Page No. :

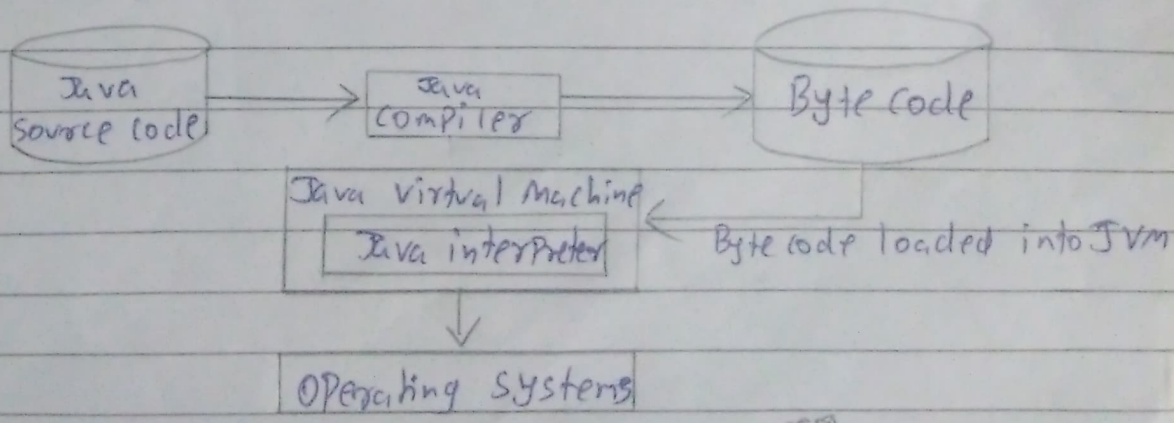
Date .

Q1 Explain Bytecode.

Ans The key that allows Java to solve both the security and portability problems just described is that the output of a Java compiler is not executable code. Rather, it is bytecode.

-> Bytecode is highly optimized set of instructions designed to be executed by the Java runtime system which is called the Java virtual Machine (JVM). In essence, the original JVM was designed as an interpreter for bytecode.

-> Translating a Java program into bytecode makes it much easier to run a program in a wide variety of environments because only the JVM needs to be implemented for each platform. once the runtime package exists for a given system any Java program can run on it.



Page No. :
Date :

Q2 Discuss features of Java.

Ans Following are the features of Java:

* simple

Java has a concise, cohesive set of features that makes it easy to learn and use.

* secure

Java provides a secure means of creating Internet applications

* Portable

Java programs can execute in any environment for which there is a Java runtime system.

* object-oriented

Java embodies the modern, object oriented Programming philosophy.

Page No. :
Date .

* Robust

Java encourage error free programming by being strictly typed and performing run time checks.

* Multithreaded

Java provides integrated support for multithreaded programming

* Architecture neutral

Java is not tied to a specific machine or OS Architecture.

* Interpreted

Java supports cross platform code through the use of Java byte code.

* High performance

The Java Bytecode is highly optimised for speed of execution.

Page No. :
Date .

* Distributed

Java was designed with the distributed environment of the Internet in mind.

* Dynamic

Java programs carry with them substantial amounts of run time information that is used to verify and resolve accesses to objects at runtime.

Page No. :
Date .

Q3 How applets in Java has a profound effect on internet?

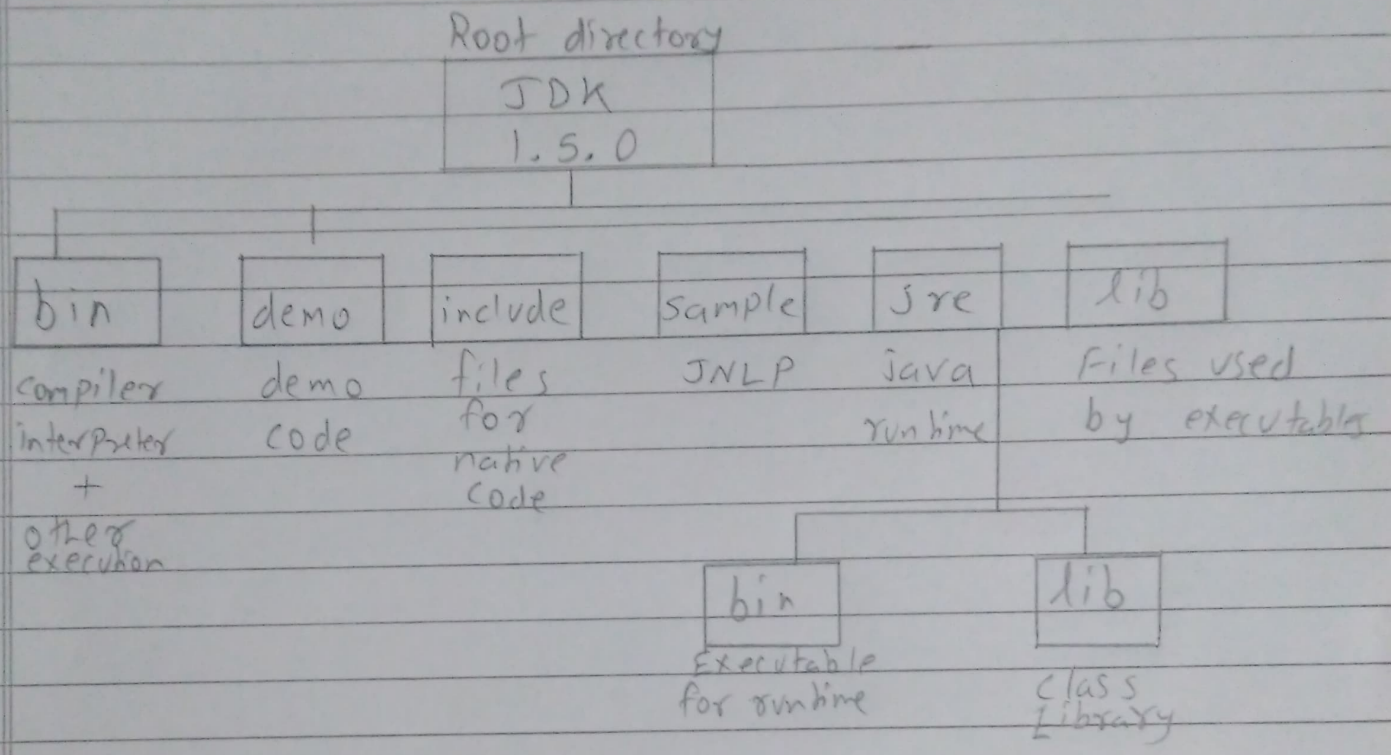
Ans The internet helped catapult Java to the forefront of programming and Java, in turn, had a Profound effect on the internet. In addition to simplifying web-programing in general, Java innovated a new type of networked Program called "applet" that changed the way the online world thought about content.

An applet is a special kind of Java Program that is designed to be transmitted over the internet and automatically executed by a Java-compatible web browser. They are typically used to display data provided by the server, handle user input or provide simple functions that execute locally, rather than on server.

The creation of the applet changed Internet programming because it expanded the universe of object that can move freely in cyberspace. An applet is dynamic self executing program, such a program is an active agent on the client computer yet it is initiated by the server.

Page No. :
Date :

Qn Explain JDK Root directory with diagram.



-> Jdk 1.5.0 is reffered to as the root directory for java it is also reffered to a a java home directory.

-> The sample directory containg the sample application that use JNLP, which is Java Network languaging Protocol that use for executing applications or applets. from a network server without the need for a browser or the need to download and install the code.

Page No. :
Date :

-> Jre directory contains the Java Runtime facilities that are used when you execute a Java Program.

-> The classes in Java libraries are stored in the Jre\lib directory and they all are packed in jre\lib directory.

Q5 Unicode

-> Unicode is a computing standard designed to consistently and uniqly encode characters used in written language throughout the world.

-> Unicode standard is used hexadecimal to express a character.

-> It uses a 16-bit code to represented a character and with 16 bit upto 65,535 (2^{16}) codes to represent a characters.

-> Java also support Unicode internally to represent character and strings each character occupy 2 bytes.

Page No. :
Date :

Unit 2 Datatype, Variables, Constants & loops.

Q1. Difference between '==' and equals() in Java.

Ans -> equals() is a method and '==' is a operator

-> we can use '==' operator for reference comparison and equals() method for content comparison.

-> simple words == checks if both objects point to the same memory location whereas equals() evaluates to the comparison of values in objects

Eg.

```
Public class Test {
        public static void main (String[] args)
        {
            String s1 = new String ("Hello");
            String s2 = new String ("Hello");

            System.out.println (s1 == s2); //fals
            System.out.println (s1.equals(s2)); //tru
        }
}
```

Page No. :
Date :

Q-2 Discuss two types of byte ordering with example. Mention the difference between two.

Ans The two types of byte ordering is

BIG ENDIAN
LITTLE ENDIAN

-> static Byte order BIG-ENDIAN
static Byte order LITTLE-ENDIAN

=> BIG ENDIAN

Constant denoting bing endian byte order. In this order the bytes of a multi-byte value are and ereal in the form must significant to the least significant.

=> LITTLE ENDIAN

Constant denoting little-endian byte order In this order the bytes of multibyte value are ordered from least significant to most significant.

Page No. :
Date :

eg. the 32 bit hex value 0x45679812 would be stored in memory as follows.

Address	00	01	02	03
Little endian	12	98	67	45
Big endian	45	67	98	12

Q3 Why strings are immutable?

Ans. → The key benefits of keeping string class as immutable are caching, security, synchronization and performance.

→ Immutable objects are particularly useful in concurrent applications, since they cannot change state, they cannot be corrupted by thread interference or ~~objec~~ observed in an inconsistent state.

→ string objects are immutable. Immutable simply means unmodifiable or unchanged.

→ once string object is created its data or state can't be changed but a new string object is created.

Page No. :
Date :

Unit 3 Defining classes.

Q-1 Difference between finally, finalize and final keyword.

Ans

Final	Finally	Finalize.
① Final is a keyword	① Finally is a block	① Finalize is a method
② Final is used to apply restriction on class method and variable Final class can't be inherited, Final method can't be overridden and Final variable value can't be changed.	② Finally is used to place important code, it will be executed whether exception is handled or not.	② Finalize is used to perform clean up processing just before is garbage collected.

Q2. Define static method, static variable, static class.

Ans Static variable

When a variable is declared as static, then a single copy of variable is created and shared among all object at class level. Static variable are essentially global variables. All instances of the class

share the same static variable.

* static method

-> static method is method that belongs to a class rather than an instance of a class. The method is accessible to every instance of a class, but methods defined in an instance are only able to be accessed by that member of class.

-> static methods do not use any instance variables of any object of the class they are defined in static methods take all the data from parameters and compute something from those parameter with no reference to variable.

* static class

A class can be made static only if it is a nested class. Nested static class doesn't need a refrence of outer class. In this case, a static class cannot access non-static members of the ~~outer~~ outer class.

Page No. :
Date :

Unit-4

Extending classes and Inheritance.

Q1 Define functional programming

Ans.
- Functional programming is the technique where we bind everything to pure function.
- The approach focuses on functional only ie; each and every task to be performed is within function.
- The functional programming uses expressions instead of statements.
- These expressions are evaluated to produce a value.
- Programs implemented using functional programming are easy to debug as they have no side effects or hidden input/output.
- Java uses Lambda expressions to enable functional programing.

Example :

```
interface Drawable {
        public void draw();
}

Drawable d = () -> System.out.println(
                "Drawing...");
d.draw();
```

Page No. :
Date :

Q2. Discuss Lamba expression and its implementation in Java.

Ans.
- The lambda expression introduces a new syntax element and operator into the Java language the new operator, sometimes referred to as the lambda operator or the arrow operator is "→".
- It divides a lambda expression into two parts. The left side specifies any parameters required by the lambda expression on the specifies the actions of the lambda expression.
- one type consists of a single expression and other type consists of a block of code.
- syntax - (argument list) → (body)
- Use of the lambda expression
  1. To provide the implementation of functional interface
  2. less coding

Page No. :
Date :

Example :

```
interface Drawable {
        public void draw();
}

public class LambdaExpression {
    public static void main (String[] args)
    {
        int width =10;
        Drawable d = new Drawable(){
            public void draw() {System.out.
                println("Drawing"+width);}
            );
            d.draw();
        }
    }
```

→ Code with Lambda

```
interface Drawable{
        public void draw();
}
public class LambdaExpression {
    public static void main(String[] args){
        int width =10;
        Drawable d = () -> (
            System.out.println("Drawing: "+
                width);
        );
        d.draw();
    }
}
```

Page No. :
Date :

Unit-5 Generics in Java

Q Discuss the benifit of generic over non-generic types.

Ans Code that uses generics has many benifits over non-generics code:

- Stronger type checkes out compile time.

A Java compiler applies strong type checking generic code and issues error if the code violates type safely. fixing compile time errors is easier than fixing runtime errors which can be difficult to find.

- Elimination of casts:

→ code snippet without generic requiring casting:

```
List list = new ArrayList();
list add ("hello");
String s = (String) list.get(a);
```

→ When re-written to use generics, the code does not requiring casting:

Page No. :
Date :

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);
```

→ Enabling programmers to implement generic algorithm

By using generics, Programmer can implement generic algorithms that work on collection of different types, can be customized and are type safe and easier to read.

Q. What is erasure with reference to generics.
Ans Generics are implemented in java is in order.

An important constrainst that governed the way generics were added to java was the need for compatibility with previous versions of java. Simply put generic code had to be compiled with preexisting, non generic code. Thus any changes to the syntax of java language, or to the JVM had to avoid breaking older code. The way java implements generic. While satisfying this constraint is through the use of erasure.

In general, here is how erasure work, when your java code is compiled, all generic type information is removed. This means

Page No. :
Date :

replacing type parameters with their bound type, which is object if no explits bound is specified and then applying the appropriate casts to maintain type compatibility with the types specified by the type arguments. The compiler also enforces this type compatibility. This approach to generics means that no type parameters exist at runtime. They are simply a source-code mechanism.

Q write shortnote on:

(1) Generic Constructors:

A constructor can be generic even if its class is not for eg in the following program, the class summation is not generic but its constructor is.

e.g.
```
class summation {
    private int sum;
    <T extends number> summation (T arg) {
        sum = 0;
        for (int i=0; i<=arg.intvalue(); i++)
            sum += i;
    }
    int getsum () {
        return sum;
    }
}
```

Page No. :
Date :

```
class genconsdemo {
    public static void main (string args[]){
        summation ob = new summation (4.0);
        system.out.println ("summation of 4.0
            is " + Ob.getsum());
    }
}
```

The Summation class computes and encapsulate the summation of the numeric value passed to its constructor. Recall that the summation of N is the sum of all the whole number between 0 and N. Because summation() specifys a type parameters that is bounded by number, a summation object can be constructed using any numeric type, including Integer, Float or Double. No matter what numeric type is used, its value is converted to Integer by calling intvalue() and the summation is computed. Therefore is not necessary for the class summation to be generic: only a generic constructor is needed

② Generic Interface

Generics also work with interface. Thus you can also have interface. Generic interface are specified just like generic classes for example.

Page No. :
Date :

```
interface containment <T> {
        boolean contains (T o);
}

Class Myclass <T> implements containment<T>{
        T [] arrayRef;
        Myclass (T[] o) {
            arrayRef=o;
        }

        Public boolean contain (T o) {
            for (T x. arrayRef)
                if(x.equals(o)) return true;
            return false;
        }
}

Class GenIfdemo{
    Public static void main (string args[]){
        Integer x = {1,2,3};
        My Class <Integer> ob = new Myclass
                    <Integer>(x);
        if (ob. contains (2))
                System. out.println ("2 is in ob");
        else
            system.out.println ("2 is not in ob");
```

```
        if (ob.contains(s))
            System.out.println("s is in ob");
        else
            System.out.println("s is not in ob");
    }
}
```

//output :

```
2 is in ob
s is not in ob.
```

③ Generic Functional Interface.

A lambda expression can't specify type parameters. so if its not generic, however functional interface associated with lambda expression is generic. In this case, the target type of lambda expression has determined by the type of arguments specified when functional interface reference is denied

* Syntax :

```
interface SomeFunc {
    T func(T t);
}
```

Page No. :
Date :

```
eg.  interface MyGeneric<T>{
         T compile(T t);
     }

Public class Generic & interface{
    Public static void main(String args[])
    {
        MyGeneric<String> reverse (str)->{
            // lambda expression
            String result = "";
            for(int i = str.length()-1; i>=0;i--)
                result t = str.charAt(i);
            return result;
        };

        MyGeneric<Integer> ~~fun~~ factorial =
            (Integer n) -> {
            // lambda expression
            int result = 1;
            for (int i=1; i<=n; i++)
                result = i * result;
            return result;
        };

        System.out.println(reverse.compute("lambda
                Generic functional interface");
        System.out.println(factorial.compute(5
    }
}
```

Page No. :
Date :

// output

ecafrenti lanoitcnuf cireneh adabmal
soho.

(4) Generic methods

Generic methods are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameters scope is limited to the method where it is declared static and non-static generic methods are allowed, as well as generic class constructors.

The syntax for a generic method includes a list of Type parameters, inside angle Methods return type. for static generic methods the type parameter section must appear before the method's return type.

The util class includes a generic method compare which compares two Pair objects.

```
Public class Util {
    public static <K, V> boolean compare(Pair
        <K, V>, P1, Pair <K, V> P2) {
    return P1.getKey().equals(P2.getKey())
        && P1.getValue().equals(P2.getValue());
    }
}
```

Page No. :
Date :

```
Public class Pair <K, V> {
    Private K key;
    Private V value;

    Public Pair (K key, V value) {
        this.key = key
        this.value = value;
    }

    Public K getkey() {
        return key;
    }
    Public V getvalue() {
        return value;
    }
}
```

→ The complete syntax for invoking this method would be :

```
Pair<Integer, String> P1 = new Pair<>(1,
                                "Apple");
Pair<Integer, String> P2 = new Pair<>(2,
                                "Pear");

boolean Same = Util.<Integer, String>
                compare (P1, P2);
```

Page No. :
Date :

Unit - 6 Comparators and Lambda expression

Q1 Discuss comparator & comparable

Ans

Comparator:

-> A comparator interface is used to order the objects of a specific class. This interface is found in java.util. Package.

-> It contains two methods
- Compare (object obj1, object obj2)
- equals (object element)

-> The first method, compare (object obj1, object obj2) compares its two input arguments & showcase the output. It returns a negative integer, zero, or a positive integer to state whether the first argument is less than, equal to, or greater than the second.

-> The second method, equals (object element), requires an object as a parameter & shows if the input object is equal to the comparator. The method will return true, only if the mentioned object is also a comparator. The order remains the same as that of the comparator.

Page No. :
Date :

Comparable :

-> Comparable is an interface which defines a way to compare an object with other objects of the same type. It helps to sort the objects that have self-tendency to sort themselves.

-> The Object must know how to order themselves

-> Eg. Roll No, age, salary...

-> This Interface is found in java.lang package and it contains only one method compareTo().

-> Comparable is not capable of sorting the objects on its own, but the interface defines a method int compareTo() which is responsible for sorting.

Page No. :
Date :

Q2 List at least 5 predefined Functional Interface?

Ans Functional Interface:

-> In java an Interface that contain only one single abstract method.

-> A functional Interface can contain default & static methods which do have an implementation.

Predefined Functional Interface:

-> In many cases, however you won't need to define your own functional Interface because JDK 8 adds a new package called java.util.function that provides several predefined ones.

Interfaces that are predefined:

(1) Unary operator

-> Apply a unary operator to an object of type T & return the result, which is also of type T. Its method is called apply().

Page No. :
Date :

(2) Binary Operator <T> :

→ Apply an operation of 2 objects of type T & return the result. Its method is called apply().

(3) Consumer<T> :

→ Apply an operation on an object of type T. Its method is called accept().

(4) Supplier<T> :

→ Return an object type T, Its method is called get()

(5) Function <T, R> :

→ Apply an operator to an object of type T and Return the result as an object of type R, its method is called apply().

Page No. :
Date :

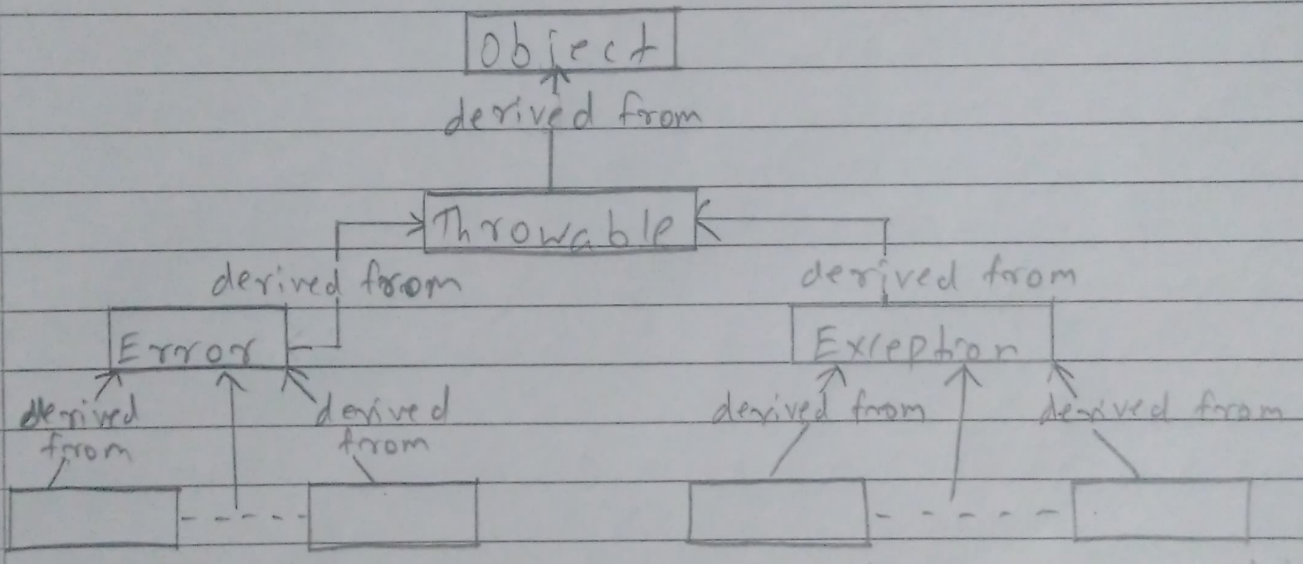
Unit - 7 Exceptions

Q1 Discuss Exception & types of Exceptions in Java.

Ans.

Two direct subclass of class Throwable - the class Error & class Exception - cover all the standard Exception.

Hierarchy to which these classes belong.



Error Exception

Error has three direct subclasses:
ThreadDeath, LinkageError, VirtualMachine Error.

Page No. :
Date :

⇒ Thread Death : A ThreadDeath exception is thrown whenever an executing thread is deliberately stopped & for the thread to be destroyed properly, you should not catch this exception.

⇒ Linkage Error : The Linkage Error exception class has subclasses that record serious error with the classes in program. Incompatibilities between classes or attempting to create an object of a non existent class type are the sort of things that cause these exception to be thrown.

⇒ Virtual Machine Error : This class has four subclasses that specify exceptions that will be thrown when a catastrophic failure of JVM occurs.

Runtime Exception Exception

Runtime Exception are treated differently because of serious error in your code.

→ quite a lot of subclasses of Runtime Exception are used to signal problems in various packages in Java class library.

Page No. :
Date :

Subclasses of Runtime Exception defined in standard Package java.lang are.

(i) Arithmetic Exception : An invalid Arithmetic condition has arisen, such as an attempt to divide an integer value by 0.

(ii) Index out of Bound Exception : To use an index that is outside the bounds of object it is applied to. This may be an array, a string object or vector object.

(iii) Negative Array size Exception : To define an array with a negative dimension.

(iv) Null Pointer Exception : An object variable containing null is used, when it should refer to an object for proper operation.

(v) Arraystore Exception : To Store an array that isn't permitted for the array type.

Page No. :
Date :

Q2 Explain commonly used methods defined by Throwable.

Ans All exception are subclasses of Throwable. So all Exception support the methods defined by Throwable.

⇒ Commonly used method defined by Throwable

1. Throwable fillInStackTrace() : Return a Throwable object that contain a completed ~~trace~~ stack trace. This object can be rethrown.

2. String getLocalized message() : Returns a localized description of Exception.

3. String getMessage() : This Returns the contents of message describe the current exception. This will typically be fully qualified name of the exception class

4. Void printStackTrace() : This will o/p the message & the stack trace of standard error output stream which is screen in case of console program.

5. Void printStackTrace (PrintStream s) : This is same as previous method except that can

specify the output stream as an argument calling previous method for an exception object e is equivalent to:

e.printStackTrace(System.error);

6. void printStackTrace(PrintWriter stream)
Sends the stack trace to the specified stream

7. String toString() : Returns a String object containing a complete description of exception. This method is called by println() when outputting a Throwable object

Program demonstrate these methods:

```
class ExcTest{
    static void genException(){
        int nums[] = new int[4];
        System.out.println("Before Exception");
        nums[7] = 10;
        System.out.println("Won't be displayed");
    }
}

class UseThrowableMethods{
    public static void main(String args[]){
        try{
            ExcTest.genException();
        }
```

Page No. :
Date :

```
catch (ArrayIndexoutofBoundsException ex)
{
    System.out.println("Standard message is:");
    System.out.println(e+c);
    System.out.println("In stack trace:");
    ex1.printStackTrace();
}
System.out.println("After catch");
}
}
```

Page No. :
Date :

Unit-10 The java.io Package.

Q1 Difference between Byte streams & character streams

Byte stream	character stream
Byte streams provide a convienient means for handling input & output of Bytes	character stream are designed for handling the input and output of character.
They are used for when reading or writing binary data	They use Unicode and therefore can be internationalized.
All byte streams are descended from InputStream & OutputStream.	All character stream classes are descended from Reader & writer.
Performs input & output operations of 8 bit bytes	Performs input & output operation of 16 bit unicode.
eg. Images, sounds etc.	eg. plain text, files, Web-pages, user keyboard input etc.

Page No. :
Date :

Q2 Explain 7 different subclasses of Input stream.

Ans ① Byte Array Input Stream :

Contains an internal buffer that contains bytes that may be read from the stream.

② File Input stream :

Obtains input bytes from a file in a file system.

③ Filter Input Stream :

Contains some inputstream, which it uses as its basic source of data, possibly transforming the data along the way or provided additional functionality.

④ Object Input stream :

Deserializes primitive data & objects previously written using an object Output Stream.

Page No. :
Date :

⑤ String Buffer Input stream :

Allows an application to create an input stream in which the bytes read are supplied by the contents of a string.

⑥ Buffered Input Streams.

Adds functionality to another inputstream, namely the ability to buffer the input and to support the mark & reset methods.

⑦ Data Input stream.

lets an application read primitive data types from an underlying input stream in a machine dependent way.

Page No. :
Date :

Unit - II Threads

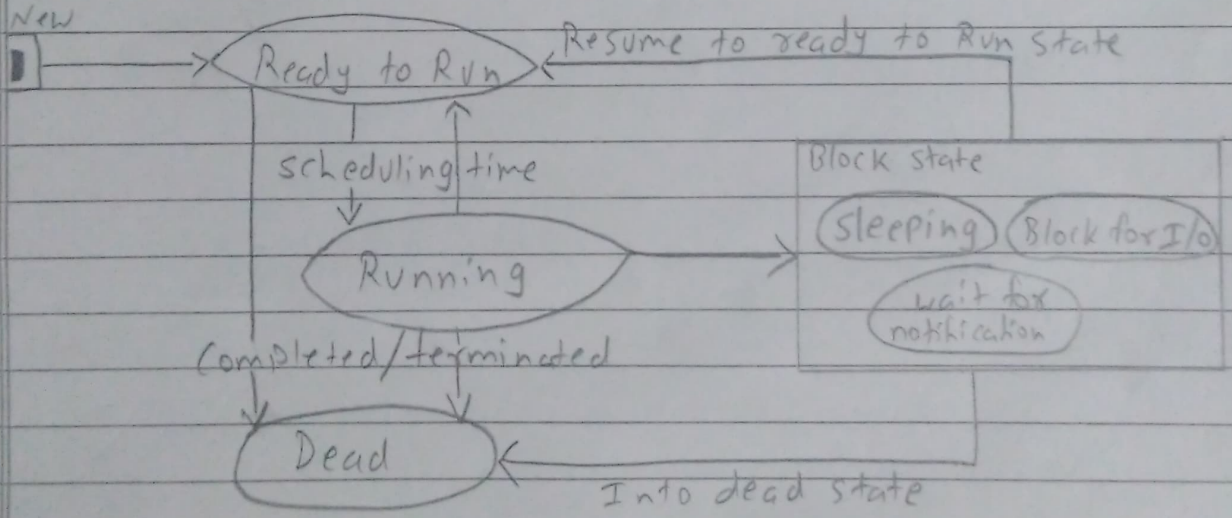
Q1 Explain Thread life-cycle with diagram

Ans

A thread passes through several states throughout its life cycle & after completing its task, it is dead.

The different states of a thread are as follows:

① New
② Ready to Run
③ Running
④ Block
⑤ Dead.



① New: At the time of thread creation, it is in the New state By calling the start() method the thread will start its execution

Page No. :
Date :

② Ready to run: After the start() method, the thread is in ready to run state. It means that now the thread is ready for CPU allocation but still the CPU is not allocated it. So, we can say that the thread is runnable but not running.

③ Running: By calling run() method the CPU time is allocated to the thread and it enters the running state. In running state, the thread executes its task for which the thread was created. After the CPU time slice expires, the thread may go back to the ready to run state if its task is uncompleted or may go to the dead state, if its tasks is completed or may go to the block state.

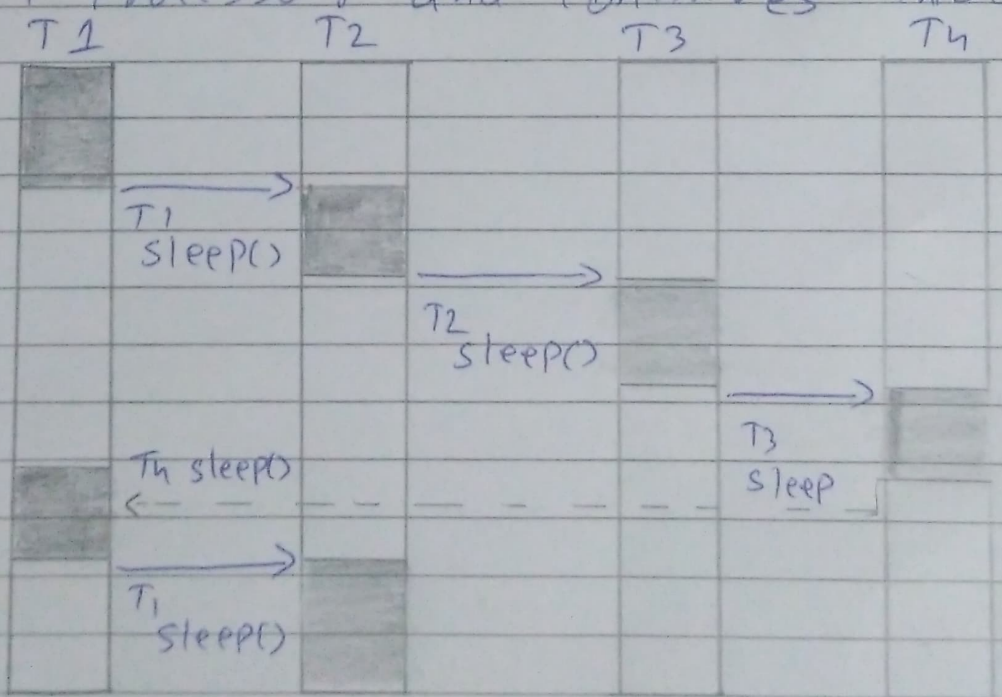
④ Block: A thread may enter the block state for several reasons. For example, the thread itself or some other thread may invoke the sleep() method to wait for an I/O operation to finish. A block thread may enter the dead or the ready to run state after its block state is complete.

⑤ Dead: A thread is dead if it completes the execution of its run() method or if its stop() method is invoked.

Page No. :
Date :

Q2 Discuss Thread scheduling

Ans The scheduling of threads depends on your Operating System to some extent, but each thread will certainly get a chance to execute with the other threads are asleep that is, when they have called this sleep method. If your operating system uses premptive multitasking, the program will work without the call to sleep method in the run() method. However if the OS doesn't schedule in this way without the sleep() call in run(), the first thread will hog the Processor and continues indefinitely.



Page No. :
Date :

- There's another yeild() method, defined in Thread class, that gives other threads a chance to execute. You would use that if you want other threads to have a looking if they are waiting, but you don't want to suspend the execution of current thread for a specific period of time. When you call sleep method for a thread, the thread will not continue for at least the time you have specified as an argument, even if no threads are waiting. Calling yeid() on the other hand, causes the current thread to resume immediately if no threads are waiting.

Q3 Discuss interprocess communication.

Ans Consider the following situation. A thread called T is executing inside a synchronised method and needs access to a resource called R that is temporary unavailable. What should T do? If T enters some Polling loop that waits for R, T ties up the object, preventing other threads to access it. A solution is to have T temporarily relinquish the control of the object, allowing another thread to run when R become available. T can be notified and resume execution. Such an approach

relies upon some form of inter thread communication in which one thread can notify another that is blocked and can be notified that it can resume execution. Java supports inter thread communications with the wait(), notify() and notifyAll() methods.

- The wait(), notify() and notifyAll() methods are part of all objects because they are implemented by object class. These methods should be called only from within a synchronized context.

- When a thread is temporarily blocked from running, it calls wait(). This causes the thread to go to sleep. and monitor for the object to be released, allowing other threads to use the object.

- At later point the sleeping thread is awakened when some other thread enters the same monitor and calls notify() or notifyAll().

- A call to notify() resumes one waiting thread. A call to notifyAll() notifies all threads, with the highest priority thread gaining access to the object.