TED (10)-3069 Reg. No. …….

(REVISION-2010) Signature ……..……

FORTH SEMESTER DIPLOMA EXAMINATION IN ENGINEERING/ TECHNOLIGY- MARCH, 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

a. Define constructor

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.

Example:

```
class Complex
{
   Complex()
   {
     body;
   }
}
```

b. Differentiate between public access and friendly access variables

- 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

```
{
public int a;  //public field a
public void show()  //public 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;}
}
```

c. Propose two way to create a thread

   a. Extending the Thread class
   b. Implementing the Runnable interface

d. Write any 4 I/O exception classes

   a. BufferReader
   b. InputStreamReader
   c. FileInputStream
   d. FileReader

e. Compare single character constant and string constant

   A character constant stored only a single character (size: 2 bytes). A string constant stores a sequence of characters. i.e. a string is a sequence of characters

PART – B

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

a. Explain basic concept of OOP

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

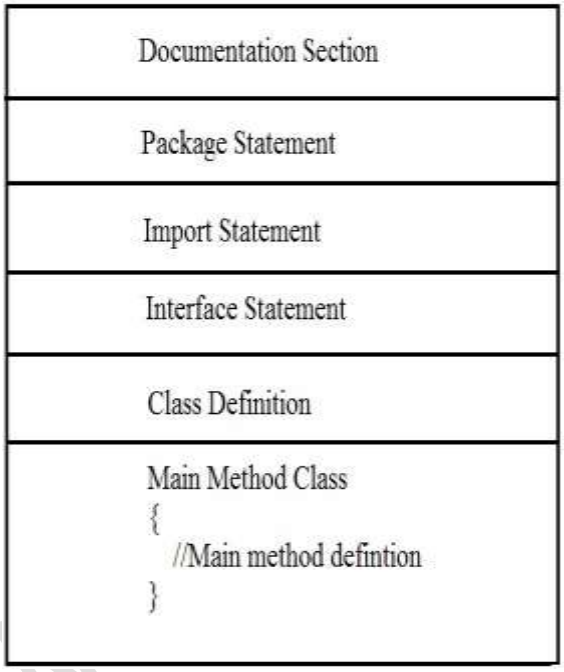
Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

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).

b. Describe the structure of java program



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.

c. Compare inheritance and classes by giving suitable example

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
Variabes	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.

d. Describe the steps to add s class to a package

It is easy to add a class to an existing package.

- In order to ad a class to an existing package just adds the package statement as the first statement before the class definition.
- Save the file in the folder that contains other classes of that package.
- Compile the file to generate the class file.

- Now the new class is added into the package and can be imported into the other programs.

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
{
    ……
    ……
}
```

e. Explain synchronization in threads with example

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:

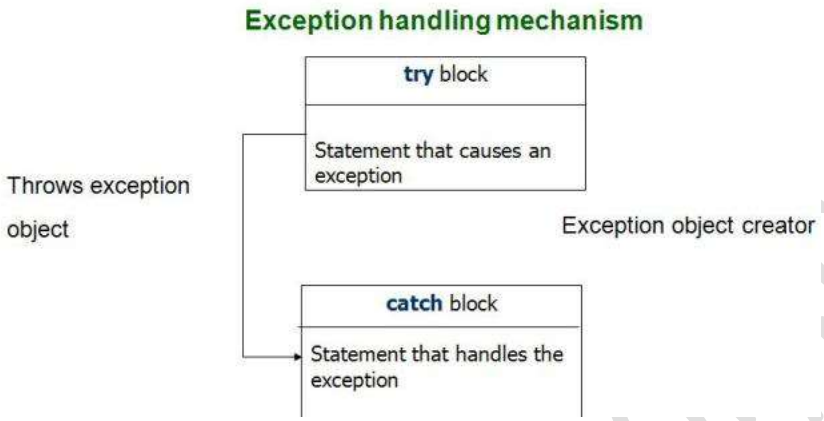
```
public class Counter
{
  private int count = 0;
  public void increment()
  {
    synchronized (this)
   {
      count++;
    }
  }
  public int getCount()
  {
    synchronized (this)
    {
      return count;
     }
  }
}
```

f. Differentiate between Applet and stand-alone application

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	Requires highest security

	security	for the system as they are untrusted

g. Describe the use of try and catch Exception in Exception handling with example (5 x 6 = 30)



A try statement is used to catch exceptions that might be thrown as your program executes. You should use a try statement whenever you use a statement that might throw an exception That way, your program won't crash if the exception occurs.

The try statement has this general form:

```
try
{
    statements that can throw exceptions
}
catch (exception-type identifier)
{
    statements executed when exception is thrown
}
finally
{
    statements that are executed whether or not exceptions occur
}
```

The statements that might throw an exception within a try block. Then you catch the exception with a catch block. The finally block is used to provide statements that are executed regardless of whether any exceptions occur.

Here is a simple example:

```
int a = 5;
int b = 0;          // you know this won't work
```

```
try
{
  int c = a / b;    // but you try it anyway
}
catch (ArithmeticException e)
{
  System.out.println("Can't do that!");
}
```

In the preceding example, a divide-by-zero exception is thrown when the program attempts to divide a by b. This exception is intercepted by the catch block, which displays an error message on the console.

PART – C

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

UNIT – I

III.

a) Explain command-line argument with example

8

The java command-line argument is an argument i.e. passed at the time of running the java program. The arguments passed from the console can be received in the java program and it can be used as an input. So, it provides a convenient way to check the behavior of the program for the different values. You can pass **N** (1,2,3 and so on) numbers of arguments from the command prompt.

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

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

compile by > javac CommandLineExample.java

run by > java CommandLineExample sonoo

Output: Your first argument is: sonoo

Frequently, a Java program needs to handle arguments specified on the command-line. For example, you might want to run your program like this:

java MyProgram 1234 www.caltech.edu "olive festival"

Somehow, your program needs to be able to access these values from the command-line. The way your program can do this is through the String[] argument passed to your main method. Note that the argument is an array of strings. Each element of the array contains one of the values specified on the command-line.

In the above example, the args array will contain the following values:

- args.length = 3
- args[0] = "1234"
- args[1] = "www.caltech.edu"
- args[2] = "olive festival"

b) Distinguish between abstract class and final 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)
Adding extra functionality	Extra functionality to the methods can be added in	No extra functionality can be added and should be used as it

OR

IV.

a) Compare POP and OOP

8

	Procedure Oriented Programming	**Object Oriented Programming**
Divided Into	In POP, program is divided into	In OOP, program is divided into

	small parts calledfunctions.	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 POP is not so easy.	OOP provides an easy way to add 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.

b) Explain method overloading with 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. 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();
        }
}
```

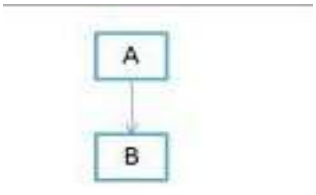
UNIT – II

V.

a) Explain different forms in inheritance

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.

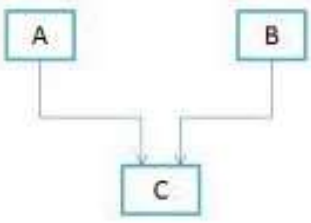


(a) Single Inheritance

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

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.

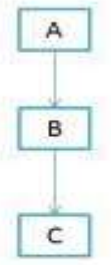


(b) Multiple Inheritance

Note 1: Multiple Inheritance is very rarely used in software projects. Using Multiple inheritance often leads to problems in the hierarchy. This results in unwanted complexity when further extending the class.

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.

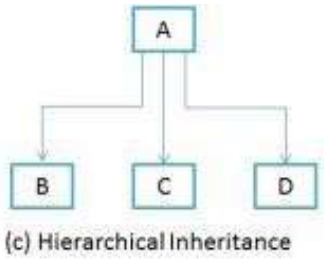


(d) Multilevel Inheritance

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

Hierarchical Inheritance

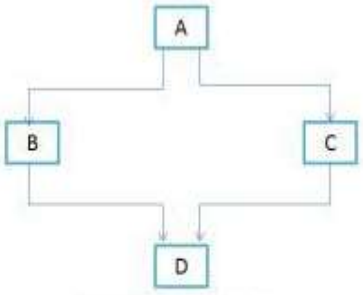
In such kind of inheritance one class is inherited by many sub classes. In below example class B,C and D inherits the same class A. A is parent class (or base class) of B,C & D. Read More at – Hierarchical Inheritance in java with example program.



(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

b) Describe the syntax of extending interface with example

7

Extending Interfaces:

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
{
        ……..
}
```

OR

VI.

a) Discuss different levels of access protection available in java

- 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) Explain the process, implementing interfaces with example

7

Interface in java is core part of Java programming language and one of the way to achieve abstraction in Java along with abstract class. Even though interface is fundamental object oriented concept ; Many Java programmers thinks Interface in Java as advanced concept and refrain using interface from early in programming career. At very basic level interface in java is a keyword but same time it is an object oriented term to define contracts and abstraction , This contract is followed by any implementation of Interface in Java. Since multiple inheritance is not allowed in Java, interface is only way to implement multiple inheritance at Type level.

1) Interface in java is declared using keyword interface and it represent a Type like any Class in Java. a reference variable of type interface can point to any implementation of that interface in Java
2) All variables declared inside interface is implicitly public final variable or constants. which brings a useful case of using Interface for declaring Constants. We have used both Class and interface for storing application wide constants and advantage of using Interface was that you can implement interface and can directly access constants without referring them with class name which was the case earlier when Class is used for storing Constants.

3) All methods declared inside Java Interfaces are implicitly public and abstract, even if you don't use public or abstract keyword. you can not define any concrete method in interface.

4) In Java its legal for an interface to extend multiple interface. for example following code will run without any compilation error: interface Session extends Serializable, Clonnable{ }

```
interface SessionIDCreator extends Serializable, Cloneable{
    String TYPE = "AUTOMATIC";
    int createSessionId();
}
class SerialSessionIDCreator implements SessionIDCreator{

    private int lastSessionId;
    @Override
    public int createSessionId() {
        return lastSessionId++;
    }

}
```

UNIT – III

VII.

a) Describe the complete life cycle of a thread with state transition diagram

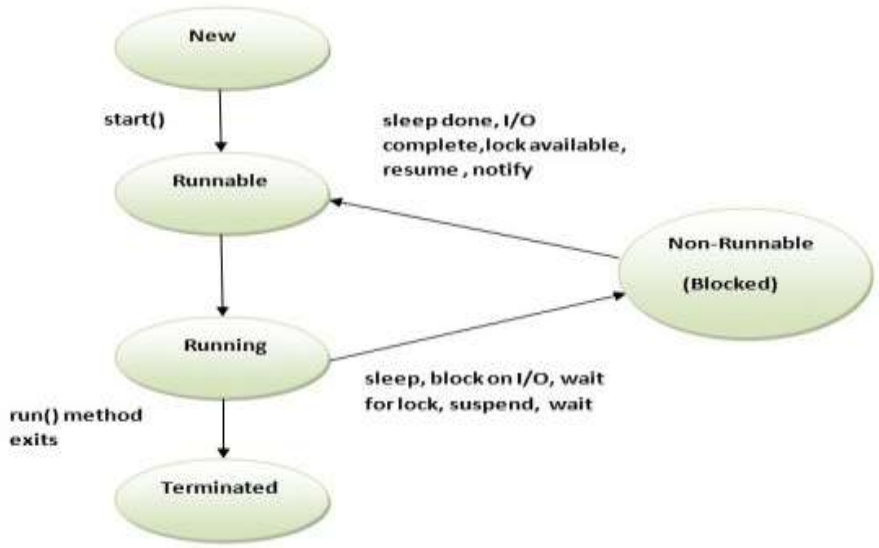
8

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.

b) Explain the general format for accessing a package

7

In a Java source file, import statements occur immediately following the package statement (if it exists) and before any class definitions. This is the general form of the importstatement:

import *pkg1*[.*pkg2*].(*classname*|*);

Here, *pkg1* is the name of a top-level package, and *pkg2* is the name of a subordinate package inside the outer package separated by a dot (.). There is no practical limit on the depth of a package hierarchy, except that imposed by the file system. Finally, you specify either an explicit *classname* or a star (*), which indicates that the Java compiler should import the entire package. This code fragment shows both forms in use:

```
Import java.util.Date;
import java.io.*;
```

Example:

```
import MyPack.*;
class TestBalance
{
  public static void main(String args[])
  {
  /* Because Balance is public, you may use Balance
  class and call its constructor. */
  Balance test = new Balance("J. J. Jaspers", 99.88);
  test.show(); // you may also call show()
  }
}
```

OR

VIII.

a) Explain the steps involved in creating a package

7

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.

b) Write note on thread exception

8

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
 }
```

UNIT – IV

IX.

a) Explain about Byte Stream Classes 8

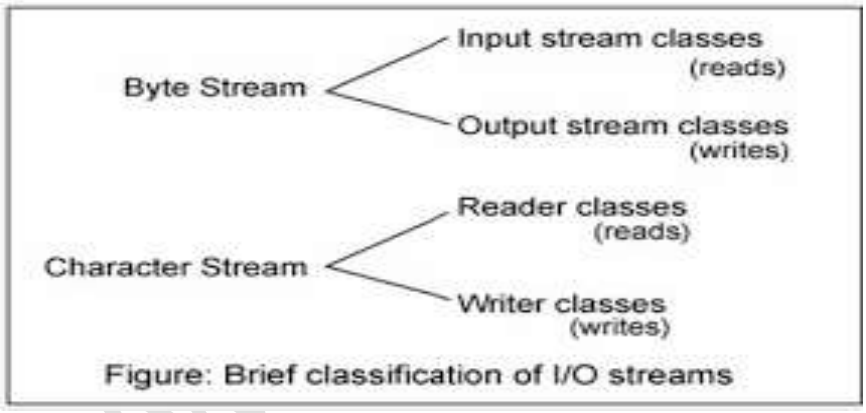


Figure: Brief classification of I/O streams

The package java.io provides two set of class hierarchies - one for handling reading and writing of bytes, and another for handling reading and writing of characters. The abstract classes InputStream and OutputStream are the root of inheritance hierarchies handling reading and writing of bytes respectively.

Input Stream classes

Input stream classes are used to read 8 bit bytes. The InputStream classes defines method for performing the following functions

1) Reading bytes
2) Closing bytes

3) Marking position in streams
4) Skipping ahead in a stream
5) Finding the number of bytes in a stream

The classes include in the hierarchy of InputStream classes are: FileInputStream, PipeInputStream, FilterInputStream

Output Stream classes

Output stream classes are used to write 8 bit bytes. The OutputStream classes defines method for performing the following functions

1) writing bytes
2) Closing stream
3) Flushing stream
4) The classes include in the hierarchy of OutputStream classes are: FileOutputStream, PipeOutputStream, FilterOutputStream etc.

b) Describe the steps involved in developing applet

7

- 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.

OR

X.

a) Describe the applet life cycle with state transition diagram

10

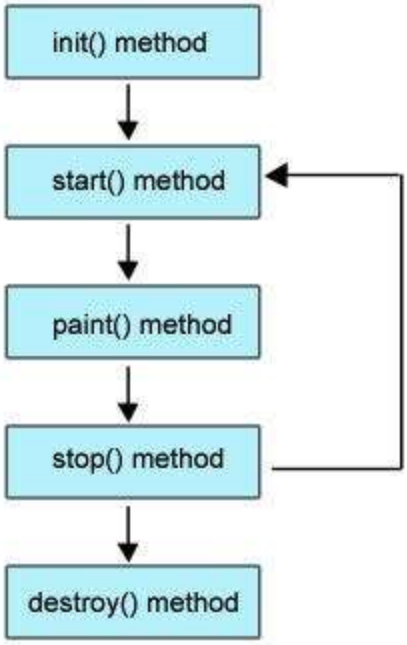
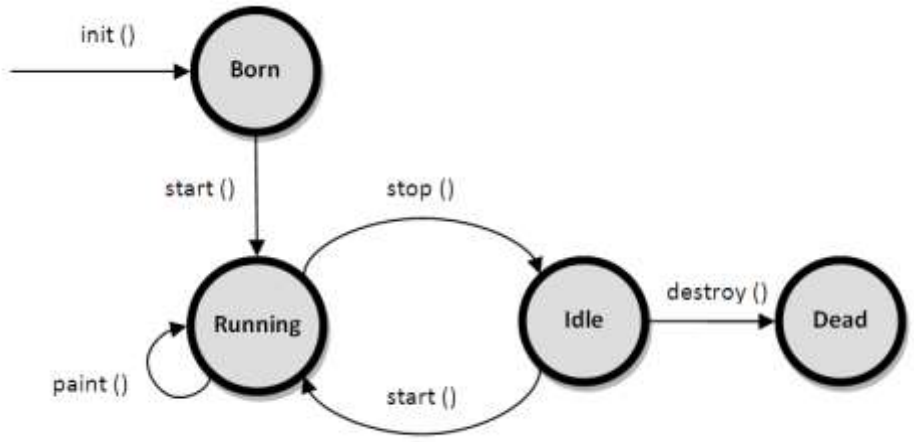


Figure: Life cycle of Applet

- **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.



b) State and explain exception and exception handling mechanism

5

An **exception** is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions. 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. Creating an exception object and handing it to the runtime system is called *throwing an exception*.

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.