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**

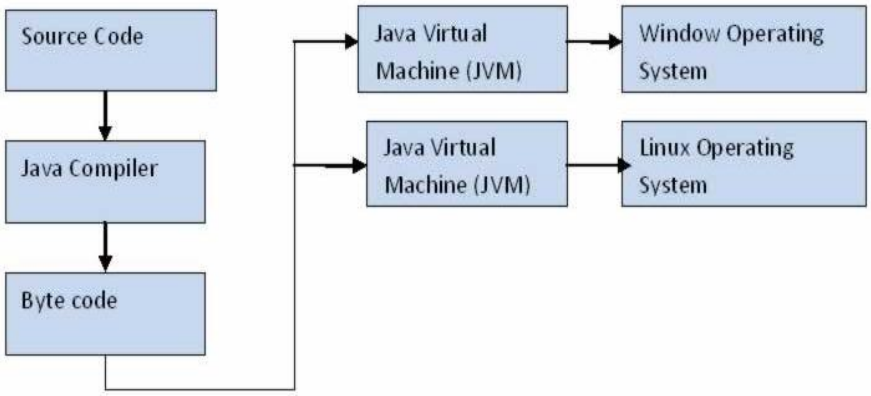
		destroyed. Eg. If an object holding some non java resources such as file handle or window character font, then before the object is garbage collected these resources should be freed. To handle such situations java provide a mechanism called finalization. In finalization, specific actions that are to 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) **Ans.**	**Explain the types of constructors in Java with suitable example.** ***(Note: Any two types shall be considered).*** 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 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). class test1 { int i; boolean b; byte bt; float ft; String s;	**4M** ***Explanation of the two types of constructors 2M*** ***Example 2M***

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

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.

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

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

SUMMER – 2019 EXAMINATION
MODEL ANSWER

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

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)
	Ans.
		Explain dynamic method dispatch in Java with suitable example.
		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.
		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()
			4M
			Explanation 2M
			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)
		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();
		}
		}
		}
			4M
			
			Correct program 4M

4.	**a)** **Ans.**	**Attempt any <u>THREE</u> of the following:** **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).	**12** **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
		}
		}
		
		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’.
			Description and example of each operator 2M

	b) Ans.	**Explain the four access specifiers in Java.** 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.	**4M** ***Each access specifiers 1M***

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

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) Ans. **Differentiate between Java Applet and Java Application (any four points)** — **4M**

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

SUMMER – 2019 EXAMINATION
MODEL ANSWER

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

	e) **Ans.**	**Write a program to copy content of one file to another file.**
		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();
		}
		}
		}
			4M
			Correct logic 2M
			Correct Syntax 2M

5.	**a)** **Ans.**	**Attempt any <u>TWO</u> of the following:** **Describe the use of any methods of vector class with their syntax.** ***(Note: Any method other than this but in vector class shall be considered for answer).*** • 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.	**12** **6M** ***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)
	
	Ans.
		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).
		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();
		}
		}
			6M
			
			Creation of two threads 4M
			
			Creating main to create and start objects of 2 threads: 2M

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

		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
		
		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
			4M for explanation
			
			2M for example

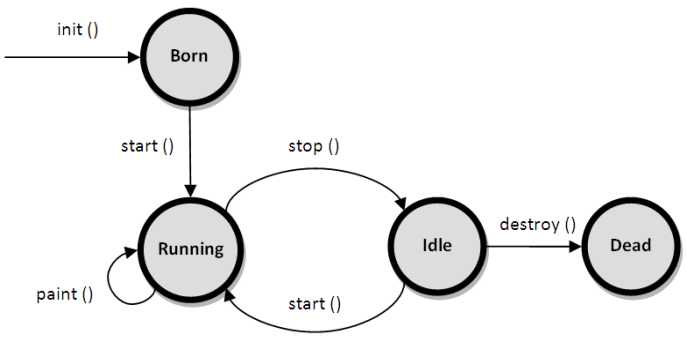
	b)
	
	Ans.
		Write a program to input name and salary of employee and throw user defined exception if entered salary is negative.
		import java.io.*;
		class NegativeSalaryException extends Exception
		{
		public NegativeSalaryException (String str)
		{
		super(str);
		}
		}
		public class S1
			6M
			
			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***