Runtime Polymorphism. Dynamic method dispatch is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time. When an overridden method is called through a superclass reference, Java determines which version (superclass/subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time. At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time. If a superclass contains a method that is overridden by a subclass, then when different types of objects are referred to through a superclass reference variable, different versions of the method are executed. Here is an example that illustrates dynamic method dispatch:	**6M** ***Explanation 3M***

```
/ A Java program to illustrate Dynamic Method
// Dispatch using hierarchical inheritance
class A
{
   void m1()
  {
      System.out.println("Inside A's m1 method");
   }
}
class B extends A
{
   // overriding m1()
   void m1()
   {
      System.out.println("Inside B's m1 method");
   }
}
class C extends A
{
   // overriding m1()
   void m1()
   {
      System.out.println("Inside C's m1 method");
   }
}

// Driver class
class Dispatch
{
   public static void main(String args[])
   {
      // object of type A
      A a = new A();

      // object of type B
      B b = new B();

      // object of type C
      C c = new C();
```

Example 3M

```
        // obtain a reference of type A
        A ref;

        // ref refers to an A object
        ref = a;

        // calling A's version of m1()
        ref.m1();

        // now ref refers to a B object
        ref = b;

        // calling B's version of m1()
        ref.m1();

        // now ref refers to a C object
        ref = c;

        // calling C's version of m1()
        ref.m1();
    }
}
```

Output:

```
Inside A's m1 method

Inside B's m1 method

Inside C's m1 method
```

Explanation:

The above program creates one superclass called A and it's two subclasses B and C. These subclasses overrides m1( ) method.

1. Inside the main() method in Dispatch class, initially objects of type A, B, and C are declared.
2. A a = new A(); // object of type A
3. B b = new B(); // object of type B
   C c = new C(); // object of type C

	c)	**Write a program to create two threads. One thread will display the numbers from 1 to 50 (ascending order) and other thread will display numbers from 50 to 1 (descending order).**	**6M**
	Ans.		***Creation of two threads 4M*** ***Creating main to create and start objects of 2 threads: 2M***

```
class Ascending extends Thread
{
  public void run()
 {
   for(int i=1; i<=15;i++)
 {
   System.out.println("Ascending Thread : " + i);
  }
 }
}

class Descending extends Thread
{
 public void run()
 {
  for(int i=15; i>0;i--) {
   System.out.println("Descending Thread : " + i);
  }
 }
}

public class AscendingDescending Thread
{
 public static void main(String[] args)
{
Ascending a=new Ascending();
a.start();
Descending d=new Descending();
d.start();
 }
}
```

6.		**Attempt any <u>TWO</u> of the following:**	**12**
	a)	**Explain the command line arguments with suitable example.**	**6M**
	Ans.	**Java Command Line Argument:** The java command-line argument is an argument i.e. passed at the time of running the java program.	

The arguments passed from the console can be received in the java program and it can be used as an input.

So, it provides a convenient way to check the behaviour of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.

Command Line Arguments can be used to specify configuration information while launching your application.

There is no restriction on the number of java command line arguments.

You can specify any number of arguments

Information is passed as Strings.

They are captured into the String args of your main method

4M for explanation

Simple example of command-line argument in java

In this example, we are receiving only one argument and printing it. To run this java program, you must pass at least one argument from the command prompt.

```
class CommandLineExample
{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
}
}
compile by > javac CommandLineExample.java
run by > java CommandLineExample sonoo
```

2M for example

b) **Write a program to input name and salary of employee and throw user defined exception if entered salary is negative.** **6M**

Ans.

```
import java.io.*;
class NegativeSalaryException extends Exception
{
 public NegativeSalaryException (String str)
 {
 super(str);
 }
}
public class S1
```

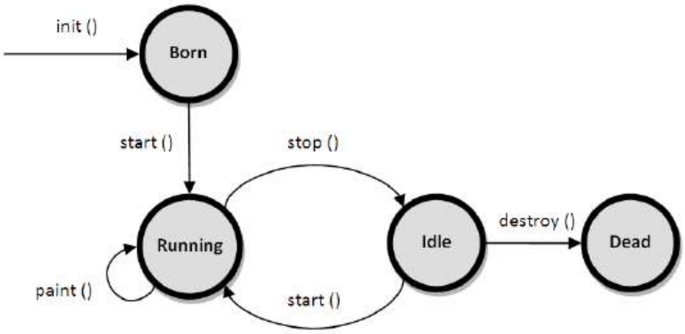
Extended Exception class with constructor 2M

```
{
 public static void main(String[] args) throws IOException
 {
BufferedReaderbr= new BufferedReader(new InputStreamReader(System.in));
  System.out.print("Enter  Name of employee");
  String name = br.readLine();
  System.out.print("Enter  Salary of employee");
  int  salary = Integer.parseInt(br.readLine());
   Try
 {
   if(salary<0)
        throw new NegativeSalaryException("Enter Salary amount
      isnegative");
System.out.println("Salary is "+salary);
  }
  catch (NegativeSalaryException a)
 {
   System.out.println(a);
  }
 }
}
```

Accepting data 1M

Throwing user defining Exception with try catch and throw 3M

c) Ans. **Describe the applet life cycle in detail.** **6M**



2M Diagram

Below is the description of each applet life cycle method:

init(): The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations

		are performed in this method. **start():** The start() method contains the actual code of the applet that should run. The start() method executes immediately after the init() method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser. **stop():** The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser. **destroy():** The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. stop() method executes just before when destroy() method is invoked. The destroy() method removes the applet object from memory. **paint():** The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of start() method and whenever the applet or browser is resized. The method execution sequence when an applet is executed is: • init() • start() • paint() The method execution sequence when an applet is closed is: • stop() • destroy()	***4M description***

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2013 Certified)

Winter – 19 EXAMINATION

Subject Name: Java Programming **Model Answer** **Subject Code: 22412**

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 judgement 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.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.		**Attempt any Five of the following:**	**10M**
	a	**Define Constructor. List its types.**	**2M**
	Ans	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. **Types of constructors:** 1. Default constructor 2. Parameterized constructor 3. Copy constructor	**Definition:1Mark** **Types: 1 Mark**
	b	**Define Class and Object.**	**2M**

	Ans	**Class**: A class is a user defined data type which groups data members and its associated functions together. **Objec**t: It is a basic unit of Object Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods.	**Definition 1 Mark each**
	c	**List the methods of File Input Stream Class.**	**2M**
	Ans	• void close() • int read() • int read(byte[] b) • read(byte[] b, int off, int len) • int available()	**Any Two Each for 1 Mark**
	d	**Define error. List types of error.**	**2M**
	Ans	• Errors are mistakes that can make a program go wrong. Errors may be logical or may be typing mistakes. An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash. Errors are broadly classified into two categories: 1. Compile time errors 2. Runtime errors	**Definition: 1m** **List: 1m**
	e	**List any four Java API packages.**	**2M**
	Ans	1.java.lang 2.java.util 3.java.io 4.java.awt 5.java.net 6.ava.applet	**1/2 Marks for one Package**
	f	**Define array. List its types.**	**2M**
	Ans	An array is a homogeneous data type where it can hold only objects of one data type. Types of Array:	**Definition 1 Mark, List 1 Mark**

<table>
<tr><td></td><td></td><td>1)One-Dimensional
2)Two-Dimensional</td><td></td></tr>
<tr><td></td><td><b>g</b></td><td><b>List access specifiers in Java.</b></td><td><b>2M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>1)public
2)private
3)friendly
4)protected
5)Private Protected</td><td><b>Any 2, 1M for each</b></td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td><b>2.</b></td><td></td><td><b>Attempt any Three of the following:</b></td><td><b>12M</b></td></tr>
<tr><td></td><td><b>a</b></td><td><b>Differentiate between String and String Buffer.</b></td><td><b>4M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>
<table>
<tr><th>String</th><th>String Buffer c</th></tr>
<tr><td>String is a major class</td><td>String Buffer is a peer class of String</td></tr>
<tr><td>Length is fixed (immutable)</td><td>Length is flexible (mutable)</td></tr>
<tr><td>Contents of object cannot be modified</td><td>Contents of object can be modified</td></tr>
<tr><td>Object can be created by assigning String constants enclosed in double quotes.</td><td>Objects can be created by calling constructor of String Buffer class using “new”</td></tr>
<tr><td>Ex:- String s=”abc”;</td><td>Ex:- StringBuffer s=new StringBuffer (“abc”);</td></tr>
</table>
</td><td><b>Any 4 Points 4 Marks</b></td></tr>
<tr><td></td><td><b>b</b></td><td><b>Define a class circle having data members pi and radius. Initialize and display values of data members also calculate area of circle and display it.</b></td><td></td></tr>
<tr><td></td><td><b>Ans</b></td><td>class abc
{</td><td><b>correct Program with correct logic 4 Mark</b></td></tr>
</table>

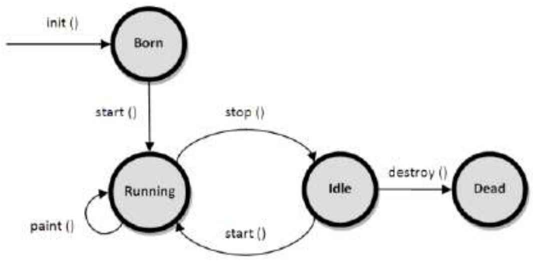
```
float pi,radius;
abc(float p, float r)
{
pi=p;
radius=r;
}
 void area()
{
float ar=pi*radius*radius;
System.out.println("Area="+ar);
}
void display()
 {
System.out.println("Pi="+pi);
System.out.println("Radius="+radius);
} }
 class area
{
public static void main(String args[])
{
abc a=new abc(3.14f,5.0f);
a.display();
```

		a.area();
		}
		}	

	c	**Define exception. State built-in exceptions.**	**4M**
	Ans	An exception is a problem that arises during the execution of a program. Java exception handling is used to handle error conditions in a program systematically by taking the necessary action **Built-in exceptions:** • **Arithmetic exception:** Arithmetic error such as division by zero. • **ArrayIndexOutOfBounds Exception:** Array index is out of bound • **ClassNotFoundException** • **FileNotFoundException:** Caused by an attempt to access a nonexistent file. • **IO Exception:** Caused by general I/O failures, such as inability to read from a file. • **NullPointerException:** Caused by referencing a null object. • **NumberFormatException:** Caused when a conversion between strings and number fails. • **StringIndexOutOfBoundsException:** Caused when a program attempts to access a nonexistent character position in a string. • **OutOfMemoryException:** Caused when there's not enough memory to allocate a new object. • **SecurityException:** Caused when an applet tries to perform an action not allowed by the browser's security setting. • **StackOverflowException:** Caused when the system runs out of stack space.	**Definition 2 Marks, List: 2 Marks**
	d	**Write syntax and example of :**	**4M**

		1) drawRect() **2)drawOval()**	
	Ans	**1)drawRect() :** drawRect () method display an outlined rectangle. **Syntax: 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);** **2) drawOval( ):** Drawing Ellipses and circles: To draw an Ellipses or circles used 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 or filled circle, specify the same width and height the following program draws several ellipses and circle. **Example: g.drawOval(10,10,50,50);**	**drawRect: 2Marks, drawOval: 2 Marks**
3.		**Attempt any Three of the following:**	
	a	**Explain the following classes.** **i)Byte stream class** **ii)Character Stream Class**	**4M**
	Ans	**i)Byte stream class:** 1) **InputStream** and **OutputStream** are designed for byte streams 2) Use the byte stream classes when working with bytes or other binary objects. 3) Input Stream is an abstract class that defines Java’s model of streaming byte input	**2M for any two points**

		4)The Input stream class defines methods for performing input function such as reading bytes, closing streams, Marking position in stream. 5) Output Stream is an abstract class that defines streaming byte output. 6) The output stream class defines methods for performing output function such as writing bytes, closing streams **ii)Character Stream Class:** 1. Reader and Writer are designed for character streams. 2. Use character stream classes when working with characters or strings. 3. Writer stream classes are designed to write characters. 4. Reader stream classes are designed to read characters. 5The two subclasses used for handling characters in file are FileReader (for reading characters) and FileWriter (for writing characters).	
	b	**Explain life cycle of Applet.**	**4M**
	Ans	When an applet begins, the AWT calls the following methods, in this sequence: 1. init( ) 2. start( ) 3. paint( ) When an applet is terminated, the following sequence of method calls takes place: 4. stop( ) 5. destroy( )	**1M for diagram ,3M for explanation**



init ( ): The **init( )** method is the first method to be called. This is where you should initialize Variables. This method is called only once during the run time of your applet.

start( ): The **start( )** method is called after **init( )**.It is also called to restart an applet after it has Been stopped. Whereas **init( )** is called once—the first time an applet is loaded—**start( )**is called each time an applet's HTML document is displayed onscreen.

Paint ( ): The **paint ( )** method is called each time your applet's output must be redrawn. 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.

Stop ( ): 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.

destroy( ): The destroy ( ) method is called when the environment determines that your applet needs to be removed completely from memory.

c **Differentiate between class and interfaces.** **4M**

Ans **1M for each point**

Class	Interface
1)doesn't Supports multiple inheritance	1) Supports multiple inheritance
2)"extend " keyword is used to inherit	2)"implements " keyword is used to inherit
3) class contain method body	3) interface contains abstract method(method without body)

4)contains any type of variable	4)contains only final variable
5)can have constructor	5)cannot have constructor
6)can have main() method	6)cannot have main() method

7)syntax
Class classname
{
Variable declaration,
Method declaration
}
	7)syntax
	Inteface Innterfacename
	{
	Final Variable declaration,
	abstract Method declaration
	}

	d	**Define type casting. Explain its types with syntax and example.**	**4M**

	Ans	**1.** The process of converting one data type to another is called casting or type casting.
		2. If the two types are compatible, then java will perform the conversion automatically.
		3. It is possible to assign an int value to long variable.
		4. However, if the two types of variables are not compatible, the type conversions are not implicitly allowed, hence the need for type casting.
		There are two types of conversion:
		1.Implicit type-casting:
		2.Explicit type-casting:
		1. Implicit type-casting:
		Implicit type-casting performed by the *compiler automatically*; if there will be no loss of precision.
		Example:
		int i = 3;
		double f;
		f = i;
			1M for definition,3M for types explanation

output:
f = 3.0

Widening Conversion:

The rule is to promote the smaller type to bigger type to prevent loss of precision, known as **Widening Conversion**.

2. Explicit type-casting:

- Explicit type-casting performed via a type-casting operator in the prefix form of (*new-type*) operand.
- Type-casting forces an explicit conversion of type of a value. Type casting is an operation which takes one operand, operates on it and returns an equivalent value in the specified type.

Syntax:

newValue = (typecast)value;

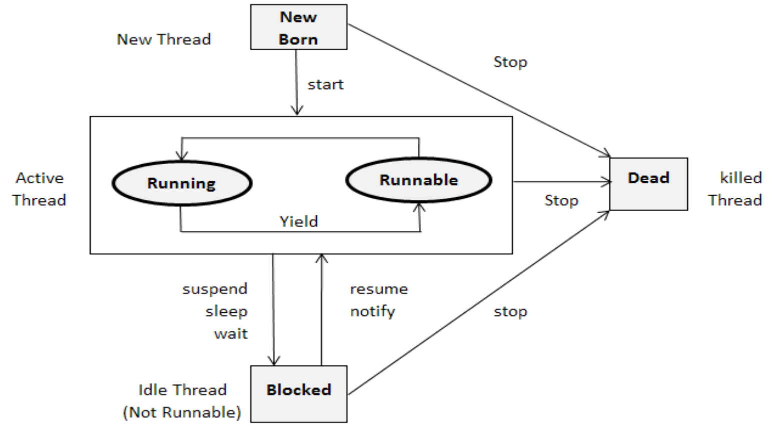
Example:

double f = 3.5;

int i;
i = (int)f; // it cast double value 3.5 to int 3.

Narrowing Casting: Explicit type cast is requires to Narrowing conversion to inform the compiler that you are aware of the possible loss of precision.

4.		**Attempt any Three of the following:**	
	a	**Explain life cycle of thread.**	**4M**

	Ans	 New Thread New Born start Stop Active Thread Running Runnable Yield Stop Dead killed Thread suspend sleep wait resume notify stop Idle Thread (Not Runnable) Blocked 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. **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. **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().	**2M for diagram,2M for explanation**

		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. **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. 1) **suspend()** : Thread can be suspended by this method. It can be rescheduled by resume(). 2) **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(). 3) **sleep():** We can put a thread to sleep for a specified time period using sleep(time) where time is in ms. It re-enters the runnable state as soon as period has elapsed /over **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	**Describe final variable and final method.**	**4M**

	Ans	**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.
		Syntax:
		final void findAverage()
		{
		//implementation
		}
		Example of declaring a final method:
		class A
		{
			2M for definition,2M for example

```
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.

Example of declaring final variable: final int size = 100;

	c	**Explain any two logical operator in java with example.**	**4M**
	Ans	**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:	**2M for each operator with eg.**

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Program demonstrating logical Operators

```
public class Test
```

```
{

public static void main(String args[])

{

boolean a = true;

boolean b = false;

System.out.println("a && b = " + (a&&b));

System.out.println("a || b = " + (a||b) );

System.out.println("!(a && b) = " + !(a && b));

}

}
```

Output:

```
a && b = false

a || b = true

!(a && b) = true
```

	d	**Differentiate between array and vector.**	**4M**
	Ans		**any four points** **1m for each point**

Array	Vector
1) An array is a structure that holds multiple values of the same type.	1)The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.

2) An array is a homogeneous data type where it can hold only objects of one data type.	2) Vectors are heterogeneous. You can have objects of different data types inside a Vector.
3) After creation, an array is a fixed-length structure.	3) The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
4) Array can store primitive type data element.	4) Vector are store non-primitive type data element.
5)Declaration of an array : int arr[] = new int [10];	5)Declaration of Vector: Vector list = new Vector(3);
6) Array is the static memory allocation.	6) Vector is the dynamic memory allocation.

	e	**List any four methods of string class and state the use of each.**	**4M**
	Ans	The java.lang.String class provides a lot of methods to work on string. By the help of these methods, We can perform operations on string such as trimming, concatenating, converting, comparing, replacing strings etc. **1) to Lowercase ():** Converts all of the characters in this String to lower case. Syntax: s1.toLowerCase() Example: String s="Sachin"; System.out.println(s.toLowerCase()); Output: sachin **2)to Uppercase():**Converts all of the characters in this String to upper case	**any four methods of string class can be considered**

Syntax: s1.toUpperCase()

Example:

String s="Sachin";

System.out.println(s.toUpperCase());

Output: SACHIN

3) trim (): Returns a copy of the string, with leading and trailing whitespace omitted.

Syntax: s1.trim()

Example:

String s=" Sachin ";

System.out.println(s.trim());

Output:Sachin

4) replace (): Returns a new string resulting from replacing all occurrences of old Char in this string with new Char.

Syntax: s1.replace(‘x’,’y’)

Example:

String s1="Java is a programming language. Java is a platform.";

String s2=s1.replace("Java","Kava"); //replaces all occurrences of "Java" to "Kava"

System.out.println(s2);

Output: Kava is a programming language. Kava is a platform.

5.		**Attempt any Three of the following:**	**12-Total Marks**
	a	**Write a program to create a vector with five elements as (5, 15, 25, 35, 45). Insert new element at 2nd position. Remove 1st and 4th element from vector.**	**6M**

	Ans	import java.util.*;
		class VectorDemo
		{
		public static void main(String[] args)
		{
		Vector v = new Vector();
		v.addElement(new Integer(5));
		v.addElement(new Integer(15));
		v.addElement(new Integer(25));
		v.addElement(new Integer(35));
		v.addElement(new Integer(45));
		System.out.println("Original array elements are ");
		for(int i=0;i<v.size();i++)
		{
		System.out.println(v.elementAt(i));
		}
		v.insertElementAt(new Integer(20),1); // insert new element at 2nd position
		v.removeElementAt(0);
		//remove first element
		v.removeElementAt(3);
		//remove fourth element
		System.out.println("Array elements after insert and remove operation ");
		for(int i=0;i<v.size();i++)
		{
		System.out.println(v.elementAt(i));
		}}}
			(Vector creation with elements – 2 M,
			Insert new element – 2M,
			Remove elements 2 M,
			(Any other logic can be considered)

	b	**Define package. How to create user defined package? Explain with example.**	**6M**
	Ans	Java provides a mechanism for partitioning the class namespace into more manageable parts. This mechanism is the package. The package is both naming and visibility controlled mechanism. Package can be created by including package as the first statement in java source code. Any classes declared within that file will belong to the specified package. Package defines a namespace in	**(Definition of package - 1M,**

which classes are stored.

The syntax for defining a package is:

package *pkg*;

Here, pkg is the name of the package

eg : package

mypack;

Package creation - 2M

Packages are mirrored by directories. Java uses file system directories to store packages. The class files of any classes which are declared in a package must be stored in a directory which has same name as package name. The directory must match with the package name exactly. A hierarchy can be created by separating package name and sub package name by a period(.) as pkg1.pkg2.pkg3; which requires a directory structure as pkg1\pkg2\pkg3.

Example - 3M

Syntax:

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*|*);

(Note Any other example can be considered)

Example:

```
package package1;
public class Box
{
        int l= 5;
        int b = 7;
        int h = 8;
        public void display()
        {
        System.out.println("Volume is:"+(l*b*h));
        }
}
```

Source file:

```
import package1.Box;
 class volume
{
```

		public static void main(String args[])
		{
		Box b=new Box();
		b.display();
		}
		}	

	c	**Write a program to create two threads one thread will print even no. between 1 to 50 and other will print odd number between 1 to 50.**	**6M**

	Ans	import java.lang.*;
		class Even extends Thread
		{
		public void run()
		{
		try
		{
		for(int i=2;i<=50;i=i+2)
		{
		System.out.println("\t Even thread :"+i);
		sleep(500);
		}
		}
		catch(InterruptedException e)
		{System.out.println("even thread interrupted");
		}
		}
		}
		class Odd extends Thread
		{
		public void run()
		{
		try
		{
		for(int i=1;i<50;i=i+2)
		{
		System.out.println("\t Odd thread :"+i);
		sleep(500);
			Creation of two threads 4M
			Creating main to create and start objects of 2 threads: 2M
			(Any other logic can be considered)

```
                        }
                }
                catch(InterruptedException e)
                {System.out.println("odd thread interrupted");
                }
        }
}
class EvenOdd
{
                public static void main(String args[])
                {
                        new Even().start();
                        new Odd().start();
                }
}
```

6.		**Attempt any Three of the following:**	**12 M**
	a	**Explain how to pass parameter to an applet ? Write an applet to accept username in the form of parameter and print "Hello <username>".**	**6M**
	Ans	**Passing Parameters to Applet** - User defined parameters can be supplied to an applet using <PARAM…..> tags. - PARAM tag names a parameter the Java applet needs to run, and provides a value for that parameter. - PARAM tag can be used to allow the page designer to specify different colors, fonts, URLs or other data to be used by the applet. **To set up and handle parameters, two things must be done.** 1. Include appropriate <PARAM..>tags in the HTML document. The Applet tag in HTML document allows passing the arguments using param tag. The syntax of <PARAM…> tag <Applet code="AppletDemo" height=300 width=300> <PARAM NAME = name1 VALUE = value1> </Applet> NAME:attribute name VALUE: value of attribute named by corresponding PARAM NAME.	**(Explanation for parameter passing - 3M,** **Correct Program – 3M**

2. Provide code in the applet to parse these parameters. The Applet access their attributes using the getParameter method.

The syntax is : String getParameter(String name);

Program

```
import java.awt.*;
import java.applet.*;
public class hellouser extends Applet
{
        String str;
        public void init()
        {
                str = getParameter("username");
                 str = "Hello "+ str;
        }
        public void paint(Graphics g)
        {
        g.drawString(str,10,100);
        }
}
<HTML>
<Applet code = hellouser.class width = 400 height = 400>
<PARAM NAME = "username" VALUE = abc> </Applet>
</HTML>
```

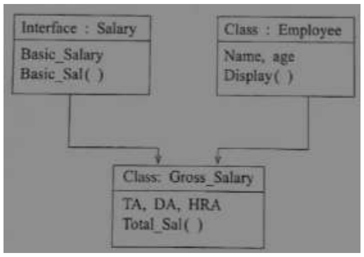
(OR)

```
import java.awt.*;
import java.applet.*;
/*<Applet code = hellouser.class width = 400 height = 400>
<PARAM NAME = "username" VALUE = abc>
</Applet>*/
public class hellouser extends Applet
{
        String str;
         public void init()
        {
                str = getParameter("username");
                str = "Hello "+ str;
        }
```

		public void paint(Graphics g)
		{
		g.drawString(str,10,100);
		}
		}	

	b	**Write a program to perform following task** **(i) Create a text file and store data in it.** **(ii) Count number of lines and words in that file.**	**6M**

	Ans	import java.util.*;
		import java.io.*;
		class Model6B
		{
		public static void main(String[] args) throws Exception
		{
		int lineCount=0, wordCount=0;
		String line = "";
		BufferedReader br1 = new BufferedReader(new InputStreamReader(System.in));
		
		FileWriter fw = new FileWriter("Sample.txt"); //create text file for writing
		System.out.println("Enter data to be inserted in file: ");
		String fileData = br1.readLine();
		fw.write(fileData);
		fw.close();
		BufferedReader br = new BufferedReader(new FileReader("Sample.txt"));
		while ((line = br.readLine()) != null)
		{
		lineCount++; // no of lines count
		String[] words = line.split(" ");
		wordCount = wordCount + words.length; // no of words count
		}
		System.out.println("Number of lines is : " + lineCount);
		System.out.println("Number of words is : " + wordCount);
		}
			Create file and store data : 3M,
			
			Get lines and word count : 3M)
			
			(Any other logic can be considered)

		}	
	c	**Implement the following inheritance**  Interface : Salary Basic_Salary Basic_Sal( ) Class : Employee Name, age Display( ) Class: Gross_Salary TA, DA, HRA Total_Sal( )	**6M**

Ans

```
interface Salary
{
        double Basic Salary=10000.0;
        void Basic Sal();
}
class Employee
{
        String Name;
        int age;
        Employee(String  n, int  b)
        {
                Name=n;
                age=b;
        }
        void Display()
        {
                System.out.println("Name of Employee :"+Name);
                System.out.println("Age  of Employee :"+age);
        }
}
class Gross_Salary extends Employee implements Salary
{
        double  HRA,TA,DA;
        Gross_Salary(String n, int b, double  h,double t,double d)
        {
        super(n,b);
        HRA=h;
```

(Interface: 1M,

Employee class: 2M,

Gross_Salary class: 3M)

```
        TA=t;
        DA=d;
        }
        public void Basic_Sal()
        {
                System.out.println("Basic Salary
:"+Basic_Salary);
        }
        void Total_Sal()
        {
                Display();
                Basic_Sal();
                double Total_Sal=Basic_Salary + TA + DA +
HRA;
                System.out.println("Total Salary :"+Total_Sal);
        }
}
class EmpDetails
{       public static void main(String args[])
        {       Gross_Salary s=new
Gross_Salary("Sachin",20,1000,2000,7000);
                s.Total_Sal();
        }
}
```

(Any other logic considered)

SUMMER – 2022 EXAMINATION
MODEL ANSWER

Subject: Java Programming **Subject Code: 22412**

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 anyequivalent 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 judgement 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.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No	Sub Q.N.	Answer	Marking Scheme
1.	 **a)** **Ans.**	**Attempt any FIVE of the following:** **Enlist the logical operators in Java.** && : Logical AND || : Logical OR ! : Logical NOT	**10** **2M** ***1M each Any two operators***
	b) **Ans.**	**Give the syntax and example for the following functions** **i) min ( )** **ii) Sqrt ( )** **i) min()** **Syntax: (Any one of the following)** static int min(int x, int y) Returns minimum of x and y static long min(long x, long y) Returns minimum of x and y static float min(float x, float y) Returns minimum of x and y static double min(double x, int y) Returns minimum of x and y	**2M** ***1M for each function with example***

		Example:
		int y= Math.min(64,45);
		ii)Sqrt()
		Syntax:
		static double sqrt(double arg) Returns square root of arg.
		Example:
		double y= Math.sqrt(64);	

	c) **Ans.**	**Define the interface in Java.** Interface is similar to a class. It consist of only abstract methods and final variables. To implement an interface a class must define each of the method declared in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.	**2M** ***1M for each point, Any two points***
	d) **Ans.**	**Enlist any four inbuilt packages in Java.** 1.java.lang 2.java.util 3.java.io 4.java.awt 5.java.net 6.java.applet	**2M** ***½ M for each package Any four packages***
	e) **Ans.**	**Explain any two methods of File Class** 1. boolean createNewFile(): It creates a new, empty file named by this abstract pathname automatically, if and only if no file with the same name exists. if(file.createNewFile()) System.out.println("A new file is successfully created."); 2. String getName(): It returns the name of the file or directory denoted by the object's abstract pathname. System.out.println("File name : " + file.getName()); 3. String getParent(): It returns the parent's pathname string of the object's abstract pathname or null if the pathname does not name a parent directory. System.out.println("Parent name : " + file.getParent());	**2M** ***1M for each method Any two methods***

		4. boolean isFile(): It returns True if the file denoted by the abstract pathname is a normal file, and False if it is not a normal file.
		System.out.println("File size (bytes) : " + file.isFile());
		5. boolean canRead(): It returns True if the application can read the file denoted by the abstract pathname, and returns False otherwise.
		System.out.println("Is file readable : " + file.canRead());
		6. boolean canWrite(): It returns True if the application can modify the file denoted by the abstract pathname, and returns False otherwise.
		System.out.println("Is file writeable : " + file.canWrite());
		7. boolean canExecute(): It returns True if the application can execute the file denoted by the abstract pathname, and returns False otherwise.
		System.out.println("Is file executable : " + file.canExecute());	

	f) **Ans.**	**Write syntax of elipse.** **Syntax:** void fillOval(int top, int left, int width, int height) The filled 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 OR **Syntax:** void drawOval(int top, int left, int width, int height) The empty 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	**2M** ***2M for correct syntax***
	g) **Ans.**	**Enlist any four compile time errors.** 1)Missing semicolon 2)Missing of brackets in classes and methods 3)Misspelling of variables and keywords. 4)Missing double quotes in Strings. 5)Use of undeclared variable. 6)Incompatible type of assignment/initialization. 7)Bad reference to object.	**2M** ***½ M for each error*** ***Any four can be considered***

2. **Attempt any <u>THREE</u> of the following:** **12**

a) **Explain any four features of Java** **4M**

Ans.

1.Object Oriented:

In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

1M for each feature Any four features

2.Platform Independent:

Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

3.Simple:

Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

4.Secure:

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

5.Architecture-neutral:

Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

6.Multithreaded:

With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

7.Interpreted: Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

b) **Write a Java program to copy the content of one file into another.** **4M**

Ans.

2M for correct logic,

2M for code

```
import java.io.*;
class filecopy
{
public static void main(String args[]) throws IOException
{
FileReader fr= new FileReader(“file1.txt");
FileWriter fo= new FileWriter("file2.txt");
int ch;
try
{
while((ch=fr.read())!= -1)
{
fo.write(ch);
}
System.out.println(“file copied successfully”);
fr.close();
fo.close();
}
finally
{
if(fr!=null)
fr.close();
if(fo!=null)
fo.close();
}}}
```

c) **Write the difference between vectors and arrays. (any four points)** **4M**

Ans.

1M for each point

Any four points

S.No	Array	Vector
1	An array is a structure that holds multiple values of the same type.	The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.
2	An array is a homogeneous data type where it can hold only objects of one data type	Vectors are heterogeneous. You can have objects of different data types inside a Vector.

3	After creation, an array is a fixed-length structure	The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created
4	Array can store primitive type data element.	Vector are store non primitive type data element.
5	Declaration of an array int arr[] = new int [10];	Declaration of Vector: Vector list = new Vector(3)
6	Array is the static memory allocation.	Vector is the dynamic memory allocation

d) **Explain exception handling mechanism w.r.t. try, catch, throw and finally.** **4M**

Ans. *1M for each*

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
}
```

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 can have one or more statements that are necessary to process the exception.

Syntax:

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

throw:
It is mainly used to throw an instance of user defined exception.
Example: throw new myException("Invalid number"); assuming myException as a user defined exception

finally:
finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block follows try or catch block.
Syntax:

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

3. **Attempt any THREE of the following:** **12**

a) **Write a Java Program to find out the even numbers from 1 to 100 using for loop.** **4M**

Ans.

```
class test
{
public static void main(String args[])
{
        System.out.println("Even numbers from 1 to 100 :");
        for(int i=1;i<=100; i++)
        {
                if(i%2==0)
                System.out.print(i+" ");
        }
}
}
```

2M for Program logic

2M for program syntax

b) **Explain any four visibility controls in Java.** **4M**

Ans. Four visibility control specifiers in Java are public, default, private and protected. The visibility control in java can be seen when concept of package is used with the java application.

1) private :The access level of a private specifier is only within the class. It cannot be accessed from outside the class.
2) default :When no specifier is used in the declaration, it is called as default specification. Default scope for anything declared in java

3M for Explanation

is implicit public. With this it can be accessed anywhere within the same package.

3) protected :The access level of a protected specifier is within the package and outside the package through derived class.
4) public :The access level of a public specifier is everywhere. It can be accessed from within the class, outside the class, within thepackage and outside the package.
5) private protected access: The visibility level is between protected access and private access. The fields are visible in all subclasses regardless of what package they are in.

These fiveaccess specifiers can be mapped with four categories in which packages in java can be managed with access specification matrix as:

Access Modifier	Public	Protected	Friendly (default)	Private protected	private
Access Location					
Same Class	Yes	Yes	Yes	Yes	Yes
Sub class in same package	Yes	Yes	Yes	Yes	No
Other classes in same package	Yes	Yes	Yes	No	No
Sub class in other packages	Yes	Yes	No	Yes	No
Non sub classes in other packages	Yes	No	No	No	No

1M for access specification table

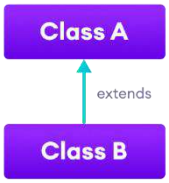
c) Ans.

Explain single and multilevel inheritance with proper example.

4M

Single level inheritance:

In single inheritance, a single subclass extends from a single superclass.



1M for each explanation

Example :

class A

{

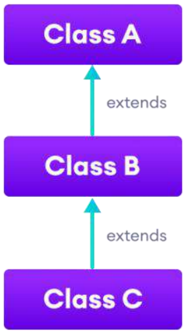
1M for each example

```
void display()
{
System.out.println("In Parent class A");
}
}
class B extends A   //derived class B from A
{
void show()
{
System.out.println("In child class B");
}
public static void main(String args[])
{
B b= new B();
b.display();  //super class method call
b.show();  // sub class method call
}
}
```

Note : any other relevant example can be considered.

Multilevel inheritance:

In multilevel inheritance, a subclass extends from a superclass and then the same subclass acts as a superclass for another class. Basically it appears as derived from a derived class.



Example:

```
class A
{
void display()
{
System.out.println("In Parent class A");
}
}
class B extends A   //derived class B from A
{
void show()
{
System.out.println("In child class B");
}
}
class C extends B   //derived class C from B
{
public void print()
{
System.out.println("In derived from derived class C");
}
public static void main(String args[])
{
C c= new C();
c.display();  //super class method call
c.show();  // sub class method call
c.print();    //sub-sub class method call
}
}
```

Note : any other relevant example can be considered.

d) **Write a java applet to display the following output in Red color. Refer Fig. No. 1.** **4M**

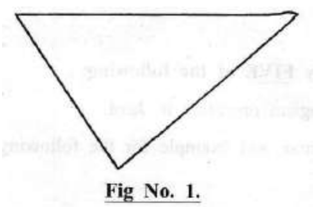


Fig No. 1.

Ans.

```
import java.awt.*;
import java.applet.*;
public class myapplet extends Applet
```

2M for correct logic

```
{
        public void paint(Graphics g)
        {
        int x[]={10,200,70};
        int y[]={10,10,100};
        g.setColor(Color.red);
        g.drawPolygon(x,y,3);
        }
}
/*<applet code=myapplet height=400 width=400>
</applet>*/
```

2M for correct syntax

4. **Attempt any THREE of the following:** **12**

a) **Explain switch case and conditional operator in java with suitable example.** **4M**

Ans. **switch…case statement:**

The switch…case statement allows us to execute a block of code among many alternatives.

1M for explanation switch case statement

1M for example

Syntax :

```
switch (expression)
 {
case value1:
   // code
break;
case value2:
   // code
break;
  ...
  ...
default:
   // default statements
  }
```

The expression is evaluated once and compared with the values of each case.

If expression matches with value1, the code of case value1 are executed. Similarly, the code of case value2 is executed if expression matches with value2.

break is a required statement, which is used to take break from switch block, if any case is true. Otherwise even after executing a case, if break is not given, it will go for the next case.
If there is no match, the code of the default case is executed.

Example :

```
// Java Program to print day of week
// using the switch...case statement
class test1{
public static void main(String[] args) {
int number = 1;
   String day;
switch (number)
{
case 1:
day = "Monday";
break;
case 2:
day= "Tuesday";
break;
case 3:
day = "Wednesday";
break;
case 4:
day= "Thursday";
break;
case 5:
day = "Friday";
break;
case 6:
day= "Saturday";
break;
case 7:
day = "Sunday";
break;
default:
day= "Invalid day";
   }
System.out.println(day);
  }
}
```

Note : any other relevant example can be considered.

Conditional Operator:

The Conditional Operator is used to select one of two expressions for evaluation, which is based on the value of the first operands. It is used to handling simple situations in a line.

Syntax:

expression1 ? expression2:expression3;

The above syntax means that if the value given in Expression1 is true, then Expression2 will be evaluated; otherwise, expression3 will be evaluated.

Example

```
class test
 {
public static void main(String[] args) {
  String result;
int  a = 6, b = 12;
result = (a==b ? "equal":"Not equal");
System.out.println("Both are "+result);
 }
}
```

Note : any other relevant example can be considered.

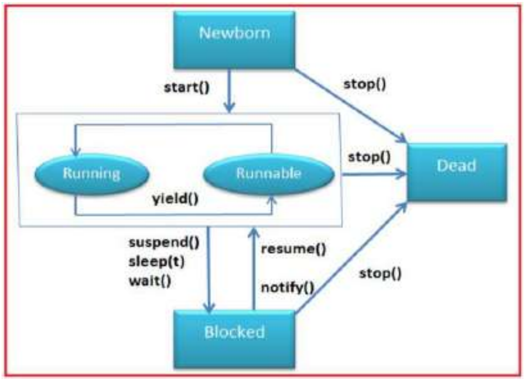
1M for explanation Conditional operator

1M for example

b) Ans. **Draw and explain life cycle of thread.** **4M**

Life cycle of thread includes following states :

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

		 Newborn
		start()
		stop()
		Running
		Runnable
		yield()
		stop()
		Dead
		suspend()
		sleep(t)
		wait()
		resume()
		notify()
		stop()
		Blocked
		New – A new thread begins its life cycle in the new state. It is also referred to as a born thread. This is the state where a thread has been created, but it has not yet been started. A thread is started by calling its start() method.
		Runnable – The thread is in the runnable state after the invocation of the start() method, but the scheduler has not selected it to be the running thread. It is in the Ready-to-run state by calling the start method and waiting for its turn.
		Running – When the thread starts executing, then the state is changed to a “running” state. The method invoked is run|().
		Blocked–This is the state when the thread is still alive but is currently not eligible to run. This state can be implemented by methods such as suspend()-resume(), wait()-notify() and sleep(time in ms).
		Dead – This is the state when the thread is terminated. The thread is in a running state and as soon as it is completed processing it is in a “dead state”. Once a thread is in this state, the thread cannot even run again.
			2M for diagram
			2M for explanation

	c) **Ans.**	**Write a java program to sort an 1-d array in ascending order using bubble-sort.** public class BubbleSort { public static void main(String[] args)	**4M** ***2M for correct logic***

```
 {
int arr[] ={3,60,35,2,45,320,5};
System.out.println("Array Before Bubble Sort");
for(int i=0; i<arr.length; i++)
        {
System.out.print(arr[i] + " ");
        }
System.out.println();
int n = arr.length;
int temp = 0;
for(int i=0; i< n; i++)
                {
for(int j=1; j < (n-i); j++)
                        {
if(arr[j-1] >arr[j])
{
           //swap elements
temp = arr[j-1];
arr[j-1] = arr[j];
arr[j] = temp;
           }
        }
                }
System.out.println("Array After Bubble Sort");
for(int i=0; i<arr.length; i++)
{
System.out.print(arr[i] + " ");
}
   }
}
```

2M for correct syntax

d) Ans. **Explain how to create a package and how to import it** **4M**

To create package following steps can be taken:

1) Start the code by keyword ‘package’ followed by package name.
   Example : package mypackage;
2) Complete the code with all required classes inside the package with appropriate access modifiers.
3) Compile the code with ‘javac’ to get .class file.

3M for steps to create

		Example: javac myclass.java to get myclass.class 4) Create a folder which is same as package name and make sure that class file of package is present inside it. If not, copy it inside this folder. To import the package inside any other program : Make use of import statement to include package in your program. It can be used with '*' to gain full access to all classes within package or just by giving class name if just one class access is required. Example : import mypackage.myclass; or importmypackage.*;	*1M to import*
	e) Ans.	**Explain** **i) drawLine** **ii) drawOval** **iii) drawRect** **iv) drawArc** i) drawLine(): It is a method from Graphics class and is used to draw line between the points(x1, y1) and (x2, y2). **Syntax :** drawLine(int x1, int y1, int x2, int y2) ii) drawOval():Its is a method from Graphics class and is used to draw oval or ellipse and circle. **Syntax** : drawOval(int x, ,int y, int width, int height) It is used to draw oval with the specifiedwidth and height. If width and height are given equal, then it draws circle otherwise oval/ellipse. iii) drawRect():It is a method from Graphics class and it draws a rectangle with the specified widthand height. **Syntax :** drawRect(int x, int y, int width, int height) iv) drawArc():It is a method from Graphics class and is used to draw a circular or elliptical arc. **Syntax :** drawArc(int x, int y, int width, int height, intstartAngle, intsweepAngle)	**4M** ***1M for each***

		where first fourare x, y, width and height as in case of oval or rect. The next two are start angle and sweep angle.When sweep angle is positive, it moves in anticlockwise direction. It is given as negative, It moves in clockwise direction.	
5.		**Attempt any TWO of the following:**	**12**
	a)	**How to create user defined package in Java. Explain with an suitable example.**	**6M**
	Ans.	A **java package** is a group of similar types of classes, interfaces and sub-packages It also provides access protection and removes name collisions.	***3M Package creation***
		Creation of user defined package: To create a package a physical folder by the name should be created in the computer.	
		Example: we have to create a package myPack, so we create a folder d:\myPack The java program is to be written and saved in the folder myPack. To add a program to the package, the first line in the java program should be package <name>; followed by imports and the program logic.	***(Note: Code snippet can be used for describing)***

		package myPack;
		import java.util;
		public class Myclass {
		//code
		}	

		Access user defined package: To access a user defined package, we need to import the package in our program. Once we have done the import we can create the object of the class from the package and thus through the object we can access the instance methods.	***3M for Example***

		import mypack.*;
		public class
		MyClassExample{
		public static void main(String a[]) {
		Myclass c= new Myclass();	

```
    }
    }
```

Example:

(Note Any other similar example can be considered)

```
 package package1;
public class Box
{
          int l= 5;
          int b = 7;
          int h = 8;
           public void display()
          {
          System.out.println("Volume is:"+(l*b*h));
          }
 }
```

Source file:

```
import package1.Box;
class volume {
public static void main(String args[])
 {
           Box b=new Box();
          b.display();
} }
```

b) **Write a Java program in which thread A will display the even numbers between 1 to 50 and thread B will display the odd numbers between 1 to 50. After 3 iterations thread A should go to sleep for 500ms.** **6M**

Ans.

3M Correct program with syntax

```
Import java.lang.*;
class A extends Thread
{
          public void run()
          {
                    try
                    {
                    for(int i=2;i<=50;i=i+2)
                    {
```

```
                System.out.println("\t A thread :"+i);
                if(i == 6) // for 3rd iteration
                sleep(500);
        }
                }
        catch(Exception e)
        {
                System.out.println("A thread interrupted");
        }
}
}
class B extends Thread
{
        public void run()
        {
        try
        {
        for(int i=1;i<50;i=i+2)
        {
                System.out.println("\t B thread :"+i);
        }
        }
        catch(Exception e)
        {
                        System.out.println("B thread interrupted");
        }
}
}
class OddEven
{
        public static void main(String args[])
        {
                new A().start();
                        new B().start();
}
}
```

3M Correct logic

	c)	**What is constructor? List types of constructor. Explain parameterized constructor with suitable example.**	**6M**
	Ans.	**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.	***2M for Definition***
		Types of constructors: 1. Default constructor 2. Parameterized constructor 3. Copy constructor 4. Constructor with no arguments or No-Arg Constructor or Non-Parameterized constructor.	***1M List types*** ***(Any 3)***
		Parameterized constructor: When constructor method is defined with parameters inside it, different value sets can be provided to different constructor with the same name.	

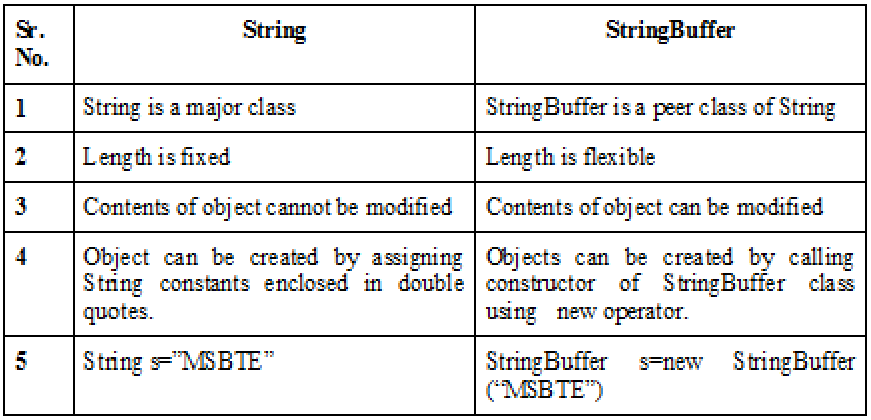
		Example
		class Student {
		int roll_no;
		String name;
		Student(int r, String n) // parameterized constructor
		{
		roll_no = r;
		name=n;
		}
		void display()
		{
		System.out.println("Roll no is: "+roll_no);
		System.out.println("Name is : "+name);
		}
			1M parameterized constructor
			2M Example
			(Any Other Example Can be

		public static void main(String a[])
		{
		Student s = new Student(20,"ABC"); // constructor with parameters
		s.display();
		}
		}
			***considered*
			*)***

6.	
	a)
	
	Ans.
		Attempt any <u>TWO</u> of the following:
		Write a Java Program to count the number of words from a text file using stream classes.
		import java.io.*;
		public class FileWordCount {
		public static void main(String are[]) throws IOException
		{
		File f1 = new File("input.txt");
		int wc=0;
		FileReader fr = new FileReader (f1);
		int c=0;
		
		try { while(c!=-1)
		{
		c=fr.read();
		if(c==(char)' ')
		wc++;
		}
		System.out.println("Number of words :"+(wc+1));
		}
		finally
		{
		if(fr!=null)
		fr.close();
		}
		}
		}
			12
			6M
			(Note : Any other relevant logic shall be considered)
			3M Correct program with syntax
			3M Correct logic

	b)	**Explain the difference between string class and string buffer class.** **Explain any four methods of string class**	**6M**

Ans.



Sr. No.	String	StringBuffer
1	String is a major class	StringBuffer is a peer class of String
2	Length is fixed	Length is flexible
3	Contents of object cannot be modified	Contents of object can be modified
4	Object can be created by assigning String constants enclosed in double quotes.	Objects can be created by calling constructor of StringBuffer class using new operator.
5	String s="MSBTE"	StringBuffer s=new StringBuffer ("MSBTE")

1M each Any 2 points

Methods of string class

1)toLowercase ():

Converts all of the characters in this String to lower case.

Syntax: s1.toLowerCase()

Example: String s="Sachin";

System.out.println(s.toLowerCase());

Output: sachin

1M each Any 4 Methods correct explanation

2) toUppercase():

Converts all of the characters in this String to upper case

Syntax: s1.toUpperCase()

Example: String s="Sachin";

System.out.println(s.toUpperCase());

Output: SACHIN

3)trim ():

Returns a copy of the string, with leading and trailing whitespace omitted.

Syntax: s1.trim()

Example: String s=" Sachin ";

System.out.println(s.trim());

Output:Sachin

4)replace (): Returns a new string resulting from replacing all occurrences of old Char in this string with new Char.

Syntax: s1.replace('x','y')

Example: String s1="Java is a programming language. Java is a platform.";

String s2=s1.replace("Java","Kava"); //replaces all occurrences of "Java" to "Kava" System.out.println(s2);

Output: Kava is a programming language. Kava is a platform

5. length():

Syntax: int length()

It is used to return length of given string in integer.

Eg. String str="INDIA"

System.out.println(str.length()); // Returns 5

6. charAt():

Syntax: char charAt(int position)

The charAt() will obtain a character from specified position .

Eg. String s="INDIA"

System.out.println(s.charAt(2)); // returns D

7. **substring():**

Syntax:

String substring (int startindex)

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

Example:

System.out.println(("Welcome".substring(3)); //come

(OR)

String substring(int startindex,int endindex)

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

Example :

System.out.println(("Welcome”.substring(3,5));//co

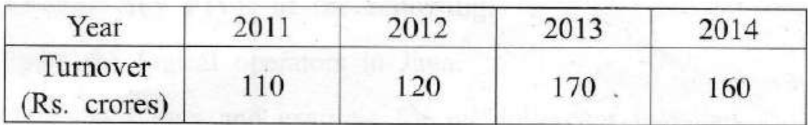
8. compareTo():

Syntax: int compareTo(Object o) or int compareTo(String anotherString)

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

```
Example. 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);
```

c) **Write a Java applet to draw a bar chart for the following values.** **6M**



Year	2011	2012	2013	2014
Turnover (Rs. crores)	110	120	170 .	160

Ans.

```
import java.awt.*;
import java.applet.*;

/* <applet code=BarChart width=400 height=400>
<param name=c1 value=110>
<param name=c2 value=120>
<param name=c3 value=170>
<param name=c4 value=160>
<param name=label1 value=2011>
<param name=label2 value=2012>
<param name=label3 value=2013>
```

2M for Applet tag

2M for

```
<param name=label4 value=2014>
<param name=Columns value=4>
</applet>
*/

public class BarChart extends Applet
{
int n=0;
 String label[];
int value[];

 public void init() {

setBackground(Color.yellow);
  try {

int n = Integer.parseInt(getParameter("Columns"));
   label = new String[n];
   value = new int[n];
   label[0]  = getParameter("label1");
   label[1]  = getParameter("label2");
   label[2]  = getParameter("label3");
   label[3]  = getParameter("label4");
   value[0] = Integer.parseInt(getParameter("c1"));
   value[1] = Integer.parseInt(getParameter("c2"));
   value[2] = Integer.parseInt(getParameter("c3"));
   value[3] = Integer.parseInt(getParameter("c4"));
   }
  catch(NumberFormatException e){}
 }
 public void paint(Graphics g)
 {
for(int i=0;i<4;i++) {
g.setColor(Color.black);
g.drawString(label[i],20,i*50+30);
g.setColor(Color.red);
g.fillRect(50,i*50+10,value[i],40);
   }
```

Syntax

2M Correct Logic

		} }	

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2013 Certified)

WINTER – 2022 EXAMINATION

Subject Name: Java Programming **Model Answer** **Subject Code:** 22412

Important Instructions to examiners:

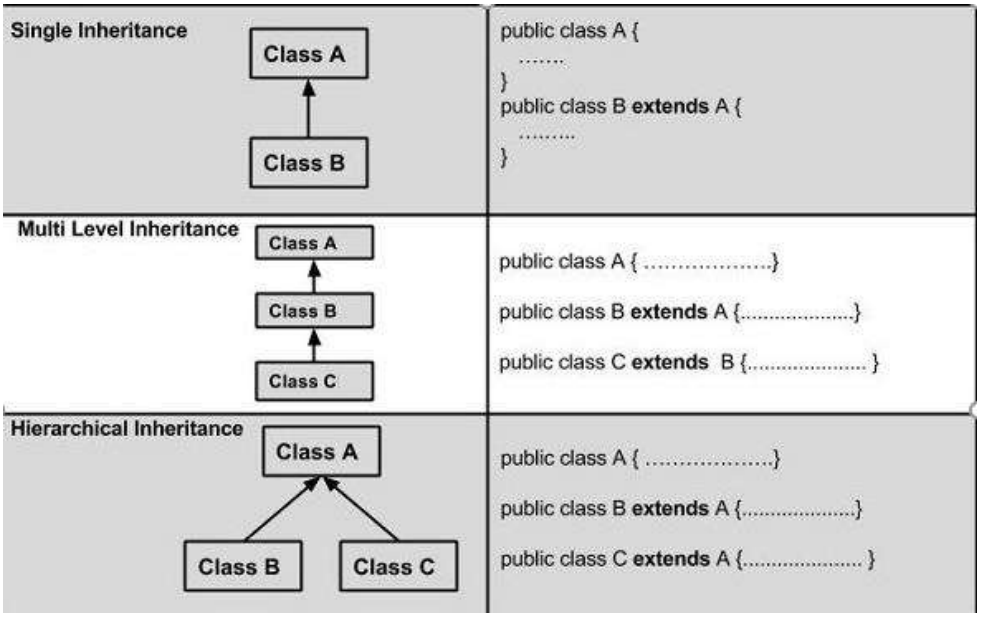
1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling 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 judgement 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.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

<table>
<tr><th>Q. No.</th><th>Sub Q. N.</th><th>Answer</th><th>Marking Scheme</th></tr>
<tr><td>1</td><td></td><td><b>Attempt any <u>FIVE</u> of the following:</b></td><td><b>10 M</b></td></tr>
<tr><td></td><td>a)</td><td><b>State any four relational operators and their use.</b></td><td><b>2 M</b></td></tr>
<tr><td></td><td>Ans</td><td>
<table>
<tr><th>Operator</th><th>Meaning</th></tr>
<tr><td>&lt;</td><td>Less than</td></tr>
<tr><td>&gt;</td><td>Greater than</td></tr>
<tr><td>&lt;=</td><td>Less than or equal to</td></tr>
<tr><td>&gt;=</td><td>Greater than or equal to</td></tr>
<tr><td>= =</td><td>Equal to</td></tr>
<tr><td>!=</td><td>Not equal to</td></tr>
</table>
</td><td>2M (1/2 M each)
Any Four</td></tr>
<tr><td></td><td>b)</td><td><b>Enlist access specifiers in Java.</b></td><td><b>2 M</b></td></tr>
<tr><td></td><td>Ans</td><td>The access specifiers in java specify accessibility (scope) of a data member, method, constructor orclass. There are 5 types of java access specifier:
• public
• private</td><td>2M (1/2 M each)
Any Four</td></tr>
</table>

		- default (Friendly) - protected - private protected	
	c)	**Explain constructor with suitable example.**	**2 M**

	Ans	Constructors are used to assign initial value to instance variable of the class.
		It has the same name as class name in which it resides and it is syntactically similar to anymethod.
		Constructors do not have return value, not even 'void' because they return the instance if class.
		Constructor called by new operator.
		Example:
		class Rect
		{
		int length, breadth;
		Rect() //constructor
		{
		length=4; breadth=5;
		}
		public static void main(String args[])
		{
		Rect r = new Rect();
		System.out.println("Area : " +(r.length*r.breadth));
		}
		}
		Output : Area : 20
			1M-
			Explanation
			1M-
			Example

	d)	**List the types of inheritance which is supported by java.**	**2 M**
	Ans		**Any two** **1 M each**

		 Single Inheritance
		Class A
		Class B
		Multi Level Inheritance
		Class A
		Class B
		Class C
		Hierarchical Inheritance
		Class A
		Class B
		Class C
		public class A {
		....
		}
		public class B **extends** A {
		....
		}
		public class A { ....}
		public class B **extends** A {...................}
		public class C **extends** B {..................... }
		public class A { ....}
		public class B **extends** A {...................}
		public class C **extends** A {..................... }	

	e)	**Define thread. Mention 2 ways to create thread.**	**2 M**

	Ans	1. Thread is a smallest unit of executable code or a single task is also called as thread.
		2. Each tread has its own local variable, program counter and lifetime.
		3. A thread is similar to program that has a single flow of control.
		<u>There are two ways to create threads in java:</u>
		1. By extending thread class
		Syntax: -
		class Mythread extends Thread
		{
		----- 	
		}
		2. Implementing the Runnable Interface
		Syntax:
		class MyThread implements Runnable
		{
		public void run()
		{
		------ 	
		}
			1 M-
			Define
			Thread
			
			1M -2ways
			to create
			thread

	f)	**Distinguish between Java applications and Java Applet (Any 2 points)**	**2 M**

Ans

Applet	Application
Applet does not use main() method for initiating execution of code	Application use main() method for initiating execution of code
Applet cannot run independently	Application can run independently
Applet cannot read from or write to files in local computer	Application can read from or write to files in local computer
Applet cannot communicate with other servers on network	Application can communicate with other servers on network
Applet cannot run any program from local computer.	Application can run any program from local computer.
Applet are restricted from using libraries from other language such as C or C++	Application are not restricted from using libraries from other language
Applets are event driven.	Applications are control driven.

1 M for each point (any 2 Points)

g) **Draw the hierarchy of stream classes.** **2 M**

Ans

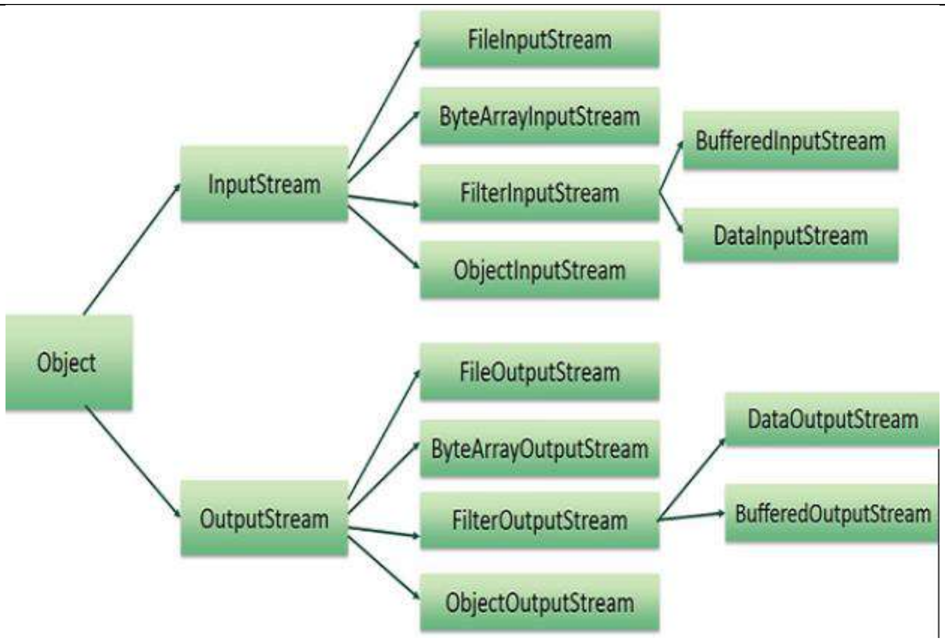


Fig: hierarchy of stream classes

2M-Correct diagram

2.		**Attempt any THREE of the following:**	**12 M**
	a)	**Write a program to check whether the given number is prime or not.**	**4 M**
	Ans	**Code:**	4M (for any correct program and logic)

```
class PrimeExample
{
 public static void main(String args[]){
  int i,m=0,flag=0;
  int n=7;//it is the number to be checked
  m=n/2;
  if(n==0||n==1){
   System.out.println(n+" is not prime number");
  }else{
   for(i=2;i<=m;i++){
    if(n%i==0){
     System.out.println(n+" is not prime number");
     flag=1;
     break;
    }
   }
   if(flag==0)  { System.out.println(n+" is prime number"); }
  }//end of else
}
}
```

Output:

```
7 is prime number
```

	b)	**Define a class employee with data members 'empid , name and salary. Accept data for three objects and display it**	**4 M**

Ans

```
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[3];
for(inti=0; i<3; i++)
{
e[i] = new employee(); e[i].getdata();
}
System.out.println(" Employee Details are : ");
for(inti=0; i<3; i++)
e[i].show();
}
}
```

4M (for correct program and logic)

	c)	**Describe Life cycle of thread with suitable diagram.**	**4 M**
	Ans	**1) Newborn State** A NEW Thread (or a Born Thread) is a thread that's been created but not yet started. It remains in this state until we start it using the start() method. The following code snippet shows a newly created thread that's in the NEW state: Runnable runnable = new NewState(); Thread t = new Thread(runnable); **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(). Runnable runnable = new NewState(); Thread t = new Thread(runnable); t.start(); **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 stop() 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	1M-digram of life cycle 3M-explanation

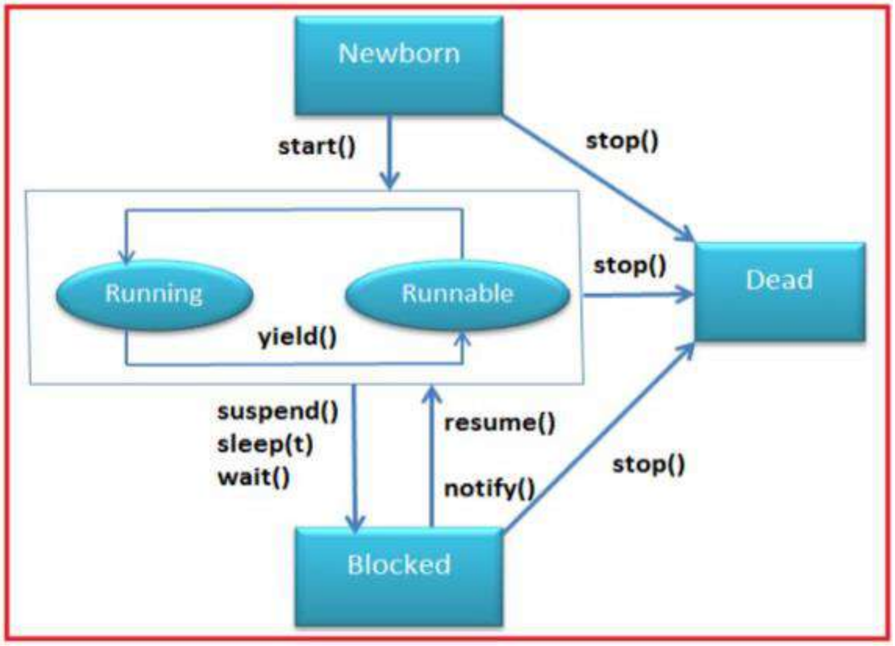


Fig: Life cycle of Thread

d) **Write a program to read a file (Use character stream)** **4 M**

Ans

```
import java.io.FileWriter;
import java.io.IOException;
public class IOStreamsExample {
  public static void main(String args[]) throws IOException {
    //Creating FileReader object
    File file = new File("D:/myFile.txt");
    FileReader reader = new FileReader(file);
    char chars[] = new char[(int) file.length()];
    //Reading data from the file
    reader.read(chars);
    //Writing data to another file
    File out = new File("D:/CopyOfmyFile.txt");
    FileWriter writer = new FileWriter(out);
    //Writing data to the file
    writer.write(chars);
    writer.flush();
    System.out.println("Data successfully written in the specified file");
  }
    }
```

4M (for correct program and logic)

3. **Attempt any <u>THREE</u> of the following:** **12 M**

	a)	**Write a program to find reverse of a number.**	**4 M**

	Ans	public class ReverseNumberExample1
		{ public static void main(String[] args)
		{
		int number = 987654, reverse =0;
		while(number !=0)
		{
		int remainder = number % 10;
		reverse = reverse * 10 + remainder;
		number = number/10;
		}
		System.out.printtln(“The reverse of the given number is: “ + reverse);
		} }
			Any Correct program with proper logic -4M

	b)	**State the use of final keyword with respect to inheritance.**	**4 M**
	Ans	Final keyword : The keyword final has three uses. First, it can be used to create the equivalent of a named constant.(in interface or class we use final as shared constant or constant.) **Other two uses of final apply to inheritance** Using final to Prevent Overriding While method overriding is one of Java’s most powerful features, To disallow a method from being overridden, specify final as a modifier at the start of its declaration. Methods declared as final cannot be overridden. **The following fragment illustrates final:** class A { final void meth() { System.out.println("This is a final method.");	Use of final keyword-2 M Program-2 M

```
}
}
class B extends A
{
void meth()
{ // ERROR! Can't override.
System.out.println("Illegal!");
}
}
```

As base class declared method as a final , derived class can not override the definition of base class methods.

c) **Give the usage of following methods** **4 M**

- **i) drawPolygon ()**
- **ii) DrawOval ()**
- **iii) drawLine ()**
- **iv) drawArc ()**

Ans **i) drawPolygon ():**

Method use with description 1 M

- drawPolygon() method is used to draw arbitrarily shaped figures.
- Syntax: void drawPolygon(int x[], int y[], int numPoints)
- The polygon‟s end points are specified by the co-ordinates pairs contained within the x and y arrays. The number of points define by x and y is specified by numPoints.

```
Example: int xpoints[]={30,200,30,200,30};
            int ypoints[]={30,30,200,200,30};
            int num=5;
           g.drawPolygon(xpoints,ypoints,num);
```

ii) drawOval ():

- To draw an Ellipses or circles used 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 or filled circle, specify the same width and height the following program draws several ellipses and circle.
- Example: g.drawOval(10,10,50,50);

ii) drawLine ():

- The drawLine() method is used to draw line which take two pair of coordinates, (x1,y1) and (x2,y2) as arguments and draws a line between them.

- The graphics object g is passed to paint() method.
- Syntax: g.drawLine(x1,y1,x2,y2);
- Example: g.drawLine(100,100,300,300;)

iv) drawArc ()

drawArc( ) 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);

d) **Write any four methods of file class with their use.** **4 M**

Ans

One method 1 M

public String getName()	Returns the name of the file or directory denoted by this abstract pathname.
public String getParent()	Returns the pathname string of this abstract pathname's parent, or null if this pathname does not name a parent directory
public String getPath()	Converts this abstract pathname into a pathname string.
public boolean isAbsolute()	Tests whether this abstract pathname is absolute. Returns true if this abstract pathname is absolute, false otherwise
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
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.
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 nondirectory 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.

4. **Attempt any <u>THREE</u> of the following:** **12 M**

a) **Write all primitive data types available in Java with their storage Sizes in** **4 M**

bytes.

Ans

Data Type	Size
Byte	1 Byte
Short	2 Byte
Int	4 Byte
Long	8 Byte
Double	8 Byte
Float	4 Byte
Char	2 Byte
boolean	**1 Bit**

Data type name, size and default value and description carries 1 M

b) **Write a program to add 2 integer, 2 string and 2 float values in a vector. Remove the element specified by the user and display the list.** **4 M**

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");
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(s7);
v.addElement(s8);
System.out.println(v);
v.removeElement(s2);
v.removeElementAt(4);
System.out.println(v);
}
}
```

Correct program- 4 M, stepwise can give marks

c) **Develop a program to create a class ‘Book’ having data members author, title and price. Derive a class 'Booklnfo' having data member 'stock position’ and** **4 M**

		method to initialize and display the information for three objects.	
	Ans	see code below	Correct program- 4 M

```
class Book
{
 String author, title, publisher;
 Book(String a, String t, String p)
 {
  author = a;
  title = t;
  publisher = p;
 }
}
class BookInfo extends Book
{
 float price;
 int stock_position;
 BookInfo(String a, String t, String p, float amt, int s)
 {
  super(a, t, p);
  price = amt;
  stock_position = s;
 }
 void show()
 {
  System.out.println("Book Details:");
  System.out.println("Title: " + title);
  System.out.println("Author: " + author);
  System.out.println("Publisher: " + publisher);
  System.out.println("Price: " + price);
  System.out.println("Stock Available: " + stock_position);
 }
}
class Exp6_1
{
 public static void main(String[] args)
 {
  BookInfo ob1 = new BookInfo("Herbert Schildt", "Complete Reference", "ABC
Publication", 359.50F,10);
  BookInfo ob2 = new BookInfo("Ulman", "system programming", "XYZ Publication",
359.50F, 20);
  BookInfo ob3 = new BookInfo("Pressman", "Software Engg", "Pearson Publication",
879.50F, 15);
  ob1.show();
```

		ob2.show();
		ob3.show();
		}
		}
		OUTPUT
		Book Details:
		Title: Complete Reference
		Author: Herbert Schildt
		Publisher: ABC Publication
		Price: 2359.5
		Stock Available: 10
		Book Details:
		Title: system programming
		Author: Ulman
		Publisher: XYZ Publication
		Price: 359.5
		Stock Available: 20
		Book Details:
		Title: Software Engg
		Author: Pressman
		Publisher: Pearson Publication
		Price: 879.5
		Stock Available: 15	

	d)	**Mention the steps to add applet to HTML file. Give sample code.**	**4 M**
	Ans	Adding Applet to the HTML file: Steps to add an applet in HTML document 1. Insert an <APPLET> tag at an appropriate place in the web page i.e. in the body section of HTML file. 2. Specify the name of the applet's .class file. 3. If the .class file is not in the current directory then use the codebase parameter to specify:- a. the relative path if file is on the local system, or b. the uniform resource locator(URL) of the directory containing the file if it is on a remote computer. 4. Specify the space required for display of the applet in terms of width and height in pixels. 5. Add any user-defined parameters using <param> tags 6. Add alternate HTML text to be displayed when a non-java browser is used. 7. Close the applet declaration with the </APPLET> tag. Open notepad and type the following source code and save it into file name	Steps – 2M Example – 2 M

“Hellojava.java”

```
import java.awt.*;
import java.applet.*;
public class Hellojava extends Applet
{
 public void paint (Graphics g)
{
g.drawString("Hello Java",10,100);
} }
```

Use the java compiler to compile the applet “Hellojava.java” file.

```
C:\jdk> javac  Hellojava.java
```

After compilation “Hellojava.class” file will be created. Executable applet is nothing but the .class file of the applet, which is obtained by compiling the source code of the applet. If any error message is received, then check the errors, correct them and compile the applet again.

We must have the following files in our current directory.

o Hellojava.java
o Hellojava.class
o HelloJava.html

If we use a java enabled web browser, we will be able to see the entire web page containing the applet.

We have included a pair of <APPLET..> and </APPLET> tags in the HTML body section. The <APPLET…> tag supplies the name of the applet to be loaded and tells the browser how much space the applet requires. The <APPLET> tag given below specifies the minimum requirements to place the HelloJava applet on a web page. The display area for the applet output as 300 pixels width and 200 pixels height. CENTER tags are used to display area in the center of the screen.

```
<APPLET CODE = hellojava.class WIDTH = 400 HEIGHT = 200 > </APPLET>
```

Example: Adding applet to HTML file:

Create Hellojava.html file with following code:

```
<HTML>
<! This page includes welcome title in the title bar and displays a welcome message. Then it specifies
the applet to be loaded and executed.
>
<HEAD> <TITLE> Welcome to Java Applet </TITLE> </HEAD>
<BODY> <CENTER> <H1> Welcome to the world of Applets </H1> </CENTER> <BR>
<CENTER>
```

```
<APPLET CODE=HelloJava.class WIDTH = 400 HEIGHT = 200 > </APPLET>
</CENTER>
</BODY>
</HTML>
```

e) **Write a program to copy contents of one file to another.** **4 M**

Ans

```
import java.io.*;
class copyf
{
public static void main(String args[]) throws IOException
{
BufferedReader in=null;
BufferedWriter out=null;
try
{
in=new BufferedReader(new FileReader("input.txt"));
out=new BufferedWriter(new FileWriter("output.txt"));
int c;
while((c=in.read())!=-1)
{
out.write(c);
}
System.out.println("File copied successfully");
}
finally
{
if(in!=null)
{
in.close();
}
if(out!=null)
{
out.close();
}
}
}
}
```

Correct program- 4 M

5. **Attempt any <u>TWO</u> of the following:** **12 M**

a) **Compare array and vector. Explain elementAT( ) and addElement( ) methods.** **6 M**

Ans

Sr. No.	Array	Vector
1	An array is a structure that holds multiple values of the same type.	The Vector is similar to array holds multiple objects and like an array; it contains components that can be accessed using an integer index.
2	An array is a homogeneous data type where it can hold only objects of one data type.	Vectors are heterogeneous. You can have objects of different data types inside a Vector.
3	After creation, an array is a fixed-length structure.	The size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.
4	Array can store primitive type data element.	Vector are store non-primitive type data element
5	Array is unsynchronized i.e. automatically increase the size when the initialized size will be exceed.	Vector is synchronized i.e. when the size will be exceeding at the time; vector size will increase double of initial size.
6	Declaration of an array : int arr[] = new int [10];	Declaration of Vector: Vector list = new Vector(3);
7	Array is the static memory allocation.	Vector is the dynamic memory allocation
8	Array allocates the memory for the fixed size ,in array there is wastage of memory.	Vector allocates the memory dynamically means according to the requirement no wastage of memory.
9	No methods are provided for adding and removing elements.	Vector provides methods for adding and removing elements.
10	In array wrapper classes are not used.	Wrapper classes are used in vector
11	Array is not a class.	Vector is a class.

4 M for any 4 correct points

1 M for elementAt()

1 M for addElement ()

elementAT( ):

The **elementAt()** method of Java Vector class is used to get the element at the specified

		index in the vector. Or The **elementAt()** method returns an element at the specified index. addElement( ): The **addElement()** method of **Java Vector** class is used to add the specified element to the end of this vector. Adding an element increases the vector size by one.	
	b)	**Write a program to create a class 'salary with data members empid', 'name' and 'basicsalary'. Write an interface 'Allowance' which stores rates of calculation for da as 90% of basic salary, hra as 10% of basic salary and pf as 8.33% of basic salary. Include a method to calculate net salary and display it.**	**6 M**
	Ans	(code below)	6 M for correct program

```
interface allowance
 {
 double da=0.9*basicsalary;
double hra=0.1*basicsalary;
double pf=0.0833*basicsalary;
void netSalary();
 }

class Salary
{
int empid;
String name;
float basicsalary;
Salary(int i, String n, float b)
{
empid=I;
 name=n;
basicsalary =b;
 }
 void display()
 {
System.out.println("Empid of Emplyee="+empid);
 System.out.println("Name of Employee="+name);
System.out.println("Basic Salary of Employee="+ basicsalary);
}
}

class net_salary extends salary implements allowance
{
 float ta;
net_salary(int i, String n, float b, float t)
{
```

```
super(i,n,b);
ta=t;
}
void disp()
{
display();
System.out.println("da of Employee="+da);
}
public void netsalary()
{
double net_sal=basicsalary+ta+hra+da;
System.out.println("netSalary of Employee="+net_sal);
}
}
class Empdetail
{
public static void main(String args[])
{
net_salary s=new net_salary(11, “abcd”, 50000);
 s.disp();
s.netsalary();
 }
 }
```

	c)	**Define an exception called 'No Match Exception' that is thrown when the passward accepted is not equal to "MSBTE'. Write the program.**	**6 M**

Ans

```
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
{
```

6 M for correct program

```
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());
}
}
}
```

6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write a program to check whether the string provided by the user is palindrome or not.**	**6 M**
	Ans	See program below	6 M for correct program

```
import java.lang.*;

import java.io.*;

import java.util.*;

class palindrome

{

public static void main(String arg[ ]) throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter String:");

String word=br.readLine( );

int len=word.length( )-1;

int l=0;

int flag=1;

int r=len;

while(l<=r)

{
```

```
if(word.charAt(l)==word.charAt(r))
{
 l++;
r--;
}
else
{
 flag=0;
break;
}
 }
if(flag==1)
 {
System.out.println("palindrome");
 }
else
{
System.out.println("not palindrome");
 }
}
 }
```

	b)	**Define thread priority ? Write default priority values and the methods to set and change them.**	**6 M**
	Ans	Thread Priority: In java each thread is assigned a priority which affects the order in which it is scheduled for running. Threads of same priority are given equal treatment by the java scheduler. Default priority values as follows	2 M for define Thread priority 2 M for

The thread class defines several priority constants as: -

default priority values

MIN_PRIORITY =1

NORM_PRIORITY = 5

MAX_PRIORITY = 10

Thread priorities can take value from 1-10.

2 M for method to set and change

getPriority(): The java.lang.Thread.getPriority() method returns the priority of the given thread.

setPriority(int newPriority): The java.lang.Thread.setPriority() method updates or assign the priority of the thread to newPriority. The method throws IllegalArgumentException if the value newPriority goes out of the range, which is 1 (minimum) to 10 (maximum).

```
        import java.lang.*;

        public class ThreadPriorityExample extends Thread
        {
         public void run()
        {
        System.out.println("Inside the run() method");
        }
         public static void main(String argvs[])
        {
        ThreadPriorityExample th1 = new ThreadPriorityExample();
        ThreadPriorityExample th2 = new ThreadPriorityExample();
        ThreadPriorityExample th3 = new ThreadPriorityExample();
          System.out.println("Priority of the thread th1 is : " + th1.getPriority());
          System.out.println("Priority of the thread th2 is : " + th2.getPriority());
          System.out.println("Priority of the thread th2 is : " + th2.getPriority());
         th1.setPriority(6);
        th2.setPriority(3);
        th3.setPriority(9);
          System.out.println("Priority of the thread th1 is : " + th1.getPriority());
          System.out.println("Priority of the thread th2 is : " + th2.getPriority());
          System.out.println("Priority of the thread th3 is : " + th3.getPriority());
System.out.println("Currently Executing The Thread : " + Thread.currentThread().gtName());
System.out.println("Priority of the main thread is : " + Thread.currentThread().getPrority();
Thread.currentThread().setPriority(10);
        System.out.println("Priority of the main thread is : " + Thread.currentThread().getPiority());
        }
```

		}	
	c)	**Design an applet to perform all arithmetic operations and display the result by using labels. textboxes and buttons.**	**6 M**
	Ans		6 M for correct program

```
import java.awt.*;
import java.awt.event.*;
public class sample extends Frame implements ActionListener {
Label l1, l2,l3;
 TextField tf1, tf2, tf3;
   Button b1, b2, b3, b4;
sample() {
l1=new Lable(“First No.”);
      l1.setBounds(10, 10, 50, 20);
      tf1 = new TextField();
      tf1.setBounds(50, 50, 150, 20);
       l2=new Lable(“Second No.”);
      l2.setBounds(10, 60, 50, 20);
      tf2 = new TextField();
      tf2.setBounds(50, 100, 150, 20);
       l3=new Lable(“Result”);
      l3.setBounds(10, 110, 150, 20);
      tf3 = new TextField();
      tf3.setBounds(50, 150, 150, 20);
      tf3.setEditable(false);
      b1 = new Button("+");
      b1.setBounds(50, 200, 50, 50);
      b2 = new Button("-");
      b2.setBounds(120,200,50,50);
      b3 = new Button("*");
      b3.setBounds(220, 200, 50, 50);
      b4 = new Button("/");
      b4.setBounds(320,200,50,50);
      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);
      b4.addActionListener(this);

      add(tf1);
      add(tf2);
      add(tf3);
      add(b1);
      add(b2);
      add(b3);
      add(b4);
```

```
            setSize(400,400);
            setLayout(null);
            setVisible(true);
         }
        public void actionPerformed(ActionEvent e) {
   String s1 = tf1.getText();
            String s2 = tf2.getText();
            int a = Integer.parseInt(s1);
            int b = Integer.parseInt(s2);
            int c = 0;
            if (e.getSource() == b1){
               c = a + b;
            }
            else if (e.getSource() == b2){
               c = a - b;
            else if (e.getSource() == b3){
               c = a * b;
            else if (e.getSource() == b4){
               c = a / b;
            }
            String result = String.valueOf(c);
   tf3.setText(result);
         }
      public static void main(String[] args) {
         new sample();
      }
      }
```

SUMMER – 2023 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject Name: **Java Programming** **Subject Code:** 22412

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 judgement 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.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No.	**Sub Q. N.**	**Answer**	**Marking Scheme**
1		**Attempt any FIVE of the following:**	**10 M**
	a)	**Define the terms with example.** **i) Class** **ii) Object**	**2 M**

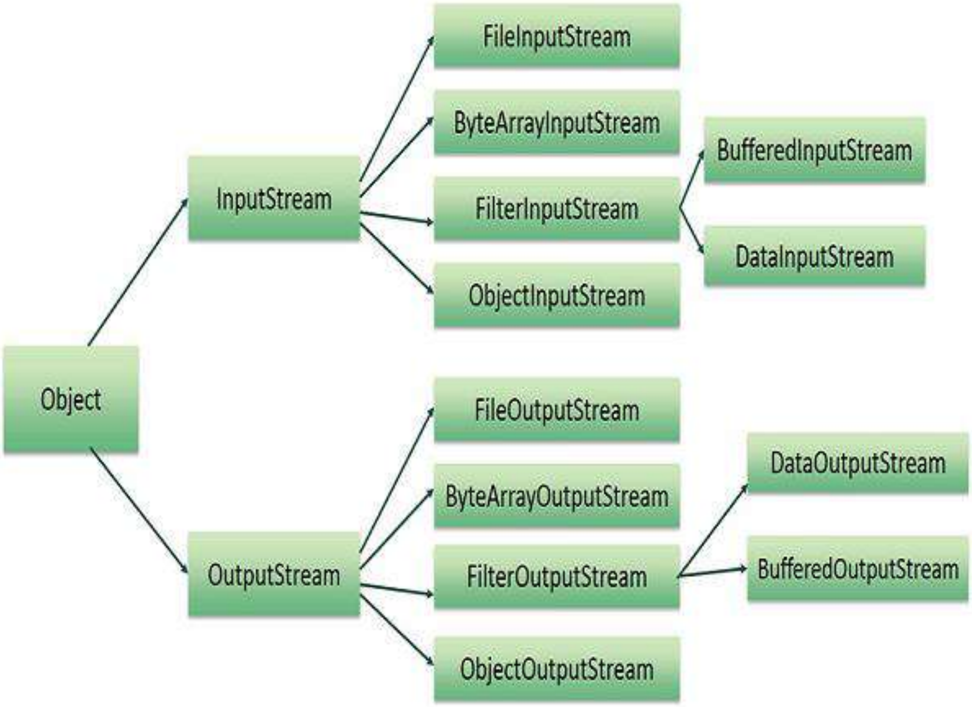
	Ans	**i) Class:** Class is a set of object, which shares common characteristics/ behavior and common properties/ attributes.
		ii) Object: It is a basic unit of Object-Oriented Programming and represents real-life entities.
		Example:
		class Student
		{
		int id;
		String name;
			1 M for any suitable class definition and 1 M for any suitable object definition

		public static void main(String args[])
		{
		Student s1=**new** Student(); //creating an object of Student
		}
		}
		In this example, we have created a Student class which has two data members id and name. We are creating the object of the Student s1 by new keyword.	

	b)	**Enlist any two access specifier with syntax.**	**2 M**
	Ans	There are 5 types of java access specifier: • public • private • default (Friendly) • protected • private protected	List any 2 access specifiers - 2M
	c)	**Give a syntax to create a package and accessing package in java.**	**2 M**
	Ans	**<u>To Create a package follow the steps given below:</u>** • Choose the name of the package • Include the package command as the first line of code in your Java Source File. • The Source file contains the classes, interfaces, etc. you want to include in the package • Compile to create the Java packages **Syntax to create a package:** package nameOfPackage; Example: package p1; **Accessing Package:** • Package can be accessed using keyword import. • There are 2 ways to access java system packages: ○ Package can be imported using import keyword and the wild card(*) but drawback of this shortcut approach is that it is difficult to determine from which package a particular member name. Syntax: import package_name.*;	syntax to create a package-1 M and accessing package-1 M

		For e.g. import java.lang.*; ○ The package can be accessed by using dot(.) operator and can be terminated using semicolon(;) Syntax: import package1.package2.classname;	
	d)	**Give a syntax of following thread method** **i) Notify ( )** **ii) Sleep ( )**	**2 M**
	Ans	i) notify() The **notify()** method of thread class is used to wake up a single thread. This method gives the notification **for only one thread** which is waiting for a particular object. Syntax: **public final void** notify() ii) sleep() Sleep() causes the current thread to suspend execution for a specified period. Syntax: public static void sleep(long milliseconds)	Syntax of notify( )-1 M and sleep ()-1 M
	e)	**Give a syntax of (param) tag to pass parameters to an applet.**	**2 M**

	Ans	User-define Parameter can be applied in applet using <PARAM…> tags. Each <PARAM…> tag has a name and value attribute.
		Syntax: <PARAM name = ……… Value = "………" >
		For example, the param tags for passing name and age parameters looks as shown below:
		<param name="name" value="Ramesh" />
		<param name="age" value="25″ />
		The *getParameter()* method of the *Applet* class can be used to retrieve the parameters passed from the HTML page. The syntax of *getParameter()* method is as follows:
		String getParameter(String param-name);
		Example:
		public void init()
		{
		n = getParameter("name");
		a = getParameter("age");
		}
			Syntax of <param> - 1 M
			And syntax of getParameter ( )- 1 M

	f)	**Define stream class and list types of stream class.**	**2 M**
	Ans	A java stream is a group of objects that can be piped together to produce the desired result. Streams are used in Java to transfer data between programs and I/O devices like a file, network connections, or consoles.  Object InputStream FileInputStream ByteArrayInputStream FilterInputStream ObjectInputStream BufferedInputStream DataInputStream OutputStream FileOutputStream ByteArrayOutputStream FilterOutputStream ObjectOutputStream DataOutputStream BufferedOutputStream Fig: Types of stream classes	Define stream class=1 M and any 2 types of stream class=1 M
	g)	**Give use of garbage collection in java.**	**2 M**
	Ans	The garbage collector provides the following uses: 1. Frees developers from having to manually release memory means destroy the unused objects 2. Allocates objects on the managed heap efficiently. 3. Reclaims objects that are no longer being used, clears their memory, and keeps the memory available for future allocations. 4. It is automatically done by the garbage collector(a part of JVM), so we don’t need extra effort.	Any 2 uses=2 M
2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Describe type casting in java with example.**	**4 M**
	Ans	In Java, **type casting** is a method or process that converts a data type into another data	Definition of

type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer.

Type casting is of two types: widening, narrowing.

Widening (Implicit)

- The process of assigning a smaller type to a larger one is known as widening or implicit.

  Byte → short → int → long → float → double

For e.g.

```
 class widening
{
public static void main(String arg[])
{
int i=100;
long l=I;
float f=l;
System.out.println(“Int value is”+i);
System.out.println(“Long value is”+l);
System.out.println(“Float value is”+f);
}
}
```

Narrowing (Explicit)

- The process of assigning a larger type into a smaller one is called narrowing.
- Casting into a smaller type may result in loss of data.
- double → long → int → short → byte

For e.g.

```
class narrowing
{
Public static void main(String[])
{
Double d=100.04;
Long l=(long) d;
Int i=(int) l;
```

type casting-1 M

Types of type casting-1 M

Example-2 M

(1 M for each example)

```
                System.out.println(“Int value is”+i);
                System.out.println(“Long value is”+l);
                System.out.println(“Float value is”
                }
}
```

b) **Differentiate between String and String Buffer Class. (any four points)** **4 M**

Ans

String class	StringBuffer class
String is a major class	StringBuffer is a peer class of String
Length is fixed (immutable)	Length is flexible (mutable)
Contents of object cannot be modified	Contents of object can be modified
Object can be created by assigning String constants enclosed in double quotes.	Objects can be created by calling constructor ofStringBuffer class using “new”
Methods of string class: toLowerCase(), toUpperCase(), replace(), trim(), equals(), length(), charAt(), concat(), substring(), compareTo()	Methods of StringBuffer class: setCharAt(), append(), insert(), append(), reverse(), delete()
Ex:- String s=”abc”‖;	Ex:- StringBuffer s=new StringBuffer (“abc”);

Any 4 correct points-4 M

c) **Write a program to create a user defined exception in java.** **4 M**

Ans **Following example shows a user defined exception as ‘Invalid Age’, if age entered by the user is less than eighteen.**

For any Correct program-4 M

```
import java.lang.Exception;
import java.io.*;
class myException extends Exception
{
myException(String msg)
{
super(msg);
}
}
class agetest
{
public static void main(String args[])
{
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
//Scanner class is also valid
try
{
System.out.println("enter the age : ");
int n=Integer.parseInt(br.readLine());
if(n < 18 )
throw new myException("Invalid Age"); //user defined exception
```

		else
		System.out.println("Valid age");
		}
		catch(myException e)
		{
		System.out.println(e.getMessage());
		}
		catch(IOException ie)
		{}
		}
		}	

	d)	**Write a program for reading and writing character to and from the given files using character stream classes.**	**4 M**

	Ans	import java.io.FileWriter;
		import java.io.IOException;
		public class IOStreamsExample {
		  public static void main(String args[]) throws IOException {
		    //Creating FileReader object
		    File file = new File("D:/myFile.txt");
		    FileReader reader = new FileReader(file);
		    char chars[] = new char[(int) file.length()];
		    //Reading data from the file
		    reader.read(chars);
		    //Writing data to another file
		    File out = new File("D:/CopyOfmyFile.txt");
		    FileWriter writer = new FileWriter(out);
		    //Writing data to the file
		    writer.write(chars);
		    writer.flush();
		    System.out.println("Data successfully written in the specified file");
		  }
		    }
			4 M (for any correct program and logic)

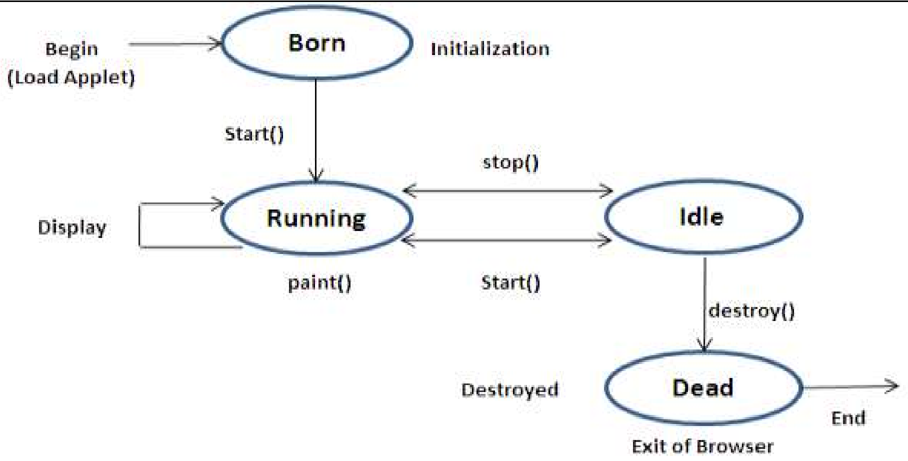
3.		**Attempt any THREE of the following:**	**12 M**
	a)	**Write a program to print all the Armstrong numbers from 0 to 999**	**4 M**

	Ans	import java.util.Scanner;
		class ArmstrongWhile
		{
		        public static void main(String[] arg)
		        {
		        int i=0,arm;
		        System.out.println("Armstrong numbers between 0 to 999");
		        while(i<1000)
			Correct logic – 4 M

```
        {
        arm=armstrongOrNot(i);
        if(arm==i)
        System.out.println(i);
        i++;
        }
        }
static int armstrongOrNot(int num)
{
        int x,a=0;
        while(num!=0)
        {
                x=num%10;
                a=a+(x*x*x);
                num/=10 ;
        }
        return a;
}
}

OR
class ArmstrongWhile
{
        public static void main(String[] arg)
        {
        int i=1,a,arm,n,temp;
        System.out.println("Armstrong numbers between 0 to 999 are");
        while(i<500)
        {
        n=i;
        arm=0;
        while(n>0)
        {
                a=n%10;
                arm=arm+(a*a*a);
                n=n/10;
        }
        if(arm==i)
                System.out.println(i);
        i++;
        }
        }
}
```

	b)	**Explain the applet life cycle with neat diagram.**	**4 M**

Ans



Applet Life Cycle: An Applet has a life cycle, which describes how it starts, how it operates and how it ends. The life cycle consists of four methods: init(), start(), stop() and destroy().

Initialization State (The init() method):

The life cycle of an Applet is begin on that time when the applet is first loaded into the browser and called the init() method. The init() method is called only one time in the life cycle on an Applet. The init() method is basically called to read the “PARAM” tag in the html file. The init () method retrieve the passed parameter through the “PARAM” tag of html file using get Parameter() method All the initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method .After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method.

We may do following thing if required.

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

Running State (The start() method): The start method of an Applet is called after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted. For Example if the user wants to return to the Applet, in this situation the start() method of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.

```
public void start()
{
……..
……..
}
```

Idle (The Stop() method): An applet becomes idle when it is stopped from running. The stop() method stops the applet and makes it invisible. Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the stop() method explicitly. The stop() method can be called multiple times in the life cycle of applet like the start () method or should be called at least one time. For example the stop() method is called by the web browser on that time When the user leaves one applet to go another applet and the start() method is called on that time when the user wants to go back into the first program or Applet.

```
 public void stop()
```

Diagram-2 M and explanation – 2 M

		{
		……..
		……..
		}
		Dead State (The destroy() method): The destroy() method is called to terminate an Applet. An Applet is said to be dead when it is removed from memory. This occurs automatically by invoking the destroy() method when we quit the browser. It is useful for clean-up actions, such as releasing memory after the applet is removed, killing off threads and closing network/database connections.
		Thus this method releases all the resources that were initialized during an applet's initialization.
		public void destroy()
		{
		……..
		……..
		}
		Display State (The paint() method): The paint() method is used for applet display on the screen.
		The display includes text, images, graphics and background. This happens immediately after the applet enters into the running state. Almost every applet will have a paint() method and can be called several times during an applet's life cycle. The paint() method is called whenever a window is required to paint or repaint the applet.
		public void paint(Graphics g)
		{
		……..
		……..
		}	

	c)	**Describe the package in java with suitable example.**	**4 M**
	Ans	• Java provides a mechanism for partitioning the class namespace into more manageable parts called package (i.e package are container for a classes). • The package is both naming and visibility controlled mechanism. Package can be created by including package as the first statement in java source code. • Any classes declared within that file will belong to the specified package. Syntax: package pkg; pkg is the name of the package eg : package mypack; • Java uses file system directories to store packages. The class files of any classes which are declared in a • package must be stored in a directory which has same name as package name. • The directory must match with the package name exactly. • A hierarchy can be created by separating package name and sub package name by a period(.) as pkg1.pkg2.pkg3; which requires a directory structure as pkg1\pkg2\pkg3. • The classes and methods of a package must be public. • 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|*);	Description-2 M, Example -2 M

- Example:

```
    package1:
    package package1;
    public class Box
    {
    int l= 5;
    int b = 7;
    int h = 8;
    public void display()
    {
    System.out.println("Volume is:"+(l*b*h));
    }
    }
}
Source file:
import package1.Box;
class VolumeDemo
{
public static void main(String args[])
{
Box b=new Box();
b.display();
}
}
```

d) **Enlist types of Byte stream class and describe input stream class and output stream class.** **4 M**

Ans

Type – 1 M, Explanation -3 M

- Byte streams class: It handles I/O operations on bytes.
- InputStream and OutputStream classes are operated on bytes for reading and writing, respectively.
- Byte streams are used in a program to read and write 8-bit bytes.
- InputStream and OutputStream are the abstract super classes of all byte streams that have a sequential nature.
- The stream is unidirectional; they can transmit bytes in only one direction.
- InputStream and OutputStream provide the Application program Interface (API) and

  partial implementation for input streams (streams that read bytes) and output streams (streams that write bytes).
- Input Stream Classes: java.io.InputStream is an abstract class that contains the basic methods for reading raw bytes of data from a stream. The InputStream class defines methods for performing the input functions like: reading bytes, closing streams, marking positions in streams, skipping ahead in a stream and finding the number of bytes in a stream.
- **Input stream class methods:**

  1. int read ()- Returns an integer representation of next available byte of input.-1 is returned at the stream end.

  2. int read (byte buffer[ ])- Read up to buffer.length bytes into buffer & returns actual number

		of bytes that are read. At the end returns –1. 3. int read(byte buffer[ ], int offset, int numbytes)- Attempts to read up to numbytes bytes into buffer starting at buffer[offset]. Returns actual number of bytes that are read. At the end returns –1. 4. void close()- to close the input stream 5. void mark(int numbytes)- places a mark at current point in input stream and remain valid till number of bytes are read. 6. void reset()- Resets pointer to previously set mark/ goes back to stream beginning. 7. long skip(long numbytes)- skips number of bytes. 8. int available()- Returns number of bytes currently available for reading. • **Output Stream Classes:** • The java.io.OutputStream class sends raw bytes of data to a target such as the console or a network server. Like InputStream, OutputStream is an abstract class. • The OutputStream includes methods that perform operations like: writing bytes, closing streams,flushing streams etc. • Methods defines by the OutputStream class are 1. void close() - to close the OutputStream 2. void write (int b) - Writes a single byte to an output stream. 3. void write(byte buffer[ ]) - Writes a complete array of bytes to an output stream. 4. void write (byte buffer[ ], int offset, int numbytes) - Writes a sub range of numbytes bytes from the array buffer, beginning at buffer[offset]. 5. void flush() - clears the buffer.	
4.		**Attempt any THREE of the following:**	**12 M**
	a)	**Describe any four features of java.**	**4 M**
	Ans	**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	Any four each features -1 M each

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.

b) **Explain any four methods of vector class with example.** **4 M**

Ans Vector class is in java.util package of java.

1 Method – 1 M

Vector is dynamic array which can grow automatically according to the requirement.

Vector does not require any fix dimension like String array and int array.

Vectors are used to store objects that do not have to be homogeneous.

Vector contains many useful methods.

Vectors are created like arrays. It has three constructor methods

- Vector list = new Vector(); //declaring vector without size
- Vector list = new Vector(3); //declaring vector with size
- Vector list = new Vector(5,2); //create vector with initial size and whenever it need to grows, it grows by value specified by increment capacity.

Method Name	**Task performed**
list.firstElement()	It returns the first element of the vector.
list.lastElement()	It returns last element of the vector
list.addElement(item)	Adds the item specified to the list at the end.
list.elementAt(n)	Gives the name of the object at nth position
list.size()	Gives the number of objects present in vector
List.capacity()	This method returns the current capacity of the vector.
list.removeElement(item)	Removes the specified item from the list.
list.removeElementAt(n)	Removes the item stored in the nth position of the list.
list.removeAllElements()	Removes all the elements in the list.
list.insertElementAt(item, n)	Inserts the item at nth position.
List.contains(object element)	This method checks whether the specified element is present inthe Vector. If the element is been found it returns true else false.
list.copyInto(array)	Copies all items from list of array.

Example:

```
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);
}
}
```

	c)	**Describe interface in java with suitable example.**	**4 M**

	Ans	Java does not support multiple inheritances with only classes. Java provides an alternate approach known as interface to support concept of multiple inheritance. An interface is similar to class which can define only abstract methods and final variables.
		Syntax:
		access interface InterfaceName
		{
		Variables declaration;
		Methods declaration;
			Interface explanation – 2 M, any suitable example – 2 M

```
}
```

Example:

```
interface sports
{
int sport_wt=5;
public void disp();
}
class test
{
int roll_no;
String name;
int m1,m2;
test(int r, String nm, int m11,int m12)
{
roll_no=r;
name=nm;
m1=m11;
m2=m12;
}
}
class result extends test implements sports
{
result (int r, String nm, int m11,int m12)
{
super (r,nm,m11,m12);
}
public void disp()
{
System.out.println("Roll no : "+roll_no);
System.out.println("Name : "+name);
System.out.println("sub1 : "+m1);
System.out.println("sub2 : "+m2);
System.out.println("sport_wt : "+sport_wt);
int t=m1+m2+sport_wt;
System.out.println("total : "+t);
}
public static void main(String args[])
{
result r= new result(101,"abc",75,75);
r.disp();
}
}
```

Output :

```
D:\>java result
Roll no : 101
Name : abc
sub1 : 75
sub2 : 75
sport_wt : 5
```

total : 155

d) **Write an applet program for following graphics method.**

i) Drawoval ( )

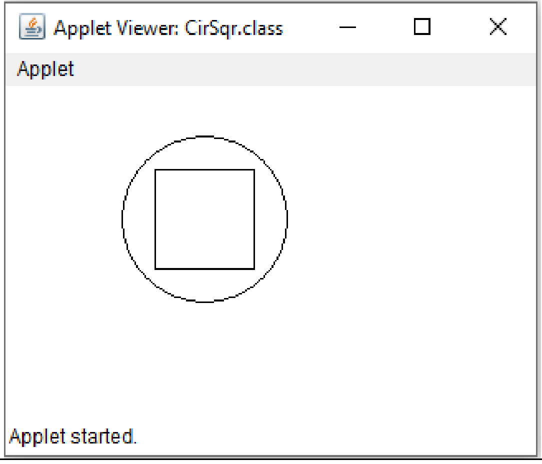
ii) Drawline ( )

4 M

Ans

```
import java.awt.*;
import java.applet.*;
public class CirSqr extends Applet
{
public void paint(Graphics g)
{
g.drawOval(70,30,100,100);
g.drawRect(90,50,60,60);
}
}
/*<applet code="CirSqr.class" height=200 width=200>
</applet> */
```

Output:



Applet started.

Any correct logic can be considered 2 M for drawoval and 2 M for drawline

e) **Enlist any four methods of file input stream class and give syntax of any two methods.**

4 M

Ans

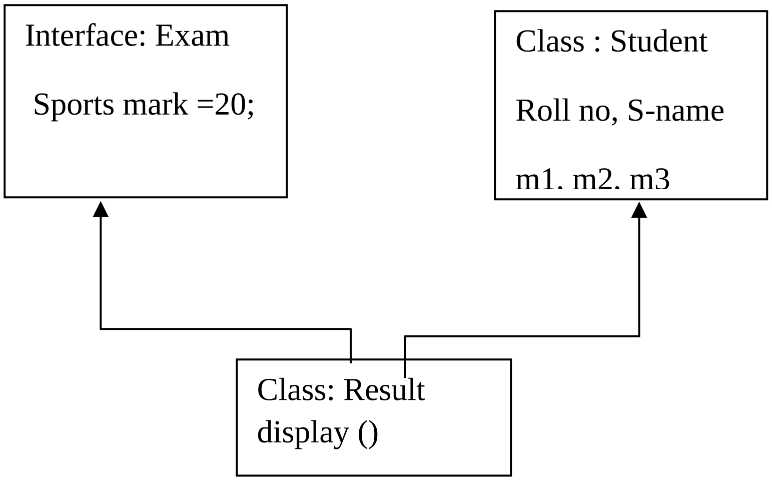
- Input Stream Classes: java.io.InputStream is an abstract class that contains the basic methods for reading raw bytes of data from a stream. The InputStream class defines methods for performing the input functions like: reading bytes, closing streams, marking positions in streams, skipping ahead in a stream and finding the number of bytes in a stream.
- Input stream class methods:
  1. int read ()- Returns an integer representation of next available byte of input.-1 is returned at the stream end.
  2. int read (byte buffer[ ])- Read up to buffer.length bytes into buffer & returns actual number
  of bytes that are read. At the end returns –1.
  3. int read(byte buffer[ ], int offset, int numbytes)- Attempts to read up to numbytes bytes
  into buffer starting at buffer[offset]. Returns actual number of bytes that are read.

One method – 1 M

		At the end returns –1. 4. void close()- to close the input stream 5. void mark(int numbytes)- places a mark at current point in input stream and remain valid till number of bytes are read. 6. void reset()- Resets pointer to previously set mark/ goes back to stream beginning. 7. long skip(long numbytes)- skips number of bytes. 8. int available()- Returns number of bytes currently available for reading.	
5.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Write a program to copy all elements of one array into another array.**	**6 M**
	Ans		6 M for any correct program and logic

```
public class CopyArray {
  public static void main(String[] args)
{
     int [] arr1 = new int [] {1, 2, 3, 4, 5};

      int arr2[] = new int[arr1.length];

            for (int i = 0; i < arr1.length; i++)
        {
              arr2[i] = arr1[i];
            }
            System.out.println("Elements of original array: ");

         for (int i = 0; i < arr1.length; i++)
        {
              System.out.print(arr1[i] + " ");
            }

            System.out.println();

            System.out.println("Elements of new array: ");

        for (int i = 0; i < arr2.length; i++)
        {
              System.out.print(arr2[i] + " ");
```

		}
		      }
		}	

	b)	**Write a program to implement the following inheritance. Refer Fig. No. 1.**  Interface: Exam Sports mark =20; Class : Student Roll no, S-name m1. m2. m3 Class: Result display () **Fig. No. 1**	**6 M**

	Ans	interface Exam
		{
		 int sports_mark=20;
		}
		class Student
		{
		        String S-name;
		        int Roll_no, m1, m2, m3;
		        Student(String n, int a, int b, int c, int d)
		{
		S-name = n;
		Roll_no = a;
			6 M for correct program

```
m1 = b;
m2 = c;
m3 = d;
}
void showdata()
{
System.out.println("Name of student :"+S-name);
System.out.println("Roll no. of the students :"+Roll_no);
System.out.println(“Marks of subject 1:”+m1);
System.out.println(“Marks of subject 2:”+m2);
System.out.println(“Marks of subject 3:”+m3);
}
}
class Result extends Student implements Exam
{
Result(String n, int a, int b, int c, int d)
{
super(n, a, b, c, d );
}
void dispaly()
{
super.showdata();
int total=(m1+m2+m3);
float result=(total+Sports_mark)/total*100;
```

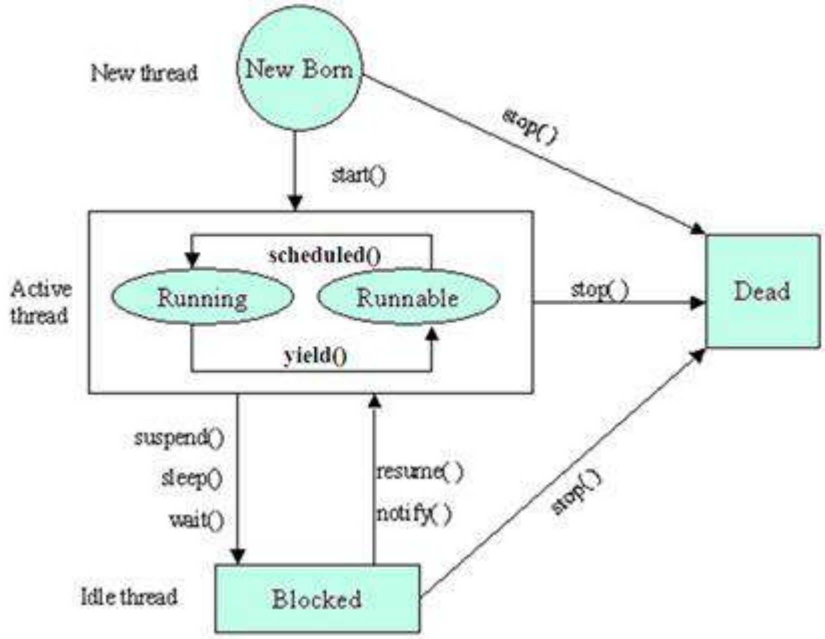
		System.out.println(“result of student is:”+result);
		}
		}
		class studentsDetails
		{
		public static void main(String args[])
		{
		Result r=new Result("Sachin",14, 78, 85, 97);
		r.display();
		}
		}	

	c)	**Write a program to print even and odd number using two threads with delay of 1000ms after each number.**	**6 M**

	Ans	class odd extends Thread
		{
		public void run()
		{
		for(int i=1;i<=20;i=i+2)
		{
		System.out.println("ODD="+i);
		try
		{
		sleep(1000);
		}
		catch(Exception e)
		{
		System.out.println("Error");
			6 M for correct program

```
}
}
}
}
class even extends Thread
{
public void run()
{
for(int i=0;i<=20;i=i+2)
{
System.out.println("EVEN="+i);
try
{
sleep(1000);
}
catch(Exception e)
{
System.out.println("Error");
}
}
}
}
class oddeven
{
public static void main(String arg[])
{
odd o=new odd();
even e=new even();
```

		o.start();
		e.start();
		}
		}	

6.		**Attempt any <u>TWO</u> of the following:**	**12 M**
	a)	**Explain thread life cycle with neat diagram.**	**6 M**
	Ans	 New thread New Born stop( ) start() scheduled() Active thread Running Runnable yield() stop( ) Dead suspend() sleep() wait() resume( ) notify( ) stop() Idle thread Blocked Thread Life Cycle Thread has five different states throughout its life. **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().	2 M for diagram, 4 M for explanation

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.

o suspend() : Thread can be suspended by this method. It can be rescheduled by resume().

o 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().

o 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 stop() 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

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

b) **Write a program to generate following output using drawline () method. Refer Fig. No. 2.** **6 M**

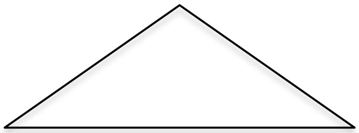


Fig. No. 2

Ans **Using drawLine() method** 6 M for correct program

```
import java.applet.*;
import java.awt.*;
public class Triangle extends Applet
{
public void paint(Graphics g)
{
g.drawLine(100,200,200,100);
g.drawLine(200,100,300,200);
g.drawLine(300,200,100,200);
}
}
/*<applet code="Triangle.class" height=300 width=200> </applet>*/
```

OR

Using drawPolygon() method

```
import java.applet.*;
import java.awt.*;
public class Triangle extends Applet
{
public void paint(Graphics g)
{
int a[]={100,200,300,100};
int b[]={200,100,200,200};
int n=4;
g.drawPolygon(a,b,n);
}
}
/*<applet code="Triangle.class" height=300 width=200> </applet>*/
```

	c)	**Explain constructor with its type. Give an example of parameterized constructor.**	**6 M**

Ans

Constructor: A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

2 M for definition of constructor and types of constructors

Types of constructors are:

Default constructor : It is constructor which is inserted by Java compiler when no constructor is provided in class. Every class has constructor within it. Even abstract class have default constructor.

By default, Java compiler, insert the code for a zero parameter constructor.

4 M for example (for any correct suitable program of parameterize

d
constructor)

Default constructor is the no arguments constructor automatically generated unless you define another constructor.

The default constructor automatically initializes all numeric members to zero and other types to null or spaces.

```
class Rect
{
 int length, breadth;
Rect() //constructor
{
length=4;
breadth=5;
}
public static void main(String args[])
{
Rect r = new Rect();
System.out.println("Area : " +(r.length*r.breadth));
}
}
```

Parameterized constructor: Constructor which have arguments are known as 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 of Parameterized Constructor

```
class Rect
 {
 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));
```

		System.out.println(“Area : ” +(r1.length*r1.breadth));
		}
		}	

22412

23124

3 Hours / 70 Marks Seat No. □□□□□□□□

Instructions – (1) All Questions are *Compulsory.*

(2) Answer each next main Question on a new page.

(3) Illustrate your answers with neat sketches wherever necessary.

(4) Figures to the right indicate full marks.

(5) Assume suitable data, if necessary.

(6) Mobile Phone, Pager and any other Electronic Communication devices are not permissible in Examination Hall.

Marks

1. Attempt any FIVE of the following : 10

a) Enlist any two logical operators and two bitwise operators.

b) Define constructor.

c) Write down the syntax of array declaration, initialization.

d) List out different ways to access package from another package.

e) Differentiate between starting thread with run( ) method and start( ) method.

f) State the classes that can an applet extend.

g) Give syntax to open a file using Inputstream class.

Marks

2. Attempt any THREE of the following : 12

a) Write a program to display ASCII value of a number 9.

b) Write a program to sort the elements of an array in ascending order.

c) Define Thread. Draw life cycle of Thread.

d) Write a program to read a file and then count number of words.

3. Attempt any THREE of the following : 12

a) Write a program which displays functioning of ATM machine, (Hint : Withdraw, Deposit, Check Balance and Exit)

b) Differentiate between method overloading and method overriding.

c) Explain applet life cycle in detail.

d) Differentiate between Byte Stream Class and Character Stream Class. (Any four points)

4. Attempt any THREE of the following : 12

a) Explain implicit and explicit type conversion with example in detail.

b) Write a program to show the use of copy constructor.

c) Write a program to show the Hierarchical inheritance.

d) Explain any four font methods with example.

e) Write a program to append content of one file into another file.

Marks

5. Attempt any TWO of the following : 12

a) Explain vector with the help of example. Explain any 3 methods of vector class.

b) Develop and Interest Interface which contains Simple Interest and Compound Interest methods and static final field of rate 25%. Write a class to implement those methods.

c) Write a program that throws an exception called "NoMatchException" when a string is not equal to "India".

6. Attempt any TWO of the following : 12

a) Write a program to print the sum, difference and product of two complex numbers by creating a class named "Complex" with separate methods for each operation whose real and imaginary parts are entered by user.

b) i) Explain Errors and its types in detail.

ii) Explain thread methods to set and get priority.

c) Write a program to draw a chessboard in Java Applet.

22412

23242

3 Hours / 70 Marks Seat No. □□□□□□□□

Instructions – (1) All Questions are *Compulsory.*

(2) Answer each next main Question on a new page.

(3) Illustrate your answers with neat sketches wherever necessary.

(4) Figures to the right indicate full marks.

(5) Assume suitable data, if necessary.

(6) Mobile Phone, Pager and any other Electronic Communication devices are not permissible in Examination Hall.

Marks

1. Attempt any FIVE of the following: 10

a) State the significance of Java Virtual Machine (JVM) in the Java programming environment.

b) Define array. List its types.

c) State use of finalize( ) method with its syntax.

d) Define the interface in Java. Write the syntax.

e) Define thread. Mention 2 ways to create thread.

f) Write syntax of draw Rect( ).

g) Give the use of <PARAM> tag in applet.

2. Attempt any THREE of the following: 12

a) Explain the concept of platform independence in Java and discuss how it is achieved. Give example to illustrate the concept.

b) What happens if you don't define any constructor in a class? Can you still create objects of that class? Explain with example.

c) Write a Java program in which thread A will display the even numbers between 1 to 50 and thread B will display the odd numbers between 1 to 50. After 3 iterations thread A should go to sleep for 500 ms.

d) Explain multilevel inheritance with example.

Marks

3. Attempt any THREE of the following: 12

a) Define a class employee with data members 'empid', 'name' and 'salary'. Accept data for three objects and display it.

b) How can the "super" keyword be used in inheritance? Give an example to demonstrate its usage.

c) Explain the following with syntax:

i) drawLine

ii) drawOval

iii) drawArc

iv) drawString

d) What is the concept of streams in Java? How do streams facilitate input and output operations?

4. Attempt any THREE of the following: 12

a) Explain any two logical operators in Java with example.

b) What is constructor? List types of constructor. Explain parameterized constructor with suitable example.

c) Implement the following inheritance. Refer Fig. No. 01.

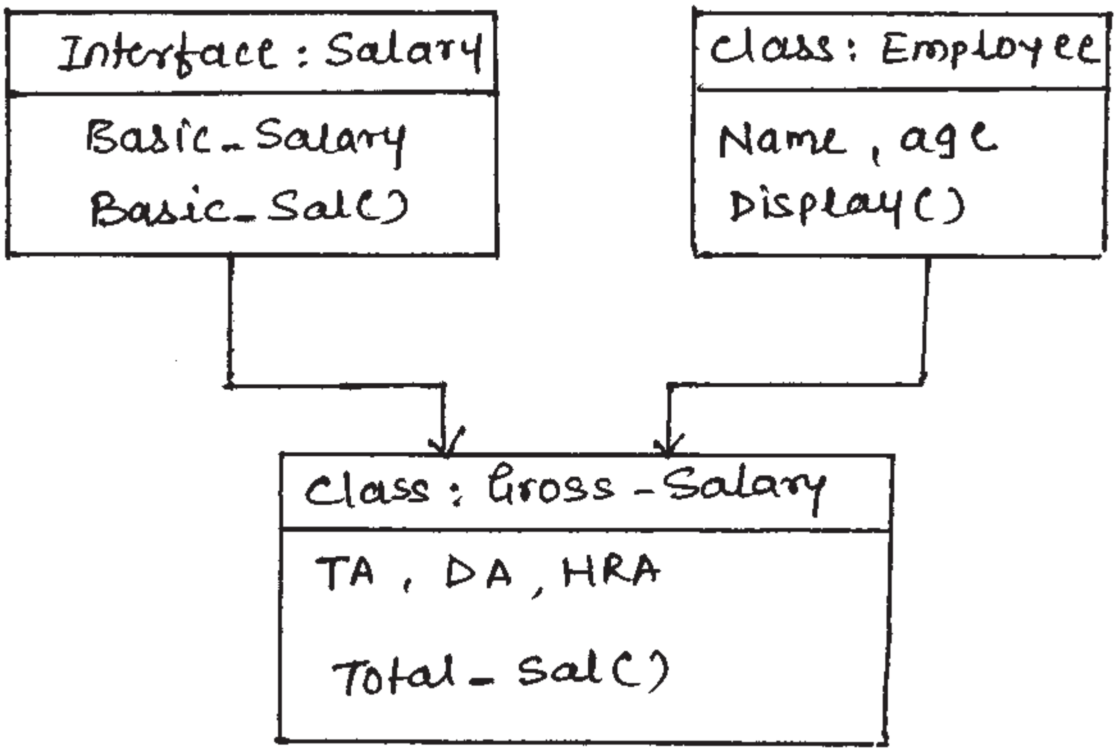


Fig. No. 01

Marks

d) Explain Life Cycle of the applet with neat diagram.

e) Create a new test file named "data.txt" using the **File** class. Write a program to count number of words of "data.txt" using stream classes.

5. Attempt any TWO of the following: 12

a) Explain the concept of argument passing and the usage of 'this' keyword in Java give example to illustrate their usage and benefits.

b) Explain the concept of packages in Java and their significance in software development. Write an example to illustrate the usage and benefits of using packages.

c) Explain the concept of exception handling in Java and its importance in robust programming. Provide an example to illustrate the implementation and benefits of exception handling.

6. Attempt any TWO of the following: 12

a) Write a program to define class Employee with members as id and salary. Accept data for five employees and display details of employees getting highest salary.

b) Define exception called 'No Match Exception' that is thrown when the password accepted is not equal to 'MSBTE'. Write the program.

c) Write a program to design an Applet showing three concentric circles filled with three different colors.

MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2013 Certified)

Winter-2024 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject Name: **Java Programming** **Subject Code:** **22412**

Important Instructions to examiners:

1) The answers should be examined by key words and not as word-to-word as given in the model answerscheme.
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 (Notapplicable 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 equivalentfigure 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 judgement 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.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1.		**Attempt any <u>FIVE</u> of the following:**	**10 M**
	a)	**Enlist any four features of java.**	**2M**
	Ans.	**Features of Java:** 1. Compiled and Interpreted 2. Platform independent and Portable 3. Object-Oriented 4. Robust and Secure 5. Distributed 6. Simple, Small and familiar 7. Multithreaded and Interactive 8. High Performance 9. Dynamic and Extensible 10. Ease of Development 11. Scalability and Performance 12. Monitoring and Manageability	**Any four correct features ½ M each**
	b)	**Write the use of `this' keyword. `**	**2M**
	Ans.	In Java, ‘this’ is a reference variable that refers to the current object. It can be used to call current class methods and fields, to pass an instance of the current class as a parameter and to differentiate between the local and instance variables. Using “this” reference can improve code readability and reduce naming conflicts.	**2M for Correct use of ‘this’ keyword**
	c)	**Write the use of super keyword.**	**2M**
	Ans.	**It's primarily used in two scenarios:** 1. **Calling Parent Class Constructor:** To invoke the parent class's constructor from the child class's constructor.	**Two uses 1M for each**

Q. No.	Sub Q. N.	Answer	Marking Scheme
		2. **Accessing Parent Class Members:** To access members (variables or methods) of the parent class, especially when there's a name conflict with members in the child class.	
	d)	**Explain the use of throw and throws.**	**2M**
	Ans.	• **throw:** All methods in java use the 'throw' statement explicitly to throw an exception from a method or any block of code. The throw is a keyword in java. Throw can be used for either checked or unchecked exception. It is mainly used to throw custom exceptions. • **throws:** It is a keyword in java. It is used to declare an exception. It is used with a method signature. Multiple exceptions can be declared through throws.	**1M for each explanation**
	e)	**Enlist the life cycle methods of Applet.**	**2M**
	Ans.	1. init() 2. start() 3. stop() 4. paint() 5. destroy()	**Any four correct methods ½ M each**
	f)	**Give two methods of file class.**	**2M**

	Ans.	1. **boolean createNewFile():** It creates a new, empty file named by this abstract pathname automatically, if and only if no file with the same name exists.
		`if(file.createNewFile())`
		`System.out.println("A new file is successfully created.");`
		2. **String getName():** It returns the name of the file or directory denoted by the objects abstract pathname.
		`System.out.println("File name : " + file.getName());`
		3. String getParent(): It returns the parents pathname string of the objects abstract pathname or null if the pathname does not name a parent directory.
		`System.out.println("Parent name : " + file.getParent());`
		4. boolean isFile(): It returns true if the file denoted by the abstract pathname is a normal file and false if it is not a normal file.
		`System.out.println("File size (bytes) : " + file.isFile());`
			Any two correct methods 1M for each

Q. No.	Sub Q. N.	Answer	Marking Scheme

		5. boolean canRead(): It returns true if the application can read the file denoted by the abstract pathname and returns false otherwise.
		System.out.println("Is file readable : " + file.canRead());
		6. boolean canWrite(): It returns true if the application can modify the file denoted by the abstract pathname, and returns false otherwise.
		System.out.println("Is file writeable : " + file.canWrite());
		7. boolean canExecute(): It returns true if the application can execute the file denoted by the abstract pathname, and returns false otherwise.
		System.out.println("Is file executable : " + file.canExecute());	

	g)	**Write the use of <PARAM> tag in an applet.**	**2M**
	Ans.	It is used for supply user defined parameters to an applet using <PARAM…> Tag has a name attribute such as color, and a value attribute such as red. Inside the applet code the applet code can refer to that parameter by name to find its value. **Syntax:** <APPLET> <PARAM =color VALUE= “red”> </APPLET>	**2M for correct use**
2.		**Attempt any THREE of the following :**	**12M**
	a)	**Write a program using sqrt() and pow() to calculate the square root and power of given number.**	**4M**

	Ans.	**Note: Any other correct logic shall be considered.**
		import java.util.Scanner;
		public class SquareRootPower {
		public static void main(String[] args) {
		Scanner scanner = new Scanner(System.in);
		System.out.print("Enter a number for Square root:");
		double number1 = scanner.nextDouble();
		// Calculate the square root
		double squareRoot = Math.sqrt(number1);
		System.out.println("Square root of " + number1 + " is: " + squareRoot);
		System.out.print("Enter a number for power: ");
		double number2 = scanner.nextInt();
			2M each for calculating the sqrt() & pow()

Q. No.	Sub Q. N.	Answer	Marking Scheme
		(continued below)	

```
        // Calculate the power of 2
        double powerOfTwo = Math.pow(number2, 2);

        System.out.println(number2 + " raised to the power of 2 is: " + powerOfTwo);

        scanner.close();
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	b)	**Write a program to accept four numbers from user using command line arguments and print the smallest number.**	**4M**
	Ans.	**Note: Any other correct logic shall be considered.**	**2M for each for accepting & printing smallest number**

```
public class SmallestNumber {
    public static void main(String[] args) {
        if (args.length != 4) {
            System.out.println("Please provide exactly 4 numbers as command-line
arguments.");
            return;
        }

        int num1 = Integer.parseInt(args[0]);
        int num2 = Integer.parseInt(args[1]);
        int num3 = Integer.parseInt(args[2]);
        int num4 = Integer.parseInt(args[3]);

        int smallest = num1;

        if (num2 < smallest) {
            smallest = num2;
        }

        if (num3 < smallest) {
            smallest = num3;
        }

        if (num4 < smallest) {
            smallest = num4;
        }

        System.out.println("The smallest number is: " + smallest);
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	c)	**Explain two ways of creating threads in java with suitable example.**	**4M**
	Ans.	Thread is an independent path of execution within a program.	**2M each way, for**

Q. No.	Sub Q. N.	Answer	Marking Scheme

There are two ways to create a thread:

1. **By extending the Thread class.**
   - In this, to create a thread is to create new class that extends thread and then to create an instance of that class.
   - **Steps :**
     1. Declare the class as extending the thread class.
     2. Implement the run() method that is responsible for executing the sequence of code that the thread with execute. The extending class must override the run() method, which is the entry point of new thread.
     3. Create a thread object and call the start() method to initiate the thread execution.

Example:

```
class MyThread extends Thread
{
        public void run()
        {
                for(int i = 1;i<=20;i++)
                {
                System.out.println(i);
                }
        }
        public static void main(String a[])
        {
                MyThread t = new MyThread();
                t.start();
        }
}
```

2. **By implementing the runnable interface.**
   - The runnable interface declares the run() method that is required for implementing thread in our programs.
   - **Steps:**
     1. Declare a class as implementing the runnable interface.
     2. Implement run() method.
     3. Create a thread by defining an object that is instantiated from this “runnable” class as the target of the thread.
     4. Call the thread’s start() method to run the thread.

Example:

```
class MyThread implements Runnable
{
        public void run()
        {
                for(int i = 1;i<=20;i++)
                {
                System.out.println(i);
```

explaining of creating threads with example

Q. No.	**Sub Q. N.**	**Answer**	**Marking Scheme**

		}
		}
		public static void main(String a[])
		{
		MyThread m = new MyThread();
		Thread t = new Thread(m);
		t.start();
		}
		}	

	d)	**Give the usage of following :** **i) set Color()** **ii) get Color()** **iii) set Foreground()** **iv) set Background**	**4M**
	Ans.	**i) set Color()** This method sets the drawing color. **Syntax:** void setColor(Color c) **ii) get Color()** This method Retrieves the current drawing color. **Syntax:** void getColor(String name) **iii) set Foreground()** This method changes the color of the text or foreground content. **Syntax:** void setForground(Color newColor) **iv) set Background()** This method used to set the background color of a component. **Syntax:** void setBackground(Color newColor)	**1M each method usage**
3.		**Attempt any THREE of the following:**	**12M**
	a)	**Write a java program to copy the contents of one file into another.**	**4M**

	Ans.	**Note: Any other correct logic shall be considered.**
		
		import java.io.*;
		class copyf
		{
		public static void main(String args[]) throws IOException
		{
		BufferedReader in=null;
		BufferedWriter out=null;
			2M Reading File & 2M copy contents to File

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
try
{
in=new BufferedReader(new FileReader("input.txt"));
out=new BufferedWriter(new FileWriter("output.txt"));
int c;
while((c=in.read())!=-1)
{
out.write(c);
}
System.out.println("File copied successfully");
}
finally
{
if(in!=null)
{
in.close();
}
if(out!=null)
{
out.close();
}
}
}
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	b)	**Write a program to create user defined exception "MIN-BAL". Throw the exception when balance in account is below 500 rupees.**	**4M**
	Ans.	**Note: Any other correct logic shall be considered.**	**2M Declaration of user defined exception & 2M Throw Exception with given condition**

```
class MinBalException extends Exception {
    public MinBalException(String message) {
        super(message);
    }
}

public class BankAccount {
    private String accountHolder;
    private double balance;

    public BankAccount(String accountHolder, double balance) {
        this.accountHolder = accountHolder;
        this.balance = balance;
    }

        public void withdraw(double amount) throws MinBalException {
        if (balance - amount < 500) {
            throw new MinBalException("Withdrawal denied! Balance cannot go below
Rs.500.");
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
        }
        balance -= amount;
        System.out.println("Withdrawal successful. Remaining balance: Rs." + balance);
    }

    public void displayBalance() {
        System.out.println("Account Holder: " + accountHolder);
        System.out.println("Current Balance: Rs." + balance);
    }

    public static void main(String[] args) {
        try {
            BankAccount account = new BankAccount("John Doe", 1000);

            account.displayBalance();

            System.out.println("\nAttempting to withdraw Rs. 600...");
            account.withdraw(600);


            System.out.println("\nAttempting to withdraw Rs. 100...");
            account.withdraw(100);

        } catch (MinBalException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
              }
```

c) **Define a class employee having data members as emp_id, name and salary. Accept and display the data for five employees.** **4M**

Ans **Note: Any other correct logic shall be considered.**

```
import java.util.Scanner;
class Employee {
    private int emp_id;
    private String name;
    private double salary;
    public Employee(int emp_id, String name, double salary) {
        this.emp_id = emp_id;
        this.name = name;
        this.salary = salary;
    }
    public void display() {
        System.out.println("Employee ID: " + emp_id + ", Name: " + name + ",
  Salary: " + salary);
    }
}
```

2M Accepting data & 2M Displaying data

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
public class EmployeeDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Employee[] employees = new Employee[5]; // Array to store 5 employee
  objects

        for (int i = 0; i < 5; i++) {
            System.out.println("\nEnter details for Employee " + (i + 1) + ":");
            System.out.print("Enter Employee ID: ");
            int emp_id = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character
            System.out.print("Enter Employee Name: ");
            String name = scanner.nextLine();
            System.out.print("Enter Salary: ");
            double salary = scanner.nextDouble();
            employees[i] = new Employee(emp_id, name, salary);
        }

        System.out.println("\nEmployee Details:");
        for (Employee emp : employees) {
            emp.display();
        }
        scanner.close();
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	d)	**Write the use of ‘Final’ key word with suitable example.**	**4M**
	Ans	The final keyword in Java is a non-access modifier that can be applied to variables, methods, and classes. It is used to restrict the user from further modifying the entity to which it is applied. This keyword plays a crucial role in ensuring immutability and preventing inheritance or method overriding.	**2M each for correct Explanation of uses with example. (Any two uses)**

Usage

1. final Variables

A final variable is a constant; once initialized, its value cannot be changed.

```
final int MAX_VALUE = 100;
```

OR

Example : final Variable

```
public class FinalVariableExample {
    public static void main(String[] args) {
        final int MAX_VALUE = 100;
        System.out.println("MAX_VALUE: " + MAX_VALUE);
        // MAX_VALUE = 200; // This will cause a compilation error
    }
}
```

In this example, MAX_VALUE is a final variable. Attempting to reassign it will result in a compilation error.

2. final Methods

A final method cannot be overridden by subclasses, ensuring that the method's implementation remains unchanged.

```
class Parent {
    public final void display() {
        System.out.println("This is a final method.");
    }
}
```

Example : final Method

```
class Parent {
    public final void display() {
        System.out.println("This is a final method.");
    }
}

class Child extends Parent {
    // public void display() { // This will cause a compilation error
    //     System.out.println("Attempting to override.");
    // }
}

public class FinalMethodExample {
    public static void main(String[] args) {
        Child obj = new Child();
        obj.display();
    }
}
```

Here, the display method in the Parent class is marked as final, preventing it from being overridden in the Child class.

3. final Classes

A final class cannot be subclassed, preventing inheritance.

```
public final class FinalClass {
    // Class implementation
}
```

Example : final Class

```
public final class FinalClass {
    public void show() {
        System.out.println("This is a final class.");
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
// class SubClass extends FinalClass { // This will cause a compilation error
// }

public class FinalClassExample {
    public static void main(String[] args) {
        FinalClass obj = new FinalClass();
        obj.show();
    }
}
```

In this example, FinalClass is declared as final, which means it cannot be extended by any other class.

Q. No.	Sub Q. N.	Answer	Marking Scheme
4.		**Attempt any THREE of the following:**	**12M**
	a)	**Enlist and explain four access specifiers in java.**	**4M**
	Ans	**Access Specifiers in Java**	**Explanation of access specifier 1M each**

1. **public**
2. **private**
3. **protected**
4. **Default (No keyword required)**

1. public

Accessible from anywhere in the program and used for classes, methods, and variables that need to be accessible globally.

Example:

```
public class PublicExample {
    public void display() {
        System.out.println("This is a public method.");
    }
}

class Main {
    public static void main(String args[]) {
        PublicExample obj = new PublicExample();
        obj.display(); // Accessible from another class
    }
}
```

2. private

Accessible only within the class where it is declared and used to enforce data encapsulation and restrict access to sensitive data.

Example:

```
class PrivateExample {
    private String message = "This is a private variable.";

    private void display() {
        System.out.println(message);
    }

    public void accessPrivate() {
        display(); // Access private method within the same class
    }
}

class Main {
    public static void main(String args[]) {
        PrivateExample obj = new PrivateExample();
        obj.accessPrivate(); // Indirectly access private method
        // obj.display(); // Compilation error
    }
}
```

3. protected

Accessible within the same package or subclasses in different packages and it is used for variables and methods that should be available to subclasses but not to unrelated classes.

Example:

```
public class ProtectedExample {
    protected void display() {
        System.out.println("This is a protected method.");
    }
}
class Subclass extends ProtectedExample {
    public static void main(String args[]) {
        Subclass obj = new Subclass();
        obj.display(); // Accessible in subclass
    }
}
```

4. Default (No keyword required)

Accessible only within the same package and used for methods and variables that don’t need to be accessed outside the package.

Example:

```
class DefaultExample {
    void display() {
        System.out.println("This is a default method.");
    }
}
class Main {
    public static void main(String args[]) {
        DefaultExample obj = new DefaultExample();
        obj.display(); // Accessible within the same package
    }
            }
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	b)	**Write a program to demonstrate the use of conditional operator and switch case statement.**	**4M**
	Ans	**Note: Any other correct logic shall be considered.**	**2M for conditional operator, 2M for switch case statement**

```
import java.util.Scanner;

public class ConditionalAndSwitchDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter two numbers (a and b): ");
        int a = scanner.nextInt();
        int b = scanner.nextInt();
        int max = (a > b) ? a : b;
        System.out.println("The greater number is: " + max);

        System.out.println("\nChoose an option:");
        System.out.println("1. Add");
        System.out.println("2. Subtract");
        System.out.println("3. Multiply");
        System.out.println("4. Divide");
        System.out.print("Enter your choice (1-4): ");
        int choice = scanner.nextInt();

        System.out.print("Enter two numbers for the operation: ");
        int x = scanner.nextInt();
        int y = scanner.nextInt();

        switch (choice) {
            case 1:
                System.out.println("Result of addition: " + (x + y));
                break;
            case 2:
                System.out.println("Result of subtraction: " + (x - y));
                break;
            case 3:
                System.out.println("Result of multiplication: " + (x * y));
                break;
            case 4:
                if (y != 0) {
                    System.out.println("Result of division: " + ((double) x / y));
                } else {
                    System.out.println("Error: Division by zero is not allowed.");
                }
                break;
            default:
                System.out.println("Invalid choice. Please choose a number between 1 and
               4."); }
        scanner.close();     } }
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	c)	**Explain how to create a package and import it with suitable example.**	**4M**
	Ans.	(see below)	**2M for creating a package with example & 2M for importing a package with example**

Ans.

To creating package involves following steps:

1. Declare the package at the beginning of a file using the form
   package packagename;
2. Define the class i.e. to be put in the package and declare it public.
3. Create a subdirectory under the directory where the main source file are stored.
4. Store the listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates .class file in the subdirectory.

Example for Create Package

```
// File: mypackage/Calculator.java
package mypackage;

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
```

//Note: Any relevant example shall be considered.

To import the package:

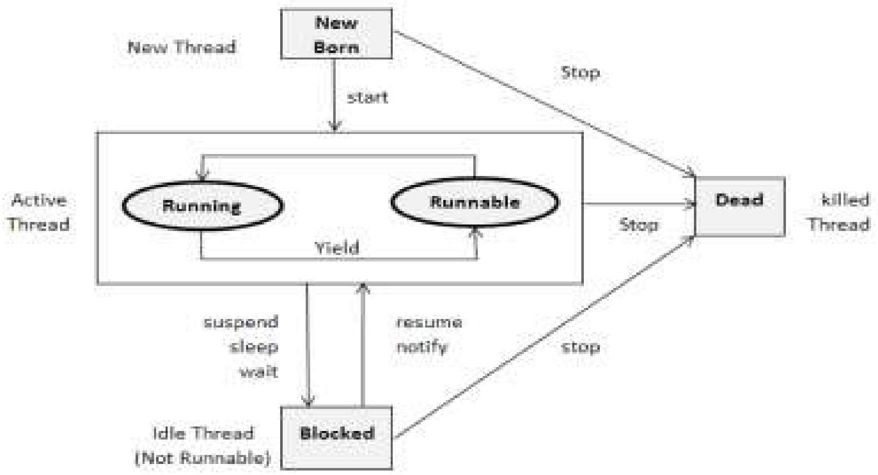
Make use of import statement to include package in your program.
It is used with ***** to gain full access to all classes within package or just by giving class name if just one class access is required.

Syntax

import mypackage.myclass;
or
import mypackage.*;

Example for Import Package

```
// File: Main.java
import mypackage.Calculator;

public class Main {
    public static void main(String args[]) {
        Calculator calc = new Calculator();
                // Using methods from the Calculator class
        System.out.println("Addition: " + calc.add(10, 5));
        System.out.println("Subtraction: " + calc.subtract(10, 5));
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	d)	**Draw and explain life cycle of thread.**	4M
	Ans	 New Thread New Born start Stop Active Thread Running Runnable Yield Stop Dead killed Thread suspend sleep wait resume notify stop Idle Thread (Not Runnable) Blocked **Fig.- Life Cycle of thread** **1. Newborn state:** When we create a thread object, the thread is born and is said to be in newborn state. The thread is not yet scheduled for running. At this state it can be scheduled for running using **start()** method or kill it using **stop()** method. If scheduled it moves to the runnable state. **2. Runnable State :** The runnable state means that thread is ready for execution and is waiting for the availability of the processor i.e. the thread has joined the queue of threads that are waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion, i.e. First-come, first-serve manner. 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. The thread that relinquishes control joins the queue at the end and again waits for its turn. This process of assigning time to threads is known as time-slicing. A thread can relinquish the control to another thread before its turn comes, by using **yield()** method. Runnable runnable = new NewState(); Thread t = new Thread(runnable); t.start(); **3. Running State :**	2M for Correct Diagram & 2M for Explanation

Q. No.	Sub Q. N.	Answer	Marking Scheme

Running State 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 preempted by a higher priority thread.

A running thread may relinquish its control in one of the following situations:

i. Relinquishing control using suspend() method.
ii. Relinquishing control using sleep() method.
iii. Relinquishing control using wait() method.

4. Blocked State :

A thread is said to be blocked when it is prevented from entering into the runnable state and subsequently the running state.

This happens when the thread is suspended, sleeping or waiting in order to satisfy certain requirements.

A blocked thread is considered **"not runnable"** but not dead and therefore fully qualified to run again.

5. Dead State :

Every thread has a life cycle.

A running thread ends its life when it has completed executing its run() method.

It is a natural death.

However, the thread can be killed by sending the stop message to it at any state thus causing a premature death to it.

A thread can be killed as soon as it is born, or while it is running, or even when it is in **"not runnable"** (blocked) condition.

e) **Explain garbage collection mechanism in java with suitable example.** **4M**

Ans. **Garbage collection:**

2M for Explanation & 2M for suitable Example

- 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.

Note: Garbage collection is performed by a daemon thread called Garbage Collector(GC). This thread calls the finalize() method before object is garbage collected.

Note: Any suitable example shall be considered.

Example:

```
public class TestGarbage1
{
 public void finalize(){
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
System.out.println("object is garbage collected");
}
 public static void main(String args[]){
  TestGarbage1 s1=new TestGarbage1();
  TestGarbage1 s2=new TestGarbage1();
  s1=null;
  s2=null;
  System.gc();
 }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
5.		**Attempt any <u>TWO</u> of the following :**	**12 M**
	a)	**Explain dynamic method dispatch mechanism with suitable example.**	**6M**
	Ans	Dynamic Method Dispatch in Java refers to the process by which a method call is resolved at runtime rather than at compile-time. It is an essential concept in Java that supports runtime polymorphism (also known as method overriding), allowing an object to determine which version of a method to call based on its actual object type, not its reference type. When a method is invoked on a reference variable, Java will determine at runtime which method to execute based on the actual object the reference variable points to, not the type of the reference variable itself. In dynamic method dispatch, a subclass provides its specific implementation of a method that is already defined in its superclass. This is called method overriding.	**3M for explanation & 3M for suitable example**

Example:

```
// Parent class
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Child class 1
class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

// Child class 2
class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("Cat meows");
    }
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
}

public class TestDynamicDispatch {
    public static void main(String[] args) {
        // Creating reference of Animal type, but pointing to Dog object
        Animal myAnimal = new Dog();
        myAnimal.sound();  // Output: Dog barks

        // Creating reference of Animal type, but pointing to Cat object
        myAnimal = new Cat();
        myAnimal.sound();  // Output: Cat meows
    }
}
```

	b)	**Write a program to check whether entered number is Armstrong number or not.**	**6M**
	Ans.	**Note: Any other correct logic shall be considered.**	**2M Variable declaration, 2M Correct logic & 2M Correct Code**

```
import java.util.Scanner;

public class ArmstrongNumber {

    public static void main(String args[]) {
        // Create a Scanner object for user input
        Scanner scanner = new Scanner(System.in);

        // Ask user for input
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();

        // Variables for calculations
        int originalNumber = num;
        int sum = 0;
        int digitCount = 0;

        // Count the number of digits in the number
        while (num > 0) {
            num /= 10;
            digitCount++;
        }

        // Reset num to original value
        num = originalNumber;

        // Calculate the sum of digits raised to the power of digitCount
        while (num > 0) {
            int digit = num % 10;
            sum += Math.pow(digit, digitCount);
            num /= 10;
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
        }

        // Check if the sum equals the original number
        if (sum == originalNumber) {
            System.out.println(originalNumber + " is an Armstrong number.");
        } else {
            System.out.println(originalNumber + " is not an Armstrong number.");
        }

        // Close the scanner
        scanner.close();
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	c)	**Design an Applet using <PARAM> tag to accept the two numbers and perform arithmetic operations.**	**6M**
	Ans.	**Note: Any other correct logic shall be considered.**	**2M Variable declaration, 2M Correct Logic & 2M Correct Code**

```
/*<!DOCTYPE html>
<html>
<head>
    <title>Simple Arithmetic Applet</title>
</head>
<body>
    <h2>Arithmetic Operations Applet</h2>
    <applet code="SimpleArithmeticApplet.class" width="300" height="200">
        <param name="num1" value="10">
        <param name="num2" value="5">
    </applet>
</body>
</html>
*/
import java.applet.Applet;
import java.awt.Graphics;

public class SimpleArithmeticApplet extends Applet {
    int num1, num2;
    int sum, difference, product;
    double quotient;

    // This method is called when the applet is initialized
    public void init() {
        // Get the values passed from the HTML page
        String num1Str = getParameter("num1");
        String num2Str = getParameter("num2");

        // Convert the strings to integers
        num1 = Integer.parseInt(num1Str);
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
        num2 = Integer.parseInt(num2Str);

        // Perform arithmetic operations
        sum = num1 + num2;
        difference = num1 - num2;
        product = num1 * num2;
        quotient = (num2 != 0) ? (double) num1 / num2 : 0; // Handle division by zero
    }

    // This method is called to paint the results on the applet window
    public void paint(Graphics g) {
        // Display the results
        g.drawString("Number 1: " + num1, 20, 20);
        g.drawString("Number 2: " + num2, 20, 40);
        g.drawString("Sum: " + sum, 20, 60);
        g.drawString("Difference: " + difference, 20, 80);
        g.drawString("Product: " + product, 20, 100);
        g.drawString("Quotient: " + quotient, 20, 120);
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
6.		**Attempt any TWO of the following :**	**12M**
	a)	**Explain life cycle of an Applet with suitable example.**	**6M**
	Ans.		**2M Diagram, 2M Explanation & 2M Example**

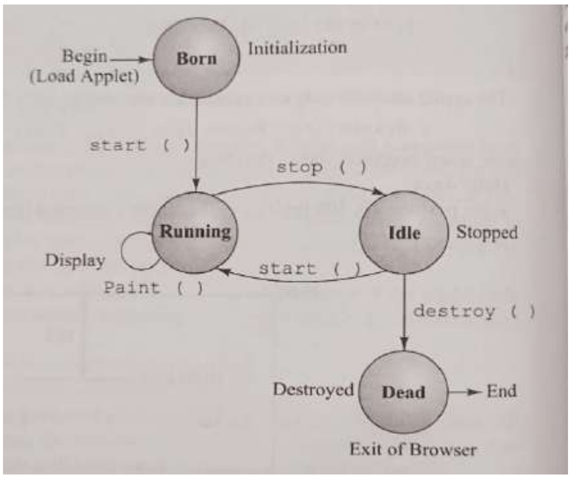


Fig. - Life Cycle of an Applet

Initialization State:

Applet enters the initialization state when it is first loaded.
This is achieved by calling the **init()** method of Applet Class.
The applet is born. At this stage, we may do the following, if required.

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

The initialization occurs only once in the applet's life cycle.

Q. No.	Sub Q. N.	Answer	Marking Scheme

		To provide any of the behaviors mentioned above, we must override the **init()** method:
		public void init()
		{
		………………….
		…………………. (Action)
		}
		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.
		Starting can also occur if the applet is already in "stopped" (idle) state.
		Note that, unlike **init()** method, the start() method may be called more than once. We may override the **start()** method to create a thread to control the applet.
		public void **start()**
		{
		………………….
		…………………. (Action)
		}
		Idle or Stopped State:
		An applet becomes idle when it is stopped from running.
		Stopping occurs automatically when we leave the page containing the currently running applet.
		We can also do so by calling the **stop()** method explicitly.
		If we use a thread to run the applet, then we must use **stop()** method to terminate the thread. This is done by overriding the **stop()** method;
		public void stop()
		{
		………………….
		…………………. (Action)
		}
		Dead State:
		An applet is said to be dead when it is removed from memory.
		This occurs automatically by invoking the **destroy()** method when we quit the browser.
		Like initialization, destroying stage occurs only once in the applet's life cycle.
		If the applet has created any resources, like threads, we may override the **destroy()** method to clean up these resources.
		public void destroy ()
		{
		………………….
		…………………. (Action)
		}
		Display State:	

Applet moves to the display state whenever it has to perform some output operations on the screen.
This happens immediately after the applet enters into the running state.
The **paint()** method is called to accomplish this task.
Every applet we have a paint method.
The default version of paint method does nothing.
Override this method if you we want anything to be displayed on the screen.

```
public void paint ()
{
      ………………….
      …………………. (Action)
 }
```

Note: Any other correct logic shall be considered.

Example:

```
/*<html>
<body>
<applet code="SimpleApplet.class" width="300" height="300">
</applet>
</body>
</html>  */

import java.applet.Applet;
import java.awt.Graphics;

public class SimpleApplet extends Applet {
    // Initialization
    public void init() {
        System.out.println("Applet Initialized");
    }

    // Start
    public void start() {
        System.out.println("Applet Started");
    }

    // Paint (called to display graphics)
    public void paint(Graphics g) {
        g.drawString("Hello, Applet!", 20, 20);
        System.out.println("Applet Painted");
    }

    // Stop
    public void stop() {
        System.out.println("Applet Stopped");
    }
```

Q. No.	Sub Q. N.	Answer	Marking Scheme

```
    // Destroy (clean-up)
    public void destroy() {
        System.out.println("Applet Destroyed");
    }
}
```

	b)	**Write a program to create two threads. One thread will display the odd numbers from 1 to 50 and the other thread will display the even numbers.**	**6M**
	Ans.	**Note: Any other correct logic shall be considered.**	**2M Creating two threads, 2M correct logic for printing odd and even numbers & 2M Main method**

```
class OddThread extends Thread {
    public void run() {
        for (int i = 1; i <= 50; i++) {
            if (i % 2 != 0) {  // Check if the number is odd
                System.out.println("Odd: " + i);
                try {
                    Thread.sleep(100);  // Sleep for 100ms to simulate work (optional)
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
        }
    }
}
class EvenThread extends Thread {
    public void run() {
        for (int i = 1; i <= 50; i++) {
            if (i % 2 == 0) {  // Check if the number is even
                System.out.println("Even: " + i);
                try {
                    Thread.sleep(100);  // Sleep for 100ms to simulate work (optional)
                } catch (InterruptedException e) {
                    System.out.println(e);
                }
            }
        }
    }
}
public class OddEvenThreads {
    public static void main(String[] args) {
        // Create instances of the threads
        OddThread oddThread = new OddThread();
        EvenThread evenThread = new EvenThread();

        // Start the threads
        oddThread.start();
        evenThread.start();
    }
}
```

Q. No.	Sub Q. N.	Answer	Marking Scheme
	c)	**Describe the use of any methods of vector class with their syntax (Any six methods)**	**6M**
	Ans.		**1M for each method (Any 4 methods)**

Sr. No.	Syntax	Task Performed
1	void addElement(Object ob)	Adds the specific component at this vector by increasing its size by one
2	int capacity()	Returns the current capacity of this vector
3	Boolean contains(Object element)	Tests if specific object is component in this vector
4	void clear()	Removes all the elements from this vector
5	Object elementAt(int index)	Returns the component at specified index
6	Enumeration elements()	Returns enumeration of components of this vector
7	Object firstElement()	Returns first component of this vector
8	Object lastElement()	Returns last element of this vector
9	int indexOf(Obejct element)	Searches for the first occurrence of the given argument
10	void insertElementAt(Object obj,int index)	Inserts specified object as component at the specified index position
11	void removeElementAt(int index)	Removes the element at the specified position in the vector
12	boolean removeElement(Obejct obj)	Removes the first occurrence of the argument from the vector
13	int size()	Returns number of components in the vector
14	void copyInto(Object[] array)	Copies the components of vector into specified array

WINTER – 2023 EXAMINATION
Model Answer – Only for the Use of RAC Assessors

Subject Name: **Java Programming** **Subject Code:** 22412

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 judgement 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.
8) As per the policy decision of Maharashtra State Government, teaching in English/Marathi and Bilingual (English + Marathi) medium is introduced at first year of AICTE diploma Programme from academic year 2021-2022. Hence if the students in first year (first and second semesters) write answers in Marathi or bilingual language (English +Marathi), the Examiner shall consider the same and assess the answer based on matching of concepts with model answer.

Q. No.	Sub Q. N.	Answer	Marking Scheme
1		**Attempt any FIVE of the following:**	**10 M**
	a)	**Enlist any two logical operators and two bitwise operators.**	**2 M**
	A l Ans	Logical Operators: 1. AND Operator (&&) – if(a && b) [if true execute else don't] 2. OR Operator (||) – if(a || b) [if one of them is true to execute else don't] 3. NOT Operator (!) – !(a<b) [returns false if a is smaller than b] Bitwise Operator: 1. Bitwise OR (|) 2. Bitwise AND (&) 3. Bitwise XOR (^) 4. Bitwise Complement (~) 5. Bitwise left shift(<<) 6. Bitwise right shift(>>)	List any two Logical operator : 2 marks List any two Bitwise operator : 2 marks

	b)	**Define constructor.**	**2 M**
	Ans	A constructor in Java is a special method that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initialvalues for object attributes.	**Correct/suitable definition- 1 M** **Syntax or**

<table>
<tr><td></td><td></td><td><b>For Example:</b>
<pre>
class Test
 {
  Test()
 {
    // constructor body
  }
 }
</pre></td><td><b>Example- 1 M</b></td></tr>
<tr><td></td><td><b>c)</b></td><td><b>Write down the syntax of array declaration, initialization.</b></td><td><b>2 M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td><b>The syntax of declaring an array in Java is given below.</b>
<pre>
datatype [] arrayName;
</pre>
Here, the datatype is the type of element that will be stored in the array, square bracket[] is for the size of the array, and arrayName is the name of the array.

<b>The syntax of initializing an array is given below.</b>
<pre>
datatype [] arrayName = new datatype [size];
</pre></td><td>1 M- array declaration
and
1 M-array initialization</td></tr>
<tr><td></td><td><b>d)</b></td><td><b>List out different ways to access package from another package.</b></td><td><b>2 M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>There are three ways to access the package from outside the package.
<ul>
<li>import package.*;</li>
<li>import package.classname;</li>
<li>fully qualified name</li>
</ul></td><td>Any 2 correct ways – 2 M</td></tr>
<tr><td></td><td><b>e)</b></td><td><b>Differentiate between starting thread with run( ) method and start() method.</b></td><td><b>2 M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>
<table>
<tr><th>start()</th><th>run()</th></tr>
<tr><td>Creates a new thread and the run() method is executed on the newly created thread.</td><td>No new thread is created and the run() method is executed on the calling thread itself.</td></tr>
<tr><td>Can’t be invoked more than one time</td><td>Multiple invocation is possible</td></tr>
<tr><td>Defined in java.lang.Thread class.</td><td>Defined in java.lang.Runnable interface and must be overridden in the implementing class.</td></tr>
</table></td><td>Any 2 valid points- 2 M</td></tr>
</table>

It starts thread to begin execution, JVM calls run method of this thread.	It is used to perform operations by thread.
Syntax: public void start()	Syntax: public void run()
A new thread will be created and it is responsible to complete the job.	No new thread will be created and main thread will be responsible to complete the job.

	f)	**State the classes that can an applet extend.**	**2 M**
	Ans	• Graphics • Font • Color	Any 2 classes-2 M
	g)	**Give syntax to open a file using Inputstream class.**	**2 M**

	Ans	Attach a file to a FileInputStream as this will enable us to read data from the file as shown below as follows:
		FileInputStream input = new FileInputStream("input.txt");
		Now in order to read data from the file, we should read data from the FileInputStream as shown below:
		ch=fileInputStream.read();
			Correct syntax-2 M

2.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Write a program lo display ASCII value of a number 9.**	**4 M**

	Ans	public class asciivalue
		{
		public static void main(String args[])
		{
		// Character whose ASCII is to be computed
		char ch = '9';
		// Creating a new variable of type int and assigning the character value.
		int ascii = ch;
		// Printing the ASCII value of above character
		System.out.println("The ASCII value of " + ch+ " is: " + ascii);
		}
		}
		Output:
		The ASCII value of 9 is: 57
			For any correct program: 4m

b) **Write a program to sort the elements of an array in ascending order.** **4 M**

Ans

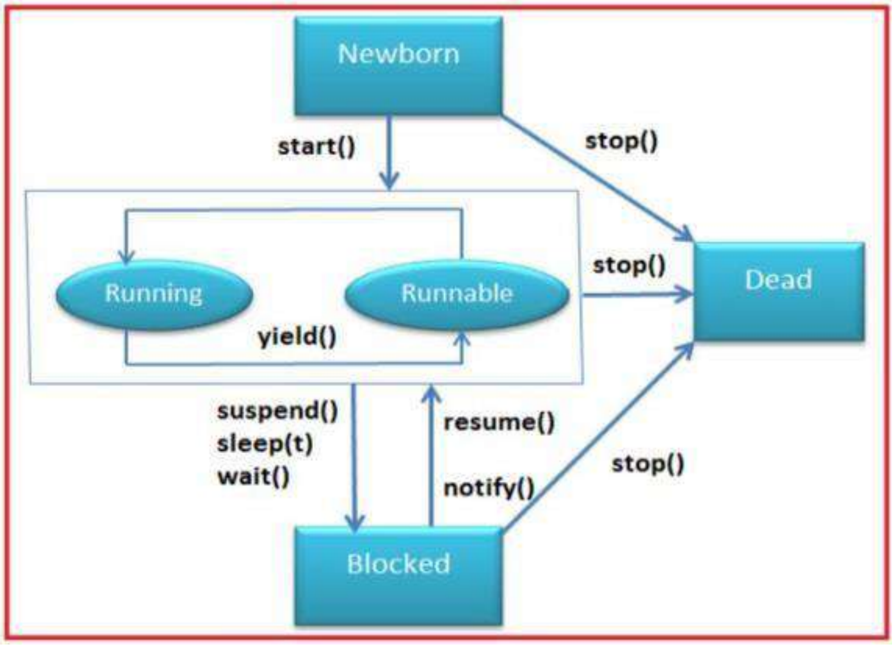
For any correct logic and program: 4m

```
class arraysort
{
public static void main(String args[])
{
int a[]={85,95,78,45,12,56,78,19};
int i=0;
int j=0;
int temp=0;
int l=a.length;

for(i=0;i<l;i++)
{ //apply bubble sort

        for(j=(i+1);j<l;j++)
        {
        if(a[i]>a[j])
        {
        temp=a[i];
        a[i]=a[j];
        a[j]=temp;
        }
        }
}
System.out.println("Ascending order of numbers:");
for(i=0;i<l;i++)
System.out.println(""+a[i]);
}
}
```

Output:

```
Ascending order of numbers:
12
19
45
56
78
78
85
95
```

c) **Define Thread. Draw life cycle of Thread.** **4 M**

	Ans	Thread is a smallest unit of executable code or a single task is also called as thread.
		Each tread has its own local variable, program counter and lifetime.
		A thread is similar to program that has a single flow of control.
		 Newborn
		start()
		stop()
		Running
		Runnable
		yield()
		stop()
		Dead
		suspend()
		sleep(t)
		wait()
		resume()
		notify()
		stop()
		Blocked
		Fig: Life cycle of Thread
			Definition of thread- 2 M
			Diagram: 2M

	d)	**Write a program to read a file and then count number of words.**	**4 M**

	Ans	// Java program to count the no. of words in a file
		import java.io.*;
		public class Test11
		{
		public static void main(String[] args) throws IOException
		{
		File file = new File("C:\\Program Files\\Java\\jdk1.7.0_80\\bin\\a.txt");
		FileInputStream fileInputStream = new FileInputStream(file);
		InputStreamReader inputStreamReader = new InputStreamReader(fileInputStream);
		BufferedReader bufferedReader = new BufferedReader(inputStreamReader);
		
		String line;
		int wordCount = 0;
		int paraCount = 0;
		while ((line = bufferedReader.readLine()) != null)
		{
		if (line.equals("")) {
			2 M - correct variable and object creation
			2 M - valid logic to count words from file.

```
paraCount += 1;
}
else {
String words[] = line.split("\\s+");
wordCount += words.length;
}
}
System.out.println("Total word count = "+ wordCount);
}
}

C:\Program Files\Java\jdk1.7.0_80\bin>javac Test11.java
C:\Program Files\Java\jdk1.7.0_80\bin>java Test11
Total word count = 8
```

3.		**Attempt any <u>THREE</u> of the following:**	**12 M**
	a)	**Write a program which displays functioning of ATM machine,** **(Hint: Withdraw, Deposit, Check Balance and Exit)**	**4 M**
	Ans	See program below	4 M for correct program Or any other relevant logic should be considered

```
import java.util.Scanner;
public class ATM_Transaction
{
public static void main(String args[] )
{
                int balance = 5000, withdraw, deposit;
                Scanner s = new Scanner(System.in);
                while(true)
                  {
                    System.out.println("Automated Teller Machine");
                    System.out.println("Choose 1 for Withdraw");
                    System.out.println("Choose 2 for Deposit");
                    System.out.println("Choose 3 for Check Balance");
                    System.out.println("Choose 4 for EXIT");
                    System.out.print("Choose the operation you want to perform:");

                   int n = s.nextInt();
                       switch(n)
                        {
```

```
          case 1:
          System.out.print("Enter money to be withdrawn:");
          withdraw = s.nextInt();
          if(balance >= withdraw)
          {
         balance = balance - withdraw;
         System.out.println("Please collect your money");
         }
         else
         {
         System.out.println("Insufficient Balance");
         }
         System.out.println("");
         break;

     case 2:
         System.out.print("Enter money to be deposited:");
         deposit = s.nextInt();
         balance = balance + deposit;
         System.out.println("Your Money has been successfully depsited");
         System.out.println("");
         break;

      case 3:
          System.out.println("Balance : "+balance);
          System.out.println("");
          break;

      case 4:
               System.exit(0);
           }
        }
      }
```

	b)	**Differentiate between method overloading and method overriding.**	**4 M**

Ans

Method Overloading	**Method Overriding**
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
Method overloading helps to increase the readability of the program.	Method overriding is used to grant the specific implementation of the method which is already provided by its parent class or superclass.
It occurs within the class.	It is performed in two classes with inheritance relationships.
Method overloading may or may not require inheritance.	Method overriding always needs inheritance.
In method overloading, methods must have the same name and different signatures.	In method overriding, methods must have the same name and same signature.
In method overloading, the return type can or can not be the same, but we just have to change the parameter.	In method overriding, the return type must be the same or co-variant.
Static binding is being used for overloaded methods.	Dynamic binding is being used for overriding methods.
Poor Performance due to compile time polymorphism.	It gives better performance. The reason behind this is that the binding of overridden methods is being done at runtime.
Private and final methods can be overloaded.	Private and final methods can’t be overridden.
The argument list should be different while doing method overloading.	The argument list should be the same in method overriding.

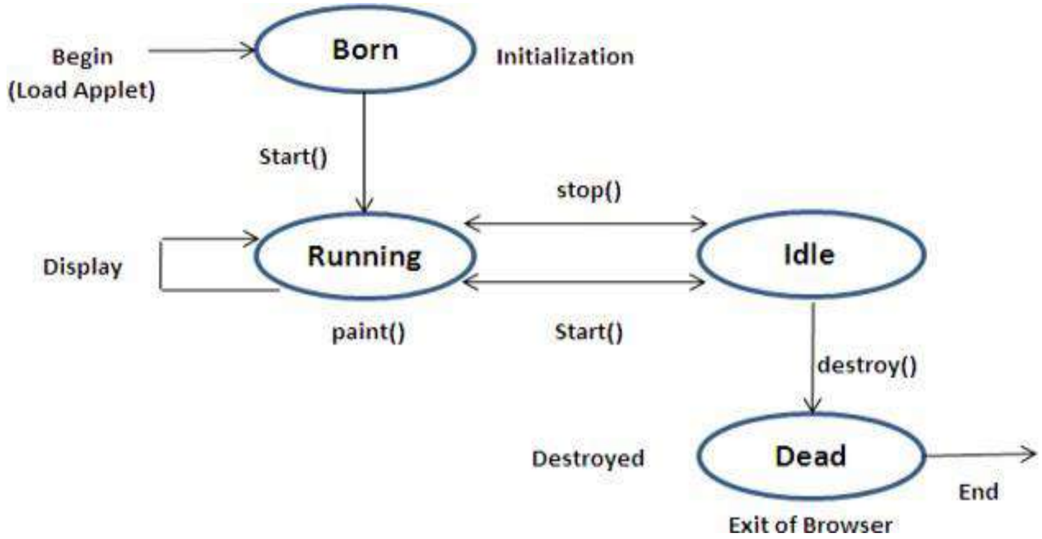
4 M for any four correct point

c) **Explain applet life cycle in detail.** **4 M**

Ans

The applet life cycle can be defined as the process of how the object is created, started, stopped, and destroyed during the entire execution of its application. It basically has five core methods namely init(), start(), stop(), paint() and destroy().These methods are invoked by the browser to execute.

2 M for diagram and 2 M for explanation



1. Initialization State (The init() method):

- The life cycle of an Applet is begin on that time when the applet is first loaded into the browser and called the init() method.
- The init() method is called only one time in the life cycle on an Applet. The init() method is basically called to read the “PARAM” tag in the html file.
- The init () method retrieve the passed parameter through the “PARAM” tag of html file using get Parameter() method All the initialization such as initialization of variables and the objects like image, sound file are loaded in the init () method.
- After the initialization of the init() method user can interact with the Applet and mostly applet contains the init() method.
- **Syntax:**

```
public void init()
{
---
---
}
```

2. Running State (The start() method):

- The start method of an Applet is called after the initialization method init(). This method may be called multiples time when the Applet needs to be started or restarted.

- For Example if the user wants to return to the Applet, in this situation the start() method of an Applet will be called by the web browser and the user will be back on the applet. In the start method user can interact within the applet.
- Syntax:-

```
public void start()
        {
        ……..
        ……..
        }
```

3. Idle (The Stop() method):

- An applet becomes idle when it is stopped from running. The stop() method stops the applet and makes it invisible.
- Stopping occurs automatically when we leave the page containing the currently running applet. We can also do so by calling the stop() method explicitly.
- The stop() method can be called multiple times in the life cycle of applet like the start () method or should be called at least one time.
- For example the stop() method is called by the web browser on that time When the user leaves one applet to go another applet and the start() method is called on that time when the user wants to go back into the first program or Applet.
- **Syntax:-**

```
public void stop()
{
……..
……..
}
```

4. Dead State (The destroy() method):

- The destroy() method is called to terminate an Applet. an Applet is said to be dead when it is removed from memory.
- This occurs automatically by invoking the destroy() method when we quit the browser. It is useful for clean-up actions, such as releasing memory after the applet is removed, killing off threads and closing network/database connections.
- Thus this method releases all the resources that were initialized during an applet’s initialization.
- **Syntax:-**

```
public void destroy()
{
……..
……..
}
```

		5. Display State (The paint() method):
		• The paint() method is used for applet display on the screen. The display includes text, images, graphics and background.
		• This happens immediately after the applet enters into the running state. Almost every applet will have a paint() method and can be called several times during an applet's life cycle.
		• The paint() method is called whenever a window is required to paint or repaint the applet.
		• **Syntax:-**
		public void paint(Graphics g)
		{
		……..
		……..
		}	

	d)	**Differentiate between Byte Stream Class and Character Stream Class. (Any four points)**	**4 M**

Ans

4 M for any correct 4 point

Byte Stream Class	**Character Stream Class**
Byte streams access the file byte by byte (8 bits).	A character stream will read a file character by character (16 bits).
Byte stream classes are classified into: 1. Input Stream Classes 2. Output Stream Classes	Character stream classes are classified into: 1. Reader class 2. Writer class
InputStream/OutputStream class is byte-oriented.	The Reader/Writer class is character-oriented.
The methods for byte streams generally work with byte data type.	The methods for character streams generally accept parameters of data type *char* parameters.
Byte-stream classes end with the suffix InputStream and OutputStream.	Character-stream classes end with the suffix Reader or Writer.
It is possible to translate character stream into byte stream with OutputStreamWriter.	It is possible to translate byte stream into a character stream with InputStreamReader.
Byte streams specifically used for reading and writing data in byte format.	The advantage of character streams, that is make it easy to write programs, which is not dependent upon a specific character encoding.
No conversion needed.	Character streams convert the underlying data bytes to Unicode, which is a costly operation.
InputStream and OutputStream are used for reading or writing binary data.	Reader and Writer uses Unicode, hence they can be internationalized. Hence in some cases they are more efficient than byte streams.

4.		**Attempt any THREE of the following:**	**12 M**
	a)	**Explain implicit and explicit type conversion with example in detail.**	**4 M**
	Ans	**Widening (Implicit)** - The process of assigning a smaller type to a larger one is known as widening or implicit. Byte → short → int → long → float → double	2 M for Implicit with example And 2 M for Explicit with example

For e.g.

```
class widening
 {
 public static void main(String arg[])
{
 int i=100;
 long l=i;
 float f=l;
 System.out.println("Int value is"+i);
System.out.println("Long value is"+l);
System.out.println("Float value is"+f);
}
}
```

<u>Narrowing (Explicit)</u>

- The process of assigning a larger type into a smaller one is called narrowing.
- Casting into a smaller type may result in loss of data.

double → long → int → short → byte

For e.g.

```
class narrowing
{
Public static void main(String[])
{
Double d=100.04;
Long l=(long) d;
Int i=(int) l;
System.out.println("Int value is"+i);
System.out.println("Long value is"+l);
System.out.println("Float value is"
}
}
```

	b)	**Write a program to show the use of copy constructor.**	**4 M**

	Ans	class student
		{
		int id;
		String name;
		student(int i, String n)
		{
		id=i;
		name=n;
		}
			4 M for any suitable correct program

```
student (student s)//copy constructor
{
id=s.id;
name=s.name;
}
void display()
{
System.out.println(id+“ ”+name)
}
public static void main(String args[])
student s1=new student(111, “ABC”);
s1.display();
student s2= new student(s1);
s2.display();
}
}
```

	c)	**Write a program to show the Hierarchical inheritance.**	**4 M**

Ans

```
import java.io.*;
abstract class shape
{
float dim1,dim2;
void getdata()
{
DataInputStream d=new DataInputStream(System.in);
try
{
System.out.println("Enter the value of Dimension1: ");
dim1=Float.parseFloat(d.readLine());
System.out.println("Enter the value of Dimension2: ");
dim2=Float.parseFloat(d.readLine());
}
catch(Exception e)
{
System.out.println("General Error"+e);
}
}
void disp()
{
System.out.println("Dimension1= "+dim1);
System.out.println("Dimension2= "+dim2);
}
abstract void area();
}
class rectangle extends shape
```

4 M for correct program
(Any relevant example can be consider)

```
{
double area1;
void getd()
{
super.getdata();
}
void area()
{
area1=dim1*dim2;
System.out.println("The Area of Rectangle is: "+area1);
}
}
class triangle extends shape
{
double area1;
void getd()
{
super.getdata();
}
void area()
{
area1=(0.5*dim1*dim2);
System.out.println("The Area of Triangle is: "+area1);
}
}
class methodover1
{
public static void main(String args[])
{
rectangle r=new rectangle();
 System.out.println("For Rectangle");
 r.getd();
 r.disp();
 r.area();
triangle t=new triangle();
 t.getd();
 t.disp();
 t.area();
}
}
```

OR

```
class A
{
  public void methodA()
  {
    System.out.println("method of Class A");
  }
}
class B extends A
{
  public void methodB()
  {
    System.out.println("method of Class B");
  }
}
class C extends A
{
 public void methodC()
 {
   System.out.println("method of Class C");
 }
}
class D extends A
{
 public void methodD()
 {
   System.out.println("method of Class D");
 }
}
class JavaExample
{
 public static void main(String args[])
 {
   B obj1 = new B();
   C obj2 = new C();
   D obj3 = new D();
   //All classes can access the method of class A
   obj1.methodA();
   obj2.methodA();
   obj3.methodA();
 }
}
```

	d)	**Explain any four font methods with example.**	**4 M**
	Ans	**Font** is a class that belongs to the **java.awt** package. Following are the methods of Font class:	2 M for any four-font method with

description and 2 M for example

Methods	**Description**
String getFamily( )	Returns the name of the font family to which the invoking font belongs.
static Font getFont(String *property*)	Returns the font associated with the system property specified by *property*. **null** is returned if *property* does not exist.
String getFontName()	Returns the face name of the invoking font.
String getName( )	Returns the logical name of the invoking font.
int getSize( )	Returns the size, in points, of the invoking font.
int getStyle( )	Returns the style values of the invoking font.
int hashCode( )	Returns the hash code associated with the invoking object.
boolean isBold( )	Returns **true** if the font includes the **BOLD** style value. Otherwise, **false** is returned.
boolean isItalic( )	Returns **true** if the font includes the **ITALIC** style value. Otherwise, **false** is returned.
boolean isPlain( )	Returns **true** if the font includes the **PLAIN** style value. Otherwise, **false** is returned.

Example:

```
import java.awt.*;
import java.applet.*;
public class Shapes extends Applet
{
Font f,f1;
String s,msg;
String fname;
String ffamily;
int size;
int style;
public void init()
{
f= new Font("times new roman",Font.ITALIC,20);
setFont(f);
msg="is interesting";
s="java programming";
fname=f.getFontName();
ffamily=f.getFamily();
size=f.getSize();
style=f.getStyle();
String f1=f.getName();
}
public void paint(Graphics g)
{
g.drawString("font name"+fname,60,44);
g.drawString("font family"+ffamily,60,77);
g.drawString("font size "+size,60,99);
g.drawString("fontstyle "+style,60,150);
g.drawString("fontname "+f1,60,190);
}
}
/*<applet code=Shapes.class height=300 width=300></applet>*/
```

	e)	**Write a program to append content of one file into another file.**	**4 M**

Ans

4 M for correct program

```
import java.io.*;
class copyf
{
public static void main(String args[]) throws IOException
{
BufferedReader in=null;
BufferedWriter out=null;
try
{
in=new BufferedReader(new FileReader("input.txt"));
out=new BufferedWriter(new FileWriter("output.txt"));

int c;
while((c=in.read())!=-1)
{
out.write(c);
}
System.out.println("File copied successfully");
}
finally
{
if(in!=null)
{
in.close();
}
if(out!=null)
{
out.close();
}
```

```
        }
    }
}
```

5. **Attempt any TWO of the following:** **12 M**

a) **Explain vector with the help of example. Explain any 3 methods of vector class.** **6 M**

Ans

- Vector is a data structure that is used to store a collection of elements. Elements can be of all primitive types like int, float, Object, etc. Vectors are dynamic in nature and accordingly, grow or shrink as per the requirement.
- Vector Class in Java is found in the java.util package.
- Vector class is a child class of the AbstractList class and implements the List interface. Therefore, we can use all the methods of the List interface.
- Vectors are known to give ConcurrentModificationException when accessed concurrently at the time of modification.
- When a Vector is created, it has a certain capacity to store elements that can be defined initially. This capacity is dynamic in nature and can be increased or decreased.
- By definition, Vectors are synchronized, which implies that at a time, only one thread is able to access the code while other threads have to wait.

Vectors are created like arrays. It has three constructor methods

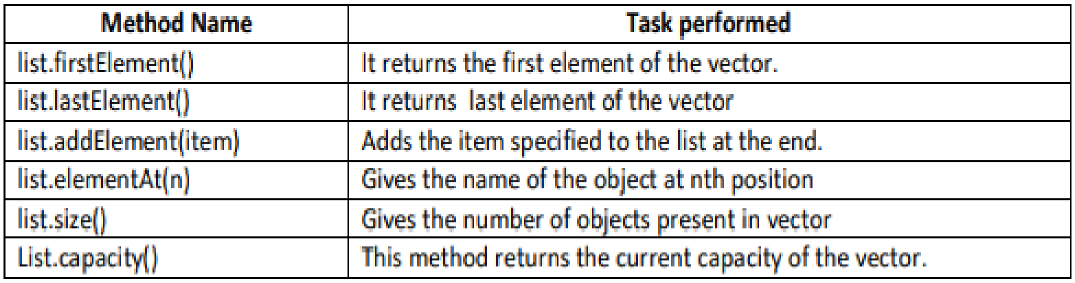
Vector list = new Vector(); //declaring vector without size
Vector list = new Vector(3); //declaring vector with size
Vector list = new Vector(5,2); //create vector with initial size and whenever it need to grows, it grows by value specified by increment capacity

Correct explaination-2 M

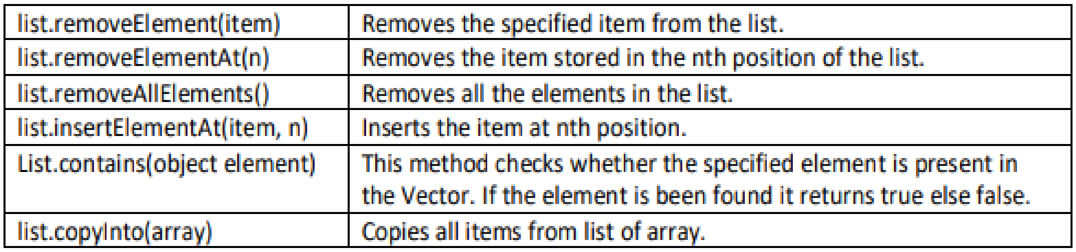
List of constructors and methods of vector class-2 M

Example – 2 M

Methods of Vector class:



Method Name	Task performed
list.firstElement()	It returns the first element of the vector.
list.lastElement()	It returns last element of the vector
list.addElement(item)	Adds the item specified to the list at the end.
list.elementAt(n)	Gives the name of the object at nth position
list.size()	Gives the number of objects present in vector
List.capacity()	This method returns the current capacity of the vector.



list.removeElement(item)	Removes the specified item from the list.
list.removeElementAt(n)	Removes the item stored in the nth position of the list.
list.removeAllElements()	Removes all the elements in the list.
list.insertElementAt(item, n)	Inserts the item at nth position.
List.contains(object element)	This method checks whether the specified element is present in the Vector. If the element is been found it returns true else false.
list.copyInto(array)	Copies all items from list of array.

Example:

import java.util.*;

```
public class Main
{
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
System.out.println("Initial Vector: " + v);
v.removeElementAt(2); // remove 3rd element
System.out.println("Current Vector: " + v);
v.removeElementAt(3); // remove 4th element
System.out.println("Current Vector: " + v);
v.insertElementAt(11,2); // new element inserted at 3rd position
System.out.println("Current Vector: " + v);
System.out.println("Size of vector after insert delete operations: " + v.size());
}
}
```

Output:

```
6
Initial Vector: [10, 20, 30, 40, 10, 20]
Current Vector: [10, 20, 40, 10, 20]
Current Vector: [10, 20, 40, 20]
Current Vector: [10, 20, 11, 40, 20]
Size of vector after insert delete operations: 5
```

b) Develop and Interest Interface which contains Simple Interest and Compound Interest methods and static final field of rate 25%. Write a class to implement those methods. **6 M**

Ans

```
 import java.util.Scanner;
 import static java.lang.Math.pow;

 interface Interest
{
   int roi=25;
   public void simpleInterest(float principle,float time);
   public void compoundInterest(float principle,float time);

}
public class InterestTest implements Interest
{

     public void simpleInterest(float principle,float time)
 {
       float si = (principle*roi*time)/100;
```

Creating correct interface with-2M

Implementing interface-1M

Calculating simple interest and compound interest-2M

```
        System.out.println("Simple interested calculate by program is : " + si);
     }
    public  void compoundInterest(float principle,float time)
   {
       double ci = principle * (Math.pow((1.0 +(roi/100)), time)) - principle;
        System.out.println("Compound interested calculate by program is : " + ci);
     }
    public static void main(String args[])
    {
       InterestTest i1 = new InterestTest();
       i1.simpleInterest(1000,2);
       i1.compoundInterest(1000,2);
      }
    }
```

Correct Main method-1M

c) **Write a program that throws an exception called "NoMatchException" when a string is not equal to "India".**

6 M

Ans

```
import java.io.*;
class NoMatchException extends Exception
{
    private String str;
   NoMatchException(String str1)
    {
       str=str1;
    }

    public String toString()
    {
       return "NoMatchException --> String is not India and string is "+str;
    }
}

class Main
{
    public static void main(String args[ ])
    {
       String str1= new String("India");
       String str2= new String("Australlia");

       try
       {
          if(str1.equals("India"))
             System.out.println(" String is : "+str1);
          else

             throw new NoMatchException(str1);

          if(str2.equals("India"))
             System.out.println("\n String is : "+str2);
```

Any Correct program – 6 M

```
            else
               throw new NoMatchException(str2);
         }
         catch(NoMatchException e)
         {
            System.out.println("\nCaught.... "+e);
         }
      }
}
```

OUTPUT:

```
String is : India

Caught.... NoMatchException --> String is not India and string is Australlia
```

6.		**Attempt any TWO of the following:**	**12 M**
	a)	**Write a program to print the sum, difference and product of two complex numbers by creating a class named "Complex" with separate methods for each operation whose real and imaginary parts are entered by user.**	**6 M**
	Ans	See program below	Correct program – 6 M

```
// Java program to add and subtract two
// complex numbers using Class
import java.util.*;

// User Defined Complex class
class Complex
 {
         // Declaring variables
         int real, imaginary;

         // Empty Constructor
         Complex()
         {
         }

         // Constructor to accept
         // real and imaginary part
         Complex(int tempReal, int tempImaginary)
         {
                  real = tempReal;
                  imaginary = tempImaginary;
         }

         // Defining addComp() method
         // for adding two complex number
         Complex addComp(Complex C1, Complex C2)
         {
                  // creating temporary variable
```

```
                Complex temp = new Complex();

                // adding real part of complex numbers
                temp.real = C1.real + C2.real;

                // adding Imaginary part of complex numbers
                temp.imaginary = C1.imaginary + C2.imaginary;

                // returning the sum
                return temp;
        }

        // Defining subtractComp() method
        // for subtracting two complex number
        Complex subtractComp(Complex C1, Complex C2)
        {
                // creating temporary variable
                Complex temp = new Complex();

                // subtracting real part of complex numbers
                temp.real = C1.real - C2.real;

                // subtracting Imaginary part of complex numbers
                temp.imaginary = C1.imaginary - C2.imaginary;

                // returning the difference
                return temp;
        }

        Complex productComp(Complex C1, Complex C2)
        {
                // creating temporary variable
                Complex temp = new Complex();

                // product of  of complex numbers
                //(a + ib) (c + id)= (ac - bd) + i(ad + bc).
                temp.real = ((C1.real*C2.real)-(C1.imaginary*C2.imaginary));
        temp.imaginary = ((C1.real*C2.imaginary) + (C1.imaginary*C2.real));

                // returning the difference
                return temp;
        }

        // Function for printing complex number
        void printComplexNumber()
        {
System.out.println("Complex number: " + real + " + " + imaginary + "i");
        }
    }
```

```
// Main Class
public class Main
 {

        // Main function
        public static void main(String[] args)
        {

                // First Complex number
                Complex C1 = new Complex(3, 2);

                // printing first complex number
                C1.printComplexNumber();

                // Second Complex number
                Complex C2 = new Complex(9, 5);

                // printing second complex number
                C2.printComplexNumber();

                // for Storing the sum
                Complex C3 = new Complex();

                // calling addComp() method
                C3 = C3.addComp(C1, C2);

                // printing the sum
                System.out.print("Sum of ");
                C3.printComplexNumber();

                // calling subtractComp() method
                C3 = C3.subtractComp(C1, C2);

                // printing the difference
                System.out.print("Difference of ");
                C3.printComplexNumber();

                // calling productComp() method
                C3 = C3.productComp(C1, C2);

                // printing the product
                System.out.print("product of ");
                C3.printComplexNumber();
        }
}
```

OUTPUT:

```
Complex number: 3 + 2i
Complex number: 9 + 5i
```

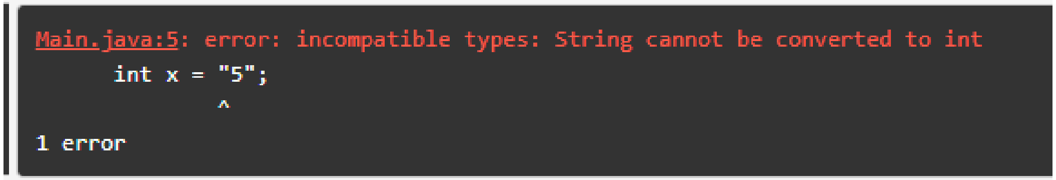
<table>
<tr><td></td><td></td><td>Sum of Complex number: 12 + 7i
Difference of Complex number: -6 + -3i
product of Complex number: 17 + 33i</td><td></td></tr>
<tr><td></td><td><b>b)</b></td><td><b>i) Explain Errors and its types in detail.</b>

<b>ii) Explain thread methods to set and get priority.</b></td><td><b>6 M</b></td></tr>
<tr><td></td><td><b>Ans</b></td><td>
An error is an issue in a program that prevents the program from completing its task. There are several types of errors that occur in Java, including syntax errors, runtime errors, and logical errors. They are

- **Syntax Errors or Compilation Errors:** These occur when the code violates the rules of the Java syntax. These errors are usually caught by the Java compiler during the compilation phase.
- Example of compile time error:

```
public class Main
{
  public static void main(String[] args)
  {
    int x = "5";
  }
}
```

OUTPUT:



```
Main.java:5: error: incompatible types: String cannot be converted to int
        int x = "5";
                ^
1 error
```

- **Runtime Errors:** These errors occur when the code encounters an unexpected behaviour during its execution. These errors are usually caused by flawed logic or incorrect assumptions in the code and can be difficult to identify and fix.
- The most common run-time errors are:

a) Dividing an integer by zero
b) Accessing an element that is out of bounds of an array
c) Trying to store value into an array of an incompatible class or type Passing parameter that is not in a valid range or value for method
d) Trying to illegally change status of thread
e) Attempting to use a negative size for an array
f) Converting invalid string to a number
g) Accessing character that is out of bound of a string

These errors can be handled by uexception handling with help of try-catch- final block
</td><td>Types of errors with example – 3 M

and

thread methods with any relevant/correct example – 3 M</td></tr>
</table>

(i) Priorities in threads

To get and set priority of a thread in java following methods are used,

1. **public final int getPriority():** java.lang.Thread.getPriority() method returns priority of given thread.
2. **public final void setPriority(int newPriority):** java.lang.Thread.setPriority() method changes the priority of thread to the value newPriority. This method throws IllegalArgumentException if value of parameter newPriority goes beyond minimum(1) and maximum(10) limit.

Example:

```
// Java Program to Illustrate Priorities in Multithreading
// via help of getPriority() and setPriority() method

// Importing required classes
import java.lang.*;

// Main class
class ThreadDemo extends Thread {

        // Method 1
        // run() method for the thread that is called
        // as soon as start() is invoked for thread in main()
        public void run()
        {
                // Print statement
                System.out.println("Inside run method");
        }

        // Main driver method
        public static void main(String[] args)
        {
                // Creating random threads
                // with the help of above class
                ThreadDemo t1 = new ThreadDemo();
                ThreadDemo t2 = new ThreadDemo();
                ThreadDemo t3 = new ThreadDemo();

                // Thread 1
                // Display the priority of above thread using getPriority() method
                System.out.println("t1 thread priority : " + t1.getPriority());

                // Thread 1
                // Display the priority of above thread
                System.out.println("t2 thread priority : " + t2.getPriority());

                // Thread 3
                System.out.println("t3 thread priority : " + t3.getPriority());
```

```
                    // Setting priorities of above threads by passing integer arguments
                    t1.setPriority(2);
                    t2.setPriority(5);
                    t3.setPriority(8);

                    System.out.println("t1 thread priority : "+ t1.getPriority());
                    System.out.println("t2 thread priority : "+ t2.getPriority());
                    System.out.println("t3 thread priority : "+ t3.getPriority());

                    // Main thread

                    // Displays the name of currently executing Thread
                    System.out.println( "Currently Executing Thread : " +
Thread.currentThread().getName());

                    System.out.println( "Main thread priority : " +
Thread.currentThread().getPriority());

                    // Main thread priority is set to 10
                    Thread.currentThread().setPriority(10);

                    System.out.println("Main thread priority : " +
Thread.currentThread().getPriority());
          }
}
```

OUTPUT:

```
t1 thread priority : 5
t2 thread priority : 5
t3 thread priority : 5
t1 thread priority : 2
t2 thread priority : 5
t3 thread priority : 8
Currently Executing Thread : main
Main thread priority : 5
Main thread priority : 10
```

	c)	**Write a program to draw a chessboard in Java Applet.**	**6 M**

Ans

Correct program – 6 M

```
import java.applet.*;
import java.awt.*;
/*<applet code="Chess" width=600 height=600>
</applet>*/
// Extends Applet Class
public class Chess extends Applet
{
          static int N = 10;
          // Use paint() method
          public void paint(Graphics g)
          {
```

```
        int x, y;
        for (int row = 0; row & lt; N; row++) {

            for (int col = 0; col & lt; N; col++) {

                // Set x coordinates of rectangle
                // by 20 times
                x = row * 20;
                // Set y coordinates of rectangle
                // by 20 times
                y = col * 20;

            // Check whether row and column are in even position
                // If it is true set Black color
                if ((row % 2 == 0) == (col % 2 == 0))
                    g.setColor(Color.BLACK);
                else
                    g.setColor(Color.WHITE);

                // Create a rectangle with
                // length and breadth of 20
                g.fillRect(x, y, 20, 20);
            }
        }
    }
}
```

Java Summer 24 - Model ans paper

Java Programming (Maharashtra State Board of Technical Education)

Java Summer 24

1.Attempt any FIVE of the following:10

a)State the significance of Java Virtual Machine (JVM) in the Java programming environment.

Ans. The Java Virtual Machine (JVM) is crucial to the Java programming environment because it enables Java applications to run on any operating system by translating Java bytecode into machine-specific code, effectively achieving platform independence, allowing developers to "write once, run anywhere" with their Java programs; it also manages memory and provides a runtime environment for executing Java applications across different platforms

b)Define array. List its types.

Ans. An array is a homogeneous data type where it can hold only
objects of one data type.
Types of Array
1)One-Dimensional
2)Two-Dimensional

c)State use of finalize() method with its syntax.

Ans.Use of finalize( ):

Sometimes an object will need to perform some action when it is destroyed. Eg. If an object holding some non java resources such as file handle or window character font, then before the object is garbage collected these resources should be freed. To handle such situations java provide a mechanism called finalization. In finalization, specific actions that are to be done when an object is
garbage collected can be defined. To add finalizer to a class define the finalize() method. The java run-time calls this method whenever it is about to recycle an object.

Syntax:

```
protected void finalize() {
}
```

d)Define the interface in Java. Write the syntax.

Ans. Interface is similar to a class.It consist of only abstract methods and final variables. To implement an interface a class must define each of the method declared in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

e)Define thread. Mention 2 ways to create thread.

Ans. Thread is a smallest unit of executable code or a single task is also called as thread.Each tread has its own local variable, program counter and lifetime. A thread is similar to program that has a single flow of control.

There are two ways to create threads in java:

1. By extending thread class

Syntax: -

```
class Mythread extends Thread
{
-----
}
```

2. Implementing the Runnable Interface

Syntax:

```
class MyThread implements Runnable
{
public void run()
{
------
}
```

f)Write syntax of draw Rect().

Ans. In Java, the drawRect() method is part of the Graphics class, which is used to draw shapes on components. The drawRect() method specifically draws a rectangle defined by its top-left corner's coordinates, width, and height.

Syntax:

```
void drawRect(int x, int y, int width, int height)
```

g)Give the use of <PARAM>tag in applet.

Ans. The PARAM element is used to provide "command-line" arguments to a Java applet, which is embedded in a document with the APPLET element.

Syntax:

```
<param name="name"value="value”>
```

Example:

```
<param name="color” value="red”>
```

2. Attempt any THREE of the following:12

a)Explain the concept of platform independence in Java and discuss how it is achieved. Give example to illustrate the concept.

Ans. Java is a platform independent language. This is possible because when a java program is compiled, an intermediate code called the byte code is obtained rather than the machine code.

Byte code is a highly optimized set of instructions designed to be executed by the JVM which is the interpreter for the byte code. Byte code is not a machine specific code. Byte code is a universal code and can be moved anywhere to any platform. Therefore java is portable, as it can be carried to any platform. JVM is a virtual machine which exists inside the computer memory and is a simulated computer within a computer which does all the functions of a computer. Only the JVM needs to be implemented for each platform. Although the details of the JVM will defer from platform to platform, all interpret the same byte code.

```
// HelloWorld.java
public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello, World!");
   }
}
```

Steps to Demonstrate Platform Independence:

Compile the Code:

Compile the Java code using the command:

```
javac HelloWorld.java
```

Run on Different Platforms:

You can run the compiled bytecode on any platform that has a JVM installed using the command:

```
java HelloWorld
```

The output will be:

```
Hello, World!
```

b)What happens if you don't define any constructor in a class?Can you still create objects of that class? Explain with example.

Ans. If you don't define any constructors in a Java class, the Java compiler automatically provides a **default constructor**. This default constructor allows you to create objects of that class. The default constructor has no parameters and initializes instance variables to their default values (e.g., 0 for int, false for boolean, and null for object references).

Detailed Explanation

1. Default Constructor

- A **default constructor** is a no-argument constructor automatically created by the Java compiler if no constructors are explicitly defined in the class.
- This constructor initializes all instance variables to their default values.

- If you define any constructor in the class (with or without parameters), the default constructor will not be created unless explicitly defined.

2. Creating Objects Without Defining a Constructor

You can still create objects of a class without defining any constructors. The default constructor takes care of that. Here’s an example to illustrate this concept:

Example:

```
// Class definition without an explicitly defined constructor
public class MyClass {
   // Instance variable
   int value; // default value will be 0

   // Another instance variable
   String name; // default value will be null

   // Method to display values
   public void display() {
      System.out.println("Value: " + value);
      System.out.println("Name: " + name);
   }
}

public class Main {
   public static void main(String[] args) {
      // Creating an object of MyClass using the default constructor
      MyClass obj = new MyClass();

      // Calling the display method to see the initialized values
      obj.display();
   }
}
```

Output:

```
Value: 0
Name: null
```

c)Write a Java program in which thread A will display the even numbers between 1 to 50 and thread B will display the odd numbers between 1 to 50. After 3 iterations thread A should go to sleep for 500 ms.

Ans.

```
class EvenThread extends Thread {
   public void run() {
        for (int i = 1; i <= 50; i++) {
            if (i % 2 == 0) { // Check if the number is even
                System.out.println("Thread A (Even): " + i);

                // After 3 iterations, put the thread to sleep
                if (i / 2 == 3) { // After the third even number (i.e., 6)
                    try {
                        System.out.println("Thread A going to sleep for 500 ms.");
                        Thread.sleep(500); // Sleep for 500 milliseconds
                    } catch (InterruptedException e) {
                        System.out.println("Thread A interrupted.");
                    }
                }
            }
        }
    }
}

class OddThread extends Thread {
    public void run() {
        for (int i = 1; i <= 50; i++) {
            if (i % 2 != 0) { // Check if the number is odd
                System.out.println("Thread B (Odd): " + i);
            }
        }
    }
}

public class EvenOddThreads {
    public static void main(String[] args) {
        EvenThread threadA = new EvenThread();
        OddThread threadB = new OddThread();

        // Start both threads
        threadA.start();
        threadB.start();

        // Wait for both threads to finish
        try {
            threadA.join();
            threadB.join();
        } catch (InterruptedException e) {
            System.out.println("Main thread interrupted.");
        }

        System.out.println("Both threads have finished execution.");
    }
}
```

d)Explain multilevel inheritance with example.

Ans.Multilevel inheritance:

In multilevel inheritance, a subclass extends from a superclass and then the same subclass acts as a superclass for another class.Basically it appears as derived from a derived class.

Example:

```
class A
{
void display()
{
System.out.println(â€œIn Parent class Aâ€);
}
}
class B extends A //derived class B from A
{
void show()
{
System.out.println(â€œIn child class Bâ€);
}
}
class C extends B //derived class C from B
{
public void print()
{
System.out.println(â€œIn derived from derived class Câ€);
}
public static void main(String args[])
{
C c= new C();
c.display(); //super class method call
c.show(); // sub class method call
c.print(); //sub-sub class method call
}
}
```

3.Attempt any THREE of the following:12

(a)Define a class employee with data members 'empid", 'name’ and "salary'. Accept data for three objects and display it.

Ans.

```
import java.util.Scanner;

class Employee {
    // Data members
    int empId;
    String name;
    double salary;

    // Method to accept employee data
    public void acceptData(Scanner sc) {
        System.out.print("Enter Employee ID: ");
        empId = sc.nextInt();
        sc.nextLine();  // Consume newline

        System.out.print("Enter Employee Name: ");
        name = sc.nextLine();

        System.out.print("Enter Employee Salary: ");
        salary = sc.nextDouble();
    }

    // Method to display employee data
    public void displayData() {
        System.out.println("Employee ID: " + empId);
        System.out.println("Employee Name: " + name);
        System.out.println("Employee Salary: $" + salary);
        System.out.println();
    }
}

public class EmployeeTest {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Creating three Employee objects
        Employee emp1 = new Employee();
        Employee emp2 = new Employee();
        Employee emp3 = new Employee();

        // Accepting data for each employee
        System.out.println("Enter details for Employee 1:");
        emp1.acceptData(sc);

        System.out.println("\nEnter details for Employee 2:");
        emp2.acceptData(sc);

        System.out.println("\nEnter details for Employee 3:");
        emp3.acceptData(sc);

        // Displaying data for each employee
        System.out.println("\nDisplaying Employee Details:");
        emp1.displayData();
        emp2.displayData();
        emp3.displayData();

        sc.close();
    }
}
```

b)How can the "super" keyword be used in inheritance? Give an example to demonstrate its usage.

Ans. The super keyword is used in inheritance to refer to the **parent class**. It serves multiple purposes:

1. **Calling the parent class constructor**: The super() call can be used to explicitly call the constructor of the parent class from within the child class constructor. If not explicitly called, Java automatically calls the parent class's default (no-argument) constructor.
2. **Accessing parent class methods**: super can be used to invoke a method from the parent class that has been overridden in the child class.
3. **Accessing parent class fields**: If a field in the child class has the same name as a field in the parent class, super can be used to distinguish the parent class field from the child class field.

Example

```
// Parent class
class Animal {
    String name = "Animal";

    // Constructor of the parent class
    public Animal() {
        System.out.println("Animal constructor called");
    }

    // Parent class method
    public void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Child class
class Dog extends Animal {
    String name = "Dog";

    // Constructor of the child class
    public Dog() {
        // Calling the parent class constructor explicitly
        super();
        System.out.println("Dog constructor called");
    }

    // Child class method overriding parent class method
    @Override
    public void sound() {
        super.sound();  // Calling the parent class method
        System.out.println("Dog barks");
    }

    // Method to show the use of super for accessing parent class field
    public void showName() {
        System.out.println("Child class name: " + name);  // Refers to the child class field
        System.out.println("Parent class name: " + super.name);  // Refers to the parent class field
    }
}
```

```
// Main class
public class Main {
   public static void main(String[] args) {
      // Create an object of the Dog class
      Dog dog = new Dog();

      // Call the overridden method
      dog.sound();

      // Show the use of super to access parent class fields
      dog.showName();
   }
}
```

Output:

```
Animal constructor called
Dog constructor called
Animal makes a sound
Dog barks
Child class name: Dog
Parent class name: Animal
```

c)Explain the following with syntax:

i)drawLine

ii)drawOval

iii)drawAre

iv) drawString

Ans.1. drawLine()

The drawLine() method is used to draw a straight line between two points in a graphical window.

Syntax:

```
public void drawLine(int x1, int y1, int x2, int y2)
```

- **x1, y1**: The starting coordinates of the line.
- **x2, y2**: The ending coordinates of the line.

Example:

```
import java.awt.*;
import javax.swing.*;

public class LineExample extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);
```

```
        // Drawing a line from point (50, 50) to point (250, 250)
        g.drawLine(50, 50, 250, 250);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Draw Line Example");
        LineExample panel = new LineExample();

        frame.add(panel);
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

2. drawOval()

The drawOval() method is used to draw the outline of an oval within a bounding rectangle.

Syntax:

public void drawOval(int x, int y, int width, int height)

- **x, y**: The top-left corner of the bounding rectangle.
- **width**: The width of the oval.
- **height**: The height of the oval.

Example:

```
import java.awt.*;
import javax.swing.*;

public class OvalExample extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Drawing an oval with top-left corner at (100, 100) and width and height of 200
        g.drawOval(100, 100, 200, 150);
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Draw Oval Example");
        OvalExample panel = new OvalExample();

        frame.add(panel);
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

3. drawArc()

The drawArc() method is used to draw the outline of an arc inside an oval. It is defined by specifying the start angle and the arc angle.

Syntax:

public void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)

- **x, y**: The top-left corner of the bounding rectangle of the oval that contains the arc.
- **width, height**: The width and height of the oval.
- **startAngle**: The starting angle of the arc, measured in degrees (0 degrees is at the 3 o'clock position).
- **arcAngle**: The angle in degrees that covers the arc (positive is counterclockwise, negative is clockwise).

Example:

```
import java.awt.*;
import javax.swing.*;

public class ArcExample extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Drawing an arc with start angle of 0 degrees and arc angle of 180 degrees
        g.drawArc(100, 100, 200, 150, 0, 180);  // A half-circle arc
    }

    public static void main(String[] args) {
        JFrame frame = new JFrame("Draw Arc Example");
        ArcExample panel = new ArcExample();

        frame.add(panel);
        frame.setSize(400, 400);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

4. drawString()

The drawString() method is used to draw a string (text) at the specified location in the window.

Syntax:

```
public void drawString(String str, int x, int y)
```

- **str**: The string to be drawn.
- **x, y**: The position where the string starts (with respect to the baseline of the first character).

Example:

```
import java.awt.*;
import javax.swing.*;
public class StringExample extends JPanel {
   public void paintComponent(Graphics g) {
      super.paintComponent(g);

      // Drawing the string "Hello World!" at position (100, 100)
      g.drawString("Hello World!", 100, 100);
   }
   public static void main(String[] args) {
      JFrame frame = new JFrame("Draw String Example");
      StringExample panel = new StringExample();

      frame.add(panel);
      frame.setSize(400, 400);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

d)What is the concept of streams in Java? How do streams facilitate input and output operations?

Ans. In Java, **streams** are used to handle input and output (I/O) operations, such as reading from and writing to files, consoles, or network connections. A stream can be thought of as a continuous flow of data from a source to a destination, allowing Java programs to process data efficiently. Streams abstract the concept of input and output and provide a consistent and simplified way to deal with data.

There are two types of streams in Java:

1. **Input Streams**: These are used to read data from a source (e.g., a file, a network, or the console).
2. **Output Streams**: These are used to write data to a destination (e.g., a file, a network, or the console).

Types of Streams in Java

Java categorizes streams into two types based on the type of data being handled:

1. **Byte Streams**: Used to handle raw binary data. They read and write data byte by byte. Examples:
    a. **InputStream**: For reading byte data.
    b. **OutputStream**: For writing byte data.
2. **Character Streams**: Used to handle textual data (characters). They read and write data character by character, making them suitable for dealing with text. Examples:
    a. **Reader**: For reading character data.
    b. **Writer**: For writing character data.

How Streams Facilitate I/O Operations

Streams in Java are designed to abstract the source or destination of data, allowing a program to handle data flow seamlessly. Streams follow a common pattern:

1. **Opening a stream**: Connects to the data source or destination.
2. **Processing data**: Reads or writes data.
3. **Closing the stream**: Releases resources once the data is processed.

Java provides various stream classes in the **java.io** package to facilitate input and output operations, such as reading from files, writing to files, or dealing with network streams. The stream abstraction hides the details of how data is managed internally, allowing the developer to focus on the higher-level logic.

Stream Classes in Java

1. Byte Stream Classes:

- **InputStream**: Abstract superclass for reading byte data.
    - Common subclasses: FileInputStream, BufferedInputStream
- **OutputStream**: Abstract superclass for writing byte data.
    - Common subclasses: FileOutputStream, BufferedOutputStream

2. Character Stream Classes:

- **Reader**: Abstract superclass for reading character data.
    - Common subclasses: FileReader, BufferedReader
- **Writer**: Abstract superclass for writing character data.
    - Common subclasses: FileWriter, BufferedWriter

4.Attempt any THREE of the following:12

a)Explain any two logical operators in Java with example.

Ans.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:

1. && (Logical AND)

- **Meaning**: This operator returns true only if **both** the conditions on its left and right sides are true. If either or both of the conditions are false, the result will be false.

2. || (Logical OR)

- **Meaning**: This operator returns true if **at least one** of the conditions on its left or right side is true. If both conditions are false, the result will be false.

3. ! (Logical NOT)

- **Meaning**: This operator **negates** or reverses the truth value of the condition it precedes. If the condition is true, !will make it false, and if it is false, ! will make it true.

Program demonstrating logical Operators

```
public class Test
{
public static void main(String args[])
{
boolean a = true;
boolean b = false;
System.out.println("a && b = " + (a&&b));
System.out.println("a || b = " + (a||b) );
System.out.println("!(a && b) = " + !(a && b));
}
}
```

Output:

```
a && b = false
a || b = true
!(a && b) = true
```

b)What is constructor? List types of constructor. Explain parameterized constructor with suitable example.

Ans. Constructors are used to initialize an object as soon as it is created. Every time an object is created using the â€˜newâ€™ keyword, a constructor is invoked. If no constructor is defined in a class, java compiler creates a default constructor. Constructors are similar to methods but with to differences, constructor has the same name as that of the class and it does not return any value.

The types of constructors are:

1. Default constructor
2. Constructor with no arguments
3. Parameterized constructor
4. Copy constructor

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

```
class Student {
```

```
int roll_no;
String name;
Student(int r, String n) // parameterized constructor
{
roll_no = r;
name=n;
}
void display()
{
System.out.println("Roll no is: "+roll_no);
System.out.println("Name is : "+name);
}
public static void main(String a[])
{
Student s = new Student(20,"ABC"); // constructor
with parameters
s.display();
}
}
```

c)Implement the following inheritance. Refer Fig. No. 01.
Ans.

d)Explain Life Cycle of the applet with neat diagram.
Ans. When an applet begins, the AWT calls the following methods, in this sequence:

1. init( )
2. start( )
3. paint( )

When an applet is terminated, the following sequence of method calls takes place:

4. stop( )
5. destroy( )

init ( ): The **init( )** method is the first method to be called. This is where you should initialize Variables. This method is called only once during the run time of your applet.
start( ): The **start( )** method is called after **init( )**.It is also called to restart an applet after it has Been stopped. Whereas **init( )** is called onceâ€”the first time an applet is loadedâ€”**start( )**is called each time an appletâ€™s HTML document is displayed onscreen.
Paint ( ): The **paint ( )** method is called each time your appletâ€™s

output must be redrawn. 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.

Stop ( ): When stop ( ) is called, the applet is probably running. You should use stop ( ) to suspend threads that donâ€™t need to run when the applet is not visible.

destroy( ): The destroy ( ) method is called when the environment determines that your applet needs to be removed completely from memory.

e)Create a new test file named "data.txt" using the File class.Write a program to count number of words of "data.txt" using stream classes.

Ans.

```
import java.io.*;
public class WordCount {
    public static void main(String[] args) {
        // Step 1: Create a new file named "data.txt"
        try {
            File file = new File("data.txt");
            // If file does not exist, create a new one
            if (!file.exists()) {
                file.createNewFile();
            }
            // Write some sample content to the file for counting words
            FileWriter writer = new FileWriter(file);
            writer.write("This is a test file.\nIt contains some words to count.");
            writer.close();
            // Step 2: Count the number of words using stream classes
            FileReader fr = new FileReader(file);
            BufferedReader br = new BufferedReader(fr);
            String line;
            int wordCount = 0;
            // Read each line from the file
            while ((line = br.readLine()) != null) {
                // Split the line into words based on spaces
                String[] words = line.split("\\s+");
                wordCount += words.length;
            }
            // Close the BufferedReader and FileReader
            br.close();
            fr.close();
            // Display the word count
            System.out.println("Total number of words in the file: " + wordCount);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output:

```
Total number of words in the file: 9
```

5.Attempt any TWO of the following:12

a)Explain the concept of argument passing and the usage of 'this' keyword in Java give example to illustrate their usage and benefits.

Ans. Argument passing is a fundamental concept in programming that refers to how data is transmitted to functions or methods when they are invoked. Understanding argument passing is essential for writing effective and efficient code. Here's a detailed explanation of the different types of argument passing, including their mechanics, advantages, and implications.

1. Types of Argument Passing

There are primarily two types of argument passing: **call by value** and **call by reference**. Some languages also use a third type called **call by name**.

a. Call by Value

- **Definition**: When a function is called, a copy of the actual argument is made and passed to the function. Changes made to the parameter inside the function do not affect the original argument.

b. Call by Reference

- **Definition**: Instead of passing a copy, a reference (or address) to the actual data is passed. Therefore, changes made to the parameter will affect the original argument.

c. Call by Name

- **Definition**: The argument expression is not evaluated until it is actually used within the function. This means that the argument can be recomputed each time it is accessed.

The this keyword is a reference variable that refers to the current object, allowing you to access the object's instance variables and methods. It is particularly useful in various scenarios, such as differentiating between instance variables and parameters, invoking constructors, and passing the current object as a parameter to another method.

Common Usages of this

1. **Distinguishing Instance Variables from Parameters**: When parameters have the same name as instance variables, this can clarify which variable is being referenced.
2. **Calling Constructors**: this can be used to call another constructor in the same class (constructor chaining).
3. **Returning the Current Object**: You can return the current object from a method using this, which is particularly useful in method chaining.
4. **Passing the Current Object**: You can pass the current object to another method or constructor.

Example:

```
class Rectangle {
  private int length;
  private int width;
  // Constructor to initialize rectangle
  public Rectangle(int length, int width) {
    this.length = length;  // Using 'this' to differentiate instance variable from parameter
    this.width = width;    // Using 'this' for clarity
  }
  // Method to calculate area
  public int calculateArea() {
    return this.length * this.width;  // 'this' is optional here but adds clarity
  }
  // Method to compare areas of two rectangles
  public boolean isLargerThan(Rectangle other) {
    return this.calculateArea() > other.calculateArea();  // Using 'this' to refer to the current
object
  }
  // Method to create a new Rectangle and return it (method chaining)
  public Rectangle resize(int newLength, int newWidth) {
    this.length = newLength;
    this.width = newWidth;
    return this;  // Returning the current object for method chaining
  }
  // Main method to demonstrate usage
  public static void main(String[] args) {
    Rectangle rect1 = new Rectangle(10, 5);
    Rectangle rect2 = new Rectangle(6, 8);
    System.out.println("Area of rect1: " + rect1.calculateArea()); // Output: 50
    System.out.println("Area of rect2: " + rect2.calculateArea()); // Output: 48
    // Comparing areas
    if (rect1.isLargerThan(rect2)) {
        System.out.println("rect1 is larger than rect2");
    } else {
        System.out.println("rect2 is larger than or equal to rect1");
    }
    // Resizing rect1 and printing its new area
    rect1.resize(15, 10);
    System.out.println("New area of rect1 after resizing: " + rect1.calculateArea()); // Output:
150
  }
}
```

Benefits of Using this

1. **Clarity**: Using this clarifies that you are referring to instance variables or methods, which can improve code readability and maintainability.
2. **Avoids Ambiguity**: In cases where local variables or parameters have the same name as instance variables, thishelps avoid confusion.
3. **Constructor Chaining**: It enables constructor chaining, allowing you to call one constructor from another, which can help reduce redundancy.
4. **Method Chaining**: By returning this, it allows for a fluent interface style of coding, making it easier to write and read code when multiple method calls are chained together.
5. **Passing Current Object**: It allows you to pass the current object to other methods or constructors, which can be useful for callbacks or factory methods.

b)Explain the concept of packages in Java and their significance in software development. XWrite an example to illustrate the usage and benefits of using packages.

Ans.a **package** is a namespace that organizes a set of related classes and interfaces. Conceptually similar to folders on your computer, packages help in structuring code in a modular way. They serve multiple purposes, including avoiding name conflicts, controlling access, and making it easier to manage and maintain code. Here's a detailed explanation of packages in Java and their significance in software development.

1. What is a Package?

A package is a collection of classes and interfaces that are grouped together under a specific namespace. In Java, packages are defined using the package keyword at the top of a Java source file.

Syntax:

package com.example.myapp; // Declaring a package

a. Built-in Packages

Java comes with a set of built-in packages that contain classes and interfaces for various functionalities. Some commonly used built-in packages include:

- java.lang: Contains fundamental classes like String, Math, and System. Automatically imported into every Java program.

- java.util: Contains utility classes, including collections like ArrayList, HashMap, and utility classes like Dateand Random.
- java.io: Provides classes for input and output through data streams, serialization, and the file system.

b. User-defined Packages

Developers can create their own packages to group related classes and interfaces according to their application’s needs. User-defined packages help maintain a clear structure and organization in large applications.

Creating a User-defined Package

1. Declare the package at the top of your Java file.
2. Compile the Java file with the package declaration.
3. The compiled class files are stored in a directory structure matching the package name.

3. Accessing Packages

To use classes and interfaces from a package, you need to import them into your Java file. The import statement is used for this purpose.

Syntax:

```
import com.example.util.MyUtility; // Importing a specific class
import com.example.util.*; // Importing all classes from the package
```

4. Significance of Packages in Software Development

The use of packages in Java software development has several advantages:

a. Name Conflicts Prevention

Packages provide a way to group related classes and interfaces under a unique namespace. This helps avoid naming conflicts, especially when different libraries or modules might have classes with the same name.

b. Code Organization and Structure

Packages allow developers to organize their code logically. By grouping related classes and interfaces, developers can create a well-structured application that is easier to navigate and maintain.

- **Hierarchy**: Packages can be nested, creating a hierarchy that reflects the logical structure of the application.

Example:

```
com
 └── example
     ├── app
     └── util
```

c. Access Control

Packages play a crucial role in access control. Java uses package-private access (default access modifier) to restrict access to classes, methods, and variables within the same package. Members declared without any access modifier are only accessible to other classes in the same package.

- **Public**: Accessible from any other class.

- **Private**: Accessible only within the class itself.
- **Protected**: Accessible within the package and by subclasses.
- **Default (Package-private)**: Accessible only within the same package.

d. Reusability

Packages promote code reusability. Once a package is created, its classes and interfaces can be reused in other projects by simply importing the package. This reduces code duplication and promotes efficient software development.

e. Maintainability

Organized code structures with packages make it easier to manage and maintain applications. Developers can locate and update classes quickly, leading to improved productivity and reduced chances of introducing bugs.

f. Collaboration

In large software projects involving multiple developers, packages facilitate collaboration by providing clear boundaries. Different teams can work on different packages, reducing the risk of conflicts and enhancing team productivity.

Example:

```
package com.example.math; // Declaring the package
public class Calculator {
    // Method to add two numbers
    public int add(int a, int b) {
        return a + b;
    }

    // Method to subtract two numbers
    public int subtract(int a, int b) {
        return a - b;
    }

    // Method to multiply two numbers
    public int multiply(int a, int b) {
        return a * b;
    }

    // Method to divide two numbers
    public double divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero.");
        }
        return (double) a / b;
    }
}
import com.example.math.Calculator; // Importing the Calculator class from the package

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator(); // Creating an instance of Calculator

        // Performing calculations
        int sum = calc.add(10, 5);
        int difference = calc.subtract(10, 5);
        int product = calc.multiply(10, 5);
        double quotient = calc.divide(10, 5);

        // Displaying the results
        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("Quotient: " + quotient);
    }
}
```

Output:

```
Sum: 15
Difference: 5
Product: 50
Quotient: 2.0
```

Benefits of using packages:

1. Namespace Management

- **Avoid Naming Conflicts**: Packages provide a unique namespace for classes and interfaces, reducing the risk of naming conflicts. For instance, two classes can have the same name as long as they are in different packages.

2. Code Organization

- **Logical Grouping**: Packages allow developers to group related classes and interfaces together logically. This makes the codebase easier to navigate and understand, especially in large applications.
- **Hierarchical Structure**: Packages can be nested, allowing for a hierarchical organization that reflects the structure of the application.

3. Access Control

- **Encapsulation**: Packages enable better access control through access modifiers. Classes, methods, and variables can be marked as public, private, protected, or package-private

(default), helping to encapsulate implementation details and exposing only necessary components.
- **Controlled Visibility**: By using package-private access, developers can restrict access to classes and methods to only those within the same package, enhancing encapsulation and security.

4. Code Reusability

- **Reusable Components**: Once a package is created, its classes and interfaces can be reused across different projects. This reduces code duplication and improves development efficiency.
- **Library Creation**: Developers can create libraries of reusable code that can be shared among multiple projects or teams, promoting best practices and standardized solutions.

5. Improved Maintainability

- **Easier Maintenance**: A well-structured package organization makes it easier to locate and update code. When changes are needed, developers can quickly find the relevant classes and methods.
- **Modularity**: Packages promote modularity in applications, enabling developers to modify or replace individual components without affecting the entire system.

6. Collaboration and Teamwork

- **Clear Boundaries**: In large teams, packages help delineate responsibilities, allowing different teams or developers to work on different packages without stepping on each other's toes.
- **Version Control**: Packages facilitate version control and dependency management by providing a clear structure for organizing changes.

7. Easier Deployment

- **Bundled Classes**: Packages allow for easier deployment of grouped classes and interfaces. Instead of deploying each class individually, you can deploy a package containing all related components.
- **Java Archive (JAR) Files**: Developers can package their classes into JAR files, making it easier to distribute and manage libraries and applications.

8. Framework and API Design

- **Frameworks**: Packages are essential for creating frameworks, allowing developers to build extensible and reusable components that others can use and build upon.
- **API Development**: Well-designed packages help organize APIs, making them easier to understand and use for other developers.

c)Explain the concept of exception handling in Java and its importance in robust programming. Provide an example to illustrate the implementation and benefits of exception handling

Ans. An **exception** is an event that disrupts the normal flow of a program's execution. It indicates that an unexpected condition has occurred, which can arise from various sources, such as invalid user input, network failures, file not found errors, or other unforeseen issues.

2. Types of Exceptions

In Java, exceptions are categorized into two main types:

a. Checked Exceptions

- **Definition**: These are exceptions that are checked at compile time. The compiler requires that these exceptions be either caught or declared in the method signature using the throws keyword.
- **Examples**: IOException, SQLException, FileNotFoundException.

b. Unchecked Exceptions

- **Definition**: These are exceptions that are not checked at compile time. They are derived from the RuntimeException class and indicate programming errors that can be avoided through proper coding practices.
- **Examples**: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.

3. Exception Handling Mechanism

Java provides a robust framework for exception handling using five key keywords:

a. try

- A block of code that may throw an exception is placed within a try block. If an exception occurs, the flow of control is transferred to the corresponding catch block.

b. catch

- This block is used to handle the exception. You can specify the type of exception to catch. Multiple catch blocks can be used to handle different exception types.

c. finally

- A finally block can be added after the try and catch blocks. It contains code that will execute regardless of whether an exception occurred or was handled. It's commonly used for resource cleanup, such as closing file streams or database connections.

d. throw

- The throw keyword is used to explicitly throw an exception from a method or block of code. It can be used to signal an error condition.

e. throws

- The throws keyword is used in a method signature to declare that a method can throw certain exceptions. This informs callers of the method about potential exceptions.

Example

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ExceptionHandlingExample {
  public static void main(String[] args) {
    String filePath = "example.txt";

    try {
      // Attempting to read from a file
      BufferedReader reader = new BufferedReader(new FileReader(filePath));
      String line;
      while ((line = reader.readLine()) != null) {
        System.out.println(line);
      }
      reader.close(); // Always close resources in the finally block or try-with-resources
    } catch (IOException e) {
      // Handling checked exception
      System.err.println("An IOException occurred: " + e.getMessage());
    } finally {
      // Code that will always execute
      System.out.println("Execution completed.");
    }
  }
}
```

4. Importance of Exception Handling in Robust Programming

Exception handling is vital for robust programming for several reasons:

a. Graceful Degradation

- By handling exceptions, programs can continue running instead of crashing. This allows applications to provide useful feedback to users, log errors, or attempt recovery, leading to a better user experience.

b. Code Clarity

- Structured exception handling makes the code more readable and understandable. It separates normal logic from error-handling logic, making it easier for developers to follow the flow of the program.

c. Resource Management

- Exception handling helps manage resources effectively. For instance, if an exception occurs while working with files or database connections, you can use the finally block to ensure that resources are released properly.

d. Debugging and Maintenance

- Catching and logging exceptions provides valuable information about what went wrong in the application. This makes it easier to debug issues and maintain the code in the long run.

e. Predictability

- Well-designed exception handling allows developers to anticipate potential failure points and handle them appropriately, leading to more predictable and stable applications.

f. Separation of Concerns

- Exception handling allows developers to separate error-handling code from regular business logic. This separation helps to create cleaner and more maintainable code.

Benefits of exception handling:

1. Improved Code Reliability

- **Graceful Error Recovery**: Exception handling allows programs to recover from unexpected errors without crashing. This means that applications can continue to operate, albeit in a limited capacity, rather than terminating abruptly.

2. Separation of Error-Handling Code

- **Cleaner Code Structure**: Exception handling separates normal logic from error-handling logic, making the code cleaner and easier to read. This modularity helps developers understand the flow of the program without getting lost in error-handling details.

3. Resource Management

- **Proper Resource Cleanup**: Using finally blocks or try-with-resources ensures that resources such as files, database connections, and network sockets are closed properly, preventing resource leaks even when exceptions occur.

4. Predictability and Control

- **Controlled Behavior**: Exception handling provides developers with control over how errors are managed. This means that applications can define specific responses to different types of exceptions, leading to more predictable behavior.

5. Enhanced Debugging and Maintenance

- **Error Logging**: Catching exceptions allows developers to log error details, which aids in diagnosing issues and understanding application failures. This information is invaluable for debugging and improving code quality.
- **Easier Maintenance**: When exceptions are handled consistently, it becomes easier to maintain and update the code. Developers can focus on specific error scenarios and implement fixes or enhancements more efficiently.

6. User-Friendly Error Messages

- **Better User Experience**: Exception handling enables the application to provide informative error messages to users instead of cryptic stack traces. This improves the user experience by helping users understand what went wrong and how to resolve it.

7. Facilitates Debugging

- **Traceability**: Exceptions can carry stack traces that help identify where the error occurred, making it easier for developers to trace the source of the problem and fix it effectively.

8. Promotes Robustness

- **Robust Applications**: Proper exception handling contributes to building robust applications that can withstand various runtime issues, ensuring that the application performs reliably even in adverse conditions.

9. Support for Checked Exceptions

- **Compile-Time Error Checking**: Checked exceptions require developers to handle them at compile time, ensuring that potential error conditions are acknowledged and managed, which can lead to fewer runtime errors.

10. Encourages Best Practices

- **Consistent Error Handling**: Exception handling encourages developers to adopt best practices in error management, leading to a more disciplined approach to coding and reducing the likelihood of overlooking error scenarios.

6.Attempt any TWO of the following:12

a)Write a program to define class Employee with members as id and salary. Accept data for five employees and display details of employcss getting highest salary.

Ans.

```
import java.util.Scanner;
class Employee {
   private int id;
   private double salary;

   // Constructor
   public Employee(int id, double salary) {
      this.id = id;
      this.salary = salary;
   }

   // Getter methods
   public int getId() {
      return id;
   }

   public double getSalary() {
      return salary;
   }
}
public class EmployeeManagement {
   public static void main(String[] args) {
      Scanner scanner = new Scanner(System.in);
```

```
        Employee[] employees = new Employee[5];
        // Input employee details
        for (int i = 0; i < employees.length; i++) {
            System.out.print("Enter ID for Employee " + (i + 1) + ": ");
            int id = scanner.nextInt();
            System.out.print("Enter Salary for Employee " + (i + 1) + ": ");
            double salary = scanner.nextDouble();
            employees[i] = new Employee(id, salary);
        }
        // Find employee with the highest salary
        Employee highestPaidEmployee = employees[0];

        for (Employee employee : employees) {
            if (employee.getSalary() > highestPaidEmployee.getSalary()) {
                highestPaidEmployee = employee;
            }
        }
        // Display details of the employee with the highest salary
        System.out.println("\nEmployee with the Highest Salary:");
        System.out.println("ID: " + highestPaidEmployee.getId());
        System.out.println("Salary: " + highestPaidEmployee.getSalary());
        scanner.close(); // Close the scanner
    }
}
```

b)Define exception called 'No Match Exception' that is thrown when the password accepted is not equal to 'MSBTE'. Write the program.

Ans.

```
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
{
throw new NoMatchException("Strings are not equal");
else
System.out.println("Strings are equal");
}
catch(NoMatchException e)
{
System.out.println(e.getMessage());
}
}
}
```

c)Write a program to design an Applet showing three concentric circles filled with three different colors.

Ans.

```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;

// Note: To run this applet, it should be in an HTML file or an IDE that supports Java Applets.
public class ConcentricCirclesApplet extends Applet {
    @Override
    public void paint(Graphics g) {
        // Set the color for the outer circle and draw it
        g.setColor(Color.RED);
        g.fillOval(50, 50, 200, 200); // Outer circle

        // Set the color for the middle circle and draw it
        g.setColor(Color.GREEN);
        g.fillOval(70, 70, 160, 160); // Middle circle

        // Set the color for the inner circle and draw it
        g.setColor(Color.BLUE);
        g.fillOval(90, 90, 120, 120); // Inner circle
    }
}
<html>
<body>
```

```
    <applet code="ConcentricCirclesApplet.class" width="400" height="400"></applet>
</body>
</html>
```