TED (10)-3069 Reg. No. ………………………….

(REVISION-2010) Signature …….……………………

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- OCTOBER, 2013

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 need of constructor

   Constructor is a method you can use to set initial values for field variables. When the object is created, Java calls the constructor first. Any code you have in your constructor will then get executed

2. List the visibility controls used in java

   1) Private
   2) Public
   3) Default
   4) protected

3. Write the purpose of API packages

   An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction.

4. Give the use of Runnable interface

A class that implements Runnable can run without subclassing Thread by instantiating a Thread instance and passing itself in as the target. In most cases, the Runnable interface should be used if you are only planning to override the run() method and no other Thread methods.

5. List the group of data stream and their use based on the data type

   1) Byte stream class that provide support for handling I/O operations on bytes

   2) Character Stream classes that provide support for managing I/O operations on characters.

PART – B

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

1. Describe the data abstraction and data encapsulation

Data Encapsulation

Java is an Object oriented programming language, That means everything in java is an object and these objects interact with each other to form an executing program. Classes are nothing but blueprints of Objects and these classes form the very basis of Data encapsulation.

Data encapsulation in it's simple form means wrapping the relevant data in a class and controlling it's access. This is also sometimes associated with another keyword - Data Hiding. When we design the class we essentially write encapsulation rules. Lets us go a little deeper to understand this concept -

We achieve data encapsulation in java by using access modifiers - Public, Protected, default, Private.

Data abstraction

Data abstraction simply means generalizing something to hide the complex logic that goes underneath. Consider a very simple example of computer science - subtraction. If we have two variables "a" and "b" we simply say their subtraction is (a-b). But what is really happening behind the scenes. These variables are stored in main memory in binary format. Processor processes the subtract information which really takes place using two's complement method. Again in processor we have gates that actually carry out the procession. Even inside gates there are pins and 1 and 0 simple means high and low signals.

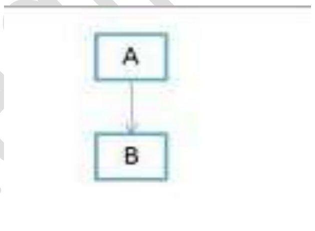
Even though the process is so complex underneath a normal software developer developing some financial application need not know every small detail. This is because the concept has been abstracted out. This is what abstraction means in a general terminology.

We achieve data abstraction in Java primarily by using Interfaces and abstract classes.

2. Explain types of inheritance with the help of diagrams

Single Inheritance

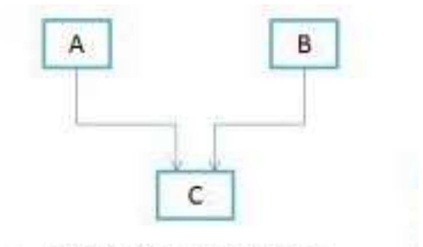
Single inheritance is damn easy to understand. When a class extends another one class only then we call it a single inheritance. The below flow diagram shows that class B extends only one class which is A. Here A is a **parent class** of B and B would be a **child class** of A.



(a) Single Inheritance

Multiple Inheritances

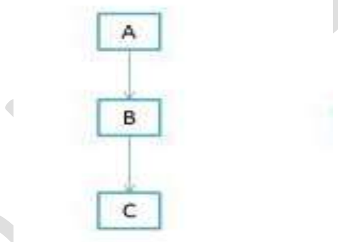
“Multiple Inheritance**”** refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with “multiple inheritance” is that the derived class will have to manage the dependency on two base classes.



(b) Multiple Inheritance

Multilevel Inheritance

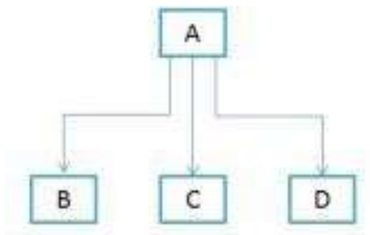
Multilevel inheritance refers to a mechanism in OO technology where one can inherit from a derived class, thereby making this derived class the base class for the new class. As you can see in below flow diagram C is subclass or child class of B and B is a child class of A. For more details and example refer – Multilevel inheritance in Java.



(d) Multilevel Inheritance

Hierarchical Inheritance

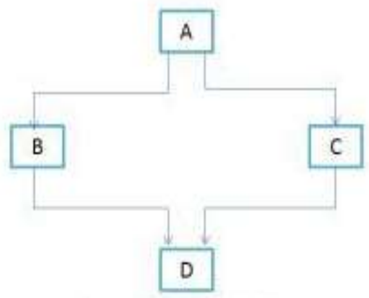
In such kind of inheritance one class is inherited by many **sub classes**. In below example class B,C and D **inherits** the same class A. A is **parent class (or base class)** of B,C & D. Read More at – Hierarchical Inheritance in java with example program.



(c) Hierarchical Inheritance

Hybrid Inheritance

In simple terms you can say that Hybrid inheritance is a combination of **Single** and **Multiple** inheritance. A typical flow diagram would look like below. A hybrid inheritance can be achieved in the java in a same way as multiple inheritances can be!! Using interfaces. yes you heard it right. By using **interfaces** you can have multiple as well as **hybrid inheritance** in Java.



(e) Hybrid Inheritance

3. Compare abstract class with interface

Property	**Abstract 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	An Interface cannot inherit any classes while it can extend many

	interfaces	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.

4. Compare the use of API packages and user define packages

   Packages are categorized as :

   1) Built-in packages (standard packages which come as a part of Java Runtime Environment)
   2) User-defined packages (packages defined by programmers to bundle group of related classes)

   Built-in Packages
   These packages consist of a large number of classes which are a part of Java API. For e.g, we have usedjava.io package previously which contain classes to support input / output operations in Java. Similarly, there are other packages which provide different functionality.

   Some of the commonly used built-in packages are shown in the table below :

Package Name	**Description**
java.lang	Contains language support classes (for e.g classes which defines primitive data types, math operations, etc.) . This package is automatically imported.
java.io	Contains classes for supporting input / output operations.
java.util	Contains utility classes which implement data structures like

	Linked List, Hash Table, Dictionary, etc and support for Date / Time operations.
java.applet	Contains classes for creating Applets.
java.awt	Contains classes for implementing the components of graphical user interface (like buttons, menus, etc.).
java.net	Contains classes for supporting networking operations.

- First statement imports Vector class from util package which is contained inside java package.
- Second statement imports all the classes from util package.

User-defined Packages

Now, we will see how to create packages and use them. Suppose, we want to create a package named relational which contains a class named Compare. First, we will create a directory named relational (the name should be same as the name of the package). Then create the Compare class with the first statement being package relational;. Now, the package is ready to be imported.

5. Writhe the naming conventions used for packages, give 2 examples

Package names should be unique and consist of lowercase letters. With small projects that only have a few packages it's okay to just give them simple (but meaningful!) names.

```
package pokeranalyzer
package mycalculator
```

Underscores may be used if necessary:

```
package com.oreilly.fish_finder;
```

Publicly available packages should be the reversed Internet domain name of the organization, beginning with a single-word top-level domain name (e.g., *com, net, org*, or *edu*), followed by the name of the organization and the project or division. (Internal packages are typically named according to the project.)

Package names that begin with java and javax are restricted and can be used only to provide conforming implementations to the Java class libraries.

In software companies and large projects where the packages might be imported into other classes, the names will normally be subdivided. Typically this will start with the company domain before being split into layers or features:

```
package com.mycompany.utilities
package org.bobscompany.application.userinterface
```

6. Describe the concept of Streams

A stream can be defined as a sequence of data. The InputStream is used to read data from a source and the OutputStream is used for writing data to a destination. InputStream and OutputStream are the basic stream classes in Java.

All the other streams just add capabilities to the basics, like the ability to read a whole chunk of data at once for performance reasons (BufferedInputStream) or convert from one kind of character set to Java's native unicode (Reader), or say where the data is coming from (FileInputStream, SocketInputStream and ByteArrayInputStream, etc.)

Input streams

The class java.io.InputStream is the base class for all Java IO input streams. If you are writing a component that needs to read input from a stream, try to make our component depend on an InputStream, rather than any of its subclasses (e.g. FileInputStream). Doing so makes your code able to work with all types of input streams, instead of only the concrete subclass

OutputStream

The class java.io.OutputStream is the base class of all Java IO output streams. If you are writing a component that needs to write output to a stream, try to make sure that component depends on an OutputStream and not one of its subclasses.

7. Differentiate applets with applications

Feature	Application	Applet
main() method	Present	Not present
Execution	Requires JRE	Requires a browser like Chrome
Nature	Called as stand-alone application as application can be executed from command prompt	Requires some third party tool help like a browser to execute
Restrictions	Can access any data or software available on the system	cannot access anything on the system except browser's services
Security	Does not require any security	Requires highest security for the system as they are un-trusted

(5 x 6 = 30)

PART – C

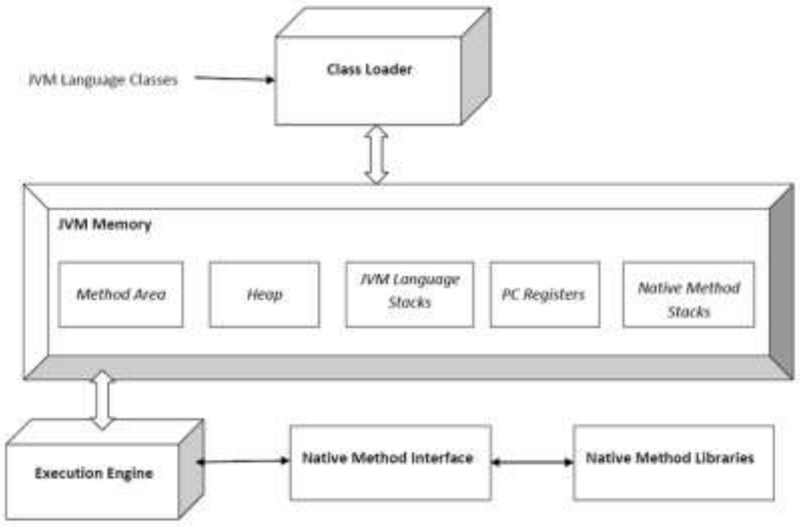
(Answer *one* full question from each unit. Each question carries 15 marks)

UNIT – I

III.

a) Explain JVM

6



A Java virtual machine (JVM) is an abstract computing machine. There are three notions of the JVM: specification, implementation, and instance. The specification is a book that formally describes what is required of a JVM implementation. Having a single specification ensures all implementations are interoperable. A JVM implementation is a computer program that implements requirements of the JVM specification in a compliant and preferably per formant manner. An instance of the JVM is a process that executes a computer program compiled into Java byte code. JVM -- a machine within a machine -- mimics a real Java processor, enabling Java byte code to be executed as actions or operating system calls on any processor regardless of the operating system. For example, establishing a socket connection from a workstation to a remote machine involves an operating system call. Since different operating systems handle sockets in different ways, the JVM translates the programming code so that the two machines that may be on different platforms are able to connect.

b) Write a java program to overload the method area to calculate area of a rectangle, square and circle

```
class MethodOverloading
{
  void area()
  {
                        float a=3.14*r*r;
                        System.out.println("Area of circle="+a);
```

```
    }
    void area(int l,int b)
    {
                        int a= l*b;
                        System.out.println("Area of rectangle="+a);
    }
  }
  class Test
  {
    public static void main(string args[])
    {
                        MethodOverloading o = new MethodOverloading();
                        o.area(5);
                        o.area(10,8);
    }
  }
```

}OR

IV.

a) Create a class "point" which is used to represent x and y co-ordinates of a point. Use constructor to initialize the point class with some values of x and y. write methods:

i. To assign values to co – ordinate(x, y)

ii. To calculate and print the distance of the point from origin [distance =$\sqrt{x^2 + y^2}$]

iii. To display x and y co-ordinates of a point object passed to it.

10

```
class Point
{
  int x,y;
  Point()
  {
    x=0;
    y=0;
  }
  void assign(int a, int b)
  {
    x=a;
    y=b;
  }
  void calculate()
  {
    float distance = java.lang.Math.sqrt((x*x)+(y*y));
    System.out.println("Distance="+distance);
  }
  void display()
  {
    System.out.println("x="+x);
    System.out.println("y="+y);
  }
}
```

b) Explain constructor overloading with example

5

Constructor overloading in java allows having *more than one constructor inside one Class*. in last article we have discussed about method overloading and overriding and constructor overloading is not much different than method overloading. Just like in case of method overloading you have multiple methods with same name but different signature; in Constructor overloading you have *multiple constructors with different signature* with only difference that Constructor doesn't have return type in Java. Those constructors will be called as overloaded constructor. Overloading is also another form of polymorphism in Java which allows having multiple constructors with different name in one Class in java.

```
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();
        }
}
```

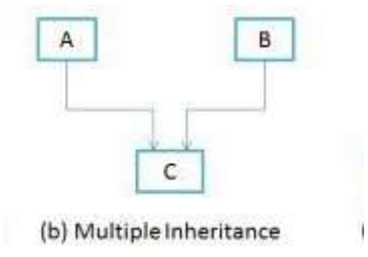
UNIT – II

V.

a) Explain the procedure for implementing multiple inheritance in java

6

"**Multiple Inheritance**" refers to the concept of one class extending (Or inherits) more than one base class. The inheritance we learnt earlier had the concept of one base class or parent. The problem with "multiple inheritance" is that the derived class will have to manage the dependency on two base classes.



(b) Multiple Inheritance

The Java programming language supports *multiple inheritance of type,* which is the ability of a class to implement more than one interface. An object can have multiple types: the type of its own class and the types of all the interfaces that the class implements. This means that if a variable is declared to be the type of an interface, then its value can reference any object that is instantiated from any class that implements the interface. As with multiple inheritance of implementation, a class can inherit different implementations of a method defined (as default or static) in the interfaces that it extends. In this case, the compiler or the user must decide which one to use.

Eg:
```
class A
{
                void showA()
                {
                    System.out.println("class A");
                }
}
Interface B
{
                void showB();
}
class C extends A implements B
{
                void showC()
                {
                    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();
                }
}
```

b) Create a class “person” with field name address and methods getdata () and putdata (). Derive 2 different classes “student” [with fields roll no. and mark] and “employee” [

with fields salary and emp. no] from “person”, overriding the method getdata(), in the sub classes “student” and “employee” to assign values to fields of “person” ,”student” and “”employee”. Override the method putdata() in the sub classes “student” and “employee” to display fields of “person”, “student” and “employee”.

9

```
class Person
{
   String name,address;
   void getData(String n, string a)
   {
     name=n;
     address=a;
   }
   void putData()
   {
     System.out.println("Name="+name);
     System.out.println("Address="+address);
   }
}

class Student extends Person
{
   int rollno,mark;
   void getData(String n, string a,int r,int m)
   {
     super.getData(n,a);
     rollno=r;
     mark=m;
   }
   void putData()
   {
     super.putData();
     System.out.println("Rollno="+rollno);
```

```
    System.out.println("Mark="+mark);
  }
}

class Employee extends Person
{
  int slary,empno;
  void getData(String n, string a,int s,int e)
  {
    super.getData(n,a);
    salary=s;
    empno=e;
  }
  void putData()
  {
    super.putData();
    System.out.println("Emp no="+empno);
    System.out.println("Salary="+salary);
  }
}
```

OR

VI.

a) Write a program which shows extension of interfaces

8

An interface can extend another interface, similarly to the way that a class can extend another class. The extends keyword is used to extend an interface, and the child interface inherits the methods of the parent interface. extends means that you take either an implementation (class) or specification (interface) and add to it with different or new functionality (or change the specification of its behaviour), thus modifying its behaviour and extend-ing it.

extend will be implantation only one class

Eg: public class demo extends demo1 *//is it true*

```
public class demo extends demo1,demo2       //is it wrong
```

Implement can be implantation one or more class

One interface can be implement one or more class

```
Eg : public interface demo
public interface demo1
public class implement demo,demo1
```

Syntax:

```
interface  ChildInterface extends ParentInterface
{
        Body;
}
Eg:
interface  A
{
        int code=101;
}
interface  B extends A
{
        void show();
}
interface C extends A,B
{
        .................
}
```

b) Compare implementing class from interface and extending class from a super class with examples

7

Implementing class from interface

1) ‘implements’ keyword is used to implement an interface
2) A class can implement more than one interface at a time
3) An interface contains abstract methods. So it is the responsibility of the child class to write body of the abstract methods

4) When a class does not fully implement an interface, then the class must be declares as abstract

Eg:

```
interface A
{
   int code=300;
   void show();
}
interface B implements A
{
  public  void show()
  {
    System.out.println("code="+code);
  }
}
```

Extending class from a super class

1) 'extends' keyword is used to extend a class
2) A class can extend only one class
3) A super class may or may not have abstract methods. When a super class contains abstract methods, then the child class must write body for that method

Eg:

```
class A
{
   int code;
   void show()
  {
    code=100;
    System.out.println("code="+code);
  }
}
class B extends A
{
  String name;
  void print()
  {
    name="java";
    System.out.println("code="+code);
  }
}
```

UNIT – III

VII.

a) Explain the use of java API packages. Write any 4 commonly used API package and brief their contents.

8

Java application programming interface (API) is a list of all classes that are part of the Java development kit (JDK). It includes all Java packages, classes, and interfaces, along with their methods, fields, and constructors. These prewritten classes provide a tremendous amount of functionality to a programmer. A programmer should be aware of these classes and should know how to use them. API). An application programming interface (API), in the context of Java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors. Similar to a user interface, which facilitates interaction between humans and computers, an API serves as a software program interface facilitating interaction. In Java, most basic programming tasks are performed by the API's classes and packages, which are helpful in minimizing the number of lines written within pieces of code.

a) Language support packages(java.lang)

   java.lang Provides classes that are fundamental to the design of the Java programming language such as String, Math, and basic runtime support for threads and processes.

   Example classes: String, Thread. It contain language support classes

b) Networking Packages(java.net)

   The java.net package provides a powerful and flexible infrastructure for networking. Many of the classes in this package are part of the networking infrastructure and are not used by normal applications; these complicated classes can make the package a difficult one to understand.

   Example classes: InetAddresss, Socket

c) i/o Package(java.io)

Java.io package provides classes for system input and output through data streams, serialization and the file system. The Java I/O package, a.k.a. java.io, provides a set of input streams and a set of output streams used to read and write data to files or other input and output sources. There are three categories of classes in java.io: input streams, output streams and everything else.

Example classes: InputStreamReader, BufferedReader

d) Utilities Package (java.util)

Java.util package contains the collections framework, legacy collection classes, event model, date and time facilities, internationalization, and miscellaneous utility classes.

b) Demonstrate the need of synchronization and give the syntax

7

The Java synchronized keyword is an essential tool in concurrent programming in Java. Its overall purpose is to only allow one thread at a time into a particular section of code thus allowing us to protect, for example, variables or data from being corrupted by simultaneous modifications from different threads. This article looks at how to use synchronized in Java to produce correctly functioning multithreaded programs. Other articles in this section look at other Java 5 concurrency facilities which have in fact superseded synchronized for certain tasks.

At its simplest level, a block of code that is marked as `synchronized` in Java tells the JVM: *"only let one thread in here at a time"*.

Imagine, for example, that we have a counter that needs to be incremented at random points in time by different threads. Ordinarily, there would be a risk that two threads could simultaneously try and update the counter at the same time, and in so doing currpt the value of the counter (or at least, miss an increment, because one thread reads the present value unaware that another thread is just about to write a new, incremented value). But by wrapping the update code in a `synchronized` block, we avoid this risk:

Eg:
```
public class Counter
{
```

```
private int count = 0;
public void increment()
{
  synchronized (this)
 {
    count++;
  }
}
public int getCount()
{
  synchronized (this)
  {
    return count;
   }
}
 }
```

OR

VIII.

a) Explain thread exceptions. Write any 4 types of catch statements of exceptions. 7

Thread is the independent path of execution run inside the program. Many Threads run concurrently in the program. Multithread are those group of more than one thread that runs concurrently in a program. Thread in a program is imported from java.lang.thread class. In Multithread, the thread runs concurrently, synchronous or asynchronous

Understand Exception in Threads.

1. A class name RunnableThread implements the Runnable interface gives you the run( ) method executed by the thread. Object of this class is runnable
2. The Thread constructor is used to create an object of RunnableThread class by passing runnable object as parameter.The Thread object has a Runnable object that implements the run( ) method.
3. The start( ) method is invoked on the Thread object . The start( ) method returns immediately once a thread has been spawned.
4. The thread ends when the run( ) method ends which is to be normal termination or caught exception.

5. runner = new Thread(this,threadName) is used to create a new thread
6. runner. start( ) is used to start the new thread.
7. public void run( ) is overrideable method used to display the information of particular thread
8. Thread.currentThread().sleep(2000) is used to deactivate the thread untill the next thread started execution or used to delay the current thread.

```
try
{
   .................
   .................
}
catch (ThreadDeath e)
{
   ................. // kill thread
}
catch (InterruptedException e)
{
   ................. // cannot handle it in the current state
}
catch (IllegalArgumentException e)
{
   ................. // illegal method argument
}
catch (Exceptio e)
{
   ................. // any other
 }
```

b) With the help of diagram, explain the life of cycle of a thread and the state of a thread.

8

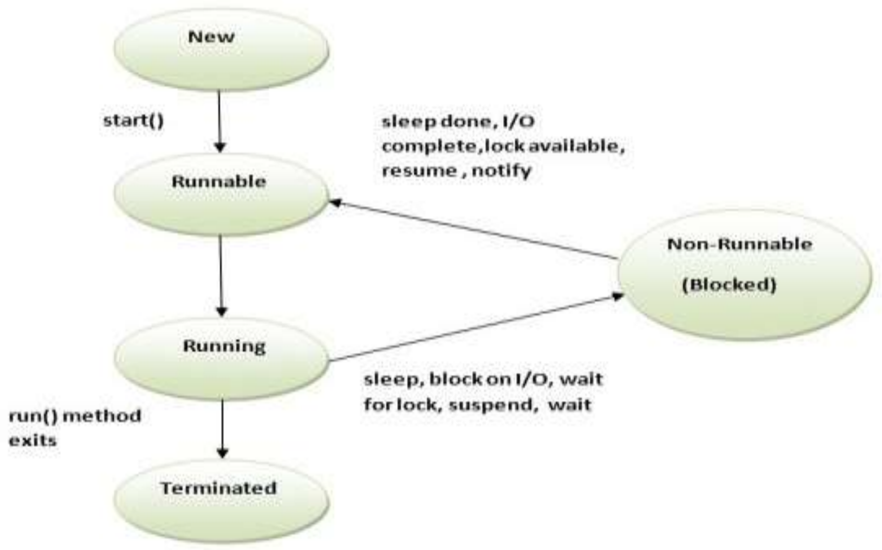
A thread can be in one of the five states. According to sun, there is only 4 states in **thread life cycle in java** new, runnable, non-runnable and terminated. There is no running state.

But for better understanding the threads, we are explaining it in the 5 states.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:

1. New
2. Runnable
3. Running
4. Non-Runnable (Blocked)

5. Terminated


New
start()
Runnable
sleep done, I/O complete, lock available, resume , notify
Non-Runnable (Blocked)
Running
sleep, block on I/O, wait for lock, suspend, wait
run() method exits
Terminated

New

When we create a new Thread object using *new* operator, thread state is New Thread. At this point, thread is not alive and it’s a state internal to Java programming.

Runnable

When we call start() function on Thread object, it’s state is changed to Runnable and the control is given to Thread scheduler to finish it’s execution. Whether to run this thread instantly or keep it in runnable thread pool before running it depends on the OS implementation of thread scheduler.

Running

When thread is executing, it’s state is changed to Running. Thread scheduler picks one of the thread from the runnable thread pool and change it’s state to Running and CPU starts executing this thread. A thread can change state to Runnable, Dead or Blocked from running state depends on time slicing, thread completion of run() method or waiting for some resources.

Blocked/Waiting

A thread can be waiting for other thread to finish using thread join or it can be waiting for some resources to available, for example producer consumer problem or waiter notifier implementation or IO resources, then it's state is changed to Waiting. Once the thread wait state is over, it's state is changed to Runnable and it's moved back to runnable thread pool.

Dead

Once the thread finished executing, it's state is changed to Dead and it's considered to be not alive.

UNIT – IV

IX.

a) Describe the steps in exception handling. List any 2 exceptions and their cause 7

The **exception handling** in java is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained. In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime. Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. The core advantage of exception handling is to maintain the normal flow of the application. Exception normally disrupts the normal flow of the application that is why we use exception handling

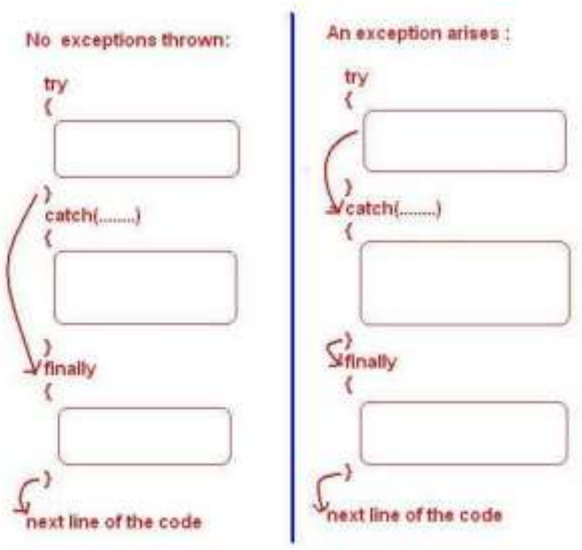
Advantages of Exception Handling

- Exception handling allows us to control the normal flow of the program by using exception handling in program.
- It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
- It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try-catch blocks.

Tasks incorporated with exception handling

1) Find the problem (Hit the exception)
2) Inform that an error has occurred(Throw the exception)

3) Receive the error information(Catch the exception)
4) Take corrective actions(Handle the exception)



Eg:

1) *ArithmeticException*: caused by math errors such as division by zero
2) *IOException*: caused by general I/O failures, such as inability to read from a file

b) Explain the different states of an applet life cycle with the help of a diagram

8

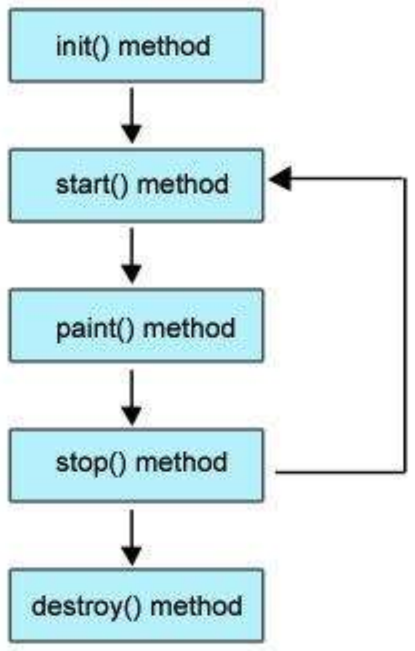
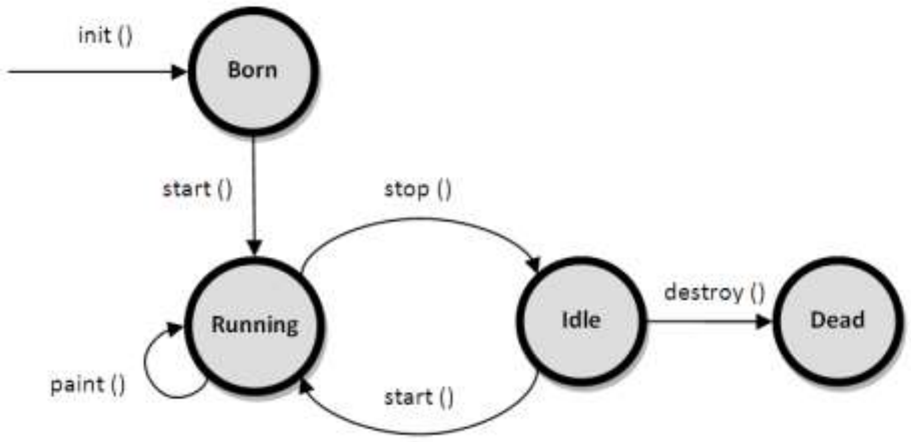


Figure: Life cycle of Applet

- **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) Summarize the steps to be done for adding an applet to a HTML document

7

Applets have the file extension "class". An example would be "Myapplet.class". Some applets consist of more than just one class file, and often other files need to be present for the applet to run (such as JPG or GIF images used by the applet). Make sure to check the documentation for the applet to see if you have all files for it to run. Before embedding an applet on your page you need to upload the required files to your server. Below is a short example showing how simple it is to embed an applet on a page.

1) Design a webpage

Eg:
```
<html>
      ……………….
      ……………….
</html>
```

2) Include the applet tag

Eg:
```
<applet code = MyApplet.class
      Width=200
```

Height=200>

</applet>

The above tag tells the browser or applet viewer to load the applet whose Applet subclass, named *MyApplet*, is in a class file in the same directory as the HTML document that contains the tag. The above tag also specifies the width and height in pixels of the applet. When a browser encounters the tag, it reserves a display area of the specified width and height for the applet, loads the bytecodes for the specified Applet subclass, creates an instance of the subclass, and then calls the instance's init() and start() methods.

b) Write a program to read data from a file and display it on the screen

8

```
import java.io.*;
class ReaderBytes
{
    public static void main(string args[]) throw IOException
     {
          int b;
          FileInputStream f = new FileInputStream(“wb.txt”);
          While((b=f.read())!=-1)
                    System.out.print((char)b);
          f.close();
     }
}
```