TED (10)-3069 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA MODEL EXAMINATION IN ENGINEERING/ TECHNOLIGY

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. List the integer data type in java and their storage size.

   - Byte - 8 bits
   - Short - 16 bits
   - Int-32 bits
   - Long-64bits

2. State the need of super keyword with example

   - super is used to refer immediate parent class instance variable.
   - super() is used to invoke immediate parent class constructor.
   - super is used to invoke immediate parent class method.

3. State the use of interfaces

   Interfaces are used to encode similarities which the classes of various types share, but do not necessarily constitute a class relationship

4. Write the name of any four system packages and their classes.

- java.lang

  Classes for primitive types, strings, math functions, threads, and exception

- java.util

Classes such as vectors, hash tables, date etc.

- java.io

  Stream classes for I/O

- java.awt

  Classes for implementing GUI – windows, buttons, menus etc

5. Write the function of exceptions

Exceptions provide the means to separate the details of what to do when something out of the ordinary happens from the main logic of a program. When an error occurs within a method, the method creates an object and hands it off to the runtime system. The object, called an *exception object*, contains information about the error, including its type and the state of the program when the error occurred

PART – B

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

1. Compare POP and OOP

	Procedure Oriented Programming	**Object Oriented Programming**
Divided Into	In POP, program is divided into small parts calledfunctions.	In OOP, program is divided into parts called objects.
Importance	In POP,Importance is not given to data but to functions as well as sequence of actions to be done.	In OOP, Importance is given to the data rather than procedures or functions because it works as a real world.
Approach	POP follows Top Down approach.	OOP follows Bottom Up approach.
Access Specifiers	POP does not have any access specifier.	OOP has access specifiers named Public, Private, Protected, etc.
Data Moving	In POP, Data can move freely from function to function in the system.	In OOP, objects can move and communicate with each other through member functions.
Expansion	To add new data and function in	OOP provides an easy way to add

	POP is not so easy.	new data and function.
Data Access	In POP, Most function uses Global data for sharing that can be accessed freely from function to function in the system.	In OOP, data can not move easily from function to function,it can be kept public or private so we can control the access of data.
Data Hiding	POP does not have any proper way for hiding data so it is less secure.	OOP provides Data Hiding so provides more security.
Overloading	In POP, Overloading is not possible.	In OOP, overloading is possible in the form of Function Overloading and Operator Overloading.
Examples	Example of POP are : C, VB, FORTRAN, Pascal.	Example of OOP are : C++, JAVA, VB.NET, C#.NET.

2. Differentiate between method and constructor with example

Constructor:	**Method**
Constructor is a special method ofa class but can't be invokeddirectly by method call. Java doc. Says (A class contains constructors that are invoked to create objects from the class blueprint.)	Methods are member of a class.
It is not a member of a class as it can neither be inherited nor invoked using dot (.) operator.	Dot (.) operator is used to invoke Non static methods via object and static methods via class name.
It has no explicit return type.	It has explicit return type, if there is nothing to return, the return type must be void
It has the same name as its class name.	Can have same name as its class name, but theexistence of return type makes it a method (unfortunately looks like constructor).

It is used to initialize the objects, members of object and then execute statements if any.	Used to execute statements.
If there is no this() as the first statement, super() will be there as first statement in the constructor.	'this' is implicitly invoked on all the member in non-static methods, but need explicit invocation in case of name confliction. Syntax: this.x, this.go()
Can be specified as public, none (default), protected or private.	Access-specifier public, none (default), protected orprivate are applicable.
Can be invoked by either a. new ClassName() b. this(args if any) c. super(args if any) d. getInstance()	Can be invoked by Class name in case of static method or by object/this in case of non-static method.

3. Distinguish between final class and abstract class

Property	**Abstract class**	**Final class**
Subclassing	Should be subclassed to override the functionality of abstract methods	Can never be subclassed as final does not permit
Method alterations	Abstract class methods functionality can be altered in subclass	Final class methods should be used as it is by other classes
Instantiation	Cannot be instantiated	Can be instantiated
Overriding concept	For later use, all the abstract methods should be overridden	Overriding concept does not arise as final class cannot be inherited
Inheritance	Can be inherited	Cannot be inherited
Abstract methods	Can contain abstract methods	Cannot contain abstract methods

Partial implementation	A few methods can be implemented and a few cannot	All methods should have implementation
Immutable objects	Cannot create immutable objects (infact, no objects can be created)	Immutable objects can be created (eg. String class)
Nature	It is an incomplete class (for this reason only, designers do not allow to create objects)	It is a complete class (in the sense, all methods have complete functionality or meaning)

4. Write the steps involved in creating a package

   Following steps are to be followed in creating and using packages.

   1. Create a package with a .class file
   2. set the classpath from the directory from which you would like to access. It may be in a different drive and directory. Let us call it as a target directory.
   3. Write a program and use the file from the package.

   Let us create a package called **forest** and place a class called **Tiger** in it. Access the package from a different drive and directory.

   1st Step: Create a package (forest) and place Tiger.class in it.

```
package forest; import java.util.*;
public class Tiger
{
        public void getDetails(String nickName, int weight)
        {
                System.out.println("Tiger nick name is " + nickName);
                System.out.println("Tiger weight is " + weight);
        }
}
```

   Order of Package Statement

The above program codingwise is very simple but is important to know the steps of package creation.

```
package forest;
import java.util.*;
public class Tiger
```

Package is a keyword of Java followed by the package name. Just writing the package statement followed by the name creates a new package; see how simple Java is to practice. For this reason, Java is known as a production language.

While creating packages, the order of statements is very important. The order must be like this, else, compilation error.

1. Package statement
2. Import statement
3. Class declaration

If exists, the package statement must be first one in the program. If exists, the import statement must be the second one. Our class declaration is the third. Any order changes, it is a compilation error.

When the code is ready, the next job is compilation. We must compile with package notation.

5. Explain synchronization related to multithreading

When we start two or more threads within a program, there may be a situation when multiple threads try to access the same resource and finally they can produce unforeseen result due to concurrency issue. For example if multiple threads try to write within a same file then they may corrupt the data because one of the threads can override data or while one thread is opening the same file at the same time another thread might be closing the same file.

So there is a need to synchronize the action of multiple threads and make sure that only one thread can access the resource at a given point in time. This is implemented using a concept called **monitors**. Each object in Java is associated with a monitor, which a thread can lock or unlock. Only one thread at a time may hold a lock on a monitor.

Java programming language provides a very handy way of creating threads and synchronizing their task by using **synchronized** blocks. You keep shared resources within this block.

Syntax:

```
synchronized (objectidentifier)
{
   // Access shared variables and other shared resources
}
```

Here, the **objectidentifier** is a reference to an object whose lock associates with the monitor that the synchronized statement represents. Now we are going to see two examples where we will print a counter using two different threads. When threads are not synchronized, they print counter value which is not in sequence, but when we print counter by putting inside synchronized() block, then it prints counter very much in sequence for both the threads.

6. Compare applet and stand-alone applications

Feature	Stand-alone 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

7. Describe passing parameters to applets with example

Parameters are passed to applets in NAME=VALUE pairs in <PARAM> tags between the opening and closing APPLET tags. Inside the applet, you read the values passed through the PARAM tags with the getParameter() method of the java.applet.Applet class.

The program below demonstrates this with a generic string drawing applet. The applet parameter "Message" is the string to be drawn.

```
import java.applet.*;
import java.awt.*;
public class DrawStringApplet extends Applet
{
    private String defaultMessage = "Hello!";
    public void paint(Graphics g)
    {
        String inputFromPage = this.getParameter("Message");
        if (inputFromPage == null) inputFromPage = defaultMessage;
        g.drawString(inputFromPage, 50, 25);
    }
}
```

You also need an HTML file that references your applet. The following simple HTML file will do:

```
<HTML>
<HEAD>
<TITLE> Draw String </TITLE>
</HEAD>
<BODY>
This is the applet:<P>
<APPLET code="DrawStringApplet" width="300" height="50">
<PARAM name="Message" value="Howdy, there!">
This page will be very boring if your
browser doesn't understand Java.
</APPLET>
```

</BODY>
</HTML>

(5 x 6 = 30)

PART – C

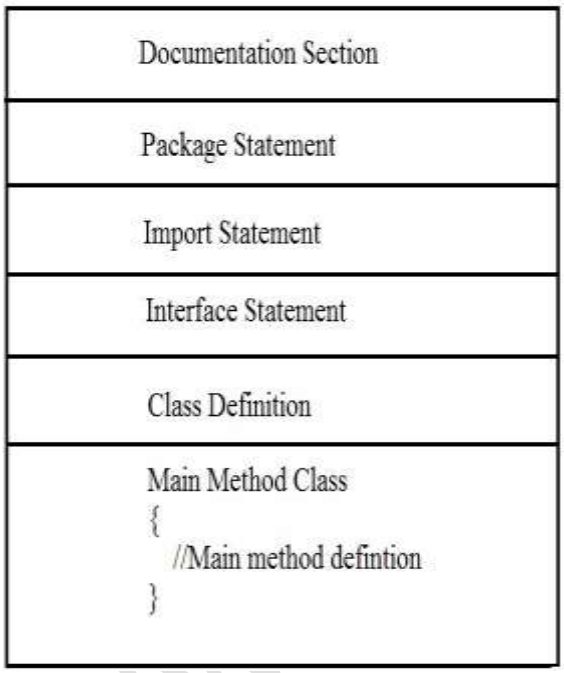
(Answer *one* full question from each unit. Each question carries 15 marks)

UNIT – I

III.

a) Derive the general structure of java program

8



Suggested

Optional

Optional

Optional

Optional

essential

Documentation Section

It includes the comments to tell the program's purpose. It improves the readability of the program. A comment is a non-executable statement that helps to read and understand a program especially when your programs get more complex. A good program should include comments that describe the purpose of the program, author name, date and time of program creation. This section is optional and comments may appear anywhere in the program

Package Statement

It includes statement that provides a package declaration. A package statement includes a statement that provides a package declaration. It must appear as the first statement in the source code file before any class or interface declaration. This statement is optional. For example: Suppose you write the following package declaration as the first statement in the source code file.

package employee;

This statement declares that all classes and interfaces defined in this source file are part of the employee package. Only one package declaration can appear in the source file.

Import statements

It includes statements used for referring classes and interfaces that are declared in other packages. An import statement is used for referring classes that are declared in other packages. The import statement is written after a package statement but before any class definition. You can import a specific class or all the classes of the package.

Interface Section

It is similar to a class but only includes constants, method declaration. Interfaces cannot be instantiated. They can only be implemented by classes or extended by other interfaces. It is an optional section and is used when we wish to implement multiple inheritance feature in the program.

Class Section

It describes information about user defines classes present in the program. Every Java program consists of at least one class definition. This class definition declares the main method. It is from where the execution of program actually starts. A class is a collection of fields (data variables) and methods that operate on the fields.

Main method class

Every program in Java consists of at least one class, the one that contains the main method. The main() method which is from where the execution of program actually starts and follow the statements in the order specified. The main method can create objects, evaluate expressions, and invoke other methods and much more. On reaching the end of main, the program terminates and control passes back to the operating system.

b) Describe method overloading with example

Method overloading

Writing two or more methods in the same class with same name but different parameters list, is known as method overloading. If we have to perform only one operation, having same name of the methods increases the readability of the program. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs. So, we perform method overloading to figure out the program quickly.

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();
        }
}
```

OR

IV.

a) Differentiate between class members and instance members with example

8

Instance members

- Instance variables are declared in a class, but outside a method, constructor or any block.
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.
- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables.
- The instance variables are visible for all methods, constructors and block in the class.
- Instance variables have default values. For numbers the default value is 0, for Booleans it is false and for object references it is null.
- Instance variables can be accessed directly by calling the variable name inside the class.

```
class Taxes
{
int count;
/*...*/
}
```

Class members

- Class variables also known as static variables are declared with the *static* keyword in a class, but outside a method, constructor or a block.
- There would only be one copy of each class variable per class, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants.

- Static variables are stored in static memory.
- Static variables are created when the program starts and destroyed when the program stops.
- Visibility is similar to instance variables
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null.
- Static variables can be accessed by calling with the class name . *ClassName.VariableName.*
- When declaring class variables as public static final, then variables names (constants) are all in upper case.

```
class Taxes
{
  static int count;
  /*...*/
}
```

Instance variable are per instance (object) basis. If you have 5 instance of one class you will have five copies of instance variable. these are also referred as *non static variable* and initialized when you create instance of any object using new() operator or by using other methods like reflection e.g. Class.newInstance(). On the other hand Class variables are declared using static keyword and they have exact same value for every instance. static or class variable are initialized when class is first loaded into JVM memory unlike instance variable which initialized when instance is created. *Static variables are similar to global variable* in C and can be used to store data which is static in nature and has same value for all instance, but at same static variable also cause subtle concurrency bugs if updated by multiple threads.

b) Explain the basic concepts of OOP

7

Object- Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

- Object

- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation
- Dynamic Binding

Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

Class

Collection of objects is called class. It is a logical entity.

Inheritance

When one object acquires all the properties and behaviors of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc. In java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something e.g. cat speaks meaw, dog barks woof etc.

Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing. In java, we use abstract class and interface to achieve abstraction.

Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

Dynamic Binding

When compiler is not able to resolve the call/binding at compile time, such binding is known as Dynamic or late Binding. Overriding is a perfect example of dynamic binding as in overriding both parent and child classes have same method. Thus while calling the overridden method, the compiler gets confused between parent and child class method (since both the methods have same name)

UNIT – II

V.

a) Explain the method of passing values to parent class constructor in java with examples

```
import com.bruceeckel.simpletest.*;
class Game {
  Game(int i) {
    System.out.println("Game constructor");
  }
}
class BoardGame extends Game {
  BoardGame(int i) {
    super(i);
    System.out.println("BoardGame constructor");
  }
}
public class Chess extends BoardGame {
  private static Test monitor = new Test();
  Chess() {
    super(11);
    System.out.println("Chess constructor");
  }
  public static void main(String[] args) {
    Chess x = new Chess();
    monitor.expect(new String[] {
      "Game constructor",
      "BoardGame constructor",
```

```
            "Chess constructor"
        });
    }
}
```

To create a sub class (child) from a Java super class (parent), the keyword extends is used. You then follow the "extends" keyword with the parent class you want to extend. We want to create a sub class from the Game class. The Game class will be the super class and the BoardGame class will be the sub class.

Just like the Game class we can create a constructor for this new BoardGame class. When we create an object from the class, Java will first of all call our constructor.

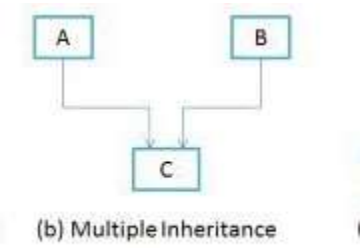
However, only one constructor can be called. If we call a new constructor from the BoardGame class all those default values we set up for the Game class fields won't get set. To get around this, there is a keyword called super. This makes a call to the constructor from the super class. If your class doesn't have default arguments, or if you want to call a base-class constructor that has an argument, you must explicitly write the calls to the base-class constructor using the super keyword and the appropriate argument list:

The name of the constructor is the same as the name of the class: Game. The first line of the code between curly brackets is the super call (note the round brackets after super). f you don't call the base-class constructor in BoardGame( ), the compiler will complain that it can't find a constructor of the form Game( ). In addition, the call to the base-class constructor *must* be the first thing you do in the derived-class constructor. (The compiler will remind you if you get it wrong.)

b) Describe the implementation of multiple inheritance in java with suitable example

"**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();
                    }
}
```

OR

VI.

a) Explain visibility modes

7

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;}
}
```

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;}
}
```

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;}
}
```

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;   }
}
```

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.

b) Illustrate method overriding with suitable example

- In Method Overriding, sub class has the same method with same name and exactly the same number and type of parameters and same return type as a super class.

- Method Overriding means method of base class is re-defined in the derived class having same signature.
- Method Overriding is to "Change" existing behavior of method.
- It is a **run time polymorphism.**
- It always requires inheritance in Method Overriding.
- In Method Overriding, methods must have **same signature.**
- In Method Overriding, relationship is there between methods of super class and sub class.
- In Method Overriding, methods have same name and same signature but in the different class.Method Overriding requires at least two classes for overriding.

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

UNIT – III

VII.

a) Explain creating and extending of threads. Give example

8

In Java, thread can be created in two different way. By Implementing runnable and by extending thread class

Implementing the Runnable Interface

The easiest way to create a thread is to create a class that implements the runnable interface. After implementing runnable interface, the class needs to implement the run() method, which is of form,

public void run();

- run() method introduces a concurrent thread into your program. This thread will end when run() returns
- You must specify the code for your thread inside run() method
- run() method can call other methods, can use other classes and declare variables just like any other normal method.

```
 class MyThread  implements Runnable
{
public void  run()
{
System.out.println("concurrent thread started running..");
}
}
class MyThreadDemo
{
public static void main( String args[] )
{
MyThread mt = new MyThread();
Thread t = new Thread(mt);
t.start();
}
}
```

To call the run() method, <b>start() method is used. On calling start(), a new stack is provided to the thread and run() method is called to introduce the new thread into the program

Extending Thread class

This is another way to create a thread by a new class that extends Thread class and create an instance of that class. The extending class must override run() method which is the entry point of new thread

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
```

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

b) Write an example of creating and accessing of packages.

7

Following steps are to be followed in creating and using packages.

4. Create a package with a .class file
5. set the classpath from the directory from which you would like to access. It may be in a different drive and directory. Let us call it as a target directory.
6. Write a program and use the file from the package.

Let us create a package called **forest** and place a class called **Tiger** in it. Access the package from a different drive and directory.

1st Step: Create a package (forest) and place Tiger.class in it.

```
package forest; import java.util.*;
public class Tiger
 {
        public void getDetails(String nickName, int weight)
        {
                System.out.println("Tiger nick name is " + nickName);
                System.out.println("Tiger weight is " + weight);
        }
}
```

Order of Package Statement

The above program codingwise is very simple but is important to know the steps of package creation.

```
package forest;
importjava.util.*;
public class Tiger
```

Package is a keyword of Java followed by the package name. Just writing the package statement followed by the name creates a new package; see how simple Java is to practice. For this reason, Java is known as a production language.

While creating packages, the order of statements is very important. The order must be like this, else, compilation error.

4. Package statement
5. Import statement
6. Class declaration

If exists, the package statement must be first one in the program. If exists, the import statement must be the second one. Our class declaration is the third. Any order changes, it is a compilation error.

When the code is ready, the next job is compilation. We must compile with package notation

OR

VIII.

a) Describe the benefits of packages.

3

- Packages can contain hidden classes that are used by the package but are not visible or accessible outside the package.
- Classes in packages can have fields and methods that are visible by all classes inside the package, but not outside.
- Different packages can have classes with the same name. For example, java.awt.Frame and photo.Frame.

b) Illustrate any 4 Java API package names and their contents.

4

a. Language support packages(java.lang)
   Example classes: String, Thread. It contain language support classes
b. Networking Packages(java.net)
   Example classes: InetAddresss, Socket
c. i/o Package(java.io)
   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.

c) Explain life cycle of a thread.

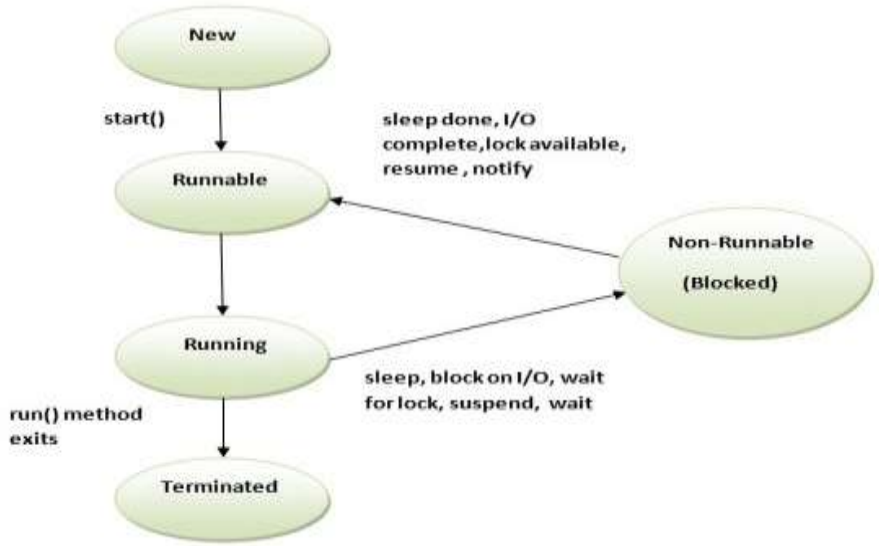
8

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

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.

Above are the different **states of thread** and it's good to know them and how thread changes it's state.

UNIT – IV

IX.

a) Explain exception handling and their tasks incorporated with exception handling

8

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

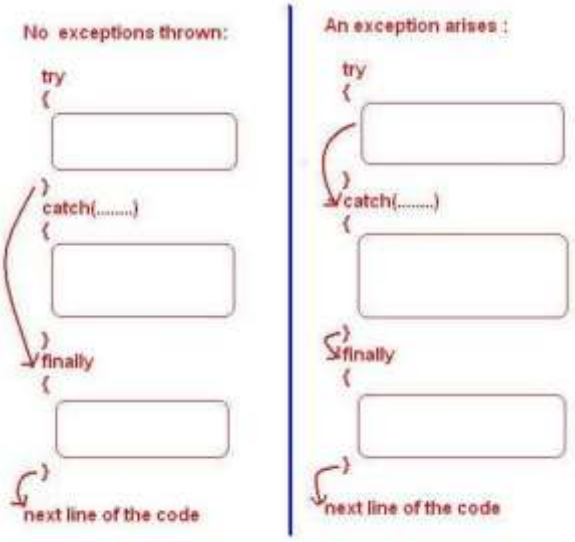
Advantages of Exception Handling

- 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)



b) Explain Applet life cycle.

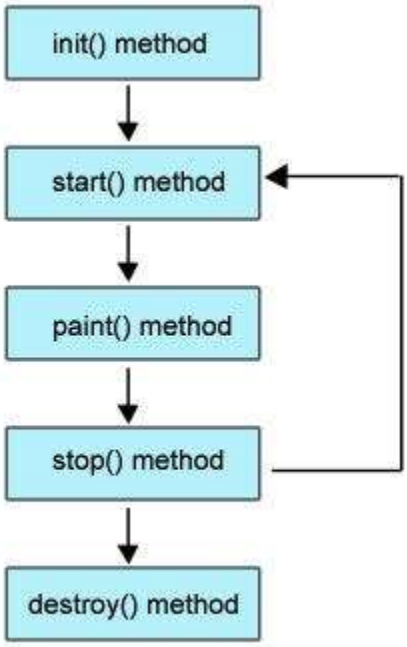
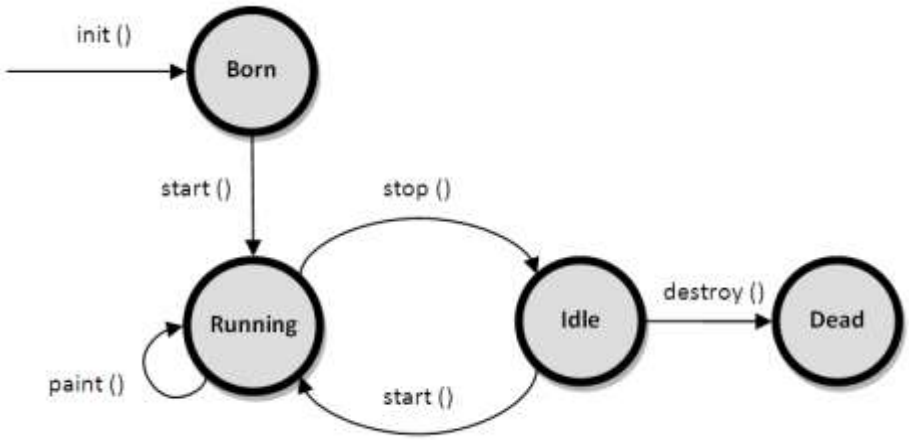


Figure: Life cycle of Applet

- **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 stateof 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) Describe the creation and running of applets. 8

Steps for creating applet

- Create a project for the applet.
- Use the Applet wizard to create an AWT applet.
- Compile and run the applet.
- Customize the applet's UI.
- Add AWT components, such as Choice, Label, and Button.
- Edit the source code.
- Deploy the applet.
- Modify the HTML file.
- Run the deployed applet from the command line.
- Test the applet.

Running

There are two methods to run a Java applet program.

- Using Java compatible web browser
    - Open any text editor like notepad. Type below code

<applet code = "test" width = 400 height = 300> </applet>

[Here width is the width of your output applet window and height is the height of your output applet window.]

- Save as a .html file in location of your java file
- After successful save, open your html file with any Java compatible web browser.
- Now you can see your applet in that browser.

- Using Applet Viewer included with your JDK.
  - Normally your JDK already contain an Applet Viewer for viewing applets (Search in your jdk folder). For using this method, add this line of code to your java program just before of your class.

    /* <applet code = "test" width = 300 height = 300> </applet> */
  - Now, totally your program should be like this:

```
import java.applet.*;
import java.awt.*;
 /* <applet code = "test" width = 300 height = 300> </applet> */
public class test extends Applet
{
 public void paint(Graphics g)
   {
     g.drawString("Hello",123,125);
   }
}
```

  - Save again. Compile your Program.
  - After successful compilation, type: appletviewer test.java to run your applet.

b) Demonstrate reading and writing of files with example

1) Reading or writing characters
2) Reading or writing Bytes

Reading or writing characters

We can read or write characters to a file. The classes used for handling characters are FileReader(for reading characters) and FileWriter (for writing characters)

Program to write characters to a file

```
Import java.io.*;
Class WriteCharacters
{
    public static void main(String args[]) throw IOException
    {
        char b[]={'a','k','b','a','r'};
        FileWriter f = new FileWriter("D:/wc.txt");
        f.write(b);
        f.close();
   }
}
```

Program to read characters from a file

```
        import java.io.*;
        class ReaderCharacters
        {
            public static void main(string args[]) throw IOException
             {
                 int b;
                 FileWriter f = new FileWriter("D:/wc.txt");
                 While((b=f.read())!=-1)
                          System.out.print((char)b);
                 f.close();
             }
        }
```

Program to read bytes to a file

```
Import java.io.*;
Class WriteBytes
{
    public static void main(String args[]) throw IOException
    {
        byte b[]={'a','k','b','a','r'};
        FileOutputStream f = new FileOutputStream("wb.txt");
        f.write(b);
        f.close();
   }
}
```

Program to read bytes from a file

```
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();
    }
}
```