TED (10)-3069 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- OCTOBER, 2011

OOP THROUGH JAVA

(Common to CT, CM and IF)

[*Time:* 3 hours

(Maximum marks: 100)

Marks

PART –A
(Maximum marks: 10)

I. Answer all questions in a sentence

1. State the use of this keyword

   The keyword is used to refer to an object that invokes a method.

2. Write the syntax of creating a subclass

```
Class subclassname extends superclassname
{
        Variable declaration;
        Method declaration;
}
```

3. List the methods used for temporarily blocking a thread

   i. Sleep()
   ii. suspend()
   iii. wait()

4. Construct a tag to display an applet

```
<APPLET
        CODE = MyApplet.class
        WIDTH = 300
        HEIGHT = 200>
</APPLET>
```

5. Develop a statement to declare and define a variable to store subject name

String subjectname=”OOP Through Java”;

PART – B

II. Answer *any five* questions. Each question carries 6 marks

1. Differentiate between class and objects with the help of a suitable example

Objects:

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tengible and intengible). The example of integible object is banking system.

An object has three characteristics:

- **state:** represents data (value) of an object.
- **behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

Class:

Java class is nothing but a template for object you are going to create or it’s a blue print by using this we create an object. In simple word we can say it’s a specification or a pattern which we define and every object we define will follow that pattern

- When we create class in java the first step is keyword class and then name of the class or identifier we can say.
- Next is class body which starts with curly braces {} and between this all things related with that class means their property and method will come here.

Eg: Mango,apple and orange are members of the class fruit.Classes are user-defined data types and behave like the built – type

Eg: Fruit mango; // will create an object mango belonging to the class Fruit.

Whereas a class is a general concept (like an Animal), an object is a very specific embodiment of that class, with a limited lifespan (like a lion, cat, or a zebra). Another way of thinking about the difference between a class and an object is that a class provides

a template for something more specific that the programmer has to define, which he/she will do when creating an object of that class.

2. Illustrate invoking of constructors.

Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor

There are basically two rules defined for the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

```
class Complex
{
    int real,image;
    Complex(int x)
{
    real = 10;
image = 20;
    }
    void display()
{
        System.out.println(real + ”+” + image + ”i”);
}
}
classMainpgm
{
    public static void main(String args[])
    {
        Complex c1=new Complex(10);
        c1.display();
    }
}
```

3. Compare overloading and overriding of methods

- Overloading happens at compile-time while Overriding happens at runtime: The binding of overloaded method call to its definition has happens at compile-time however binding of overridden method call to its definition happens at runtime.
- Static methods can be overloaded which means a class can have more than one static method of same name. Static methods cannot be overridden, even if you declare a same static method in child class it has nothing to do with the same method of parent class.

- The most basic difference is that overloading is being done in the same class while for overriding base and child classes are required. Overriding is all about giving a specific implementation to the inherited method of parent class.
- Static binding is being used for overloaded methods and dynamic binding is being used for overridden/overriding methods.
- Performance: Overloading gives better performance compared to overriding. The reason is that the binding of overridden methods is being done at runtime.
- private and final methods can be overloaded but they cannot be overridden. It means a class can have more than one private/final methods of same name but a child class cannot override the private/final methods of their base class.
- Return type of overloaded methods should be same however in case of method overriding the return type of overriding method can be different from overridden method.
- Argument list should be different while doing method overloading. Argument list should be same in method Overriding.

Eg. For method overloading

```
Class Add
{
int sum(int a, int b)
  {return a + b;   }
int sum(int a)
 {return a + 10;   }
}
```

Eg. For method overriding

```
Class A  // Super Class
{
void display(intnum)
 {  printnum ;   }
}
//Class B inherits Class A
Class B //Sub Class
{
void display(intnum)
 {  printnum ;    }
}
```

4. Explain finalizer methods

Java supports a concept called finalization, which is just opposite to initialization. Sometimes an object will need to perform some action when it is destroyed. For example, if an object is holding some non-java resource such as a file handle or window character font, then you might want to make sure these resources are freed before an object is destroyed. To handle such situations, java provides a mechanism called finalization. By using finalization, you can define specific actions that will occur when an object is just about to be reclaimed by the garbage collector. To add a finalizer to a class, you simply define the **finalize()** method. The java run time calls that method whenever it is about to recycle an object of that class. Inside the **finalize()** method you will specify those actions that must be performed before an object is destroyed. The garbage collector runs periodically, checking for objects that are no longer referenced by any running state or indirectly through other referenced objects. Right before an asset is freed, the java run time calls the **finalize()**method on the object.

The **finalize()** method has this general form:

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

5. Describe various levels of access protection available for packages

a. Public

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe

```
class Example
{
publicint a;  //public field a
public void show()  //public method show
  {Body;}
}
```

b. Private

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself. Private access modifier is the most restrictive access level. Class and interfaces cannot be private.

```
class Example
```

```
{
privateint a;  //private field a
private void show()  //private method show
  {Body;}
}
```

c. Default/friendly access

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public

```
class Example
{
int a;  //default field a
void show()  //default method show
  {Body;}
}
```

d. Protected

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

```
class Example
{
int a;  //protected field a
void show()  //protected method show
  {   Body;  }
}
```

e. Private protected

Private protected members' visibility lies in between protected and private access. These members are visible in all sub-classes regardless of what package they are in.

6. Consider a class represents an account in a bank. Let the minimum balance for the account be Rs.1000. Write a java program that throws an exception when a withdrawal results in the balance to a value less than Rs.1000.

```
classMyException extends Exception
{
        MyException(string message)
        {
                super(message);     }
}
class Account
{
        int balance=5000;
        void withdraw(int amount)
        {
                try
                {
                        int b= balance – amount;
                        if(b<=1000)
                        throw new MyException(“minimum balance error”);
                        else
                        system.out.println(“transaction successful”);
                }
                catch(MyException e)
                {
                        system.out.println(e.getMessage());
}       }       }

class Test
{
        public static void main(String args[])
        {
                intamt=Integer.parseInt(arg[0]);
                Account 2a = new Account()
                a.withdraw(amt);
        }
}
```

7. Compare initialize and start life cycle stages of an applet

Initialization state

init(): The applet's voyage starts here. In this method, the applet object is created by the browser. Because this method is called before all the other methods, programmer can utilize this method to instantiate objects, initialize variables, setting background and

foreground colors in GUI etc.; the place of a constructor in an application. It is equivalent to born state of a thread.

At this stage we can do

- Create object needed by applet
- Setup initial values
- Load images or fonts
- Setup colors

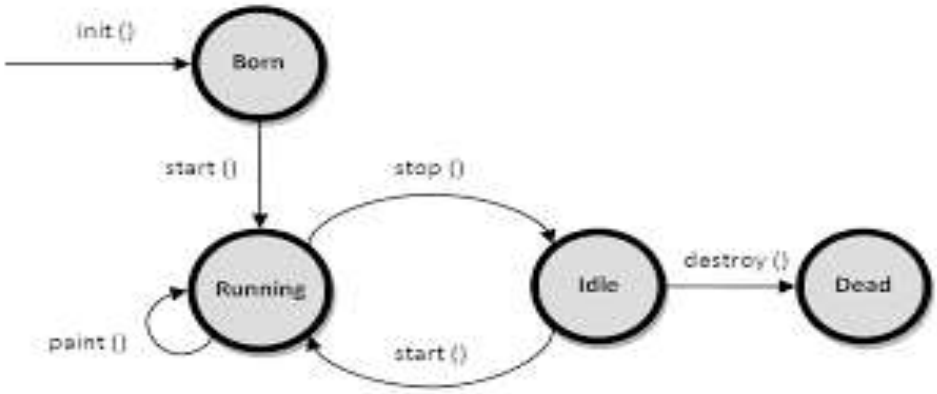
  The initialization occurs only once in the applet's lifecycle.

  public void init()

  {

  .....

  }



Running state

In init() method, even through applet object is created, it is in**inactive** state. An inactive applet is not eligible for microprocessor time even though the microprocessor is idle. To make the applet active, the init() method calls start() method. In start() method, applet becomes active and thereby eligible for processor time. **start()** method is called by the init() method. This method is called a number of times in the life cycle; whenever the applet is **deiconifed** , to make the applet active.

public void start()

{

.....

}

(5 x 6 = 30)

PART – C

(Answer *one* full question from each unit. Each question carries 15 marks)

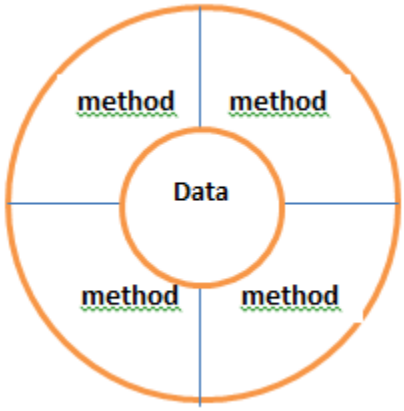
UNIT – I

III.

a) Describe the way in which the data and methods are organized in an object-oriented program

8

OOP threats data as a critical element in the program development and does not allow it to flow freely around the system. It ties data more closely to the method that operate on it and protect it from unintentional modification by other methods. OOP allow it to decompose a problem into a number of entities called objects and then build data and methods around these entities. The combination of data and method make up an object



b) Write a program which demonstrates constructor overloading

7

Constructor Overloading Program

```
class Example
{
    inta,b;
    Example()
    {
        a = 10; b = 20;
    }
    Example(int x)
    {
        a = x; b = x;
```

```
        }
        Example(intx,int y)
        {
                a = x; b = y;
        }
        void show()
        {
                System.out.println(a);
                System.out.println(b);

        }
}
class Test
{
        public static void main(String args[])
        {
                Example e1=new Example();
                e1.show();
                Example e2=new Example(50);
                e2.show();
                Example e3=new Example(100,30);
                e3.show();
        }
}
```

OR

IV.

a) Write a short note on

i. Data abstraction and data encapsulation

Encapsulate means to hide. Encapsulation is also called data hiding.You can think Encapsulation like a capsule (medicine tablet) which hides medicine inside it. Encapsulation is wrapping, just hiding properties and methods. Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world. Class is the best example of encapsulation. OOPS makes use of encapsulation to enforce the integrity of a type (i.e. to make sure data is used in an appropriate manner) by preventing programmers from accessing data in a non-intended manner. Through encapsulation, only a predetermined group of

functions can access the data. The collective term for datatypes and operations (methods) bundled together with access restrictions (public/private, etc.) is a class.

Abstraction refers to showing only the necessary details to the intended user. As the name suggests, abstraction is the "abstract form of anything". We use abstraction in programming languages to make abstract class. Abstract class represents abstract view of methods and properties of class. Abstraction is implemented using interface and abstract class while Encapsulation is implemented using private and protected access modifier.

ii. Dynamic binding and message passing

Message Passing is nothing but sending and receiving of information by the objects same as people exchange information. So this helps in building systems that simulate real life. Following are the basic steps in message passing. Message passing simply means that (at a very abstract level) the fundamental mechanism of program execution is objects sending each other message. The important point is that the name and structure of these messages is not necessarily fixed beforehand in the source code and can itself be additional information

- Creating classes that define objects and its behavior.
- Creating objects from class definitions
- Establishing communication among objects

In OOPs **Dynamic Binding** refers to linking a procedure call to the code that will be executed only at run time. The code associated with the procedure in not known until the program is executed, which is also known as late binding.

- in the context of OOP typically refers to the binding of methods to messages
- methods varying dynamically entails much of the power of the OO approach
- main source of power in an OO language
- search for method (code body) to bind to a message starts from the class to which the receiver currently (i.e., at run-time) is an instance of, and and proceeds up the class inheritance hierarchy from there (static binding initiates

the search from the class to which the reference variable pointing to the receiver was declared to be an instance of)

- if no method found anywhere, a run-time error (method not found) is reported and this is typically the only error in a Smalltalk program ever detected and reported

Eg:

```
Mammal m;
Cow c;

if (user input)
  m = new Cow;   // if static binding used, run method in class Mammal
                    bound to run message here
              // if dynamic binding used, run method in class Cow
                    bound to run message here
m.run
else
  c = new Cow;
c.run
```

8

b) Illustrate method overloading with an example

7

Method overloading

Writing two or more methods in the same class with same name but different parameters list, is known as method overloading

Eg:

```
class Complex
{
        intreal,image;
        void assign()
        {
            real=10;
            image=3;
        }
        void assign(int x, int y)
        {
            real = x;
            image = y;
        }
        void show()
```

```
        {
            System.out.println(real+"+"+image+"i");
        }
}
class Test
{
        public static void main(String args[])
        {
            Complex c = new Complex();
            c.assign(6);
            c.show();
        }
}
```

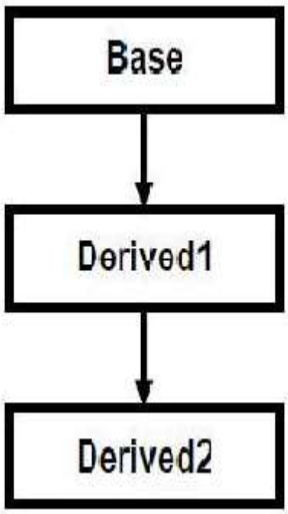
UNIT – II

V.

a) Describe multilevel and hierarchical inheritance with examples

8

In multilevel, one-to-one ladder increases. Multiple classes are involved in inheritance, but one class extends only one. The lowermost subclass can make use of all its super classes' members. Multilevel inheritance is an indirect way of implementing multiple inheritances.

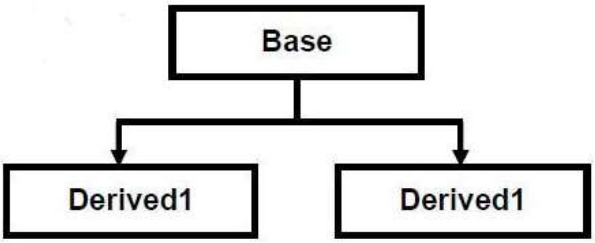


Eg: class A

```
    {
        ……
        ……
```

```
}
class B extends A
{
        ……
        ……
}
class C extends B
{
        ……
        ……
}
```

In hierarchical type of inheritance, one class is extended by many subclasses. It is one-to-many relationship. A realtime example is available at dynamic binding –



Eg:
```
class A
{
        ……
        ……
}
class B extends A
{
        ……
        ……
}
class C extends A
{
        ……
        ……
}
```

b) Distinguish between an interface and a class

Property	**Class**	**Interface**

Instantiation	Can Be Instantiated	Cannot be instantiated
State	Each Object created will have its own state	Each objected created after implementing will have the same state
Behavior	Every Object will have the same behavior unless overridden.	Every Object will have to define its own behavior by implementing the contract defined.
Inheritance	A Class can inherit only one Class and can implement many interfaces	An Interface cannot inherit any classes while it can extend many interfaces
Variables	All the variables are instance by default unless otherwise specified	All the variables are static final by default, and a value needs to be assigned at the time of definition
Methods	All the methods should be having a definition unless decorated with an abstract keyword	All the methods are abstract by default and they will not have a definition.

OR

VI.

a) Compare this and super keywords

7

this(current class)	**super(super class)**
It is a keyword used to store current object reference.	It is a keyword used to store super class object in sub class.

Pre define instance variable used to hold current object reference.	Pre define instance variable used to hold super class object reference through sub class object.
Used to separate state of multiple objects of same class and also used to separate local variables and class level variables in a non-static method if both have same name.	Used to separate super class and subclass members if both have same name.
It must be used explicitly if non-static variables and local variables or parameter name is same.	It must be used explicitly if super class and sub class members have same names.
Can't be referred from static context. It can be printed, means can be called from System.out.println.	Can't be referred from static context.It can't be printed, means cannot be called from System.out.println.
For example System.out.println(this.x);	For example System.out.println(super.x); it leads to compile time error.

b) Develop a java program that support multiple inheritance

8

```
class A
{
                voidshowA()
                {
                    System.out.println("class A");
                }
}
Interface B
{
                voidshowB();
}
class C extends A implements B
{
                voidshowC()
                {
                    System.out.println("class C");
                }
                public void showB()
                {
```

```
                    System.out.println("interface B");
                }
}
class Multi
{
                public static void main(String args[])
                {
                    C c1=new C();
                    c1.showA();
                    c1.showB();
                    c1.showC();
                }
}
```

UNIT – III

VII.

a) Explain adding classes and interfaces to packages 8

It is easy to add a class or an interface to an existing package. Consider the following package

```
package p1;
public class X
{
   ……
   ……
}
```

The package p1` consist one public class(X). Suppose to add another class Y to this package. This can be done as follows

1) Place the package statement
2) Define the class and make it public

Eg:

```
package p1;
public class Y
{
   ……
   ……
}
```

3) Store this as Y.java file under the directory p1.

4) Compile Y.java file. This will create a Y.classfile and place it in p1

b) Explain stopping and blocking of threads

7

Stopping and blocking of threads

When we want to stop a thread from running, then call the stop() method.

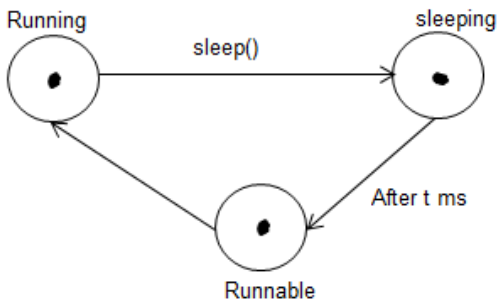
Eg: aThread.stop();

This statement causes the thread to move to the dead state. A thread will also move to the dead state automatically when it reaches the end of its method. The stop() method may be used when the pre-mature death of a thread is desired.

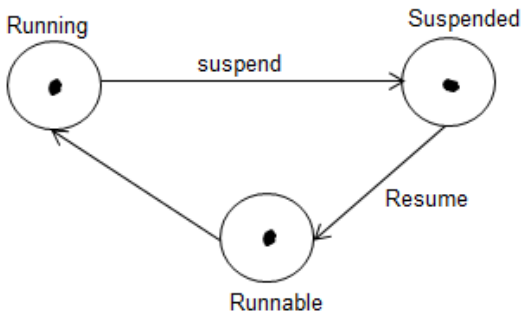
Blocking of a thread

A thread can also be temporarily blocked from entering into the runnable and subsequently running state by using the following methods:

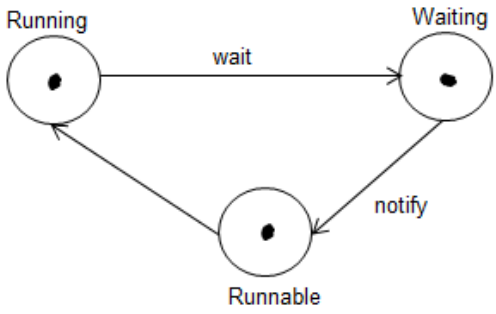
1) sleep()



2) suspend()



3) wait()



These methods cause the thread to go into the blocked state. The thread will return to the runnable state when specified time is elapsed in the case of sleep(), the resume() method is invoked in the case of suspend(), and the notify() method is called in the case of wait().

OR

VIII.

a) List the benefits of organizing classes into packages

7

- The classes contained in the package of other programs can be easily re-used
- Two classes in two different packages can have the same name
- Packages provide a way to hide classes
- Packages provide a way for separating design from coding
- It shows that the classes and interfaces in the package are related.
  Often a group of classes and interfaces are related according to functionality so naturally they should be grouped into a package. An excellent example is thejava.io package which groups a set of classes that all perform input and output functions.
- You know where to find the classes you want if they're in a specific package.
  If you are looking for a specific class and you know what functionality it provides you will naturally be able to find it in the right package. If you are looking for an InputStreamReader you will find it in the input and output package, ie: java.io
- The names of your classes and interfaces won't be in conflict with those of other programmers.

If you decide to implement your own String class and you place it in a package it will be distinguishable from the java.lang.String class that comes with the Java API.

- You can restrict the access to your classes.

  When you only want your code or program to access certain code, for example in a library, then using packages makes access control to source code much easier.

b) Explain the different ways of creation of threads. Give example

8

A thread can be created in 2 ways:

1) Extending the Thread class
2) Implementing the Runnable interface

Extending the Thread class

It includes the following steps:

1. Declare the class as extending the Thread class

```
class A extends Threads
{
        ……
        ……
}
```

2. Implement the run() method

```
public void run()
{
……
……
}
```

3. Create a thread object and call the start() method

```
A a1 = new A();
A1.start();
```

Implementing the Runnable interface

It include the following steps

1. Declare the class as implementing the Runnable interface

```
class A implements Runnable
{
……
……
}
```

2. Implement the run() method

```
public void  run()
{
……
……
}
```

3. Create an object of the class

```
A a1 = new A();
```

4. Create an object of the Thread class which takes the above object as parameter

```
Thread t1 = new Thread(a1);
```

5. Calls the start() method

```
t1.start();
```

UNIT – IV

IX.

a) Write a program to check IP address validation with possible expectations and errors, that is 0 – 255.0 – 255.0 – 255.0 – 255

8

```
import java.io.*;
importjava.util.stringTokenizer;
classMyException extends Exception
{
                    MyException(String message)
                    {
                        super(message);
                    }
}

class Test
{
public static void main(String args[])
  {
try
   {
                    InputStreamReaderip = new
InputStreamReader(System.in);
                    BufferReaderbr = new BufferReader(ip);
                    int count = 1;
                    System.out.println("Enter IP Address");
```

```
                    String ipaddr = br.readLine();
                    StringTokenizerst = new StringTokenizer(ipaddr,".");
                    while(st.hasMoreElement())
                    {
                      String s = String.valueOf(st.nextElement());
                    int n = Integer.parseInt(s);
                    if(n<0||n>255||count>4)
                         throw new MyException("Invalid IP Address");
                    count++;
                    }
                    System.out.println("correct IP Addres");
        }
    catch(MyException e)
        {
                    System.out.println(e.getMessage());
        }
    catch(IOException e)
        {
                    System.out.println(ioe.getMessage());
        }
      }
    }
```

b) Compare the paint stage with the rest of the life cycle stages of an applet.

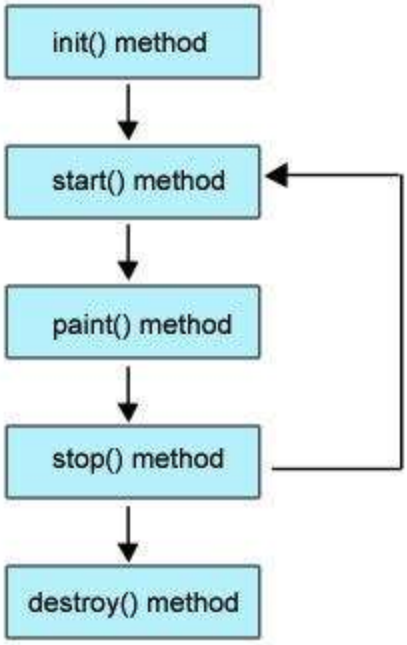


Figure: Life cycle of Applet

- **Born or initialization state (init()):** The applet's voyage starts here. In this method, the applet object is created by the browser. Because this method is called before all the other methods, programmer can utilize this method to instantiate objects, initialize variables, setting background and foreground colors in GUI etc.; the place of a constructor in an application. It is equivalent to born state of a thread.
- **Running state (start()):** In init() method, even though applet object is created, it is ininactive state. An inactive applet is not eligible for microprocessor time even though the microprocessor is idle. To make the applet active, the init() method calls start() method. In start() method, applet becomes active and thereby eligible for processor time.
- **Display state (paint()):** This method takes a java.awt.Graphics object as parameter. This class includes many methods of drawing necessary to draw on the applet window. This is the place where the programmer can write his code of what he expects from applet like animation etc. This is equivalent to **runnable state**of thread.
- **Idle state (stop()):** In this method the applet becomes temporarily inactive. An applet can come any number of times into this method in its life cycle and can go back to the active state (paint() method) whenever would like. It is the best place to have cleanup code. It is equivalent to the **blocked state** of the thread.

- **Dead or destroy state (destroy()):** This method is called just before an applet object is garbage collected. This is the end of the life cycle of applet. It is the best place to have cleanup code. It is equivalent to the **dead state** of the thread.

OR

X.

a) Write an applet that receives three numeric values as input from the user and then display the largest of the three on the screen. Write a HTML page to test the applet.

8

```
importjava.awt.*;
importjava.awt.event.*;
importjava.applet.*;
public class Largest extend Applet implements ActionListener
{
TextField t1,t2,t3,t4;
  Label l1,l2,l3,l4;
  Button b;
public void init()
  {
                    t1 =new TextField(10);
                    t2 =new TextField(10);
                    t3 =new TextField(10);
                    t4 =new TextField(10);

                    l1= new Label("1st no.");
                    l2= new Label("2nd no.");
                    l3= new Label("3rd no.");
                    l4= new Label("4th no.");

                    b= new Button("find Largest");

                    add(l1);
                    add(l2);
                    add(l3);
                    add(l4);
                    add(t1);
                    add(t2);
                    add(t3);
                    add(t4);
```

```
                    add(b);
                    b.addActionListener(this);
 }
public void acctioPerformed(ActionEventae)
 {
                    int a = Integer.parseInt(t1.getText());
                    int b = Integer.parseInt(t2.getText());
                    int c = Integer.parseInt(t3.getText());
                    if(a>b && a>c)
                    t4.setText(a+"");
                    else if(b>c)
                    t4.setText(b+"");
                    else
                      t4.setText(c+"");
 }
}
```

***save this file as Largest.java, then compile it.**

HTML page(web page)

```
<html>
      <applet code=Largest.class
            Width=200
            Height=200>
      </applet>
</html>
```

b) Write a java program that will count the number of characters in a file.


```
import java.io.*;
classCharCounter
{
public static void main(String args[]) throw IOException
 {
                    FileInputStream f= new FileInputStream("te.txt");
                    intb,count=0;
                    while(b=f.read()!=-1)
                    {
                    if(b!='')
                         count++;
                    }
                    System.outprintln("No.ofcharacters="+count);
}}
```