

OOOP through JAVA (JNTUK-R19-2-1-ECE)
UNIT V: MULTITHREADING, EXCEPTION HANDLING, COLLECTIONS,
JAVA BEANS, NETWORK PROGRAMMING

Syllabus:

Multithreading in java: Thread life cycle and methods, Runnable interface, Thread synchronization, Exception handling with try-catch-finally, Collections in java, Introduction to JavaBeans and Network Programming.

1. INTRODUCTION TO MULTITHREADING:

- Multithreading is a programming concept where a program (process) is divided into two or more subprograms (process), which can be implemented at the same time in parallel.
- A multithreaded program contains two or more parts that can run concurrently. Each part of such a program is called a thread, and each thread defines a separate path of execution.
- A process consists of the memory space allocated by the operating system that can contain one or more threads.
- A thread cannot exist on its own; it must be a part of a process.
- A thread is a single sequential flow of control within a program
- There are two distinct types of Multitasking
 - Processor-Based multitasking
 - Thread-Based multitasking

Process-Based Vs Thread-based multitasking:

- Process-Based multitasking is a feature that allows your computer to run two or more programs concurrently.
- For example, you can listen to music and at the same time chat with your friends on Facebook using browser.
- In Thread-based multitasking, thread is the smallest unit of code, which means a single program can perform two or more tasks simultaneously.
- For example, a text editor can print and at the same time you can edit text provided that those two tasks are performed by separate threads.

Multithreading	Multitasking
<ul style="list-style-type: none"> ▪ It supports execution of multiple parts of a single program simultaneously ▪ All threads share common address space ▪ Context switching is low cost ▪ Interthread communication is inexpensive ▪ Thread is light weight task ▪ It is under the control of java 	<p>It supports execution of multiple programs simultaneously.</p> <p>Each process has its own address space</p> <p>Context switching is high cost</p> <p>Interprocess communication is expensive</p> <p>process is heavy weight task</p> <p>It is not under the control of java</p>

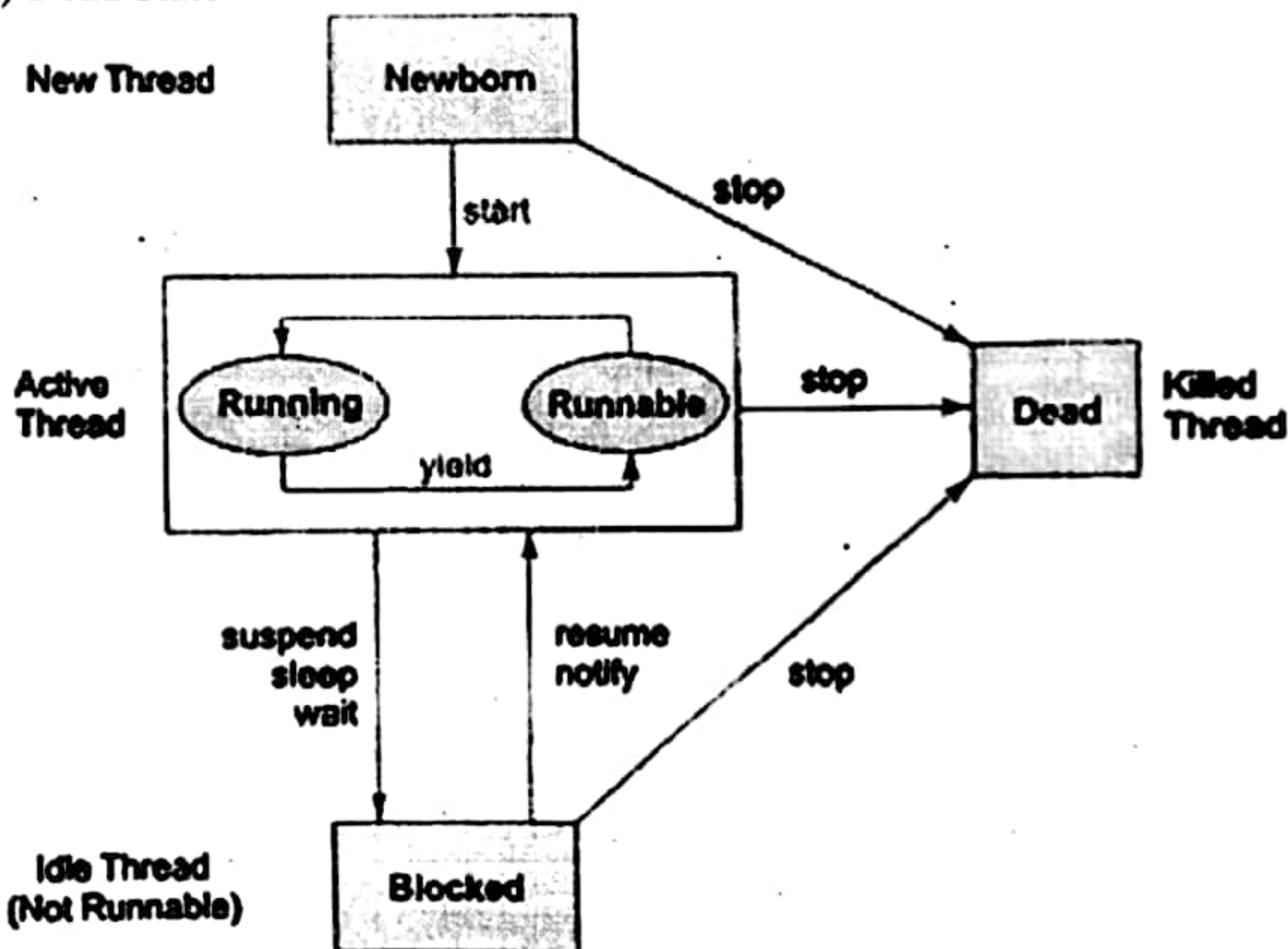
Benefits of Multithreading:

- Enables programmers to do multiple things at one time
- Programmers can divide a long program into threads and execute them in parallel which eventually increases the speed of the program execution
- Improved performance and concurrency
- Simultaneous access to multiple applications

2. LIFE CYCLE OF THREAD:

A thread can be in any of the five following states

- Newborn State
- Runnable State
- Running State
- Blocked State
- Dead State



i) Newborn State:

- When a thread object is created a new thread is born and said to be in Newborn state.

ii) Runnable State:

- If a thread is in this state it means that the thread is ready for execution and waiting for availability of the processor.
- If all threads in queue are of same priority then they are given time slots for execution in round robin fashion

iii) Running State:

- It means that the processor has given its time to the thread for execution.
- A thread keeps running until the following conditions occurs
 - * Thread give up its control on its own and it can happen in the following situations

- A thread gets suspended using **suspend()** method which can only be revived with **resume()** method
- A thread is made to sleep for a specified period of time using **sleep(time)** method, where time in milliseconds
- A thread is made to wait for some event to occur using **wait ()** method. In this case a thread can be scheduled to run again using **notify ()** method.

* A thread is pre-empted by a higher priority thread

iv) Blocked State:

- If a thread is prevented from entering into runnable state and subsequently running state, then a thread is said to be in Blocked state.

v) Dead State:

- A runnable thread enters the Dead or terminated state when it completes its task or otherwise terminates.

*) Main Thread:

- Every time a Java program starts up, one thread begins running which is called as the main thread of the program
- It is the one that is executed when your program begins.
- Child threads are produced from main thread
- Often it is the last thread to finish execution as it performs various shut down operations

3. CREATING A THREAD:

- Java defines two ways to create threads
 - By implementing the Runnable interface
 - By extending the Thread class
- The two methods used while creating threads are
 - public void run()
 - public void start()

public void run():

- It is used to perform action for a thread.

public void start():

- It starts the execution of the thread.
- Then JVM calls the run() method

By implementing the Runnable interface:

- To do this we must perform the following steps:
 - Declare a class as implementing the **Runnable** interface
 - Implement the **run()** method
 - Create a **Thread** by defining an object that is instantiated from this "runnable" class as the target of the thread
 - Call the thread's **start()** method to run the thread.

Example:

```
class t1 implements Runnable
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t1 obj1 = new t1();
        Thread t = new Thread(obj1);
        t.start();
    }
}
```

By extending the Thread class:

- It includes the following steps:

- Declare the class as extending the Thread class.
- Override the "run()" method that is responsible for running the thread.
- Create a thread and call the "start()" method to instantiate the Thread Execution.

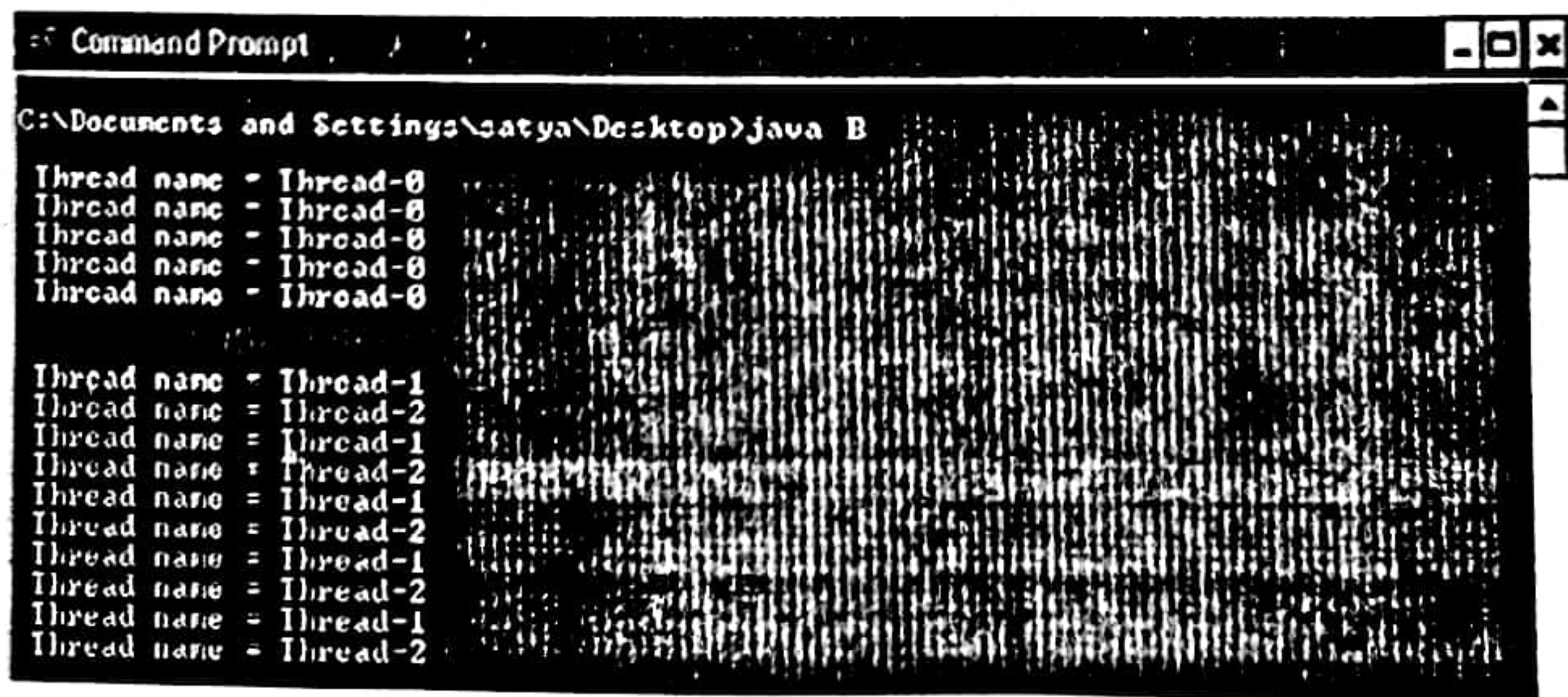
```
class t2 extends Thread
{
    public void run()
    {
        System.out.println("Thread is Running");
    }
    public static void main(String args[])
    {
        t2 obj1 = new t2();
        obj1.start();
    }
}
```



4. CREATING MULTIPLE THREADS:

```
public class B extends Thread {  
  
    public void run() {  
        System.out.println();  
        for (int i = 1; i <= 5; i++) {  
            System.out.println(" Thread name = "+ Thread.currentThread().getName());  
        }  
    }  
  
    public static void main(String[] args) {  
        B t1 = new B();  
        t1.start();  
  
        B t2 = new B();  
        t2.start();  
  
        B t3 = new B();  
        t3.start();  
    }  
}
```

Output :



```
Command Prompt  
C:\Documents and Settings\satya\Desktop>java B  
Thread name = Thread-0  
Thread name = Thread-0  
Thread name = Thread-0  
Thread name = Thread-0  
Thread name = Thread-0  
  
Thread name = Thread-1  
Thread name = Thread-2  
Thread name = Thread-1  
Thread name = Thread-2  
Thread name = Thread-1  
Thread name = Thread-2  
Thread name = Thread-1  
Thread name = Thread-2  
Thread name = Thread-1  
Thread name = Thread-2
```

5. METHODS OF THREAD CLASS:

Methods	Description
<code>static Thread currentThread()</code>	Returns a reference to the currently executing thread.
<code>static int activeCount()</code>	Returns the current number of active threads.
<code>long getID()</code>	Returns the identification of thread.
<code>final String getName()</code>	Returns the thread's name.
<code>final void join()</code>	Waits for a thread to terminate.
<code>void join (long m)</code>	Waits at the most for 'm' milliseconds for the thread to die.
<code>void join (long m, int n)</code>	Waits at the most for 'm' milliseconds and 'n' nanoseconds for the thread to die.
<code>void run()</code>	Entry point for the thread.
<code>final void setDaemon(boolean how)</code>	If how is true, the invoking thread is set to daemon status.
<code>boolean isInterrupted()</code>	Returns true if the thread on which it is called has been interrupted.
<code>final boolean isDaemon()</code>	Returns true if the invoking thread is a daemon thread.
<code>final boolean isAlive()</code>	Returns boolean value stating whether a thread is still running.
<code>void interrupt()</code>	Interrupts a thread.
<code>static boolean holdsLock(Object anyObj)</code>	Returns true if the invoking thread holds the lock on anyObj.
<code>Thread.State getState()</code>	Returns the current state of the thread.
<code>final int getPriority()</code>	Returns the priority of the thread.
<code>static boolean interrupted()</code>	Returns true if the invoking thread has been interrupted.
<code>final void setName(String thrdName)</code>	Sets a thread's name to thrdName.
<code>final void setPriority(int newPriority)</code>	Sets a thread's priority to newPriority.
<code>static void sleep(long milliseconds)</code>	Suspends a thread for a specified period of milliseconds.
<code>void start()</code>	Starts a thread by calling its run() method.
<code>void destroy()</code>	Destroys the thread, without any clean up.
<code>static int enumerate (Thread[] thrdArray)</code>	Copies into the specified array, every active thread of thread's group and sub group.
<code>static void yield()</code>	Cause the current executing thread to pause and allow other threads to execute.

6. THREAD SYNCHRONIZATION:

- Synchronization in java is the capability *to control the access of multiple threads to any shared resource.*

- Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Why use Synchronization:

The synchronization is mainly used to

- i) To prevent thread interference.
- ii) To prevent consistency problem.

Types of Synchronization:

There are two types of synchronization

- i) Process Synchronization
- ii) Thread Synchronization

Thread Synchronization:

There are two types of thread synchronization mutual exclusive and inter-thread communication.

- i) Mutual Exclusive
 - a) Synchronized method.
 - b) Synchronized block.
 - c) Static Synchronization.
- ii) Cooperation (Inter-thread communication in java)

Mutual Exclusive:

Mutual Exclusive helps keep threads from interfering with one another while sharing data. This can be done by three ways in java:

- i) by synchronized method
- ii) by synchronized block
- iii) by static synchronization

Concept of Lock in Java:

- Synchronization is built around an internal entity known as the lock or monitor.
- Every object has an lock associated with it.
- By convention, a thread that needs consistent access to an object's fields has to acquire the object lock before accessing them, and then release the lock when it's done with them.
- From Java 5 the package `java.util.concurrent.locks` contains several lock implementations.

Understanding the problem without Synchronization:

In this example, there is no synchronization, so output is inconsistent

```
class Table{
void printTable(int n){//method not synchronized
for(int i=1;i<=5;i++){
System.out.println(n*i);
try{
Thread.sleep(400);
}catch(Exception e){System.out.println(e);}
}
}
}
```

```

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1 {
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}

```

Output:

```

5
100
10
200
15
300
20
400
25
500

```

Java synchronized method:

- If you declare any method as synchronized, it is known as synchronized method.
- Synchronized method is used to lock an object for any shared resource.

- When a thread invokes a synchronized method, it automatically acquires the lock for that object and releases it when the thread completes its task.

//example of java synchronized method

```
class Table{
    synchronized void printTable(int n){//synchronized method
        for(int i=1;i<=5;i++){
            System.out.println(n*i);
            try{
                Thread.sleep(400);
            }catch(Exception e){System.out.println(e);}
        }
    }
}
```

```
class MyThread1 extends Thread{
    Table t;
    MyThread1(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(5);
    }
}
```

```
class MyThread2 extends Thread{
    Table t;
    MyThread2(Table t){
        this.t=t;
    }
    public void run(){
        t.printTable(100);
    }
}
```

```
public class TestSynchronization2{
    public static void main(String args[]){
        Table obj = new Table();//only one object
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```

Output:

- 5
- 10
- 15
- 20
- 25
- 100
- 200
- 300
- 400
- 500

7. EXCEPTION HANDLING IN JAVA

i) Error:

- Errors are the wrongs that can make a program go wrong
- An error may produce
 - An incorrect output or
 - May terminate the execution of the program abruptly or
 - Even may cause the system to crash

Types of errors:

- Errors are of two types
 - a) Compile-time errors
 - b) Run-time errors

a) Compile-time Errors:

- All syntax errors will be detected and displayed by the java compiler

Examples:

- Missing Semicolons
- Missing or Mismatching brackets in classes and methods
- Misspelling of identifiers and keywords
- Missing double quotes in strings
- Use of undeclared variables
- Incompatible types in assignments

b) Runtime Errors:

- Sometimes a program may compile successfully but may not run properly
- Such programs
 - may produce wrong results due to wrong logic or
 - may terminate due to errors such as stack overflow

Examples:

- Dividing an integer by zero
- Accessing an element that is out of bounds of an array

ii) Exception:

- An exception is a condition that is caused by a run-time error in the program

Example:

- Dividing an integer by zero

Exception Types:

- Exceptions are broadly classified into two categories:
- *checked* and *unchecked* exceptions,

- Checked exceptions are those for which the compiler checks to see whether they have been handled in your program or not.

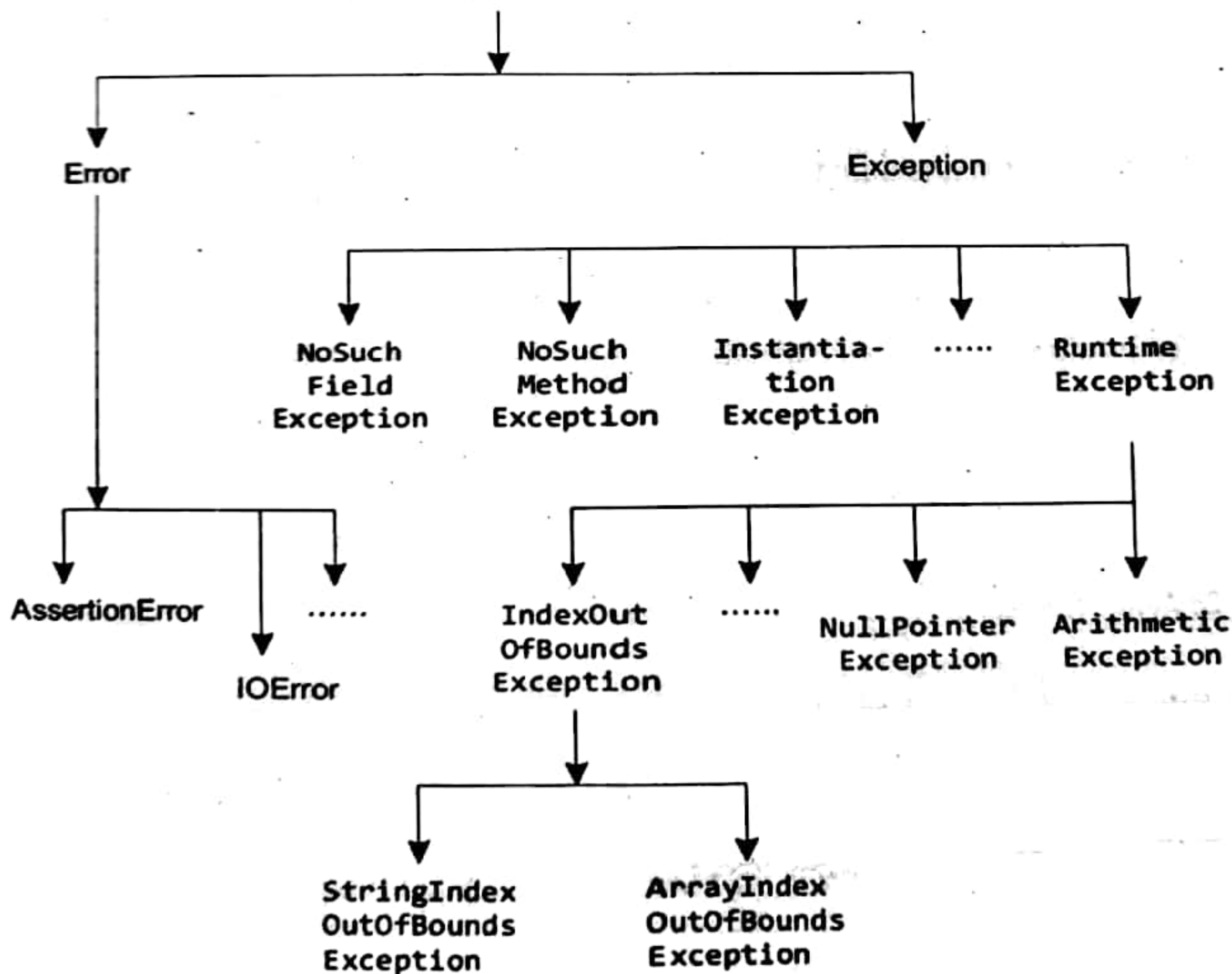
- Unchecked or runtime exceptions are not checked by the compiler.

- Checked and Unchecked Exception Classes are depicted in a table as,

Checked Exceptions	Unchecked Exceptions
ClassNotFoundException	ArithmeticException
NoSuchFieldException	ArrayIndexOutOfBoundsException
NoSuchMethodException	NullPointerException
InterruptedException	ClassCastException
IOException	BufferOverflowException
IllegalAccessExcepion	BufferUnderflowException

- Exception Hierarchy is depicted as,

java.lang.Throwable

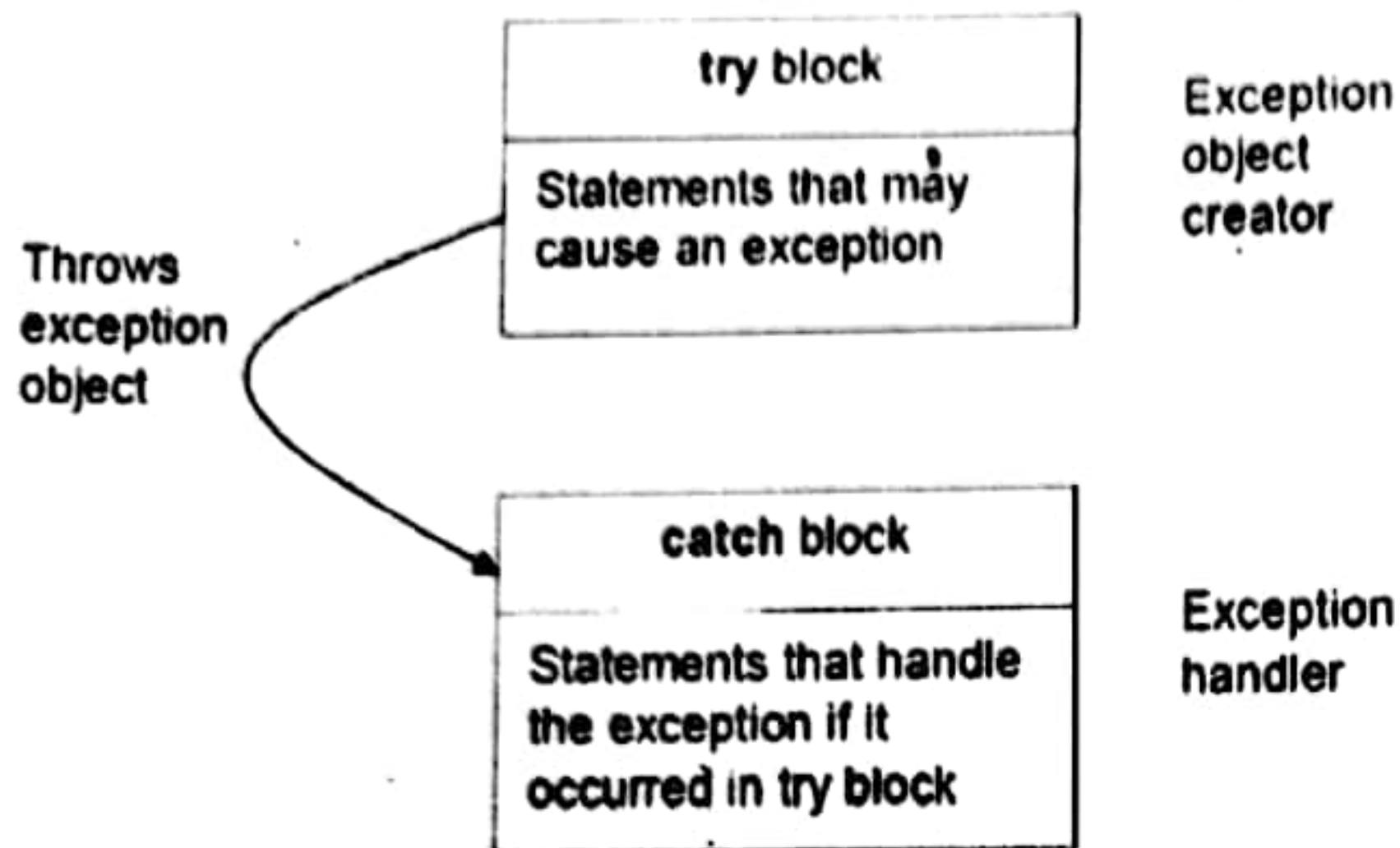


iii) Exception Handling:

- Exception handling is the process of responding to exceptions when a computer program runs.

- It performs the following tasks
 - Find the problem (**Hit** the exception)
 - Inform that an error has occurred (**Throw** the exception)
 - Receive the error information (**Catch** the exception)
 - Take corrective actions (**Handle** the exception)

- **Exception Handling Mechanism** in java is depicted as,



iv) Exception Handling Techniques:

Java provides five keywords for exception handling:

- try, catch, throw, throws, and finally

a) try...catch:

- The try/catch block can be placed within any method that you feel can throw exceptions.
- All the statements to be tried for exceptions are put in a try block and immediately following the try is the catch block.
- catch block is used to catch any exception raised from the try block.
- If exception occurs in any statement in the try block, the following statements are not executed and control immediately passes to the corresponding catch block.

Example:

```

class Division {
    public static void main(String[] args)
    {
        int a=7, b=0, result;
        try
        {
            result = a / b;
            System.out.println("Result = " + result);
        }
        catch (ArithmeticException e)
        {
            System.out.println("Exception caught: Division by zero");
        }
    }
  
```

```
}  
}  
}
```

Output:

Exception caught: Division by zero

b) Multiple Catch Clauses:

- A single try can have multiple catch clauses, for catching specific exceptions.
- As soon as an exception is thrown, the first appropriate catch clause responsible for handling that exception is located and the exception is passed to it.

Example:

```
class multitrydemo  
{  
public static void main(String args[])  
{  
try  
{  
int a=10,b=5;  
int c=a/b;  
int d[]={0,1};  
System.out.println(d[10]);  
System.out.println(c);  
}  
catch(ArithmeticException e)  
{  
System.out.println(e);  
}  
catch(ArrayIndexOutOfBoundsException e)  
{  
System.out.println(e);  
}  
System.out.println("After the catch statement");  
}  
}
```

Output:

java.lang.ArrayIndexOutOfBoundsException: 10
After the catch statement

c) throw Keyword:

- The throw keyword is used to explicitly throw an exception.
- Whether implicit or explicit, objects of exception need to be created before they are thrown.
- Execution of the program is suspended and the runtime environment looks for the appropriate catch to handle the exception.

- throw is more useful when we want to throw a user-defined exception.

Syntax:
throw new NullPointerException(); // throw new ThrowableInstance

Example:

```
class throwdemo
{
public static void main(String args[])
{
try
{
throw new NullPointerException("demo");
}
catch(NullPointerException e)
{
System.out.println(e);
}
}
}
```

Output:

java.lang.NullPointerException: demo

d) throws keyword:

- The throws is added to the method signature to let the caller know about what exceptions the called method can throw.
- It is the responsibility of the caller to either handle the exception (using try...catch mechanism) or it can also pass the exception (by specifying throws clause in its method declaration).
- If all the methods in a program pass the exception to their callers (including main()), then ultimately the exception passes to the default exception handler.
- A method should use either of the two techniques—try/catch or throws.
- Usually (for checked exceptions specifically), it is the catch or specify mechanism that is used.
- A method can throw more than one exception; the exception list is specified as separated by commas.

Syntax:

```
public void divide(int a, int b) throws ArithmeticException,IllegalArgumentException
```

Example:

```
import java.io.*;
class M{
void method()throws IOException{
throw new IOException("device error");
}
}
class throwsdemo{
public static void main(String args[])throws IOException{//declare exception
```

```
M m=new M();
m.method();
```

```
System.out.println("normal flow...");
```

```
}
}
```

Output:

Runtime Exception

e) finally Block:

- The finally block is always executed in try-catch-finally statements irrespective of whether an exception is thrown from within the try/catch block or not.
- Statements following the exception in a try block are not executed.
- Some statements are mandatory to execute such as the statements related to the release of resources.
- All these statements can be put in a finally block.

Syntax:

```
try
{...}
catch(Throwable e)
{...}
finally
{....}
```

Example1:

```
class finallydemo
{
public static void main(String args[])
{
try
{
int a=10,b=0;
int c=a/b;
System.out.println(c);
}
catch(ArithmeticException e)
{
System.out.println(e);
}
finally
{
System.out.println("This is inside finally block");
}
}
```

```
}  
}
```

Output:

java.lang.ArithmeticException: / by zero
This is inside finally block

Example2:

```
class finallydemo  
{  
public static void main(String args[])  
{  
try  
{  
int a=10,b=5;  
int c=a/b;  
System.out.println(c);  
}  
catch(ArithmeticException e)  
{  
System.out.println(e);  
}  
finally  
{  
System.out.println("This is inside finally block");  
}  
}  
}
```

Output:

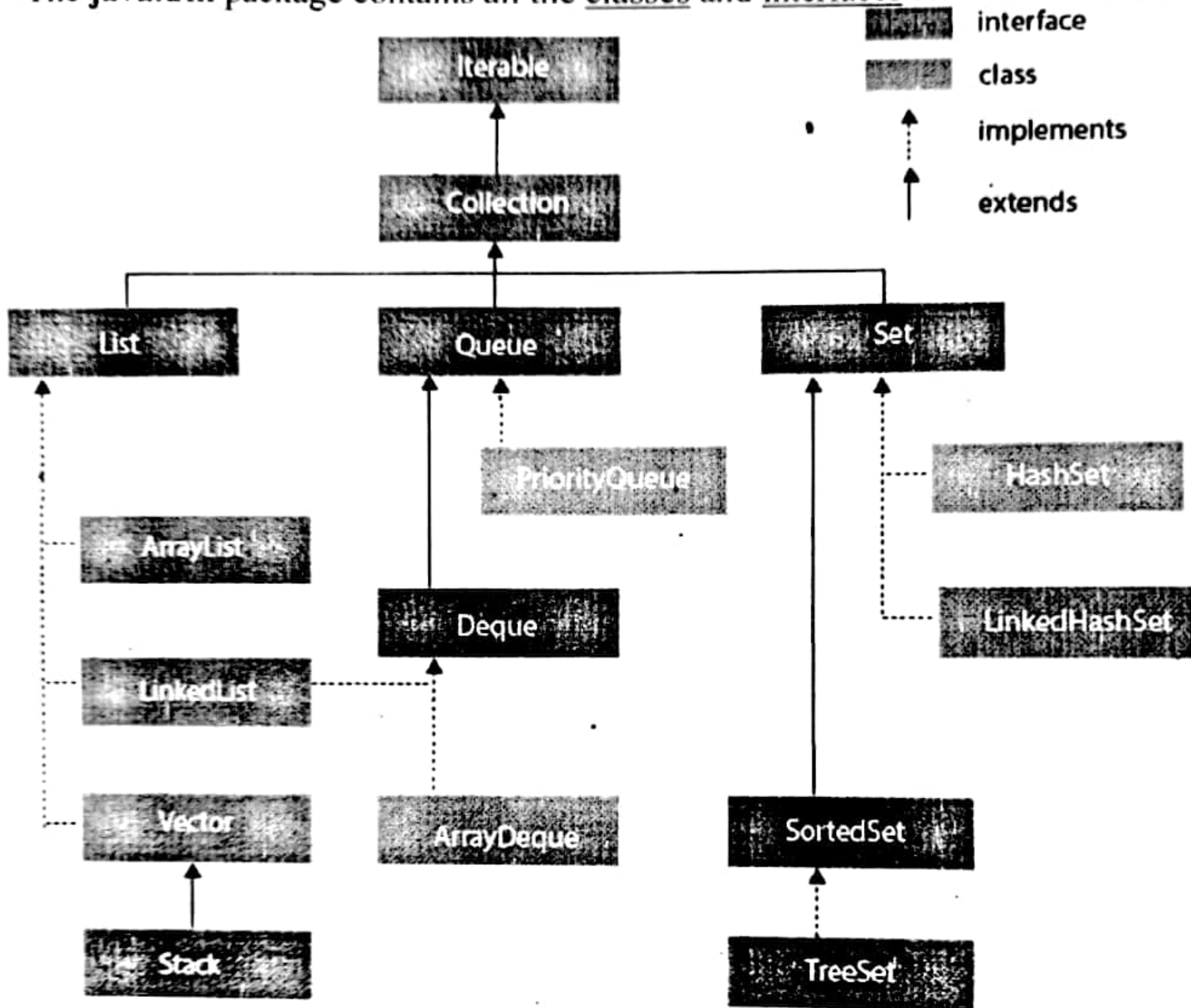
2

This is inside finally block

8. COLLECTIONS IN JAVA:

- The Collection in Java is a framework that provides an architecture to store and manipulate group of objects.
- Java Collections can achieve all the operations that you perform on a data such as searching, sorting, insertion, manipulation, and deletion.
- Java Collection means a single unit of objects.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).
- A Collection represents a single unit of objects, i.e., a group.
- A Framework
 - provides readymade architecture.
 - represents a set of classes and interfaces.
 - is optional.

- The Collection framework represents a unified architecture for storing and manipulating a group of objects.
- It has,
 - Interfaces and its implementations, i.e., classes
 - Algorithm
- The hierarchy of Collection framework.
- The `java.util` package contains all the classes and interfaces for the Collection framework



9. INTRODUCTION TO JAVABEANS AND NETWORK PROGRAMMING:

i) JAVABEAN:

- A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

- According to Java white paper, it is a reusable software component.
- A bean encapsulates many objects into one object so that we can access this object from multiple places.
- Moreover, it provides easy maintenance.

Simple example of JavaBean class:

//Employee.java

```
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

How to access the JavaBean class?

- To access the JavaBean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

JavaBean Properties:

- A JavaBean property is a named feature that can be accessed by the user of the object.
- The feature can be of any Java data type, containing the classes that you define.
- A JavaBean property may be read, write, read-only, or write-only.
- JavaBean features are accessed through two methods in the JavaBean's implementation class:

a) getProperty Name ():

- For example, if the property name is firstName, the method name would be getFirstName() to read that property.
- This method is called the accessor.

b) setProperty Name ():

- For example, if the property name is firstName, the method name would be setFirstName() to write that property.
- This method is called the mutator.

Advantages of JavaBean:

- The following are the advantages of JavaBean:
 - o The JavaBean properties and methods can be exposed to another application.
 - o It provides an easiness to reuse the software components.

Disadvantages of JavaBean:

- The following are the disadvantages of JavaBean:
 - o JavaBeans are mutable. So, it can't take advantages of immutable objects.
 - o Creating the setter and getter method for each property separately may lead to the boilerplate code.

ii) JAVA NETWORKING:

- Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

- Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking:

- a. sharing resources
- b. centralize software management

Java Networking Terminology:

- The widely used java networking terminologies are given below:

- a) IP Address
- b) Protocol
- c) Port Number
- d) MAC Address
- e) Connection-oriented and connection-less protocol
- f) Socket

a) IP Address:

- IP address is a unique number assigned to a node of a network e.g. 192.168.0.1..
- It is composed of octets that range from 0 to 255.
- It is a logical address that can be changed.

b) Protocol:

- A protocol is a set of rules basically that is followed for communication. For example:
 - o TCP
 - o FTP
 - o Telnet
 - o SMTP
 - o POP etc.

c) Port Number:

- The port number is used to uniquely identify different applications.
- It acts as a communication endpoint between applications.
- The port number is associated with the IP address for communication between two applications.

d) MAC Address:

- MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller).
- A network node can have multiple NIC but each with unique MAC.

e) Connection-oriented and connection-less protocol:

- In connection-oriented protocol, acknowledgement is sent by the receiver.
- So it is reliable but slow.
- The example of connection-oriented protocol is TCP.
- But, in connection-less protocol, acknowledgement is not sent by the receiver.
- So it is not reliable but fast.
- The example of connection-less protocol is UDP.

f) Socket:

- A socket is an endpoint between two way communication.

java.net package:

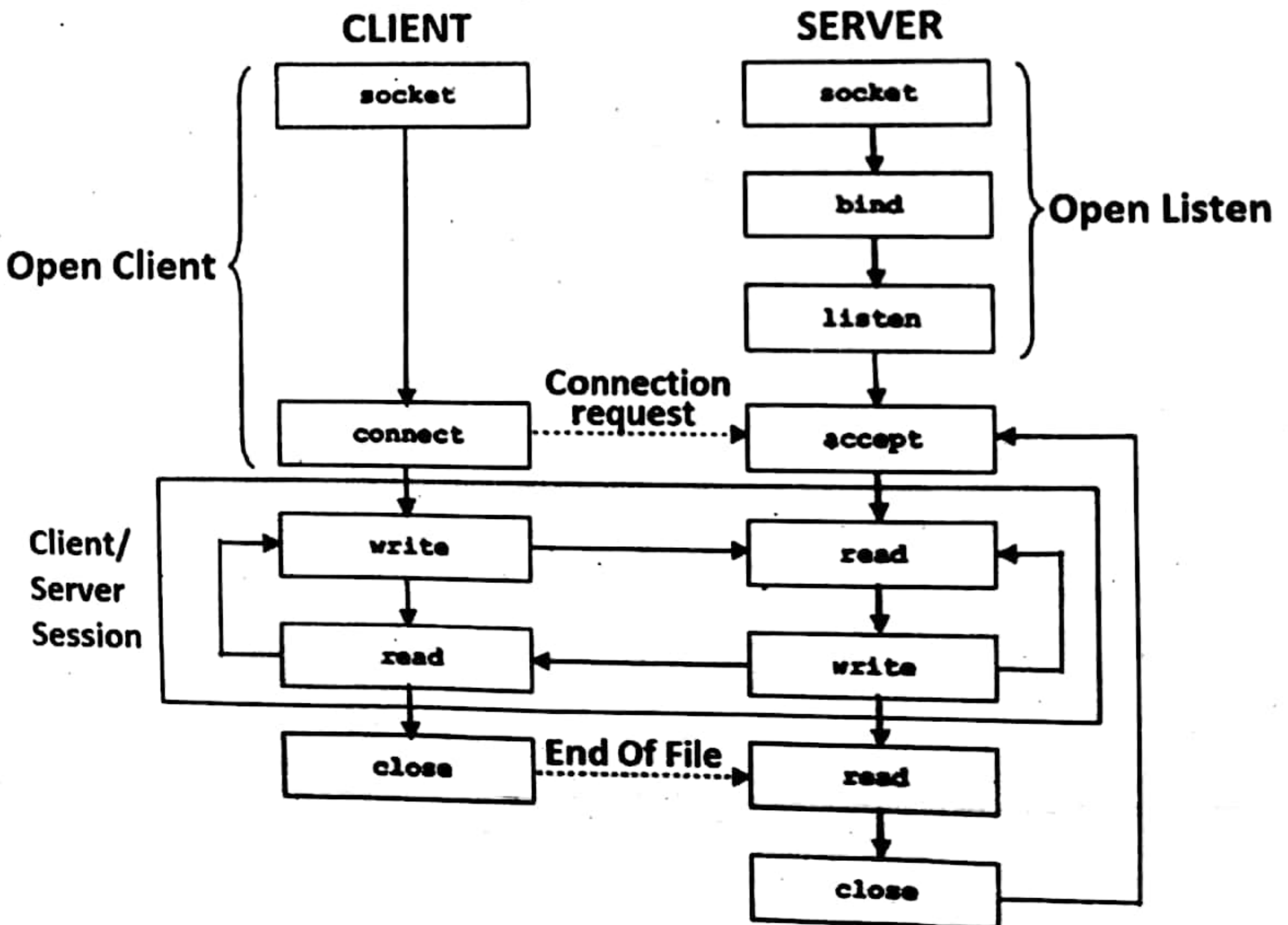
- The java.net package provides many classes to deal with networking applications in Java.

- A list of these classes is given below:

- Authenticator
- CacheRequest
- CacheResponse
- ContentHandler
- CookieHandler
- CookieManager
- DatagramPacket
- DatagramSocket
- DatagramSocketImpl
- InterfaceAddress
- JarURLConnection
- MulticastSocket
- InetSocketAddress
- InetAddress
- Inet4Address
- Inet6Address
- IDN
- HttpURLConnection
- HttpCookie
- NetPermission
- NetworkInterface
- PasswordAuthentication
- Proxy
- ProxySelector
- ResponseCache
- SecureCacheResponse
- ServerSocket
- Socket
- SocketAddress
- SocketImpl
- SocketPermission
- StandardSocketOptions
- URI
- URL
- URLClassLoader
- URLConnection
- URLDecoder
- URLEncoder
- URLStreamHandler

iii) JAVA SOCKET PROGRAMMING:

- Java Socket programming is used for communication between the applications running on different JRE.
- Java Socket programming can be connection-oriented or connection-less.
- Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- The client in socket programming must know two information:
 - a) IP Address of Server, and
 - b) Port number.
- Here, we are going to make one-way client and server communication.
- In this application, client sends a message to the server, server reads the message and prints it.
- Here, two classes are being used: Socket and ServerSocket.
- The Socket class is used to communicate client and server.
- Through this class, we can read and write message.
- The ServerSocket class is used at server-side.
- The accept() method of ServerSocket class blocks the console until the client is connected.
- After the successful connection of client, it returns the instance of Socket at server-side.



SOCKET API

Socket class:

- A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.
- Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class:

- The ServerSocket class can be used to create a server socket.
- This object is used to establish communication with the clients.
- Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming:

Creating Server:

- To create the server application, we need to create the instance of ServerSocket class.
- Here, we are using 6666 port number for the communication between the client and server.
- You may also choose any other port number.
- The accept() method waits for the client.
- If clients connects with the given port number, it returns an instance of Socket.
 - a) `ServerSocket ss=new ServerSocket(6666);`
 - b) `Socket s=ss.accept();//establishes connection and waits for the client`

Creating Client:

- To create the client application, we need to create the instance of Socket class.
- Here, we need to pass the IP address or hostname of the Server and a port number.
- Here, we are using "localhost" because our server is running on same system.
 - a) `Socket s=new Socket("localhost",6666);`

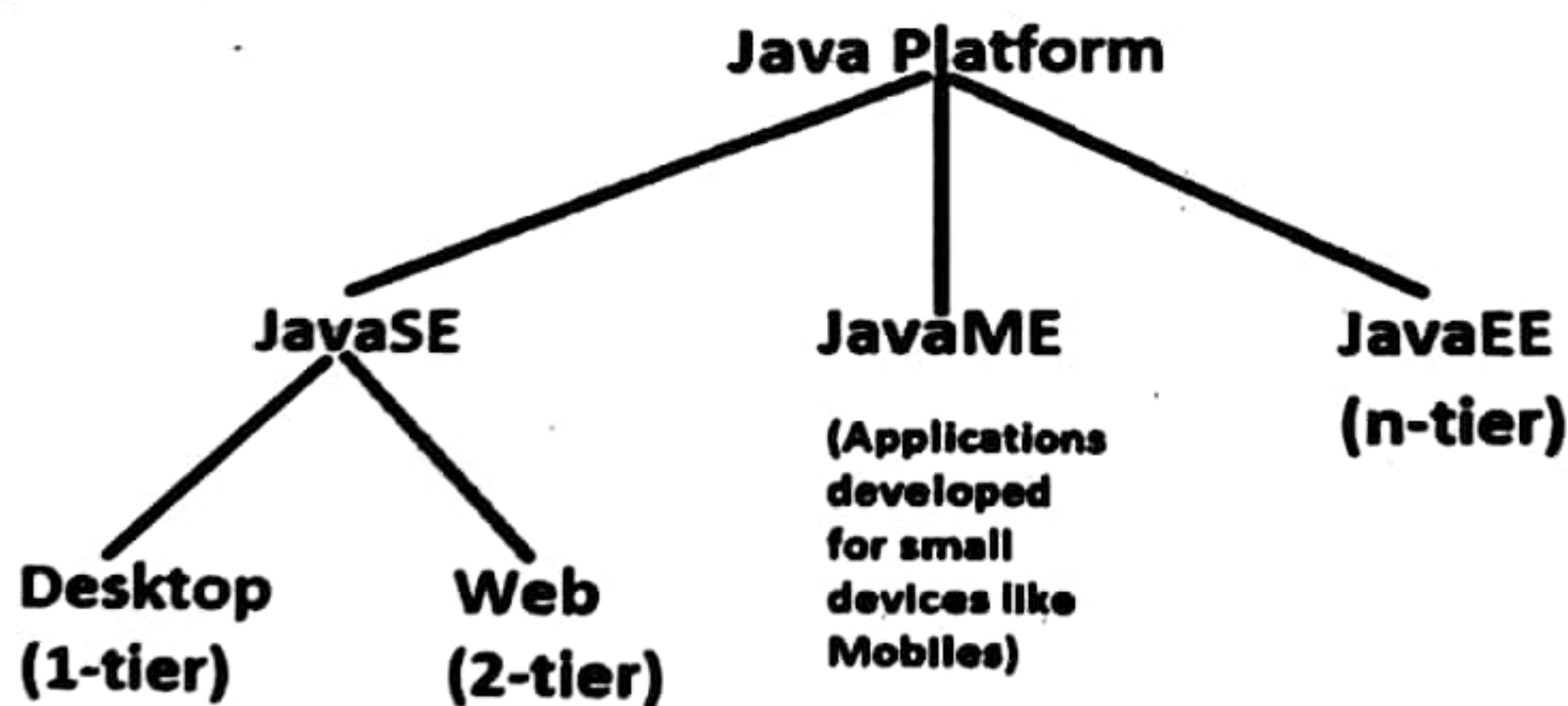
MISSING TOPICS AND ADDITIONAL TOPICS:

1. History of Java:

- Java is an Object-Oriented Programming Language.
- It was developed by Sun Micro Systems of USA in 1991, by a team of professionals led by James Gosling, later it was sold by Oracle in 2010.
- Java was designed for the development of software for consumer electronic devices like TVs, VCRs, toasters and such other electronic machines.
- The first name of Java is OAK.
- Later, it was renamed to Java in 1995.
- The term 'Java' in USA is generally a slang used for coffee.
- Java is also the name of a coffee produced on the islands of Java in Indonesia.
- Java is an open-source language.
- Open-source means it is freely available to the market and any company can develop their own applications using java.
- Java is the most popular language today because of 4 properties in it,
 - Platform Independency
 - Multithreading
 - Internet Programming
 - Distributed Programming

2. Java Platform:

- Java platform is a bundle of related programs used to develop and run the applications written in java language.
- It has three editions.



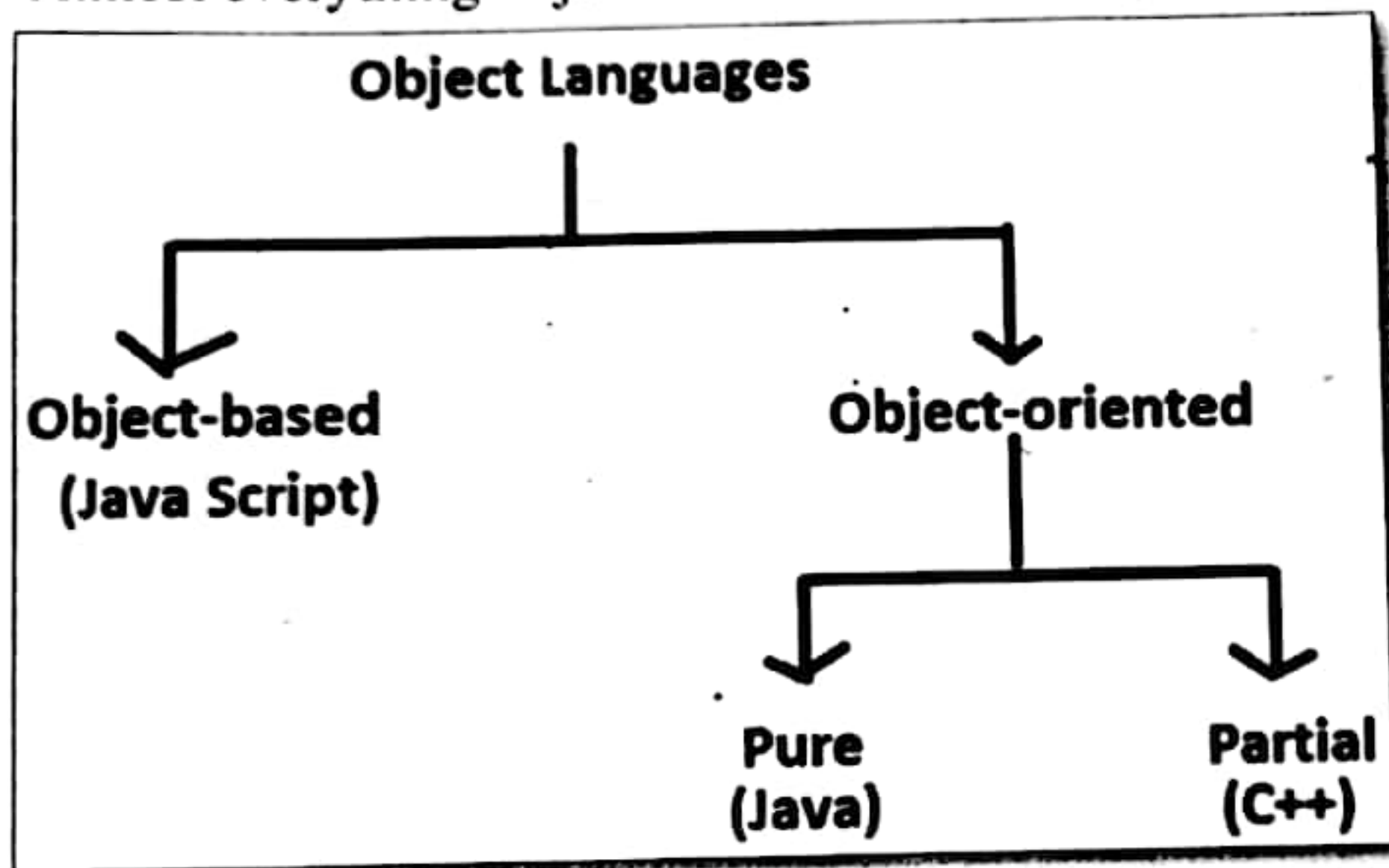
- JavaSE (Java Platform Standard Edition) is used to develop desktop (1-tier) and web applications (2-tier) using java.
- Java Desktop applications are also called as Standalone applications.
- JavaME (Java Platform Micro Edition) is used to develop the software to run the applications on small devices such as mobiles, set-top boxes and so on.
- JavaEE ((Java Platform Enterprise Edition) is used to develop multitier, distributed and server-side applications in java.
- The tools required in JavaME are different from JavaSE and JavaEE.

3. JAVA FEATURES/ BUZZWORDS/ CHARACTERISTICS:

- The different features of java are,
 - i) Object-oriented
 - ii) Compiled and Interpreted
 - iii) Platform Independent
 - iv) Portable
 - v) Robust and Secure
 - vi) Distributed
 - vii) Simple, Small and Familiar
 - viii) Dynamic and Extensible
 - ix) Multithreaded and Interactive
 - x) High Performance and Internet-based

i) Object-oriented:

- Almost everything in java is in terms of an object



- Java is said to be pure object-oriented programming language
- In java, complete program code and data reside within objects and classes.
- It comes with an extensive set of classes arranged in packages.

ii) Compiled and Interpreted:

- Generally a computer language will be either compiled or interpreted but java combines both these approaches.

- Java is a hybrid implementation system.

- In java, the compiler used is javac tool and interpreter used is JVM (Java Virtual Machine)

iii) Platform Independent:

- Java is an operating system independent and processor independent programming language.

- Because after compilation, it generates a mediator code called Bytecode.

- Bytecode is later translated into machine code, so that a Java program execution is started according to the operating system.

- Because of Bytecode, the mission of java is WORA (Write Once Run Anywhere).

iv) Portable:

- Java is portable means it is easy to carry from one system to other system.

v) Robust and Secure:

- Java is robust means it is simple and it has dynamic memory management.

- Java Security is of two types
 - Declarative Security
 - Programmatic Security
- Because of applets, java applications are virus-free applications.
- In J2EE, a concept called JAAS, By using this we can provide security by means of username and password.

vi) Distributed:

- Java is a distributed language.
- Java not only has the ability to share data but also programs.
- This enables many programmers residing at different locations to work on a single project.
- A distributed application contains 3 logics which runs on a network.
 - Presentation Logic
 - Business or Application Logic
 - Data Access Logic
- The most useful distributed technology today is EJB (Enterprise Java Beans).

vii) Familiar, Simple, and Small:

- Java is a familiar language.
- It is modeled on C and C++ languages.
- Java uses several constructs of C and C++.
- In fact, Java is a simplified version of C++.
- Java is not only small but also a simple language.
- Many features of C and C++ are not included in java like pointers, goto statement, operator overloading, multiple inheritance directly,...etc.

viii) Dynamic and Extensible:

- Java is a dynamic language, which is capable of dynamically linking in new class libraries, methods and objects.
- Java also supports extensibility.
- It supports functions written in other languages like C and C++.

ix) Multithreaded and Interactive:

- Java supports multithreading, i.e, it is not necessary for an application to finish one task before starting another.
- If two or more tasks are running currently, then it is called multithreading.
- It is a thread-based multitasking.
- Java 6.0 supports Hyperthreading (Simultaneous Multithreading).
- Java also works on interactive systems like touch screens, virtual reality systems,...,etc.

x) High Performance and Internet-based:

- Because of using Bytecode, java performance is excellent for an interpreted language.
- In order to reduce overhead during runtime, Java architecture is designed carefully.
- Multithreading improved the overall execution speed of Java programs.
- Java is internet-based because it is supported by all kinds of internet servers and resources.
- Active messaging (Chatting) is developed in java by yahoo.com in 1996.
- The internet-based applications in Java are JSP, Java Servlets, and JDBC.

4. APPLICATIONS OF JAVA:

- Applications of OOP beginning to gain importance in many areas.
- The promising areas for application of OOP include,
 - Real-time Systems
 - Simulation and Modeling
 - Object-oriented Databases
 - Hypertext, Hypermedia and Expertext
 - Artificial Intelligence and Expert Systems
 - Neural Networks and parallel Programming
 - Decision Support and Office Automation Systems
 - CIM/ CAD/ CAD Systems

5. INTRODUCTION TO OOP:

- OOP means Object-oriented programming.
- OOP is a programming paradigm
- It deals with the concepts of object to build programs and software applications.
- It is modeled around the real world.
- The world we live in is full of objects.
- Every object has a well-defined *identity*, *attributes*, and *behavior*.
- The features of object-oriented programming also map closely to the real-world features like
 - Inheritance
 - Abstraction
 - Encapsulation
 - Polymorphism

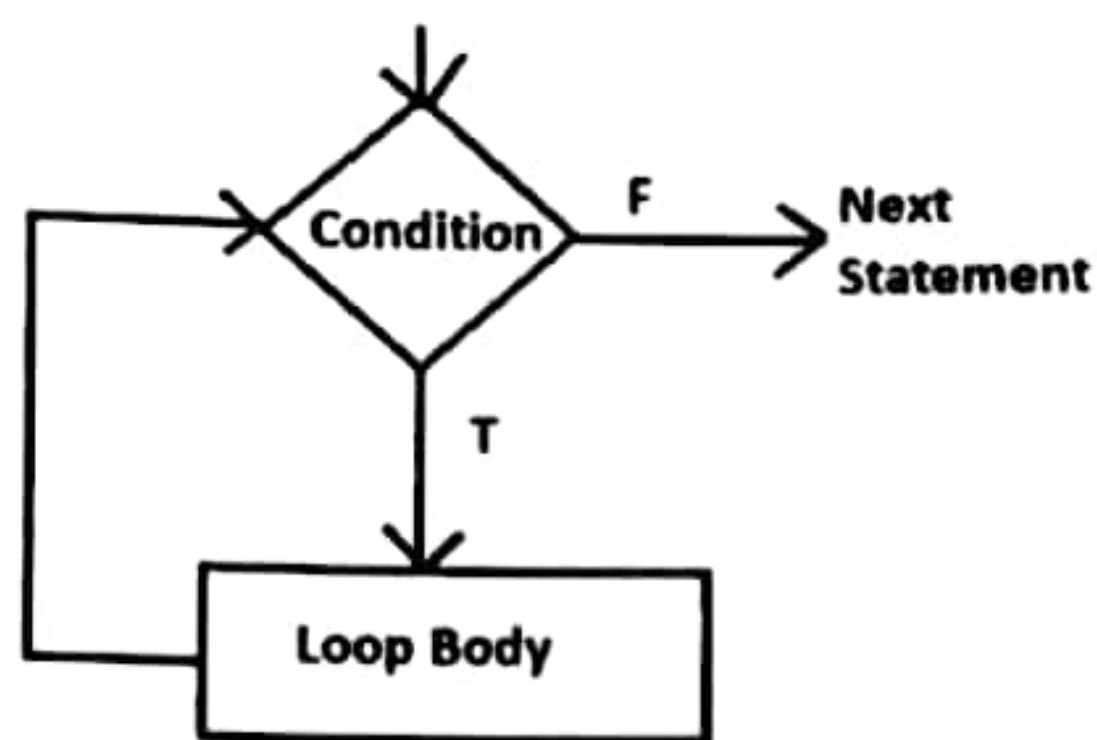
6. PROCEDURAL PROGRAMMING LANGUAGE AND OBJECT-ORIENTED LANGUAGE:

- The Procedural languages create self-contained units called procedures.
- Example languages are C, Fortran, Pascal, Ada
- The Object oriented languages create objects and procedures.
- Example languages are Java, C++, C#, Visual Basic

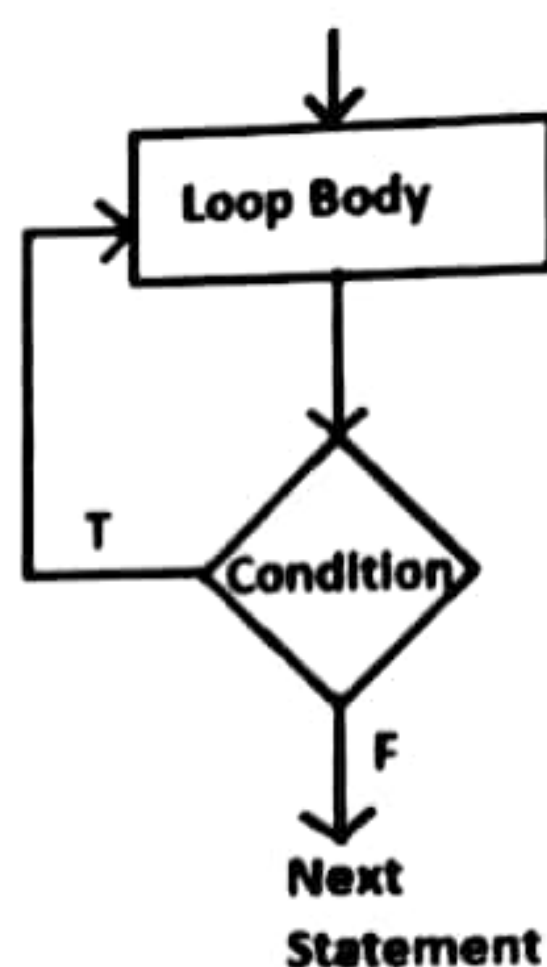
POP Vs OOP:

Procedural Language	POP	OOP
<ul style="list-style-type: none"> ● Separate data from functions that operate on them. 		<ul style="list-style-type: none"> ● Encapsulate data and methods in a class.
<ul style="list-style-type: none"> ● Not suitable for defining abstract types. 		<ul style="list-style-type: none"> ● Suitable for defining abstract types.
<ul style="list-style-type: none"> ● Debugging is difficult. 		<ul style="list-style-type: none"> ● Debugging is easier.
<ul style="list-style-type: none"> ● Difficult to implement change. 		<ul style="list-style-type: none"> ● Easier to manage and implement change.
<ul style="list-style-type: none"> ● Not suitable for larger programs and applications. 		<ul style="list-style-type: none"> ● Suitable for larger programs and applications.
<ul style="list-style-type: none"> ● Analysis and design not so easy. 		<ul style="list-style-type: none"> ● Analysis and design made easier.
<ul style="list-style-type: none"> ● Faster. 		<ul style="list-style-type: none"> ● Slower.
<ul style="list-style-type: none"> ● Less flexible. 		<ul style="list-style-type: none"> ● Highly flexible.
<ul style="list-style-type: none"> ● Data and procedure based. 		<ul style="list-style-type: none"> ● Object oriented.
<ul style="list-style-type: none"> ● Less reusable. 		<ul style="list-style-type: none"> ● More reusable.
<ul style="list-style-type: none"> ● Only data and procedures are there. 		<ul style="list-style-type: none"> ● Inheritance, encapsulation, and polymorphism are the key features.
<ul style="list-style-type: none"> ● Use top-down approach. 		<ul style="list-style-type: none"> ● Use bottom-up approach.
<ul style="list-style-type: none"> ● Only a function call another. 		<ul style="list-style-type: none"> ● Object communication is there.
<ul style="list-style-type: none"> ● Example: C, Basic, FORTRAN. 		<ul style="list-style-type: none"> ● Example: JAVA, C++, VB.NET, C#.NET.

7. Pretest Loops Vs Posttest loops:



Pretest Loops



Posttest Loops

	Pretest loop	Posttest Loop
Initialization	1	1
Number of tests	n+1	n
Loop body executed	n	n
Updating executed	n	n
Minimum iterations	0	1
Types	while, for	do-while

8. Break Vs Continue:

Break	continue
1.exits from the current block or loop	1.loop takes next iteration
2.control passes to next statement	2.control passes to beginning of the loop
3.terminates the program	3.never terminates the program.

9. Counter-Controlled Loops and Condition-Controlled Loops:

a) Counter-Controlled Loops:

- This type of loops will use counter-variable to repeat the statements number of times.

Example:

```
fact=1,n=5,i;
for(i=n;i>=1;i--)
{
    fact=fact*i;
}
System.out.println("Factorial="+fact);
```

b) Condition-Controlled Loops:

- These loops are also called as Event Controlled Loops.

- This type of loops will use condition to repeat the statements number of times.

Example:

```

n=123,rev=0,d;
while(n!=0)
{
d=n%10;
rev=(rev*10)+d;
n=n/10;
}
System.out.println("Reverse="+rev);

```

10. GARBAGE COLLECTOR:

- The Java runtime environment has a garbage collector
- It periodically frees the memory used by objects that are no longer needed.
- The garbage collector runs either synchronously or asynchronously in a low priority daemon thread.
- The garbage collector executes *synchronously* when the system runs out of memory or *asynchronously* when the system is idle.
- The garbage collector can be invoked to run at any time by calling `System.gc()` or `Runtime.gc()`.

Basic approaches used by garbage collectors:

- There are two basic approaches used by garbage collectors are
 - *Reference counting* and
 - *Tracing*.

Reference Counting Approach:

- Reference counting maintains a reference count for every object.
- A newly created object will have count as 1.
- Throughout its lifetime, the object will be referred to by many other object thus incrementing the reference count and as the referencing object move to other objects, the reference count for that particular object is decremented.
- When reference count for a particular object is 0, the object can be garbage collected.

Tracing Approach::

- Tracing technique traces the entire set of objects (starting from root) and all objects having reference on them are marked in some way.
- Tracing garbage collector algorithm popularly known as is *mark and sweep* garbage collector scans Java's dynamic memory areas for objects, marking those objects that are referenced.
- After all the objects are investigated, the objects that are not marked (not referenced) are assumed to be garbage and their memory is reclaimed.

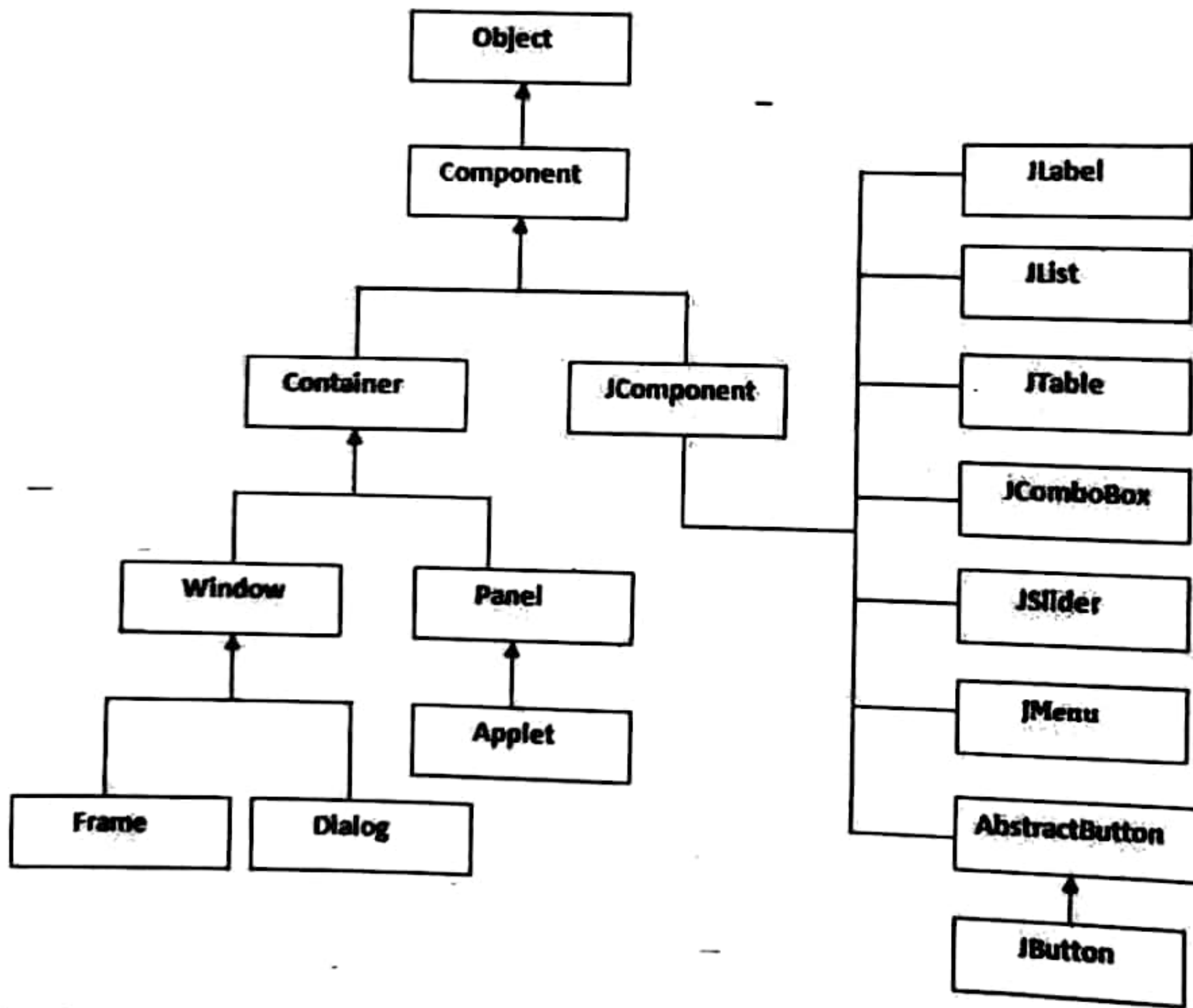
MORE ABOUT JAVA SWINGS:

i) AWT and Swing:

No.	Java AWT	Java Swing
1)	AWT components are platform-dependent .	Java swing components are platform-independent .

2)	AWT components are heavyweight .	Swing components are lightweight .
3)	AWT doesn't support pluggable look and feel .	Swing supports pluggable look and feel .
4)	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT doesn't follow MVC (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing follows MVC .

ii) Hierarchy of Java Swing classes



Simple Java Swing Example:

```

import javax.swing.*;
public class FirstSwingExample {
public static void main(String[] args) {
JFrame f=new JFrame();//creating instance of JFrame
  
```

```
JButton b=new JButton("click");//creating instance of JButton  
b.setBounds(130,100,100, 40);//x axis, y axis, width, height
```

```
f.add(b);//adding button in JFrame
```

```
f.setSize(400,500);//400 width and 500 height
```

```
- f.setLayout(null);//using no layout managers  
f.setVisible(true);//making the frame visible  
}  
}
```

