Java - Notes - Chapter 1 - The Java Programming Environment

+ Java language is both a technology and a platform.

+ Language:

- Programming language is a notation to write program.
  e.g. C, C++, C# etc.
- Every programming language has following elements
  * Syntax
  * Semantics
  * Type System [Data types]
  * Operator Set
  * Built in features
  * Standard library and runtime system.
- Java has all above elements hence Java is considered as programming language.

+ Technology :

- Using language and technology we can develop the application.
- Let's consider example of asp.net. Asp.net helps us to create Web application and web services so it is classified as technology but it is not a language. To implement it we should use C# as a programming language.
- Using java, it is possible to create different types of application hence it is considered as technology.

+ Platform :

- A platform is the hardware or software environment in which a program runs.
- Platforms can be described as a combination of the operating system and underlying hardware(e.g MS Windows, Linux, Solaris OS, Mac OS etc) or software only platform(e.g MS.NET, Java).
- The Java platform has two components:
  * Java virtual machine[JVM]
  * Java application programming interface[Java API].

+ History of Java :

- Java goes back to 1991, when a group of Sun engineers, led by Patrick Naughton and James Gosling, wanted to design a small computer language that could be used for consumer devices like cable TV switchboxes. Since these devices do not have a lot of power or memory, the language had to be small and generate very tight code. Also, as different manufacturers may choose different CPU's, it was important that the language not be tied to any single architecture. The project was code-named "Green"

- The requirements for small, tight, and platform-neutral code led the team to design a portable language that generated intermediate code for a virtual machine. The Sun people came from a UNIX background, so they based their language on C++.

- Gosling decided to call his language The people at Sun later realized that Oak was the name of an existing computer language, so they changed the name to Java.

- In 1992, the Green project delivered its first product, called "*7."It was an extremely intelligent remote control. Unfortunately, no one was interested in producing this at Sun, and the Green people had to find other ways to market their technology. However, none of the standard consumer electronics companies were interested either. The group then bid on a project to design a cable TV box that could deal with emerging cable services such as video-on-demand. They did not get the contract.

- The Green project spent all of 1993 and half of 1994 looking for people to buy its technology. No one was found.

- First Person was dissolved in 1994. While all of this was going on at Sun, the World Wide Web part of the Internet was growing bigger and bigger. The key to the World Wide Web was the browser translating hypertext pages to the screen. In 1994, most people were using Mosaic, a noncommercial web browser that came out of the supercomputing center at the University of Illinois in 1993.

- In the SunWorld interview, Gosling says that in mid-1994, the language developers realized that we could build a real cool browser. It was one of the few things in the client/server mainstream that needed some of the weird things we'd done: architecture-neutral, real-time, reliable, secure-issues that weren't terribly important in the workstation world. So we built a browser

- The actual browser was built by Patrick Naughton & Jonathan Payne and evolved into the HotJava browser, which was designed to show off the power of Java. The builders made the browser capable of executing Java code inside web pages. This "proof of technology was shown at SunWorld on May 23, 1995, and inspired the Java craze that continues today.

+ Java Platforms:

- Java is not specific to any processor or operating system as Java platforms have been implemented for a wide variety of hardware and operating systems with a view to enable Java programs to run identically on all of them. Different platforms target different classes of device and application domains:

- **Java Card:** A technology that allows small Java-based applications to be run securely on smart cards and similar small-memory devices.

- **Java ME (Micro Edition) :** Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. It is often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.

- Java SE (Standard Edition) : Java Platform, Standard Edition or Java SE is a widely used platform for development and deployment of portable code for desktop environments

- Java EE (Enterprise Edition) : Java Platform, Enterprise Edition or Java EE is a widely used enterprise computing platform developed under the Java Community Process. The platform provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

JDK and JRE :

- SDK is software development kit which is required to develop application.
- SDK = software development tools + documentation + [libraries + runtime environment].
- JDK is java platforms SDK. It is a software development environment used for developing Java applications and applets.
- JDK - java tools + java docs + JRE.
- JRE - Java API (java class libraries) + Java virtual machine(jvm).
- All core java fundamental classes are part of rt.jar file. JVM and "rt.jar" is integrated part of JRE.
- To develop application it is necessary to have both i.e. JDK and JRE on developer's machine.
- To deploy java application on client's machine, it is necessary to install only JRE on client's machine.

+ **Java SE Naming and Versions:**

- The Java Platform name has changed a few times over the years.
- Java was first released in January 1996 and was named Java Development Kit, abbreviated JDK.
- Version 1.2 was a large change in the platform and was therefore rebranded as Java 2. Full name: Java 2 Software Development Kit, abbreviated to Java 2 SDK or J2SDK.
- Version 1.5 was released in 2004 as J2SDK 5.0 â€“dropping the "1."from the official name and was further renamed in 2006. Sun simplified the platform name to better reflect the level of maturity, stability, scalability, and security built into the Java platform. Sun dropped the "2" from the name. The development kit reverted back to the name "JDK" from "Java 2 SDK". The runtime environment has reverted back to "JRE" from "J2RE."

- JDK 6 and above no longer use the "dot number"at the end of the platform version.

+ **Directory structure of JDK :**

- "/usr/lib/jvm/java-8-openjdk" is a installation directory of JDK in Ubuntu. Following is the directory structure of JDK.

Directory structure	Description
openjdk	[The name may be different].
|	
|- bin	The compiler and other java tools.
|- docs	Library documenation in HTML format.
|- include	Files for compiling native methods.
|- jre	Java runtime environment files.
|- lib	Library files.
|- src	The library source code[Extract src.zip].
|- man	

+ Simple Hello World Application:

- Let us write simple "Hello World" application using editor[e.g vim/gedit/Leafpad].

```
-       class MyProgram
        {
                public static void main( String[] args )
                {
                        System.out.println("Hello World.");
                }
        }
```

- Steps to Compile and execute java application.

```
export PATH=/usr/bin/                   //To locate java tools

javac -d . MyProgram.java               //output : MyProgram.class

export CLASSPATH=.;                     //To locate .class file

java MyProgram                          //To execute java app
```

- In the Java programming language, all source code is first written in plain text files ending with the .java extension. Those source files are then compiled into .class files by the javac compiler. A .class file does not contain code that is native to your processor; it instead contains byte codes - the machine language of the Java Virtual Machine (Java VM). The java launcher tool then runs your application with an instance of the Java Virtual Machine.

+ Bytecode :

- java compiler is responsible for generating bytecode.
- It is architecure neutral code. Bytecode is also called as a virtual or managed code. - Using "javap -c"command we can view bytecode.
- Bytecode is an object oriented assembly language code which can be understand by the java virtual machine [JVM].

+ Path and Classpath :

- PATH is an operating systems environment variable which is used to locate "javac" file.

- CLASSPATH is a java platforms environment varibale which is used to locate ".class" file.

+ Let us seperate .java file from .class file. Consider the following directory structure.

```
- Complex[ dir ]*
 |
 |- src[ dir ]
 |       |
```

```
|       |- MyProgram.java
|- bin[ dir ]
|
```

- Now, let us compile and execute MyProgram.java file from Complex directory.

```
export PATH=/usr/bin/;

javac -d ./bin/ ./src/MyProgram.java

export CLASSPATH=./bin/;

java MyProgram
```

- After compilation MyProgram.class file will be stored in bin folder. See the following directory structure.

```
Complex[ dir ]*
|
|- src[ dir ]
|       |
|       |- MyProgram.java
|- bin[ dir ]
|       |
|       |- MyProgram.class
|
```

+ Java 8 New Features:

- The newest version of the Java platform, Java 8, was released more than a year ago.There are very few good reasons to do this, because Java 8 has brought some important improvements to the language.

- There are many new features in Java 8.

1. Lambda expressions.
2. Stream API for working with Collections.
3. Asynchronous task chaining with CompletableFuture.
4. Java Date and Time API.

Authors of java have written an influential "White Paper" that explains their design goals and accomplishments.

+ Simple :

- Since syntax of java is simpler than syntax of C/C++, Java programming language is considered as simple.
- Consider the following points:
  * In Java, there is no need for header files.
  * It does not support pointer and pointer arithmetic.
  * It does not structure and union.
  * It does not support friend concept, operator overloading and virtual base class.
  * It does not support multiple [implementation] inheritance.
  * It supports new operator but not delete operator.
  * There is no concept of declaration and definition separately.

- Another aspect of being simple is being small. One of the goals of java is to enable the construction of software that can run stand-alone in small machines. The size of the basic interpreter and class support is about 40K; the basic standard libraries and thread support add another 175K.

+ Object Oriented:

- Since java supports all major and minor pillars of oops, it is considered as object oriented programming language.
- Java programming language was influenced from its previous successors programming language like C++. Java developers did not just took everything and implemented in Java but they analyzed the current challenges in the existing language and then included what is necessary.
- The object oriented model in Java is simple and easy to extend and also the primitive types such as integers, are retained for high-performance.

+ Robust :

- Following features of Java make it Robust.
  * Platform Independent: Java program are written once and executed on any platform this makes the job of developer easier to develop programs and not code machine dependent Coding.

  * Object Oriented Programming Language: helps to break the complex code into easy to understand objects and manage high complexity programs in distributed team environment.

  * Memory Management: In traditional programming language like C, C++ user has to manage memory by allocating and deallocating memory which leads to memory leaks in the program. In Java, memory management is taken care by the Java Virtual Machine and safe from memory crashes. All the allocation and clean of the memory is done automatically.

* Exception Handling: In Java, developers are forced to handle the exception during the development time. All the possible exception are errored out during the compilation of the program. This way when the exception happens during runtime there is proper exception handling mechanism already coded in the program.

+ Architecture-Neutral :

- The major challenge when Java was developing is to have programming language with which a program can be developed and executed anytime in future. With changing environments of hardware, processor, Operating system there was need to have program still adopt to this architecture changes
- Java code does not depend on the underlying architecture and only depends on it JVM thus accomplish the architecture neutral programming language.
- MS.NET is single platform for multiple technology and Java is single technology for multiple platforms.

+ Portable :

- Java programs are not dependant on underlying hardware or operating system. The ability of java program to run on any platform makes java portable.
- Number of datatypes and their size is constant / same on all the platforms which makes java portable.
- Since java is portable java do not support sizeof operator.

+ Secure :

- When Java programs are executed they donâ€™t instruct commands to the machine directly. Instead Java Virtual machine reads the program (ByteCode) and convert it into the machine instructions. This way any program tries to get illegal access to the system will not be allowed by the JVM. Allowing Java programs to be executed by the JVM makes Java program fully secured under the control of the JVM.

+ High Performance:

- Performance of java is slower than performance of C and C++.

- When java programs are executed, JVM does not interpret entire code into machine instructions. If JVM attempts to do this then there will huge performance impact for the high complexity programs. JVM was intelligently developed to interpret only the piece of the code that is required to execute and untouched the rest of the code.

- Just In Time compiler (JIT) has taken place of interpreter which helps to increase performance of the application.

+ Multithreaded :

- If we want to utilize hardware resources(e.g CPU)efficiently then we should use threrad.
- Since java supports multithreading, it is easy to write program that responds to the user actions and helps developers to just implement the logic based on the user action instead to manage the complete multi-tasking solution.

+ Dynamic :

- Methods of the class are by default considered as a virtual.
- Java is designed to adapt to an evolving environment hence it is truly dynamic programming language.
- Java programs access various runtime libraries and information inside the compiled code (Bytecode). This dynamic feature allows to update the pieces of libraries without affecting the code using it.

+ Distributed :

- Due tot RMI(Remote Method Invocation), java is considered as a distributed.
- using RMI, program can invoke method of another program across a network and get the output.

+ Coding Conventions:

- Camel case naming convention: In this case of naming convention, first character of each word except first word should be capital.

e.g parseInt.

- Use "camel case" naming convention for following elements

* fields
* method
* method parameter
* local variable

- Pascal case naming convention : In this case of naming convention, first character of each word including first word should be capital.

e.g StringBuilder.

- Use "pascal case" naming convention for following elements.

* Type name[Interface/class/enum].
* File name.

- package name must be in lower case.
- Constant must be in upper case only.

+ Simple Java Program :

- Java is pure object oriented programming language so you can not write anything global. In C/C++ we write main function globly but in Java it must be member of class. For method names java follows camel case naming convention so name of entry point method is in lowercase.
- Consider the following code:

```
import java.lang.*;     //Optional to import
class Program
{
        public static void main( String[] args )
        {
                System.out.println("Hello SunBeam");
        }
}
```

+ Characteristics of entry point method :

- main is considered as entry poit method in java.
- syntax : â€œpublic static void main(String[] args)â€• .
- JVM invokes main method.

- we can overload main method in java.
- per class we can write entry point method.

+ java.lang.System class :

```
package java.lang;
import java.io*;
public final class System extends Object
{
        //Fields
        public static final InputStream in;
        public static final PrintStream out;
        public static final PrintStream err;
        //Methods
        public static Console console();
        public static void exit(int status);
        public static void gc();
        public static void loadLibrary(String libname);
        ....
}//end of System class.
```

+ System.out.println :

- System : It is a final class declared in java.lang package and java.lang package is declared in rt.jar file.
- Out : It is an object of PrintStream class and. Out is declared as public static final field in System class.
- Println : It is non static overloaded method of PrintStream class.

+Java.io.PrintStream class :

```
package system.io;
public class PrintStream extends ....
{
        //Constructors
        public void flush();
        public void print(String s);
        //Other overloaded print methods
        public void println(String x);
```

```
        //Other overloaded println methods
        public PrintStream format( String format, Object... args );
        public void close();
    }//end of class
```

+ Comments :

- If we want to maintain documentation of source code then we should use comments.
- Java supports three types comments
  - * //Single line comment.
  - * /* Multi line comments */
  - * /** Documentation comments */
- If we want to generate java code documentation using java source code then we should use
- javadoc tool. It comes with jdk. using javadoc we can generate documentation in HTML format.

+ Data Types :

- Data type describes three things:
  1. How much memory is required to store the data
  2. Which kind of data memory can hold
  3. which operation we can perform on the data.
- Java is strictly as well as strongly type checked language. In java, data types are classified as:
  1. primitive / value types
  2. non primitive / reference types

- Value Types :

Type	Size	Default value
boolean	undefined	false
byte	1 byte	0
char	2 bytes	'\u0000'
short	2 bytes	0
int	4 bytes	0
long	8 bytes	0L
float	4 bytes	0.0f
double	8 bytes	0.0d

- Wrapper Class:

- Java has designed classes corresponding to every primiitive type. It is called as wrapper class.
- All wrapper classes are final classes i.e we can extend it.
- All wrapper classes are declared in java.lang package.
- Consider the hierarchy:

```
* java.lang.Object
        |- Boolean      < final >
        |- Character    < final >
        |- Number       < abstract >
                |- Byte  < final >
                |- Short         < final >
                |- Integer       < final >
                |- Long          < final >
                |- Float < final >
                |- Double        < final >
```

- Widening: We can convert state of object of narrower type into wider type. it is called as widening.

```
int num1 = 10;
double num2 = num1;          //widening
```

- Narrowing : We can convert state of object of wider type into narrower type. It is called narrowing.

```
double num1 = 10.5;
int num2 = ( int ) num1;     //narrowing
```

- Reference Type:

- Only 4 types are reference types in java
- following are the reference types in java
    1. interface
    2. class
    3. enum
    4. array

+ Command Line Arguments:

```
public class Program
{
   public static void main(String[] args)
   {
        int num1 = Integer.parseInt( args[ 0 ] );
        float num2 = Float.parseFloat( args[ 1 ] );
        double num3 = Double.parseDouble( args[ 2 ] );
        String operator = args[ 3 ];
        switch( operator )
        {
        case "+":
          double result = num1 + num2 + num3;
          System.out.println("Result :          "+result);
        }
   }
}
//Input from termial : java Program 10 20.1f 30.2d +
```

+ Console Input Output :

- In java there is no standard way to take input from console.
- we can use any one of the following way to take input :

1. Console is a class declared in java.io package

```
Console console = System.console();
String text = console.readLine();
```

2. JOptionPane is a class declared in javax.swing package

```
String text = JOptionPane.showInputDialog("Enter text");
JOptionPane.showMessageDialog(null, text);
```

3. BufferedReader is class declared in java.io package

```
BufferedReader reader = null;
reader = new BufferedReader(new InputStreamReader(System.in));
String text = reader.readLine();
```

4. Scanner is a class declared in java.util package.

```
Scanner sc = new Scanner(System.in);
String text = sc.nextLine();
```

+ Boxing :

- It is the process of converting state of object of value type into reference type.
- Consider the example:

```
int number = 10;
String strNumber = String.valueOf( number );
```

- If boxing is done implictly then it is called autoboxing.
- Consider the example:

```
Object obj = 10;        //AutoBoxing
```

it is internally works like:

```
Object obj = new Integer( 10 );
```

+ UnBoxing :

- It is the process of converting state of object of reference type into value type.
- Considert the example:

```
String strNumber = "125";
int number = Integer.parseInt( strNumber );
```

- If unboxing is done impliitly then it is called as autounboxing
- Considert the example:

```
Integer obj = new Integer(10);
int number = obj;       //AutoUnboxing
```

Java Notes - Chapter 4 - Package

+ Path :

- It is operating system platform's environment variable which is used to locate java tools.
- export PATH=/usr/bin/

+ Classpath :

- It is Java platforms environment variable which is used to locate .class file.
- export CLASSPATH=./bin

+ Package:

- If we want to group functionally equivalent or functionally related classes together then we should use package.
- Package can contain following elements:
  * Sub package
  * Interface
  * Class
  * Enum
  * Error
  * Exception
  * Annotation
- **Advantages of package**
  * To avoid name clashing/ collision/ ambiguity
  * To group functionally related classes together.
- If we want add class in package then we should use package keyword.

e.g

```
package p1;
class Complex
{       }
```

- package statement must be the first statement in .java file.

```
import java.util.Scanner;
package p1;    //Error
class Complex
{       }
```

- we can not declare multiple packages in single .java file.

```
package p1;
package p2;    //Error
class Complex
{       }
```

- package name is physically mapped with folder.
- Default access modifier of the class is package(default).

```
package p1;
??? class Complex      // ??? --> package level private
{       }
```

- If we want to access any Type outside the package then we should declare that type public.
- class can have either package or public access modifier.

```
package p1;
public class Complex
{       }
```

- Name of public class and .java file must be same hence in single .java file we cannot write multiple public classes.
- .java file can contain multiple non-public classes but It can contain at the most only one public class.

- If we want to use packaged class outside the package then we should use either fully qualified class name or import statement.
- Consider the example:

```
class Program
{
	public static void main( String[] args )
	{
		p1.Complex c1 = new p1.Complex(); //Ok
	}
}
or

import p1.Complex;   //Look Here
class Program
{
	public static void main( String[] args )
	{
		Complex c1 = new Complex();		//Ok
	}
}
```

- We can access packaged class in unpackaged class.
- If we declare class without package then it is considered as a member of default package.
  We cannot import default package hence it is not possible to access unpackaged class in packaged class.

```
"Complex.java"
public class Complex
{
	//TODO
}

"Program.java"
package p2;
public class Program
{
	public static void main( String[] args )
	{
		Complex c1 = new Complex();		//Error
	}
}
```

- We can add multiple classes in single package. In this case we can access class directly within same package.

```
"Complex.java"
package p1;
public class Complex
{
	//TODO
```

```
}

"Program.java"
package p1;
public class Program
{
        public static void main( String[] args )
        {
                Complex c1 = new Complex();         //Ok
        }
}
```

- we can declare package inside package. It is called sub package.

```
package p1.p2;          //Here p2 is sub package
public class Complex
{
        //TODO
}
```

- If we want to access types outside package then we should use import statement and if we want to access static members of packaged class outside package directly then we should use static import.

```
import static java.lang.Math.PI;
import static java.lang.Math.pow;
import static java.lang.System.out;

public class Program
{
   public static void main(String[] args)
   {
        double radius = 10;
        double area = PI * pow( radius, 2);
        out.println("Area       :       "+area);
   }
}
```

- Most commonly used packages in java:
  * java.lang
  * java.util
  * java.io
  * java.net
  * java.rmi
  * java.lang.reflect
  * java.sql
  * java.rmi

- Since java.lang package is by default imported in every .java file, it is optional to import java.lang package.

+ **Jar:**
  - using jar tool we can group packages together. Jar tool creates jar file.
  - If we want to create reusable component in java then we should take help of jar file.
  - Steps to create jar file from eclipse:
    * create java project and add classes in it.
    * Right click on project--> Export-->Java-->Jar File--Next-->Brozse... -->.jar --> Finish.
  - **Steps to add .jar file in runtime classpath or buildpath:**
    * create java project.
    * Right click on project-->Build Path-->Configure Build Path-->Java Build Path--> Libraries--> Add External Jars--> Select jar file and then click on ok.
    * import package and use types.

Java Notes - Chapter 5 - OOPS - PART I

+ Class:

- It is a collection of fields and methods
  - * Field - data member is called field in java
  - * Method - member function is called method in java
- Class is reference type i.e instance of class get space on heap section.
- We can achieve encapsulation using class.
- In java every class is by default extended from java.lang.Object class.

+Java.lang.Object:

- Object is non final concrete class declared in java.lang package.
- In java every class extends Object class directly or indirectly.
- Methods declared in Object class:

```
* protected Object clone() throws CloneNotSupportedException
* public boolean equals(Object obj)
* public int hashCode()
* public String toString()
* protected void finalize() throws Throwable
* public final Class<?> getClass()
* public final void wait()throws InterruptedException
* public final void wait(long timeout)throws InterruptedException
* public final void wait(long timeout,int nanos) throws InterruptedException
* public final void notify()
* public final void notifyAll()
```

+ Instance:

- Object is called instance in java.
- To create instance it is necessary to use new operator.
- Consider the following code snippet:

```
* Complex c1;          //reference
* new Complex();       //instance
* Complex c1 = new Complex();
  Complex c2 = c1;     //Shallow Copy of references.
```

+ Constructor:

- It is method of class which is used to initialize the object or instance.
- Java do not support default argument. Hence it is possible to call one constructor from another constructor. It is called constructor chaining.
- If we want to reuse implementation of existing constructor then we should use constructor chaining.
- using this statement we can achieve constructor chaining.
- this statement must be the first statement in constructor.

```
class Complex
{
   private int real;
   private int imag;
   public Complex()
   {
```

```
        this(0,0);      //ctor chaining
    }
    public Complex(int real, int imag)
    {
        this.real = real;
        this.imag = imag;
    }
}
```

+ Final variable:

- After storing value inside varible, if we dont want to modify value of that variable then we should declare such variable as final.
- final is keyword in java.
- we can declare reference as a final but we can not declare object as a final.

+ Static in java:

- In java we cannot declare local variable as a static.
- If we want to share value of field in all the instances of same class then we should declare field as a static.
- Static field do not get space inside object hence we should access it using class name.
- To initialize the static field we should use static block. we can write multiple static blocks inside a class.
- JVM executes static block at the time of class loading.
- To access state of non-static member we should define non static method inside class and to access state of static member we should define static method inside a class.
- In non-static method we can access static as well as non-static members.
- Static method do not get this reference hence in static method we cannot access non-static members.
- Using object/instance we can access non static members inside static method.

+ Singleton Class:

- a class from which we can create only one instance is called singleton class.
- Consider the following code.

```
class Singlton
 {
  private Singleton()
  {   }
  private static Singleton singleton;
  public static Singleton getInstance()
  {
      if( singleton == null )
         singleton = new Singleton();
      return singleton;
  }
}
```

+ Array:

- It is collection of same type of elements where each element get space continuously.
- It is collection of fixed elements i.e it cannot grow/shrink at runtime.
- Arrays are reference types in java i.e to create array instance we should use new operator
- There are three types of array in java:
  * Single dimensional
  * Multi-dimensional
  * Ragged Array

+ Single Dimensional Array:

```
- int arr[];                          //Array reference : Ok
- int[] arr;                          //Array reference : Recommended
- int[] arr = new int[ 3 ];           //Array instance
- int[] arr = new int[ ]{ 10, 20, 30 }; //Initialization
- int[] arr = { 10, 20, 30 };         //Initialization : valid
```

- Accessing elements using for loop

```
for( int index = 0; index < arr.length; ++ index )
    System.out.println( arr[ index ] );
```

- Accessing elements using for each loop

```
for( int element : arr )
    System.out.println( element );
```

+ Multi-Dimensional Array:

- Array of array which contains same number of elements is called Multi-dimensional Array.

```
- int arr[][];                        //Array reference : Ok
- int[] arr[];                        //Array reference
- int[][] arr;                        //Array reference : Recommended
- int[][] arr = new int[2][3];        //Array instance
- int[][] arr = new int[2][3]{ {10, 20, 30}, {40, 50, 60} };   //Initialization
- int[][] arr = { {10, 20, 30}, {40, 50, 60} };          //Initialization
```

- Accessing elements using for loop

```
for( int row = 0; row < arr.length; ++ row )
{
    for( int col = 0; col < arr[ row ].length; ++ col )
    {
        System.out.println(arr[row][col]);
    }
}
```

- Accessing elements using for each loop

```
for( int[] arrRef : arr )
{
    for( int element : arrRef )
    {
        System.out.println(element);
```

```
        }       }
```

+ Ragged Array:

- Array of array which contains diff. number of elements is called ragged Array.
- int arr[][]; //Array reference : Ok
- int[][] arr; //Array reference : Recommended
- int[][] arr = new int[3][]; //Array of references

```
arr[ 0 ] = new int[ 2 ];
arr[ 1 ] = new int[ 3 ];
arr[ 2 ] = new int[ 4 ];
```

- Accessing elements using for loop

```
for( int row = 0; row < arr.length; ++ row )
{
        for( int col = 0; col < arr[ row ].length; ++ col )
        {
                System.out.println(arr[row][col]);
        }
}
```

- Accessing elements using for each loop

```
for( int[] arrRef : arr )
{
        for( int element : arrRef )
        {
                System.out.println(element);
        }
}
```

+ Array of Value Types:

- If we create array of value type then it contains value. Default value is depends on default value of datatype.

e.g boolean[] arr = new boolean[3];

+ Array of Rerefences:

- If we create array of reference type then it contains reference. Default value is null.

e.g Complex[] arr = new Complex[3];

+ Array of Objects:

```
Complex[] arr = new Complex[ 3 ];  //Array of references
for( int index = 0; index < arr.length; ++ index )
```

```
arr[ index ] = new Complex();
```

+ In java, every element is pass to the function by value. If we want to pass element to the function by reference then we should use reference.

```
Public static void swap(int[] arr)
{
    int temp = arr[ 0 ];
    arr[ 0 ] = arr[ 1 ];
    arr[ 1 ] = temp;
}
public static void main(String[] args)
{
    int num1 = 10, num2 = 20;
    int[] arr = new int[ ]{ num1, num2 };
    Program.swap(arr);
    num1 = arr[ 0 ]; num2 = arr[ 1 ];
    System.out.println(num1+"    "+num2);
}
```

+ Object Oriented Programming:

- It's a method of programming in which real world problems are solved using classes and objects.
- Java is object oriented programming language.
- A language which supports all major and minor pillars of OOP is called pure object oriente programming language.

+ Major Pillars / Parts / Elements of OOPS:

- Abstraction
  * It is the process of getting essential things from object.
  * Abstraction may changes from object to object.
  * Using abstraction we can achieve simplicity.
- **Encapsulation**
  * Binding of data and code together is called encapsulation / Process of implementing abstraction is called encapsulation.
  * Using class we can achieve encapsulation.
- **Modularity-**
  * Process of dividing complex system into small modules is called modularity.
  * using package and jar file we can achieve modularity.
  * Using modularity we can minimize module dependancy.
- **Hierarchy**
  * Level / Order / ranking of abstraction is called hierarchy.
  * Main job of hierarchy is to achieve modularity.
  * 4 Types of heirarchy:
    # has-a / part-of --> Composition
    # is-a / kind-of --> Inheritance
    # use-a --> Dependancy
    # creates-a --> Instantiation

+ Minor Pillars / Parts / Elements of OOPS:

- Typing / Polymorphism
  * Ability of an object to take multiple forms is called polymorphism.
  * Using polymorphism we can reduce maintenance of the system.
  * We can achieve it using
    # Method Overloading
    # Method overriding
- **Concurrency**
  * Process of executing multiple task simultaneously is called concurrency.
  * We can achieve it using threading.
- **Persistence**
  * Process of maintaing state of the object in file or database is called persistence.

+ Composition:

- Composition is also called as containment.
- If has-a relationship exist between two types then we shoud use composition.

  e.g Car has-a engine
- If object/component is part of another object then it is called as composition.
- In java composition do not implies physical containment. Composition is achieved using reference variable only.
- In other words object outside the object is called composition.
- Consider the example:

```
class Person
{
        private String name = new String();
        private Date birthDate = new Date();
        private Address currAddress = new Address();

        //TODO
}
```

+ Inheritance:

- Inheritance is also called as generalization.
- If is-a relationship exist between two types then we should use inheritance.
- Without modifing existing class if we want to extend meaning of the class then we should use inheritance.
- In java, parent class is called super class and child class is called sub class.
- to create sub class we should use "extends" keyword.
- consider following example:

```
class Shape
{       }
class Circle extends Shape
{       }
```

- Sub class can extend at the most only one super class. i.e. multiple implementation inheritance is not allowed:

```
class A{        }
class B{        }
class C extends A, B   //Not Allowed in java
{       }
```

- Java supports only public mode of inheritance.

	In same package			In diff package	
	Same class	Sub class	Non sub class	Sub class	Non Sub class
private	A	NA	NA	NA	NA
package	A	A	A	NA	NA
protected	A	A	A	A	NA
public	A	A	A	A	A

- If we create instance of sub class then first super class constructor gets called and then sub class constructor gets called.
- In Java class do not contain destructor.
- using super statement, we can call any super class constructor from sub class constructor.
- If we want to access members of super class in method of sub class then we should use super keyword.

+ Upcasting and Downcasting:

- we can convert reference of sub class into reference of super class. It is called upcasting.

```
e.g.  Employee emp = new Emloyee("Sandeep",33,45000);
      Person p = emp;        //Upcsting.
```

- We can convert reference of super class into reference of sub class. It is called downcasting.

```
e.g   Person p =  new Emloyee("Sandeep",33,45000);
      Employee emp = ( Employee ) p;      //Downcasting
```

- If downcasting fails then JVM throws ClassCastException.

- In java all the methods are by default virtual hence using super class reference variable we can call method of sub class. It is called dynamic method dispatch.

+ Rules of method overriding:

1. Signature of super class and sub class method should be same.
2. Access modifier of super class and sub class method should be same or it should be wider than super class method.

3. Return type of super class and sub class method should be same or it should be sub type of return type specified in super class method.
4. Method name, number of parameter and type of parameter pass to the method must be same.
5. Checked exception list specified in sub class method should be same ot it should be subset of exception list specified in super class method.

+ Difference between == and equals method:

- if we want to compare state of object of value type then we should use == operator.

```
int num1 = 10;
int num2 = 10;
if( num1 == num2 )
   System.out.println("Equal");
else
   System.out.println("Not Equal");
//Output : Equal
```

- if we want to compare state of variable of reference type then also we should use == operator.

```
Complex c1 = new Complex(10,20);
Complex c2 = new Complex(10,20);
if( c1 == c2 )
   System.out.println("Equal");
else
   System.out.println("Not Equal");
//Output : Not Equal
```

- if we want to compare state of object of reference type then we should use equals method.

```
Complex c1 = new Complex(10,20);
Complex c2 = new Complex(10,20);
if( c1.equals( c2 ) )
   System.out.println("Equal");
else
   System.out.println("Not Equal");
//Output : Equal
```

* if class do not contain equals method then super class equals method gets called. And if any super class do not contain equals method then Object class equals method gets called. Object.equals always compares object reference.

* To compare state of the object we should override equals method in class.
* For example:

```
class Complex
{
   private int real;
   private int imag;
   @Override
   public boolean equals(Object obj)
   {
        if( obj != null )
        {
           Complex other = (Complex)obj;
           if( this.real == other.real && this.imag == other.imag )
                return true;
        }
        return false;
   }
}
```

+ Final method:

- If implementation of super class method is logically complete then we should declare super class method as final.
- We can not override final method in sub class. But final method inherit into sub class.
- Some of the final methods are:
  * public final Class<?> getClass()
  * public final void wait() throws InterruptedException
  * public final void notify()
  * public final void notifyAll()

+ Final class:

- If implementation of all the methods is logically complete then instead of declaring method as final we should declare class as final.
- we can not extend final class i.e we can create sub class of final class.
- Some of the final classes are:
  * All Wrapper classes.
  * java.lang.System
  * java.lang.String
  * java.lang.StringBuffer
  * java.lang.StringBuilder
  * java.lang.Math

+ Abstract method:

- If implementation of super class method is logically 100% incomplete then we should declare super class method as abstract.
- Abstract method do not contain body.
- If method is abstract then it is compulsory to declare class as abstract.
- if super class contains abstract method then sub class should override that method in sub class or sub class should be declared as abstract.
- we cannot override private, final and static method in sub class hence such keywords cannot be used with abstract method.
- Some of the abstract methods declared in Number class are:

```
* public abstract int intValue()
* public abstract float floatValue()
* public abstract double doubleValue()
* public abstract long longValue()
```

+ Abstract class:

- If implementation of any class is logically incomplete then we should declare class as abstract. - If class contains abstract method then we should declare class as abstract.
- Without declaring method as abstract we can declare class as abstract.
- We cannot create instance of abstract class but we can create reference of abstract class.
- Abstract class can contain constructor.
- Some of the abstract classes are:
  * java.lang.Enum
  * java.lang.Number

+ Fragile base class problem:

- If we make changes in super class method then it is necessary to recompile all the sub classes is called fragile base class problem.
- To avoid fragile base class problem we should declare super type as a interface.

+ Interface:

- Set of rules is called standard and standard is also called as specification.
- If we want to define specification for the sub classes then we should use interface.
- There are a number of situations in software engineering when it is important for disparate groups of programmers to agree to a "contract" that spells out how their software interacts. Each group should be able to write their code without any knowledge of how the other group's code is written. Generally speaking, interfaces are such contracts.
- Interface is keyword in java.

- Interface is reference type.
- Interface an contain only constants, method signatures, default methods, static methods, and nested types.
- Interfaces cannot be instantiated , they can only be implemented by classes or extended by other interfaces.
- Observe the following statements:

```
Interface : I1, I2, I3
Class     : C1, C2, C3

* I2 implements I1          //Incorrect
* I2 extends I1             //correct : Interface inheritance
* I3 extends I1, I2         //correct : Multiple interface inheritance
* C2 implements C1          //Incorrect
* C2 extends C1             //correct : Implementation Inheritance
* C3 extends C1,C2          //Incorrect : Multiple Implementation Inheritance
* I1 extends C1             //Incorrect
* I1 implements C1          //Incorrect
* c1 implements I1          //correct : Interface implementation inheritance
* c1 implements I1,I2 //correct : Multiple Interface implementation inheritance
* c2 implements I1,I2 extends C1     //Incorrect
* c2 extends C1 implements I1,I2     //correct
```

- If we want to implement interface then we should use implements keyword.

```
interface A
{
   void f1();
}
class B implements A
{
   @Override
   public void f1()
   {    }
}
```

- If class implement multiple interfaces which contain methods with same signature then sub class can override it only once.

```
interface A
{
   void f1();
}
interface B
```

```
{
   void f1();
}
class C implements A, B
{
   @Override
   public void f1()
   {    }
}
```

- If class implement multiple interfaces which contain methods with same name and different return type then sub class can not override it.

```
interface A
{
   int f1();
}
interface B
{
   double f1();
}
class C implements A, B
{
   //Error: Can not override method in sub class
}
```

+ Default Methods:

- Designing interfaces have always been a tough job because if we want to add additional methods in the interfaces, it will require change in all the implementing classes.

As interface grows old, the number of classes implementing it might grow to an extent that it is not possible to extend interfaces.

That is why when designing an application, most of the frameworks provide a base implementation class and then we extend it and override methods that are applicable for our application.

- To overcome such problem, in java 8 , we can add default methods in interface.

- For creating a default method in java interface, we need to use " default " keyword with the method signature.

```
interface A
{
   void f1(String str);
   default void log(String str)
   {
```

```
        System.out.println("A log :"+str);
    }
}
```

- Notice that log(String str) is the default method in the Interface A. Now when a class will implement interface A, it is not mandatory to provide implementation for default methods of interface. This feature will help us in extending interfaces with additional methods, all we need is to provide a default implementation.

- We know that Java doesn't allow us to extend multiple classes because it will result in the "Diamond Problem" where compiler cannot decide which superclass method to use. With the default methods, the diamond problem would arise for interfaces too.

- Some key points about default method:

* Java interface default methods will help us in extending interfaces without having the fear of breaking implementation classes.

* Java interface default methods has bridge down the differences between interfaces and abstract classes.

* Java 8 interface default methods will help us in avoiding utility classes, such as all the Collections class method can be provided in the interfaces itself.

* Java interface default methods will help us in removing base implementation classes, we can provide default implementation and the implementation classes can chose which one to override.

* One of the major reason for introducing default methods in interfaces is to enhance the Collections API in Java 8 to support lambda expressions.

* A default method cannot override a method from java.lang.Object.

* Java interface default methods are also referred to as Defender Methods or Virtual extension methods.

+ Interface Static Method:

- Java interface static method is similar to default method except that we canâ€™t override them in the implementation classes.

- Important points about java interface static method:

* Java interface static method is part of interface, we can't use it for implementation class objects.

* Java interface static methods are good for providing utility methods, for example null check, collection sorting etc.

* Java interface static method helps us in providing security by not allowing implementation classes to override them.

* We can't define interface static method for Object class methods.

* We can use java interface static methods to remove utility classes such as Collections and move all of it's static methods to the corresponding interface, that would be easy to find and use.

+ Functional Interface:

- An interface with exactly one abstract method is known as Functional Interface.
- A new annotation @FunctionalInterface has been introduced to mark an interface as Functional Interface.
- @FunctionalInterface annotation is a facility to avoid accidental addition of abstract methods in the functional interfaces. It's optional but good practice to use it.
- Functional interfaces enables us to use lambda expressions to instantiate them.
- java.util.function package contains functional interface that are targeted to lambda expression.

+ Exception Handling:

- An exception is an event, which occurs during the execution of a program that disrupts the normal flow of the program's instructions.
- To manage OS resources carefully, we should handle exception.
- To handle exception we should use following keywords:
  * try - to inspect the exception
  * throw - to generate the exception
  * catch - to handle the exception
  * throws - to redirect the exception
  * finally - to release the local resources.

- java.lang.Throwable is super class of Error and exception in java.
- Exceptional conditions that are internal to the application are exceptions.
- Exceptional conditions that are external to the application, and that the application usually cannot recover from are errors.

+ Throwable class:

- Constructors:

```
* public Throwable()
* public Throwable(String message)
* public Throwable(Throwable cause)
* public Throwable(String message,Throwable cause)
```

- Methods:

```
* public final void addSuppressed(Throwable exception)
* public StackTraceElement[] getStackTrace()
* public Throwable getCause()
* public String getMessage()
* public void printStackTrace()
```

+ Types of Exception:

- Checked and UnChecked are the types of exception. These are the types of exception designed for compiler.

- Checked Exception

* java.lang.Exception and all its sub classes except RuntimeException class are classified as checked exception classes.
* It is compulsory to handle checked exception otherwise compiler generates error.
* e.g InterruptedException, ClassNotFoundException etc.

- UnChecked Exception

* java.lang.RuntimeException and all its sub classes are classified as checked exception classes.
* It is optional to handle unchecked exception.

* e.g NullPointerException, ArithmeticException, ClassCastException etc.

+ StackTrace :

- A stack trace provides information on the execution history of the current thread and lists the names of the classes and methods that were called at the point when the exception occurred.
- A stack trace is a useful debugging tool that you'll normally take advantage of when an exception has been thrown.

+ Syntax:

```
try
{
   BufferedReader reader = null;
   reader = new BufferedReader(new InputStreamReader(System.in));
   System.out.print("Enter num1      :       ");
   int num1 = Integer.parseInt(reader.readLine());
   System.out.print("Enter num2      :       ");
   int num2 = Integer.parseInt(reader.readLine());
   int result = num1 / num2;
   System.out.println("Result :        "+result);
   reader.close();
}
catch (NumberFormatException e)
{
   e.printStackTrace();
}
catch (ArithmeticException e)
{
   e.printStackTrace();
}
catch (IOException e)
{
   e.printStackTrace();
}
```

- It is possible to handle all the exceptions in single catch block.

```
try
{
   BufferedReader reader = null;
   reader = new BufferedReader(new InputStreamReader(System.in));
```

```
        System.out.print("Enter num1      :       ");
        int num1 = Integer.parseInt(reader.readLine());
        System.out.print("Enter num2      :       ");
        int num2 = Integer.parseInt(reader.readLine());
        int result = num1 / num2;
        System.out.println("Result :        "+result);
        reader.close();
    }
    catch (NumberFormatException | ArithmeticException | IOException e)
    {
        e.printStackTrace();
    }
```

- If you want to release local resources then you should define finally block

```
    Scanner sc = null;
    try
    {
        sc = new Scanner(System.in);
        System.out.print("Enter num1      :       ");
        int num1 = sc.nextInt();
        System.out.print("Enter num2      :       ");
        int num2 = sc.nextInt();
        int result = num1 / num2;
        System.out.println("Result :        "+result);
    }
    catch ( ArithmeticException | InputMismatchException e)
    {
        e.printStackTrace();
    }
    finally
    {
        sc.close();
    }
```

- try-with-resources

* A resource is an object that must be closed after the program is finished with it.
* Instance of class which implements either AutoCloseable or Closeable interface is called resource.

* If we use resource with try then there is no need to close the resource in finally. When block of code gets executed normally or by throwing exception then close method of resource gets called automatically.

* consider the following code:

```
try( Scanner sc = new Scanner(System.in); )
{
   System.out.print("Enter num1      :       ");
   int num1 = sc.nextInt();
   System.out.print("Enter num2      :       ");
   int num2 = sc.nextInt();
   int result = num1 / num2;
   System.out.println("Result :        "+result);
}
catch ( ArithmeticException | InputMismatchException e)
{
   e.printStackTrace();
}
```

- A try-with-resources statement can have catch and finally blocks just like an ordinary try statement.
- Some exceptions are designed to handle. If we do not handle it then compiler generates error.

Consider the following code:

```
public static void print()
{
      for( int i = 1; i < 10; ++ i )
      {
         System.out.println(i);
         Thread.sleep(300);  //Compiler error
      }
}
```

- sleep method throws InterruptedException, which is checked exception. We must handle it.

```
public static void print()
 {
      try
      {
         for( int i = 1; i < 10; ++ i )
         {
            System.out.println(i);
            Thread.sleep(300);        //Now Ok
         }
      }
```

```
        catch (InterruptedException e)
        {
           e.printStackTrace();
        }
    }
```

- If we want to redirect exception to caller of the method then we can use throws clause.

```
public static void print() throws InterruptedException
{
        for( int i = 1; i < 10; ++ i )
        {
           System.out.println(i);
           Thread.sleep(300);
        }
}
```

- throws clause can be used to redirect any checked as well as unchecked exception.

Generic Catch:

- Exception class can keep reference of any checked as well as unchecked exception hence it is used to write generic catch.
- Remember, compiler do not allows us to handle super type exception first. So generic catch must appear at last.

Summary:

- using try, catch, throw, throws and finally we can handle exception in java.
- try block may have multiple catch block but single try block must have at lest one catch/ finally or resource.
- we can handle multiple exceptions in single catch block also.
- generic catch must appear at last.
- Handling checked exceptions is compulsory. If we do not handle it then compler gives err.
- To handle checked exception we should use either try-catch block or we should redirect exception to caller method using throws clause.
- To release local resources we should use finally block.
- JVM always execute finally block. If we write System.exit(0) in try and catch block then JVM do execute finally block.

Chained Exceptions:

- Generally exceptions are handled by throwing new exception. This process of handling exception is called exception chaining.
- Following Methods and constructors in Throwable that support chained exceptions:

```
* Throwable(Throwable)
* Throwable(String, Throwable)
* Throwable initCause(Throwable)
* Throwable getCause()
```

- consider the following example:

```
try
{
        //TODO
}
catch (SQLException e)
{
   throw new SerevletException("Other SQLException", e); // Exception Chaining
}
```

+ **Consider the following code snippets:**

1.

```
int[] arr = new int[ ]{ 10, 20, 30 };
int element = arr[ 3 ]; //ArrayIndexOutOfBoundsException
```

2.

```
String str = "Sandeep";
char ch = str.charAt(7); //StringIndexOutOfBoundsException
```

3.

```
ArrayList<Integer> list = new ArrayList<>();
list.add(10);
list.add(20);
list.add(30);
int element = list.get(3); //IndexOutOfBoundsException
```

4.

```
Object obj = new Integer(10);
//downcasting
String str = ( String )obj; // ClassCastException
```

5.

```
String str = "A123";
int number = Integer.parseInt(str); // NumberFormatException
```

+ Generics:

- Like object, we can pass data type as a argument to the function. Hence parameterized type is generics. In C++, it is called as template.

- First let us discuss generic class using java.lang.Object class. Consider the following code.

```
class Box
{
	private Object object;
	public Object getObject()
	{
		return object;
	}
	public void setObject(Object object)
	{
		this.object = object;
	}
}
```

1.

```
Box box = new Box();
int num1 = 10;
box.setObject(num1);//box.setObject(new Integer(num1));
int num2 = (int)box.getObject();	//Ok
```

2.

```
Box box = new Box();
Date dt1  = new Date();
box.setObject(dt1);			//UpCasting
Date dt2 = (Date)box.getObject();	//DownCasting
```

3.

```
Box box = new Box();
Date dt1  = new Date();
box.setObject(dt1);			//UpCasting
String str = (String)box.getObject();	//DownCasting : ClassCastException
```

+ Why Generics?

1. Generics gives us stronger type checking at compile time.
2. It completely eliminates explicit type casting.
3. Using generics we can write generic algorithm.

Let's us consider following code.

```
class Box< T >          //T is Type parameter
{
        private T object;
        public T getObject()
        {
                return object;
        }
        public void setObject(T object)
        {
                this.object = object;
        }
}
```

1.

```
Box<int> box = new Box<int>();      //Error
```

In above code, int is type argument. Type argument used in generics must be reference type.

2.

```
Box<Integer> box = new Box<Integer>();    //OK
```

If we want to store primitive values in generic collection then we should use wrapper class.

3.

```
Box<Date> box = new Box<Date>();
box.setObject(new Date());
String str = (String)box.getObject();  //Compiler error : src and dest must be Date  type.
```

4.

```
Box<Date> box = new Box<Date>();
box.setObject(new Date());
Date date = box.getObject();  //OK
```

+ Most Commonly used type parameters in java:

T	-	Type
N	-	Number
E	-	Element
K	-	Key

V - Value
A, U - Second Type Parameters.

We can specify multiple type parameters too.

```
interface Map<K,V>
{
	void put( K key, V value );
	K getKey();
	V getValue();
}
class HashTable<K,V> implements Map<K,V>
{
	private K key;
	private V value;
	public void put( K key, V value )
	{
		//TODO : Assignment for programmer
	}
	@Override
	public K getKey()
	{
		return this.key;
	}
	@Override
	public V getValue()
	{
		return this.value;
	}
}
public class Program
{
	public static void main(String[] args)
	{
		Map<Integer,String> map = new HashTable<>();
		map.put(1, "Sandeep");
		map.put(2, "Prathamesh");
		map.put(3, "Soham");
	}
}
```

+ Bounded Type Parameter :

- Sometimes we need to put restrictions on a type that can we used as type argument. Using bounded type parameter we can put restriction on type argument.

```
class Box< N extends Number >       //N extends Number => is bounded type parameter.
{
        private N object;
        public N getObject()
        {
                return object;
        }
        public void setObject(N object)
        {
                this.object = object;
        }
}
```

- as shown in above code, box object can store only numbers.

1.
```
Box<Integer> box = new Box<>();          //Allowed
```
2.
```
Box<String> box = new Box<>();           //Not allowed
```

+ Wild Card:

- In generics ? is called wild card which indicates unknown type.
- There are 3 types of wild card in generics.
    1. UnBounded Wild Card.
    2. Upper Bounded Wild Card.
    3. Lower Bounded Card.

+ UnBounded Wild Card :

```
public static void printList( List<?> list )
{
        for (Object object : list)
        {
                System.out.println(object);
        }
```

```
}

List<Integer> list1 = new ArrayList<>();
//TODO: Add element in list1
Program.printList(list1);

List<String> list2 = new Vector<>();
//TODO: Add element in list2
Program.printList(list2);

List<Date> list3 = new LinkedList<>();
//TODO: Add element in list3
Program.printList(list3);
```

- As shown in above code, list can store reference of any List<> collection which contains any type of element.

+ Upper Bounded Wild Card:

```
public static void printList( List<? extends Number > list )
{
        for (Number number : list)
        {
                System.out.println(number);
        }
}
List<Integer> list1 = new ArrayList<>();
//TODO: Add element in list1
Program.printList(list1);

List<Double> list2 = new Vector<>();
//TODO: Add element in list2
Program.printList(list2);

List<String> list3 = new ArrayList<>();
//TODO: Add element in list3
Program.printList(list3);               //Not Allowed
```

- As shown in above code, list can store reference of any List<> collection which contains only number.

+ Lower Bounded Wild Card:

```
public static void printList( List<? super Integer > list )
{
	for (Object object : list)
	{
		System.out.println(object);
	}
}
```

```
List<Integer> list1 = new ArrayList<>();
//TODO: Add element in list1
Program.printList(list1);

List<Double> list2 = new Vector<>();
//TODO: Add element in list2
Program.printList(list2);		//Not Allowed

List<String> list3 = new ArrayList<>();
//TODO: Add element in list3
Program.printList(list3);		//Not Allowed
```

- As shown in above code, list can store reference of any List<> collection which contains elements of integer or its super type.

```
public static void printList( List<Number > list )
{
	//TODO
}
public static void main(String[] args)
{
	List<Integer> list1 = new ArrayList<Integer>();
	Program.printList(list1);	//Error
}
```

- In case of type argument, inheritance is not allowed. As shown in above code, type argument used in List is Number and type argument used in ArrayList is integer. It must be same.

+ Marker Interface:

- An interface which do not contain any member is called marker interface.
- following are the marker interface:
  * java.lang.Cloneable
  * java.util.EventListener
  * java.util.RandomAccess
  * java.io.Serializable
  * java.rmi.Remote

+ Commonly used interfaces in java:

- java.lang.Cloneable
- java.lang.AutoCloseable
- java.lang.Comparable<T>
- java.util.Comparator<T>
- java.lang.Iterable<T>
- java.util.Iterator<E>
- java.lang.Runnable

+ Cloneable:

- It is interface declared in java.lang package.
- It is marker interface.
- consider the following code:

```
Complex c1 = new Complex();
Complex c2 = c1;      //Shallow Copy of references
```

- If we want to create new object from existing object then we should override clone() method.

```
@Override
  public Complex clone() throws CloneNotSupportedException
  {
    Complex other = ( Complex )super.clone();
    return other;
  }
```

- If we want to create shallow copy of current object then we should use "super.clone()".
- Without implementing Cloneable interface, if we try to use clone method then it throws CloneNotSupportedException.

+ Comparable:

- It is interface declared in java.lang package.
- "int compareTo(T o)" is method declared in Comparable interface.

- If we want to sort array of object of same type then we should use Comparable interface.
- Consider the following example:

```
class Employee implements Comparable<Employee>
{
   @Override
   public int compareTo(Employee other)
   {
        if( this.empid < other.empid )
           return -1;
        else if( this.empid > other.empid )
           return 1;
        else
           return 0;
   }
}
public class Program
{
   public static Employee[] getEmployees()
   {    }
   public static void main(String[] args)
   {
        Employee[] arr = Program.getEmployees();
        Arrays.sort(arr);
   }
}
```

+ Comparator:

- It is interface declared in java.util package.
- It is functional interface.
- "int compare(T o1,T o2)" is method of Comparator interface.
- If we want to sort array of objects of different types then we should use Comparator interface.
- Consider following example:

```
abstract class Person{ }
class Student extends Person{        }
class Employee extends Person{       }
class SortById implements Comparator<Person>
{
```

```
    @Override
    public int compare(Person p1, Person p2)
    {
        //TODO
        return 0;
    }
}
public class Program
{
    public static Person[] getPersons()
    {   }
    public static void main(String[] args)
    {
        Person[] persons = Program.getPersons();
        Comparator<Person> comparator = new SortById();
        Arrays.sort(persons,comparator);
    }
}
```

+ **Iterator:**

- It is smart pointer which is used to traverse the collection.
- In java for each loop is called iterator.
- We can traverse elements of array or object which implements java.lang.Iterable interface.

Consider the folloiwing code

```
class Node
{       }
class LinkedList implements Iterable<Integer>
{
    private Node head = null;
    //TODO
    @Override
    public Iterator<Integer> iterator()
    {
      Iterator<Integer> itr = new LinkedListIterator( this.head );
      return itr;
    }
}
```

```
class LinkedListIterator implements Iterator<Integer>
{
   Node trav;
   LinkedListIterator( Node trav )
   {
         this.trav = trav;
   }
   @Override
   public boolean hasNext()
   {
      return this.trav != null;
   }
   @Override
   public Integer next()
   {
     int data = trav.data;
     trav = trav.next;
     return data;
   }
}
```

+ Nested Class:

- In Java we can write class inside scope of another class, it is called as nested class.
- Outer class can be declared as package level private or public only but nested class can be declared as private, package level private, protected or public.
- If we want to design class for implementation of another class then it should be nested.
- Why Use Nested Classes?
  * It is a way of logically grouping classes that are only used in one place:

    If a class is useful to only one other class, then it is logical to embed it in that class and keep the two together.
  * It increases encapsulation:

    Consider two top-level classes, A and B, where B needs access to members of A that would otherwise be declared private. By hiding class B within class A, A's members can be declared private and B can access them. In addition, B itself can be hidden from the outside world.
  * It can lead to more readable and maintainable code:

    Nesting small classes within top-level classes places the code closer to where it is used.

- Types of Nested Classes:

* Inner class:

If we declare nested class as a non-static then it is called as inner class.

```
#   class A                          //A.class
    {
        class B //Inner class       //A$B.class
        {
            //TODO
        }
    }
# A.B b = new A().new B();   //Instantiation
```

As with instance methods and variables, an inner class is associated with an instance of its enclosing class and has direct access to that object's methods and fields. Also, because an inner class is associated with an instance, it cannot define any static members itself.

For the sake of simplicity consider non static nested class as non-static method of the class.

using object (of inner class) we can access members of inner class inside method of outer class.

Members of Outer class are directly accessible in methods of inner class.

```
# Shadowing:
class Outer
{
        private int num1 = 10;
        class Inner
        {
                private int num1 = 20;
                public void print( )
                {
                        int num1 = 30;
                        System.out.println(Outer.this.num1); //10
                        System.out.println(this.num1);              //20
                        System.out.println(num1);           //30
                }
        }
}
```

If implementation of nested class is dependent on outer class then nested class should be declared as non static(Inner class).

There are two special kinds of inner classes: local classes and anonymous classes.

*** Static Nested class:**

```
# if we declare nested class as static then it is called as static nested  class.
#       class A                                         //A.class
        {
                static class B   //static nested class     //A$B.class
                {
                        //TODO
                }
        }
#       A.B b = new A.B();    //Instantiation
```

As with class methods and variables, a static nested class is associated with its outer class. And like static class methods, a static nested

class cannot refer directly to instance variables or methods defined in its enclosing class: it can use them only through an object reference.

for the sake of simplicity, consider static nested class as static method of the class.

Using object (of static nested class) we can access members of static nested class inside method of outer class.

Static members of Outer class are directly accessible in methods of static nested class. To access non static members it is necessary to use object.

If implementation of nested class do not dependent on outer class then nested class should be declared as static.

- Example of nested class.

```
class LinkedList
{
	private static class Node
	{
		//TODO
	}
	private Node head;
	public class LinkedListIterator
	{
		Node node = head;
		//TODO
	}
}
```

+ Local Class:

- In java, we can write class inside block/method. It is called as local class.
- Local class can be categorized as:
  * Method local inner class
  * Method local anonymous inner class

- Method local inner class:

* Local class cannot be declared as static hence it is also called as method local inner class.

* Instance of local class are accessible within method only.

- Method Local Anonymous Inner Class:

* We can define class without name, it is called anonymous class.

* In java, we can create anonymous class within block/method only so it cannot be static. Hence anonymous class is also called as method local inner class.

* Anonymous classes enable you to make your code more concise.

* The anonymous class expression consists of the following:

The new operator

The name of an interface to implement or a class to extend.

Parentheses that contain the arguments to a constructor, just like a normal class instance creation expression.

Note: When you implement an interface, there is no constructor, so you use an empty pair of parentheses.

A body, which is a class declaration body.

* Example 1:

```
Runnable r = new Runnable()
{
        @Override
        public void run()
        {
                System.out.println("Inside run");
        }
};
r.run();
```

* Example 2:

```
abstract class Shape
{
        public abstract void calculateArea();
}
public class Program
```

```
        {
                public static void main(String[] args)
                {
                        Shape sh = new Shape()
                        {
                                @Override
                                public void calculateArea()
                                {
                                        System.out.println("calculateArea()");
                                }
                        };
                        sh.calculateArea();
                }
        }
```

* Example 3:

```
        Object obj = new Object()
        {
                @Override
                public String toString()
                {
                        return "string";
                }
        };
        System.out.println(obj.toString());
```

An object which contains more than one element is called collection. In java data structure classes are called collection classes. Collection classes are also called as container classes. Library of reusable classes which are used to develop the application is called framework. Library of reusable collection classes which are used to develop java application is called java's collection framework.

Java is abstraction technology so, developer should not worry about implementation of collection rather he/she should worry about its use.

To use java collection framework, we should import java.util package. We are going to use following interfaces in collection framework.

java.lang.Iterable
java.util.Iterator
java.lang.Comparable
java.util.Comparator
java.util.Enumeration
java.util.Collection
java.util.List
java.util.ListIterator
java.util.Set
java.util.SortedSet
java.util.NavigableSet
java.util.Queue
java.util.Deque
java.util.Map
java.util.Map.Entry
java.util.SortedMap
java.util.NavigableMap

+ Collection:

- It is root interface in java collection framework.
- The JDK does not provide any direct implementations of this interface.
- This interface is typically used to pass collections around and manipulate them where maximum generality is desired.
- **Methods Of Collection Interface**
  * boolean add(E e)
  * boolean addAll(Collection<? extends E> c)
  * boolean contains(Object o)
  * boolean containsAll(Collection<?> c)

* boolean remove(Object o)
* boolean removeAll(Collection<?> c)
* boolean retainAll(Collection<?> c)
* void clear()
* boolean isEmpty()
* int size()
* Object[] toArray()
* <T> T[] toArray(T[] a)

* default boolean removeIf(Predicate<? super E> filter)
* default Stream<E> stream()
* default Stream<E> parallelStream()
* default Spliterator<E> spliterator()

+ **List:**

- It is sub interface of Collection interface.
- ArrayList, Vector, Stack, LinkedList etc implements List interface. These are reffered as List collection.
- List collections store data in ordered/sequential fashion [unsorted].
- From List collection, user can access element using integer index.
- In list collection we can insert duplicate elements.
- List collections allows us to insert multiple null elements.
- We can travserse elements of list collection using Iterator and ListIterator.
- If we want to store elements of reference type in List collection then reference type should contain equals method.
- Due to inheritance all above methods are implicitly methods of List. Following are List specific methods.

* void add(int index,E element)
* boolean addAll(int index, Collection<? extends E> c)
* E get(int index)
* E set(int index, E element)
* int indexOf(Object o)
* int lastIndexOf(Object o)
* ListIterator<E> listIterator()
* ListIterator<E> listIterator(int index)
* boolean remove(Object o)
* List<E> subList(int fromIndex, int toIndex)

* default void sort(Comparator<? super E> c)
* default void replaceAll(UnaryOperator<E> operator)

Following are the synchronized collection classes in java:

1. Vector
2. Stack
3. HashTable

+ ArrayList:

- It is dynamically growable/shrinkable array.
- It is unsynchronized collection.
- using Collections.synchronizedList() method, we can make it synchronized.
- It stores data in ordered/sequential fashion [unsorted].
- From ArrayList, user can access element using integer index.
- In ArrayList we can insert duplicate elements.
- It allows us to insert multiple null elements.
- We can travserse elements of ArrayList using Iterator and ListIterator.
- If we want to store elements of reference type in ArrayList then reference type should contain equals method.
- Initial capacity of ArrayList is 10 elements.
- If ArrayList is full then its capacity gets increased by half of its exsiting capacity.
- Ctor declared in ArrayList:
  * public ArrayList() //Initial capacity - 10
  * public ArrayList(int initialCapacity)
  * public ArrayList(Collection<? extends E> c)
- Methods declared in ArrayList:
  * public void ensureCapacity(int minCapacity)
  * public void trimToSize()
  * public void forEach(Consumer<? super E> action)

+ Vector:

- It is dynamically growable/shrinkable array.
- It is in java since JDK 1.0. It is legacy class.
- It is synchronized collection.
- It stores data in ordered/sequential fashion [unsorted].

- From Vector, user can access element using integer index.
- In Vector we can insert duplicate elements.
- It allows us to insert multiple null elements.
- We can traverse elements of Vector using Iterator , ListIterator and Enumeration.
- If we want to store elements of reference type in Vector then reference type should contain equals method.
- Initial capacity of Vector is 10 elements.
- If Vector is full then its capacity gets increased its exsiting capacity.
- Ctor declared in Vector:
    * public Vector()
    * Vector(int initialCapacity)
    * public Vector(Collection<? extends E> c)
- Methods declared in Vector:
    * public void ensureCapacity(int minCapacity)
    * public void addElement(E obj)
    * public int capacity() //Not exist in ArrayList
    * public void copyInto(Object[] anArray)
    * public E elementAt(int index)
    * public Enumeration<E> elements()
    * public E firstElement()
    * public void forEach(Consumer<? super E> action)
    * public void insertElementAt(E obj,int index)
    * public E lastElement()
    * public boolean removeElement(Object obj)
    * public void removeElementAt(int index)
    * public void removeAllElements()
    * public void setElementAt(E obj,int index)
    * public void setSize(int newSize)
    * public void trimToSize()

+ Stack:

- It is sub class of Vector
- It is synchronized collection
- The Stack class represents a last-in-first-out (LIFO) stack of objects.
- A more complete and consistent set of LIFO stack operations is provided by the Deque interface:

```
Deque<Integer> stack = new ArrayDeque<Integer>();
```

- Methods declared in Stack:
    * public boolean empty()
    * public E peek()
    * public E pop()
    * public E push(E item)
    * public int search(Object o)

+ LinkedList:

- It implements List and Deque interface
- Its implementation is based on Doubly Linked List.
- Implementation of LinkedList is not synchronized.
- We can make it synchronized using Collections.synchronizedList() method.
- It stores data in ordered/sequential fashion [unsorted].
- From LinkedList, user can access element using integer index.
- In LinkedList we can insert duplicate elements.
- It allows us to insert multiple null elements.
- If we want to store elements of reference type in LinkedList then reference type should contain equals method.
- Ctor declared in LinkedList:
    * public LinkedList()
    * public LinkedList(Collection<? extends E> c)
- Methods declared in LinkedList:
    * public void addFirst(E e)
    * public void addLast(E e)
    * public Iterator<E> descendingIterator()
    * public E getFirst()
    * public E getLast()
    * public E element()
    * public E removeFirst()
    * public E removeLast()

+ Enumeration:

- It is interface declared in java.util package
- Using Enumeration we can traverse collection only in forward direction.
    - The functionality of this interface is duplicated by the Iterator interface. In addition, Iterator adds an optional remove operation, and has shorter method names.
- Methods of Enumeration:

+ boolean hasMoreElements()
+ E nextElement()

- e.g

```
Vector<Integer> vector = new Vector<>();
vector.add(10);
vector.add(20);
vector.add(30);
Enumeration<Integer> e = vector.elements();
while( e.hasMoreElements())
{
        int element = e.nextElement();
        System.out.println(element);
}
```

+ **Iterator:**

- It is a interface declared in java.util package
- Iterator takes the place of Enumeration in the Java Collections Framework.
- Iterators differ from enumerations in two ways:
    * Iterators allow the caller to remove elements from the underlying collection during the iteration.
    * Method names have been improved.
- Methods of Iterator:
    * boolean hasNext()
    * E next()
    * default void remove()
    * default void forEachRemaining(Consumer<? super E> action)
- e.g

```
Vector<Integer> vector = new Vector<>();
vector.add(10);
vector.add(20);
vector.add(30);
Iterator<Integer> itr = vector.iterator();
while( itr.hasNext())
{
        int element = itr.next();
        System.out.println(element);
```

```
}
```

+ ListIterator:

- It is interface declared in java.util package
- It is sub interface of Iterator
- It is designed to use with List Collection only.
- It allows the programmer to traverse the list in either direction
- During travsersing we can add and remove elements
- Methods declared in ListIterator
  - * boolean hasNext()
  - * E next()
  - * boolean hasPrevious()
  - * E previous()
  - * void add(E e)
  - * void set(E e)
  - * void remove()
  - * int nextIndex()
  - * int previousIndex()

- e.g

```
Vector<Integer> vector = new Vector<>();
vector.add(10);
vector.add(20);
vector.add(30);

ListIterator<Integer> itr = vector.listIterator();
while( itr.hasNext())
        System.out.print(itr.next()+"  ");                    //10 20 30
System.out.println();
while(itr.hasPrevious())
        System.out.print(itr.previous()+"       ");           //30 20 10
```

+ ConcurrentModificationException:

- This exception may be thrown by methods that have detected concurrent modification of an object when such modification is not permissible.
- For example, it is not generally permissible for one thread to modify a Collection while another thread is iterating over it. In general, the results of the iteration are undefined under these

circumstances. Some Iterator implementations may choose to throw this exception if this behavior is detected.

- Iterators that do this are known as fail-fast iterators.

- e.g

```
List<String> list = new ArrayList<String>();
list.add("1");
list.add("2");
list.add("3");
list.add("4");
list.add("5");
Iterator<String> itr = list.iterator();
while( itr.hasNext())
{
        String value = itr.next();
        System.out.println("Value     :         "+value);
        if( value.equals("3"))
                list.remove("3");       //ConcurrentModificationException
}
```

+ Set:

- It is sub interface of Collection.
- HashSet, LinkedHashSet, TreeSet etc. implements Set interface. These are Set Collections.
- Set models the mathematical set abstraction.
- The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.
- Methods of Set interface:
    * boolean add(E e)
    * boolean addAll(Collection<? extends E> c)
    * boolean contains(Object o)
    * boolean containsAll(Collection<?> c)
    * boolean remove(Object o)
    * boolean removeAll(Collection<?> c)
    * boolean retainAll(Collection<?> c)
    * void clear()
    * boolean isEmpty()
    * int size()
    * Object[] toArray()

* <T> T[] toArray(T[] a)

* default boolean removeIf(Predicate<? super E> filter)
* default Stream<E> stream()
* default Stream<E> parallelStream()
* default Spliterator<E> spliterator()

+ SortedSet:

- It is sub interface of Set.
- It maintains its elements in ascending order, sorted according to the elements' natural ordering (Comparable) or according to a Comparator provided at SortedSet creation time.
- Methods declared in SortedSet:
  * E first()
  * E last()
  * SortedSet<E> headSet(E toElement)
  * SortedSet<E> tailSet(E fromElement)
  * SortedSet<E> subSet(E fromElement,E toElement)

+ NavigableSet:

- It is a sub interface of SortedSet.
- It is sortedSet having navigation methods.
- TreeSet class Implements NavigableSet interface.
- Methods declared in NavigableSet:
  * E ceiling(E e)
  * E floor(E e)
  * E higher(E e)
  * E lower(E e)
  * E pollFirst()
  * E pollLast()
  * Iterator<E> descendingIterator()
  * NavigableSet<E> descendingSet()

+ TreeSet:

- It is a class which implements NavigableSet interface.
- It is based on TreeMap collection.
- It do not contain duplicate elements
- We can not store null elements in TreeSet.

- It is unsynchronized collection.
- Using Collections.synchronizedSortedSet() method we can make it synchronized.
- It stores data in sorted format on the basis of either Comparable or Comparator.
- If we want to store elements of reference type in TreeSet then Reference type must implement Comparable or Comparator interface.

+ Searching Algorithm:

1. Linear search / Sequential Search
   - We can use it search element in any sorted as well as unsorted collection.
   - If collection contains large number of elements then it is time consuming

2. Binary Search

- We can use it only on sorted array
- It is faster than Linear search

3. Hashing

- If we want to search element in constant time then we should use hashing
- It is based on hashcode. An integer number that can be generated by processing state of the object is called hashcode. Hash function is responsible for generating hashcode.
- If objects are hashed to the same slot then it is called as collision.
- To avoid collision we can use collision removal techniques(Open addressing, separate chaining etc.)
- In seperate chaining, array of linked list is maintained. Each linked list is called as bucket.
- Load Factor = (no of buckets) / bo of elements.

+ HashSet:

- It is a class wich implements Set interface.
- It is based on HashTable.
- It do not contain duplicate elements
- It can contain null element
- It is unsynchronized collection.
- using Collections.synchronizedSet we can make it sychronized.
- HashSet is much faster than TreeSet but do not give guarantee of ordering.
- An instance of Hashtable has two parameters that affect its performance: initial capacity and load factor.
- The capacity is the number of buckets in the hash table, and the initial capacity is simply the capacity at the time the hash table is created.
- One thing worth keeping in mind about HashSet is that iteration is linear in the sum of the number of entries and the number of buckets (the capacity). Thus, choosing an initial capacity that's too high can waste both space and time. On the other hand, choosing an initial capacity that's too low wastes time by copying the data structure each time it's forced to increase its capacity. If you don't specify an

initial capacity, the default is 16. In the past, there was some advantage to choosing a prime number as the initial capacity. This is no longer true. Internally, the capacity is always rounded up to a power of two.

- If we want to add object of reference type in HashSet then refrence type should override equals and hashCode method.
- Ctor declared in HashSet:
    * public HashSet()
    * public HashSet(Collection<? extends E> c)
    * public HashSet(int initialCapacity, float loadFactor)
    * public HashSet(int initialCapacity)

- Methods declared in HashSet:
    * public boolean add(E e)
    * public void clear()
    * public Object clone()
    * public boolean contains(Object o)
    * public boolean isEmpty()
    * public Iterator<E> iterator()
    * public boolean remove(Object o)
    * public int size()
    * public Spliterator<E> spliterator()

+ LinkedHashSet:

- It is a subclass of HashSet
- Its implimentation is based on HashTable and LinkedList.
- It gives gurantee of order of elements.
- It runs nearly as fast as HashSet.
- It do not contain duplicate elements
- It can contain null element
- It is unsynchronized collection.
- using Collections.synchronizedSet we can make it sychronized.

+ Queue:

- It is sub interface of Collection.
- PriorityQueue implements Queue interface
- Queue Collection maintains elements in FIFO.
- Queue implementations generally do not allow insertion of null elements

- Methods declared in Queue:
    * boolean add(E e)
    * E element()
    * boolean offer(E e)
    * E peek()
    * E poll()
    * E remove()

-

	Throws exception	Returns special value
Insert	add(e)	offer(e)
Remove	remove()	poll()
Examine	element()	peek()

+ Deque:

- The name deque is short for "double ended queue" and is usually pronounced "deck".
- It is sub interface of Queue.
- This interface defines methods to access the elements at both ends of the deque.

-

	First Element (Head)		Last Element (Tail)	
	Throws exception	Special value	Throws exception	Special value
Insert	addFirst(e)	offerFirst(e)	addLast(e)	offerLast(e)
Remove	removeFirst()	pollFirst()	removeLast()	pollLast()
Examine	getFirst()	peekFirst()	getLast()	peekLast()

- LinkedList class implements Deque interface.

+ Map:

- It is interface declared in java.util package.
- It is a part of collection framework but it do not extend Collection interface
- This interface takes the place of the Dictionary class, which was a totally abstract class rather than an interface.

- Properties, HashTable, HashMap, LinkedHashMap, TreeMap etc implements Map interface. It is also called as Map collection.
- Map collection stores elements in key/Value pair format.
- A map cannot contain duplicate keys but it can contain duplicate values.
- Each key can map to at most one value.
- Abstract methods declared in Map:
    * void clear()
    * boolean containsKey(Object key)
    * boolean containsValue(Object value)
    * Set<Map.Entry<K,V>> entrySet()
    * V get(Object key)
    * boolean isEmpty()
    * Set<K> keySet()
    * V put(K key,V value)
    * void putAll(Map<? extends K,? extends V> m)
    * V remove(Object key)
    * int size()
    * Collection<V> values()

+ Map.Entry:

- It is nested interface declared in Map.
- Methods of Entry interface:
    * K getKey()
    * V getValue()
    * V setValue(V value)

+ HashTable:

- It is class which implements Map interface.
- Since it has implemented Map interface, it stores elements in key/value pair format.
- In HashTable Key and value can not be null.
- Key must be unique, value can be duplicate
- It is synchronized collection.
- To successfully store and retrieve objects from a hashtable, the objects used as keys must implement the hashCode method and the equals method.
- Ctor declared in HashTable:
    * public Hashtable()
    * public Hashtable(int initialCapacity)

* public Hashtable(int initialCapacity, float loadFactor)
* public Hashtable(Map<? extends K,? extends V> t)

+ HashMap:

- It is Hash table based implementation of the Map interface.
- The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.
- In HashMap key and value can be null
- Key must be unique, value can be duplicate
- Implementation of HashMap is unsynchronized.
- Using Collections.synchronizedMap method we can make it synchronized.
- To successfully store and retrieve objects from a HashMap, the objects used as keys must implement the hashCode method and the equals method.
- Ctor declared in HashMap:
  * public HashMap()
  * public HashMap(int initialCapacity)
  * public HashMap(int initialCapacity, float loadFactor)
  * public HashMap(Map<? extends K,? extends V> t)

+ LinkedHashMap:

- Hash table and linked list implementation of the Map interface, with predictable iteration order.
- This implementation differs from HashMap in that it maintains a doubly-linked list running through all of its entries.
- In LinkedHashMap key and value can be null
- Key must be unique, value can be duplicate
- Implementation of LinkedHashMap is unsynchronized.
- Using Collections.synchronizedMap method we can make it synchronized.

+ SortedMap:

- It is sub interface of Map.
- A SortedMap is a Map that maintains its entries in ascending order, sorted according to the keys' natural ordering, or according to a Comparator provided at the time of the SortedMap creation.
- Methods declared in Sorted map.
  * Comparator<? super K> comparator()
  * K firstKey()
  * SortedMap<K,V> headMap(K toKey)

* K lastKey()
* SortedMap<K,V> subMap(K fromKey, K toKey)
* SortedMap<K,V> tailMap(K fromKey)

+ NavigableMap:

- It is sub interface of SortedMap.
- It is sortedMap having navigation methods. A NavigableMap may be accessed and traversed in either ascending or descending key order.
- Methodes of NavigableMap:
  * Map.Entry<K,V> ceilingEntry(K key)
  * K ceilingKey(K key)
  * NavigableSet<K> descendingKeySet()
  * NavigableMap<K,V> descendingMap()
  * Map.Entry<K,V> firstEntry()
  * Map.Entry<K,V> floorEntry(K key)
  * K floorKey(K key)
  * Map.Entry<K,V> higherEntry(K key)
  * Map.Entry<K,V> lastEntry()
  * Map.Entry<K,V> lowerEntry(K key)
  * K lowerKey(K key)
  * NavigableSet<K> navigableKeySet()
  * Map.Entry<K,V> pollFirstEntry()
  * Map.Entry<K,V> pollLastEntry()

+ TreeMap:

- A Red-Black tree based NavigableMap implementation.
- The map is sorted according to the natural ordering of its keys(Comparable), or by a Comparator provided at map creation time.
- In TreeMap, Key cannot be null but value can be null.
- It is unsynchronized collection.
- To make it synchronized we should use Collections.synchronizedSortedMap() method.
- If we want to use object of reference type as a key then reference type should implement either Comparable or Comparator interface.

+ **Persistence:**
  - It is minor pillar of oops.
  - It is the process of maintain state of the object either file or databases.

+ **File:**
  - A container which holds collection of records on HDD is called file.
  - General classification of files:
    * Binary files (.dat)
      - can open with specific application.
      - requires less processing hence it is faster.
      - e.g. jpg, mp3, .class etc
    * Text files(.txt)
      - can open with any text editor.
      - requires more processing hence it is slower.

+ **Stream:**
  - It is an object which is used to perform operations on file(read/write/append)---

+ **Standard Streams in java:**
  - System.in --> associated with keyboard
  - System.out --> associated with monitor
  - System.err --> associated with monitor

+ **File related terminology:**
  - Path : It consist of root directory, sub directories, path seperator and file names.
    e.g c:\Sandeep\Java\SimpleHello\src\Program.java
  - Absolute path : It is a path of file from root directory.
    e.g c:\Sandeep\Java\SimpleHello\src\Program.java
  - Relative path : It is path of file from curret directory.
    e.g .\src\Program.java
  - Path Seperators:
    Windows - \
    Linux - /
    Mac OS - . (dot)

+ File IO:
  - To manipulate files, we should use interfaces and classes declared in java.io package.
  - Types declared in io package are device independent.

- Interfaces declared in java.io package:
    * FilenameFilter
    * Flushable
    * Closeable
    * DataInput
    * DataOutput
    * ObjectInput
    * ObjectOutput
    * Serializable

- Classes declared in java.io package:
    * Console
    * File

    * InputStream
    * OutputStream
    * FileInputStream
    * FileOutputStream
    * BufferedInputStream
    * BufferedOutputStream
    * DataInputStream
    * DataOutputStream
    * ObjectInputStream
    * ObjectOutputStream
    * PrintStream

    Above classes are required to manipulate binary files

    * Reader
    * Writer
    * FileReader
    * FileWriter
    * BufferedReader
    * BufferedWriter
    * InputStreamReader

* PrintWriter

above classes are required to manipulate text files

+ java.io.File:

- It represents files and directories of of OS.
- Instances of the File class are immutable; that is, once created, the pathname represented by a File object will never change.
- File ctor:
  * File(String pathname)
- Some of the important methods of File class.
  * public boolean createNewFile() throws IOException
  * public boolean delete()
  * public boolean exists()
  * public String getName()
  * public long getFreeSpace()
  * public long getUsableSpace()
  * public long getTotalSpace()
  * public boolean isDirectory()
  * public boolean isFile()
  * public long lastModified()
  * public long length()
  * public File[] listFiles()
  * public File[] listFiles(FileFilter filter)
  * public boolean mkdir()
-

```
File file = new File("/media/sandeep/DOCUMENTS/E-Book/C++");
if( file.exists())
{
    File[] files = file.listFiles(new FilenameFilter()
    {
        @Override
        public boolean accept(File dir, String name)
        {
            if(name.endsWith(".pdf"))
                return true;
```

```
                    return false;
                }
            });
            for (File f : files)
                System.out.println(f.getName());
        }
```

- The java.nio.file package defines interfaces and classes for the Java virtual machine to access files, file attributes, and file systems. This API may be used to overcome many of the limitations of the ava.io.File class.

+ I/O Streams:

- An I/O Stream represents an input source or an output destination. A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.

- Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects. Some streams simply pass on data; others manipulate and transform the data in useful ways.

- No matter how they work internally, all streams present the same simple model to programs that use them: A stream is a sequence of data. A program uses an input stream to read data from a source.

- java.lang.Object

|- java.io.InputStream

|- java.io.OutputStream

|- java.io.Reader

|- java.io.Writer

- Byte Streams :

* Programs use byte streams to perform input and output of 8-bit bytes.

* All byte stream classes are descended from InputStream and OutputStream.

//Program to write single character at a time in file

```
public static void writeRecord( String pathname ) throws Exception
{
    try( FileOutputStream outputStream = new FileOutputStream(pathname))
    {
        char ch = 'A';
```

```
                while( ch <= 'Z' )
                {
                        outputStream.write(ch);
                        ++ ch;
                }
        }
}
//Program to read single character at a time in file
public static void readRecord( String pathname ) throws Exception
{
        try( FileInputStream inputStream = new  FileInputStream(pathname))
        {
                int data;
                while( ( data = inputStream.read( ) ) != -1 )
                {
                        char ch = (char)data;
                        System.out.print(ch+" ");
                }
        }
}
```

* Examples we've seen is unbuffered I/O. This means each read or write request is handled directly by the underlying OS. This can make a program much less efficient, since each such request often triggers disk access, network activity, or some other operation that is relatively expensive.

* To reduce this kind of overhead, the Java platform implements buffered I/O streams. Buffered input streams read data from a memory area known as a buffer; the native input API is called only when the buffer is empty. Similarly, buffered output streams write data to a buffer, and the native output API is called only when the buffer is full.

* There are four buffered stream classes used to wrap unbuffered streams:
BufferedInputStream and BufferedOutputStream create buffered byte streams, while
BufferedReader and BufferedWriter create buffered character streams.

* A program can convert an unbuffered stream into a buffered stream :

```
inputStream = new BufferedInputStream(new FileInputStream(pathname));
outputStream = new BufferedOutputStream(new FileOutputStream(pathname));
```

- Data Streams:

* Data streams support binary I/O of primitive data type values (boolean, char, byte, short, int, long, float, and double) as well as String values.

* All data streams implement either the DataInput interface or the DataOutput interface.

* outputStream = new DataOutputStream(new BufferedOutputStream(...));

* A data output stream lets an application write primitive Java data types to an output stream .

* inputStream = new DataInputStream(new BufferedInputStream(...));

* A data input stream lets an application read primitive Java data types from an underlying input stream.

* Notice that DataStreams detects an end-of-file condition by catching EOFException,

* Also notice that each specialized write in DataStreams is exactly matched by the corresponding specialized read.

- Object Streams:

* Object streams support I/O of objects.

* The object stream classes are ObjectInputStream and ObjectOutputStream.

* These classes implement ObjectInput and ObjectOutput, which are subinterfaces of DataInput and DataOutput.

* readObject() is method of ObjectInputStream and writeObject is method of ObjectOutputStream.

* Process of converting state of the object into bytes is called serialization and converting bytes into Object is called deserialization.

* To serialize state, class must implement Serializable marker interface.

* Without implementing Serializable interface, if we try to serialize state of the object then JVM throws NotSerializableException.

* If we do not serialize state of any specific field then we should declare such field as transient.

* State of static and transient field can not be serialized.

* serialVersionUID is used to ensure that same class (that was used during serialization) is loaded during deserialization. serialVersionUID is used for version control of object.

* e.g. private static final long serialVersionUID = -5410036443708494562L;

* //Serialization

```
stream = new ObjectOutputStream( new BufferedOutputStream(...));
Employee emp1 = new Employee("Sandeep",33,45000.50f);
stream.writeObject( emp1 );
```

```
*       //Deserialization
        stream = new ObjectInputStream(new BufferedInputStream(...));
        Employee emp2 = ( Employee )stream.readObject();
        //TODO
```

- Character Streams:

* The Java platform stores character values using Unicode conventions.

* Character stream I/O automatically translates this internal format to and from the local character set

* I/O with character streams is no more complicated than I/O with byte streams.

* All character stream classes are descended from Reader and Writer.

* Character streams are often "wrappers" for byte streams. The character stream uses the byte stream to perform the physical I/O, while the character stream handles translation between characters and bytes. FileReader, for example, uses FileInputStream, while FileWriter uses FileOutputStream.

* There are two general-purpose byte-to-character "bridge" streams: InputStreamReader and OutputStreamWriter. Use them to create character streams when there are no prepackaged character stream classes that meet your needs.

Java Notes - Chapter 14 - String Handling

+ Using Following classes, we can manipulate strings in java.

- java.lang.Character
- java.lang.String
- java.lang.StringBuffer
- java.lang.StringBuilder
- java.util.StringTokeniizer

+ ASCII versus UNICODE:

- ASCII and Unicode are two character encodings. Basically, they are standards on how to represent different characters in binary so that they can be written, stored, transmitted, and read in digital media.

- The main difference between the two is in the way they encode the character and the number of bits that they use for each.

- ASCII originally used seven bits to encode each character. This was later increased to eight with Extended ASCII to address the apparent inadequacy of the original.

- In contrast, Unicode uses a variable bit encoding program where you can choose between 32, 16, and 8-bit encodings.

- Major advantage of Unicode is that at its maximum it can accommodate a huge number of characters. Because of this, Unicode currently contains most written languages and still has room for even more. This includes typical left-to-right scripts like English and even right-to-left scripts like Arabic. Chinese, Japanese, and the many other variants are also represented within Unicode

- In order to maintain compatibility with the older ASCII, which was already in widespread use at the time, Unicode was designed in such a way that the first eight bits matched that of the most popular ASCII page. So if you open an ASCII encoded file with Unicode, you still get the correct characters encoded in the file.

+ Character:

- It is a wrapper class of char data type.

- In addition, this class provides several methods for determining a character's category (lowercase letter, digit, etc.) and for converting characters from uppercase to lowercase and vice versa.

- Character information is based on the Unicode Standard.

- The char data type are based on the original Unicode specification, which defined characters as fixed-width 16-bit entities.

- The range of legal code points is now U+0000 to U+10FFFF, known as Unicode scalar value.

- The set of characters from U+0000 to U+FFFF is sometimes referred to as the Basic Multilingual Plane (BMP). Characters whose code points are greater than U+FFFF are called supplementary characters.

- The Java platform uses the UTF-16 representation in char arrays and in the String and StringBuffer classes.

- More about utf:

  * Unicode Transformation Format(UTF) is an algorithmic mapping from every unicode code point to unique byte sequence.

  * UTF-8 and UTF-16 are variable length encodings.

  * UTF-8 is most common on web. UTF-16 is used by Java and Windows.UTF-8 and UTF-32 are used by linux and various Unix systems.

  * UTF-16 encodes characters into specific binary sequences using either one or two 16-bit sequences.

+ String :

- It is a final class declared in java.lang package

- It implements Serializable, Comparable<String>, CharSequence interfaces.

- In java String is not built in type. It is a reference type.

- A String represents a string in the UTF-16 format.

- we can create object of String with or without new operator.

  * String str1 = new String("Sandeep"); //OK

  * String str2 = "SunBeam"; //OK

- String str = "Soham" is equivalent to:

  char data[] = {'S', 'o', 'h','a', 'm' };

  String str = new String(data);

- If we try to modify string then new string object gets created i.e. String objects are constant/ immutable. Because String objects are immutable they can be shared.

- The Java language provides special support for the string concatenation operator (+), and for conversion of other objects to strings.

- Some of the ctor's of String:

  * public String();

  * public String(char[] value)

  * public String(byte[] bytes)

  * public String(String original)

  * public String(StringBuffer buffer)

  * public String(StringBuilder builder)

- Methods declared in String:

* public char charAt(int index)
* public int codePointAt(int index)
* public int compareTo(String anotherString)
* public int compareToIgnoreCase(String str)
* public String concat(String str)
* public boolean endsWith(String suffix)
* public boolean equalsIgnoreCase(String anotherString)
* public static String format(String format,Object... args)
* public byte[] getBytes()
* public int indexOf(int ch)
* public int indexOf(String str)
* public String intern()
* public boolean isEmpty()
* public int lastIndexOf(int ch)
* public int lastIndexOf(String str)
* public int length()
* public boolean matches(String regex)
* public String[] split(String regex)
* public boolean startsWith(String prefix)
* public String substring(int beginIndex)
* public String substring(int beginIndex, int endIndex)
* public char[] toCharArray()
* public String toLowerCase()
* public String toUpperCase()
* public String trim()
* public static String valueOf(int i)
* overloaded valueOf methods.

- Some useful twisters:

1.
```
String str1 = new String("SunBeam");
String str2 = new String("SunBeam");
if( str1 == str2 )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Not Equal
```

2.
```
String str1 = new String("SunBeam");
String str2 = new String("SunBeam");
if( str1.equals(str2) )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Equal
```

3.
```
String str1 = "SunBeam";
String str2 = "SunBeam";
if( str1 == str2 )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Equal
```

4.
```
String str1 = "SunBeam";
String str2 = "SunBeam";
if( str1.equals( str2 ) )
        System.out.println("Equal");
```

```
else
        System.out.println("Not Equal");
//Output : Equal
```

5.
```
String str1 = "SunBeam";
String str2 = new String("SunBeam");
if( str1 == str2 )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Not Equal
```

6.
```
String str1 = "SunBeam";
String str2 = new String("SunBeam");
if( str1.equals( str2 ) )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Equal
```

7.
```
String str1 = "Sun"+"Beam";
String str2 = "SunBeam";
if( str1 == str2  )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Equal
```

8.
```
String str = "Sun";
```

```
String str1 = str + "Beam";
String str2 = "SunBeam";
if( str1 == str2  )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Not Equal
```

9.
```
String str = "Sun";
String str1 = (str + "Beam").intern();
String str2 = "SunBeam";
if( str1 == str2  )
        System.out.println("Equal");
else
        System.out.println("Not Equal");
//Output : Equal
```

- Literal strings within the same class in the same package represent references to the same String object.

- Literal strings within different classes in the same package represent references to the same String object.

- Literal strings within different classes in different packages likewise represent references to the same String object.

- Strings computed by constant expressions are computed at compile time and then treated as if they were literals.

- Strings computed by concatenation at run time are newly created and therefore distinct.

- The result of explicitly interning a computed string is the same string as any pre-existing literal string with the same contents.

- String Pooling:

* When compiling source code, compiler must process each literal string and emit the string into metadata. If the same literal string appears several times in source code, emitting all of these strings into metadata will bloat the size of the resulting file.

* To remove this bloat, many compilers write the literal string into metadata only once.

* An ability of a compiler to merge multiple occurrences of single string into single instance can reduce the size. This process is called String Pooling.

+ Regular expression: //Assignment for reader

+ StringBuffer:

- It is a final class declared in java.lang package.
- It implements Serializable and CharSequence interface.
- A string buffer is like a String, but can be modified i.e it is mutable string object.
- It is synchronized.
- equals and hashCode methods are not overiden in StringBuffer.
- It is slower than StringBuilder class.
- Ctor's declared in StringBuffer:
    * public StringBuffer() //Initial capacity - 16 char
    * public StringBuffer(int capacity)
    * public StringBuffer(String str) //Initial capacity - 16 + string length
- Some of the methods declared in StringBuffer:
    * public StringBuffer append(String str)
    * //Other overloaded append methods.
    * public int capacity()
    * public int length()
    * public char charAt(int index)
    * public void setCharAt(int index,char ch);
    * public StringBuffer reverse() //Not available in string
    * public String substring(int start)
    * public void trimToSize()

- Some useful twisters:

```
1.  StringBuffer sb1 = new StringBuffer("Sandeep");
    StringBuffer sb2 = new StringBuffer("Sandeep");
    if( sb1 == sb2)
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Not Equal
```

```
2.  StringBuffer sb1 = new StringBuffer("Sandeep");
    StringBuffer sb2 = new StringBuffer("Sandeep");
    if( sb1.equals(sb2))
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
    //Output : Not Equal
```

```
3.  String str1 = new String("Sandeep");
    StringBuffer str2 = new StringBuffer("Sandeep");
    if( str1  == str2 )        //Compiler Error
        System.out.println("Equal");
    else
        System.out.println("Not Equal");
```

```
4.  String str1 = new String("Sandeep");
    StringBuffer str2 = new StringBuffer("Sandeep");
    if( str1.equals(str2) )
```

```
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
```

+ StringBuilder:

- It is a final class declared in java.lang package.
- It implements Serializable and CharSequence interface.
- A string buffer is like a String, but can be modified i.e it is mutable string object.
- It is unsynchronized.
- equals and hashCode methods are not overiden in StringBuilder
- It is faster than StringBuffer class.
- Ctor's declared in StringBuilder:
    * public StringBuilder() //Initial capacity - 16 char
    * public StringBuilder(int capacity)
    * public StringBuilder(String str) //Initial capacity - 16 + string length
- This class provides an API compatible with StringBuffer, but with no guarantee of synchronization.
- **Some useful Twisters:**

1.
```
StringBuilder sb1 = new StringBuilder("SunBeam");
StringBuilder sb2 = new StringBuilder("SunBeam");
if( sb1 == sb2 )
    System.out.println("Equal");
else
    System.out.println("Not Equal");
//Output : Not Equal
```

2.
```
StringBuilder sb1 = new StringBuilder("SunBeam");
StringBuilder sb2 = new StringBuilder("SunBeam");
if( sb1.equals(sb2) )
```

```
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
3.      StringBuilder sb1 = new StringBuilder("SunBeam");
        StringBuffer sb2 = new StringBuffer("SunBeam");
        if( sb1 == sb2 )        //Compiler error
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
```

```
4.      StringBuilder sb1 = new StringBuilder("SunBeam");
        StringBuffer sb2 = new StringBuffer("SunBeam");
        if( sb1.equals(sb2) )
            System.out.println("Equal");
        else
            System.out.println("Not Equal");
        //Output : Not Equal
```

+ StringTokenizer:

- java.util.StringTokenizer is a class.

- It implements Enumeration interface.

- The string tokenizer class allows an application to break a string into tokens.

- StringTokenizer is a legacy class that is retained for compatibility reasons although its use is discouraged in new code. It is recommended that anyone seeking this functionality use the split method of String or the java.util.regex package instead.

- Ctor's declared in StringTokenizer:

```
* public StringTokenizer(String str)
* public StringTokenizer(String str, String delim)
* public StringTokenizer(String str, String delim, boolean returnDelims)
```

- Methods:

* public int countTokens()
* public boolean hasMoreTokens()
* public String nextToken()
* public String nextToken(String delim)

- example:

```
String str = "www.sunbeaminfo.com";
String delim = ".";
StringTokenizer stk = new StringTokenizer(str,".");
while( stk.hasMoreTokens())
{
        String token = stk.nextToken();
        //TODO
}
```

+ Reflection:

- In Java, the process of analyzing and modifying all the capabilities of a class at runtime is called Reflection.

- Reflection API in Java is used to manipulate class and its members which include fields, methods, constructor, etc. at runtime.

- One advantage of reflection API in java is, it can manipulate private members of the class too.

- The java.lang.reflect package provides many classes to implement reflection in java.

- Methods of the java.lang.Class class are used to gather the complete metadata of a particular class.

+ Application of Reflection:

- RAD tools use reflection to drag n drop components at runtime.

- To access values of private fields, debugger use reflection.

- To implement intellisense feature, IDE's use reflection.

- To create stub object, RMI use reflection.

- TO analyze members of the class, javap tool use reflection.

+ How to get complete information about a class?

1. using getClass() method:

```
Integer obj = new Integer(10);
Class<?> c=  obj.getClass();
```

2. using .class syntax:

```
Class<?> c = Number.class;
```

3. using Class.forName() method:

```
Class<?> c = Class.forName("java.lang.Object");
```

Methods declared in java.lang.Class:

1. public static Class<?> forName(String className) throws ClassNotFoundException
2. public Annotation[] getAnnotations()
3. public ClassLoader getClassLoader()
4. public Constructor<?>[] getConstructors()throws SecurityException
5. public Field getField(String name) throws NoSuchFieldException, SecurityException
6. public Field[] getFields() throws SecurityException
7. public Method getMethod(String name, Class<?>... parameterTypes);
8. public Method[] getMethods() throws SecurityException

9. public Class<?>[] getInterfaces()

10. public int getModifiers()

11. public String getName()

12. public Package getPackage()

13. public InputStream getResourceAsStream(String name)

14. public String getSimpleName()

15. public T newInstance() throws InstantiationException, IllegalAccessException

+ Classes declared in java.lang.reflect Package :

Following is a list of various Java classes in java.lang.package to implement reflection.

Field: This class is used to gather declarative information such as datatype, access modifier, name and value of a variable.

Method: This class is used to gather declarative information such as access modifier, return type, name, parameter types and exception type of a method.

Constructor: This class is used to gather declarative information such as access modifier, name and parameter types of a constructor.

Modifier: This class is used to gather information about a particular access modifier.

+ Middleware example:

```
class Convert
{
  public static Object changeType(String strValue, Parameter parameter )
  {
      switch( parameter.getType().getSimpleName())
      {
      case "int":
        return Integer.parseInt(strValue);
      }
      return null;
  }
}
public class Program
{
```

```
public static void main(String[] args)
{
    try(Scanner sc = new Scanner(System.in))
    {
      System.out.print("F.Q. Class Name        :        ");
      String className = sc.nextLine();
      Class<?> c = Class.forName(className);
      System.out.print("Method Name   :        ");
      String methodName = sc.nextLine();
      for( Method method : c.getMethods())
      {
          if( method.getName().equalsIgnoreCase(methodName))
          {
            Parameter[] parameters = method.getParameters();
            Object[] arguments = new Object[ parameters.length ];
            for( int index = 0; index < parameters.length; ++ index )
            {
                System.out.println(parameters[ index ].getName()+" : ");
                arguments[ index ] = Convert.changeType(sc.nextLine(), parameters[ index ]);
            }
            Object result = method.invoke(c.newInstance(), arguments);
            System.out.println("Result :        "+result);
          }
      }
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
} }
```

+ Metadata:

- Data about data is called as metadata.

- Metadata has may uses. Here are some of them are:

  * Metadata removes need for header and library files when compiling.

  * IDE's intellisense feature parses metadata to tell us what methods type offers.

  * Metadata allows an object fields to be serialized into a memory block, remoted to another machine and then de-serialized , recreating the object and its state on the remote machine.

  * Metadata allows the garbage collector to track lifetime of objects.

+ Annotations:

- Using annotations we can add metadata programatically.

- Annotations have no direct effect on the operation of the code they annotate.

- Annotations have a number of uses, among them:

  * Information for the compiler: Annotations can be used by the compiler to detect errors or suppress warnings.

  * Compile-time and deployment-time processing: Software tools can process annotation information to generate code, XML files, and so forth.

  * Runtime processing â€" Some annotations are available to be examined at runtime.

  * Many annotations replace comments in code.

+ Annotations Basics:

- The at sign character (@) indicates to the compiler that what follows is an annotation.

  e.g @Override

  public String toString { ... }

- The annotation can include elements, which can be named or unnamed, and there are values for those elements:

  @SuppressWarnings(value = "unchecked")

  void myMethod() { ... }

  If there is just one element named value, then the name can be omitted, as in:

  @SuppressWarnings("unchecked")

  void myMethod() { ... }

- If the annotation has no elements, then the parentheses can be omitted.

- If the annotations have the same type, then this is called a repeating annotation:

```
@Author(name = "Jane Doe")
@Author(name = "John Smith")
class MyClass { ... }
```

+ Types of Annotation

- Marker Annotation
- Single-Value Annotation
- Multi-Value Annotation

+ Declaring an Annotation Type:

- The annotation type definition looks similar to an interface definition where the keyword interface is preceded by the at sign (@)

```
@interface ClassPreamble
{
  String author();
  String date();
  int currentRevision() default 1;
  String lastModified() default "N/A";
  String lastModifiedBy() default "N/A";
  // Note use of array
  String[] reviewers();
}
```

-
```
@ClassPreamble (
author = "Sandeep Kulange",
date = "7/02/2017",
currentRevision = 6,
lastModified = "15/07/2016",
lastModifiedBy = "Smita Kadam",
// Note array notation
```

```
reviewers = {"Nikhil", "Ritika", "Sangita"}
)
public class Generation3List extends Generation2List
{    }
```

- To make the information in @ClassPreamble appear in Javadoc-generated documentation, you must annotate the @ClassPreamble definition with the @Documented annotation

```
@Documented
@interface ClassPreamble
{       }
```

+ Annotations That Apply to Other Annotations:

- @Retention
- @Documented
- @Target
- @Inherited
- @Repeatable

- @Retention: It is used to specify to what level annotation will be available.

RetentionPolicy	Availability
SOURCE	refers to the source code, discarded during compilation.
CLASS	refers to the .class file, available to compiler but not to JVM.
RUNTIME	refers to the runtime, available to java compiler and JVM .

- @Documented: It is used to signal to the JavaDoc tool that your custom annotation should be visible in the JavaDoc for classes using your custom annotation.

- @Traget: This meta annotation says that this annotation type is applicable for only the element (ElementType) listed. Possible values for ElementType are:

1. CONSTRUCTOR
2. FIELD
3. LOCAL_VARIABLE
4. METHOD
5. PACKAGE
6. PARAMETER
7. TYPE

- @Inherited: It signals that a custom Java annotation used in a class should be inherited by subclasses inheriting from that class.

- @Repeatable: It is java 8 annotation. There are some situations where you want to apply the same annotation to a declaration or type use.

```
@Alert(role="Manager")
@Alert(role="Administrator")
public class UnauthorizedAccessException extends SecurityException
{ ... }
```

+ Enum :

- If we want to give name to the literals then we should use Enum.
- Enum is reference type in java.
- All enums are implicitly extended from java.lang.Enum class which is abstract.
- Methods declared in java.lang.Enum class:
    * public final String name()
    * public final int ordinal()
    * public static <T extends Enum<T>> T valueOf(Class<T> enumType, String name)
    * public final Class<E> getDeclaringClass()

+ Enum Declaration :

- enum Color

```
{
    RED, BLUE, GREEN;
}
```

- After compilation enum looks like as follows:

```
final class Color extends java.lang.Enum<Color>
 {
        public static final Color RED;

        public static final Color GREEN;

        public static final Color BLUE;

        public static Color[] values();

        public static Color valueOf( String color );
 }
```

- Since every enum is implicitly final class we cannot extend enum.
- Compiler adds values and valueOf method at compile time in enum.
- As shown in code above enum members are objects of enum.

- Let us see how to access it:

```
Color[] colors = Color.values();
for (Color color : colors)
{
   String name = color.name();
   int ordinal = color.ordinal();
   System.out.printf("%-15s%-5d\n",name, ordinal);
}
```

- Now Let us see how to give name to the literals:

```
enum Day
{
   SUN("SunDay",1),MON("MonDay",2),TUES("TuesDay",3);
   private String name;
   private int number;
   private Day(String name,int number)
   {
        this.name = name;
        this.number = number;
   }
   public String getName()
   {
        return name;
   }
   public int getNumber()
   {
        return number;
   }
}
```

- Let us see how to access it:

```
Day[] days = Day.values();
for (Day day : days)
{
   String name = day.name();
   int ordinal = day.ordinal();
   String dayName = day.getName();
   int dayNumber = day.getNumber();
}
```

- we can override methods in enum.

+ **Concurrency:**

- It is minor pillar of oops.
- A large subset of programming problems can be solved using sequential programming. For some problems, however, it becomes convenient or even essential to execute several parts of program in parallel.
- Parallel programming can produce great improvements in program execution speed.
- Process of executing multiple task parallely is called concurrency.

+ **Multitasking:**

- An ability of any operating system to execute single task at a time is called single tasking.
- An ability of any operating system to execute multiple task at a time is called multitasking.
- Multitasking can be process based/thread based multi-tasking.

+ **Process Based Multitasking:**

- Program in execution is called process / running instance of a program is called process.
- More about Process:
    * Each process run in separate address space.
    * Each process has its own set of resources.
    * A process contains at least one thread.
    * Creating and disposing process is relatively time consuming task.
- When CPU executes multiple processes, it first save state of process into process control block (PCB) and then switches to another process. It is called context switch. But context switch is heavy hence process based multi-tasking is called heavy weight multitasking.

+ **Thread Based Multitasking:**

- Light weight process is called thread. In other words it is a seperate path of execution which runs independantly.
- If we want to utilize harware resources (CPU) efficiently then we should use thread.
- More about thread:
    * Threads belonging to the same process share the process's address space and code.
    * Thread creation is very echonomical.
- Since thread gets executed inside same process they do not require PCB. i.e context switching is not required. So thread based multitasking is faster than process based multitasking.

+ **MultiThreading:**

- If we create application using single thread then it is called as single threaded application and if we create application using multiple threads then it is called as multithreaded application.
- Java is multithreaded programming language.
- Java threading is based on the low level pthreads approach which comes from C.

- Java's multithreading is preemptive, which means that a scheduling mechanism provides time slices for each thread and context switching to the another thread, so that each one is given a reasonable amount of time to drive its task.

- If JVM starts execution of java application then it also starts execution of two threads:
  * Main Thread
    # It is user thread.
    # It is responsible for invoking main method.
  * Garbage Collector
    # It is daemon thread.
    # It is responsible for reclaming memory of unused object.

- If we want to manipulate threads in java then we should use types declared in java.lang package.

+ Thread creation in Java:

- In java, we can create thread using two ways:
  1. By implementing java.lang.Runnable interface.
  2. By extending java.lang.Thread class.
- Runnable is functional interface which contains "void run()" method.
- Thread is a class which implements Runnable interface.
- Members of Thread class:

```
-----------------------------------------------------
Nested Type
-----------------------------------------------------
class Thread
{
        public static enum State
        {

NEW,RUNNABLE,BLOCKED,WAITING,TIMED_WAITING,TERMINATED
        }
}
-----------------------------------------------------
Fields
-----------------------------------------------------
public static final int MIN_PRIORITY = 1
public static final int NORM_PRIORITY = 5
public static final int MAX_PRIORITY = 10
-----------------------------------------------------
```

Constructor's

```
public Thread()
Thread(String name)
Thread(Runnable target)
Thread(Runnable target, String name)
Thread(ThreadGroup group, Runnable target)
Thread(ThreadGroup group, Runnable target, String name)
```

Method's

```
public static Thread currentThread();
public final String getName()
public final void setName(String name)
public final int getPriority()
public final void setPriority(int newPriority)
public Thread.State getState()
public final ThreadGroup getThreadGroup()
public final boolean isAlive()
public final boolean isDaemon()
public final void setDaemon(boolean on);
public final void join() throws InterruptedException
public static void sleep(long millis) throws InterruptedException
public static void yield()
public void start()
public final void suspend()    //@Deprecated
public final void resume()     //@Deprecated
public final void stop()       //@Deprecated
public void destroy()          //@Deprecated
```

+ Thread creation using Runnable interface:

```
class CThread implements Runnable
{
   private Thread thread;
   public CThread( String name )
   {
```

```
        this.thread = new Thread(this, name);
        this.thread.start();
    }
    @Override
    public void run()
    {
        //TODO : Business logic
    }
}
public class Program
{
    public static void main(String[] args)
    {
        new CThread("User Thread#1");
    }
}
```

+ Thread creation using Thread class:

```
class CThread extends Thread
{
    public CThread( String name )
    {
        super( name );
        this.start();
    }
    @Override
    public void run()
    {
        //TODO : Business logic
    }
}
public class Program
{
    public static void main(String[] args)
    {
        new CThread("User Thread#1");
```

```
    }
}
```

+ Difference between process based and thread based multitasking:

- If class do not extends any class then we can create thread by extending Thread class.
- If class extends any class then we can create thread by implementing Runnable interface.
- If class extends Thread then sub class can ommit from threading.
- If class implements Runnable then all sub classes must participate in threading.

+ Relation between th.start() and run() method:

thread.start() do not call run method rather it register's the thread with operating system via JVM and then run method gets called on object which implements either Runnable interface or extends Thread class.

+ Types of thread:

1. User Thread:
   - User thread is also called as non daemon thread.
   - New thread created is by default treated as user thread.
   - Life span / execution of user thread is do not depends on execution of daemon thread.
2. Daemon Thread:
   - Daemon thread is also called as background thread.
   - using th.setDaemon(true) we can convert user thread into daemon thread.
   - Life span / execution of daemon thread is depends on execution of user thread.

+ Thread termination:

Thread can terminate due to following reason:

1. if run method executes successfully.
2. during execution of run if any exception occurs
3. during execution of run, if jvm encounters return statement.

+ Thread Life Cycle:

A thread can be in any one of four states:

1. New:

A thread remains in this state only momentarily, as it is being created. The thread is not yet running. When thread is in new state, the program has not started executing code inside of it.

2. Runnable:

Once you invoke start method, the thread is in runnable state.

This means that a thread can be run when the time-slicing mechanism has CPU cycles available for thread. Thus thread might or might not be running at any moment but

there is nothing to prevent it from being run if the scheduler can arrange it. That is, it's not dead or block.

3. Blocked and Waiting:

The thread can be run, but something prevents it. While a thread is in blocked state, the scheduler will simply skip it and not give any CPU time. Until thread reenters the runnable state it won't perform any operations.

A task can become blocked for the following reason:

- You have put the task to sleep by calling Thread.sleep(...) method.
- You have suspended the execution of thread with wait
- The task is waiting for some I/O to complete.
- The task is trying to call synchronized method on another object, and that object's lock is not available because it has already been acquired by another task.

4. Dead:

A thread in dead or terminated state is no longer schedulable and will not receive any CPU time. Its task is completed and it is no longer runnable.

A thread can terminate due to following reason:

- It dies a natural death because the run method exist normally.
- It dies abruptly because an uncaught exception terminates the run method.

+ Thread Priorities:

- In java programming language every thread has a priority. Thread Priority talks about how much time should spent for thread.
- By default, a thread inherits priority of the thread that constructed it.
- We can set the priority to any value between MIN_PRIORITY and MAX_PRIORITY.
- We can increase or decrease priority of thread using setPriority() method.

```
e.g th.setPriority( Thread.NORM_PRIORITY + 3 );
```

- If the priority is not in the range MIN_PRIORITY to MAX_PRIORITY then jvm throws IllegalArgumentException.
- Whenever thread scheduler has a chance to pick a new thread, it prefers thread with higher priority. However thread priorities are highly system dependent. When virtual machine relies on thread implementation of the host platform, the java thread priorities are mapped to the priority levels of the host platform, which may have more or fewer thread priority levels.

- For example, Windows has 7 priority levels. Some of the java priorities will map to the same OS level. In Oracle JVM for linux, thread priorities are ignored all together- all threads have same priority.

+ Yielding:

- If you know that you've accomplished what you need to during one pass through a loop in your run() method, you can give a hint to the thread scheduling mechanism that you have done enough and that some other task might as well have the CPU. This hint takes the form of the yield() method.
- When you call yield(),you are suggesting that other threads of the same priority might be run.

+ Joining Thread:

- One thread may call join on another thread to wait for the second thread to complete before proceeding.
- You may also call join() with timeout argument so that if the target thread doesn't finish in that period of time, the call to join returns anyway.

+ Critical Section:

- Sometimes, you only want to prevent multiple thread access to part of the code inside method instead of entire method. This section of a code is called critical section.

+ Resource Locking / UnLocking:

- When multiple threads try to access same object at the same time, it is called as "race condition".
- This may result in corrupting object state and hence will get unexpected results.
- To avoid this Java provides sync primitive known as "monitor".
- Each java object is associated with a monitor object. The monitor can be locked or unlocked by the thread.
- If monitor is already locked, another thread trying lock it will be blocked until monitor is unlocked.
- The thread who had locked monitor, is said to be owner of monitor and only that thread can unlock the monitor.
- Java provides "synchronized" keyword to deal with monitors.
  - synchronized block:

```
* example:
        void someMethod()
        {
                // ...
                synchronized(obj)
                {
                        //TODO
                }
        }
```

- When certain part of the method should be executed by single thread at a time, we can use "synchronized" block.
- When a thread begin execution of the sync block, it will lock the monitor associated with given object (obj) and continue execute within that block.
- Meanwhile if another thread try to execute same sync block, it will try to lock the object monitor again.
- Since object is already locked, the second thread will be blocked, until first thread release the lock (at the end of sync block).
- When first thread release the lock, the waiting second thread will acquire it and continue to execute the block.

- Synchronized Method:

* When non-static synchronized method is invoked by the thread, it will lock the monitor associate with current object ("this"). When method completes, thread will release the lock.

* When static synchronized method is invoked by the thread, it will lock the monitor object associated with "ClassName.class" object.

* This ensure that two threads cannot execute same synchronized method on the same object.

example:

```
class Account
{
	private double balance;
	// ...
	public synchronized void withdraw(double amt)
	{
		// get cur balance from db
		// balance = balance - amt;
		// update new balance into db
	}
	public synchronized void deposit(double amt)
	{
		// get cur balance from db
		// balance = balance + amt;
		// update new balance into db
	}
	public double getBalance()
	{
		// get cur balance from db
		// return balance;
```

```
                }
            }
```

* a1.withdraw(...) and a1.withdraw(...) executed by two different threads at the same time will not execute parallely. (As "a1" object is locked by thread1 and thread2 must wait for it).

* a1.withdraw(...) and a1.deposit(...) executed by two different threads at the same time will not execute parallely. (As "a1" object is locked by thread1 and thread2 must wait for it).

* a1.withdraw(...) and a2.withdraw(...) executed by two different threads at the same time can execute parallely. As lock is acquired by two threads on two different objects.

* a1.withdraw(...) and a2.getBalance(...) executed by two different threads at the same time can execute parallely. Because getBalance() method is not synchronized and hence the thread is neither trying lock the object nor will get blocked.

* If all methods in a class are synchronized, only one thread can perform any operation on object of that class. Such class is called as "synchronized" / "thread-safe" class. E.g. StringBuffer, Vector, Hashtable, etc.

+ Thread Synchronization:

- When multiple threads on same monitor try to communicate with each other then it is called inter thread communication.
- Inter thread communication implies synchronization.
- using wait(), notify() and notifyAll() methods we can achieve synchronoization.
- Since we perform synchronization and notification mechanism on objects monitor, wait / notify / notiifyAll belongs to java.lang.Object.
- Consider the following example

```
class Program
{
    void someMethod()throws InterruptedException
    {
        String str = new String("Hello");
        synchronized( this )
        {
            str.wait(); //Exception : IllegalMonitorStateException
        }
    }
}
```

- In multiprocessor environment, if we want to sync data between the threads then we should declare data as volatile.

+ Deadlock:

- It is possible for one task to get stuck waiting for another task, which in turns waits for another task and so on, until the chain leads back to task waiting on the first one. You get continuous loop of tasks waiting on each other and no one can move. This is called deadlock.

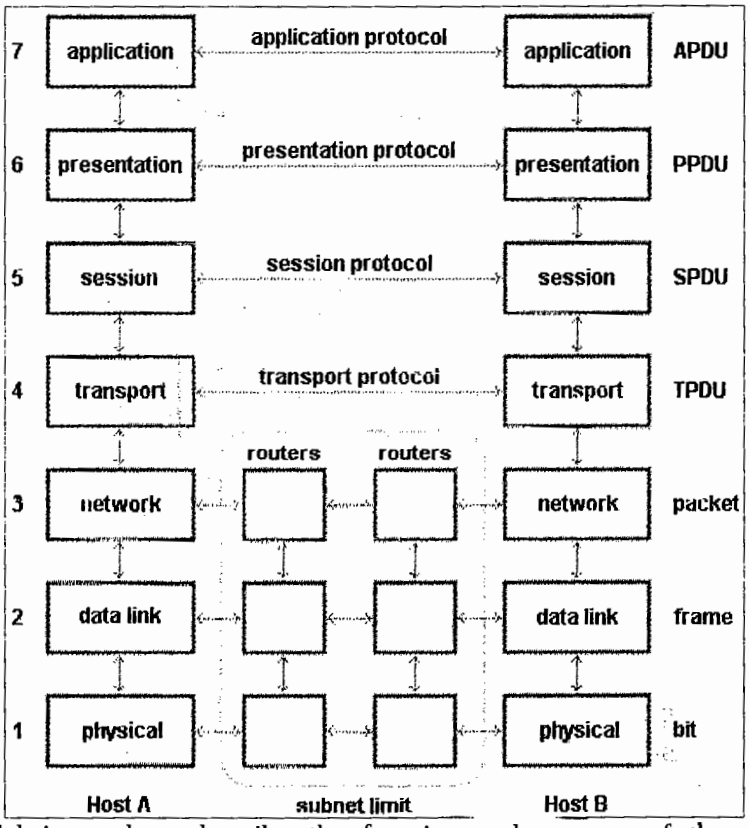
- To avoid dead lock programmer should write logic.

1.OSI reference model:

The OSI model has been the de facto reference model for networking protocols since the mid-1990s. The OSI model is formally known as Open Systems Interconnection (OSI) model ISO/IEC 7498-1.

The OSI model is a conceptual model rather than a technical specification. This means it is used to discuss, describe, compare, and contrast actual technologies rather than directly mandating elements of technology.

The OSI model is comprised of seven layers, with layer one positioned at the bottom of the layer stack, and layer seven at the top. The layers have assigned names as well as number references.



The OSI model is used to describe the function and purpose of the various elements in network communications.

In theory, data is received by the protocol stack into layer 7, the application layer, from software. The received data is labeled as a service data unit (SDU). Each layer adds its own layer-specific header to the SDU, thus creating a payload data unit (PDU). The PDU is then passed down to the next layer below, where it becomes the SDU of that layer. This process of traversing down the layer stack is known as encapsulation. Once layer 1, the physical layer, receives the PDU from layer 2, the data link layer, the data is transmitted over the network medium (i.e., twisted pair cable, fiber optic cable, or wireless).

When a network interface receives a signal of data from the network medium, it processes the PDU in reverse. This reverse unpacking process is known as de-encapsulation. Each layer reads its corresponding header of the PDU, processes and removes the header from the PDU, creating an SDU, then passes the SDU to the layer above. This is repeated until layer 7, the application layer, receives its PDU and passes the actual data to software.

Layer 7 - Application

Layer 7, the application layer, is the interface between the protocol stack and application software. The software might be client utilities or server services.

The application layer is assigned the responsibility to check whether a remote communication partner is available, confirm communications with that partner are possible, and evaluate whether or not there are sufficient resources to maintain a communication.

Layer 6 – Presentation

This layer is usually part of an operating system and converts incoming and outgoing data from one presentation format to another.

Layer 5 - Session

The session layer, or layer 5, manages the connections between computers. Connection management includes establishing, maintaining, and terminating the links between networked systems. This layer provides for full-duplex, half-duplex, and simplex communications.

Layer 4 - Transport

This layer manages packetization of data, then the delivery of the packets, including checking for errors in the data once it arrives. On the Internet, TCP and UDP provide these services for most applications as well.

Layer 3 - Network

This layer handles the addressing and routing of the data (sending it in the right direction to the right destination on outgoing transmissions and receiving incoming transmissions at the packet level).

Internet Protocol (IP) is the most recognized protocol that operates at this layer. Currently, IPv4 is the most widely used version; however, IPv6 is quickly gaining in popularity. IPv6 got its official global Internet kick-off on June 6, 2012.

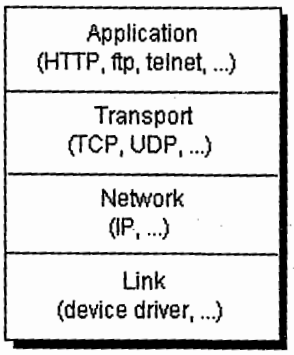
Layer 2 - Data Link

This layer sets up links across the physical network, putting packets into network frames. The most common standard technologies in use at the data link layer are Ethernet and Wireless.

Layer 1 - Physical

The physical layer, or layer 1, is the interface between the logical software of the network protocol and the hardware network interface card. It is at this layer that the conversion from the binary data of the layer 2 PDU occurs into the transmission technology encoding of the message bits, such as voltage variations, light pulses, or radio wave modulations.

2. Networking Basics:



Computers running on the Internet communicate to each other using either the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), as this diagram illustrates:

When you write Java programs that communicate over the network, you are programming at the application layer. Typically, you don't need to concern yourself with the TCP and UDP layers. Instead, you can use the classes in the java.net package. These classes provide system-independent network communication. However, to decide which Java classes your programs should use, you do need to understand how TCP and UDP differ.

TCP

When two applications want to communicate to each other reliably, they establish a connection and send data back and forth over that connection. This is analogous to making a telephone call. If you want to speak to Aunt Beatrice in Kentucky, a connection is established when you dial her phone number and she answers. You send data back and forth over the connection by speaking to one another over the phone lines. Like the phone company, TCP guarantees that data sent from one end of the connection actually gets to the other end and in the same order it was sent. Otherwise, an error is reported.

TCP provides a point-to-point channel for applications that require reliable communications. The Hypertext Transfer Protocol (HTTP), File Transfer Protocol (FTP), and Telnet are all examples of applications that require a reliable communication channel. The order in which the data is sent and received over the network is critical to the success of these applications. When HTTP is used to read from a URL, the data must be received in the order in which it was sent. Otherwise, you end up with a jumbled HTML file, a corrupt zip file, or some other invalid information.

Definition:

TCP (Transmission Control Protocol) is a connection-based protocol that provides a reliable flow of data between two computers.

UDP

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data, called datagrams, from one application to another. Sending datagrams is much like sending a letter through the postal service: The order of delivery is not important and is not guaranteed, and each message is independent of any other.

Definition:

UDP (User Datagram Protocol) is a protocol that sends independent packets of data, called datagrams, from one computer to another with no guarantees about arrival. UDP is not connection-based like TCP.

For many applications, the guarantee of reliability is critical to the success of the transfer of information from one end of the connection to the other. However, other forms of communication don't require such strict standards. In fact, they may be slowed down by the extra overhead or the reliable connection may invalidate the service altogether.

Consider, for example, a clock server that sends the current time to its client when requested to do so. If the client misses a packet, it doesn't really make sense to resend it because the time will be incorrect when the client receives it on the second try. If the client makes two requests and receives packets from the server out of order, it doesn't really matter because the client can figure out that the packets are out of order and make another request. The reliability of TCP is unnecessary in this instance because it causes performance degradation and may hinder the usefulness of the service.

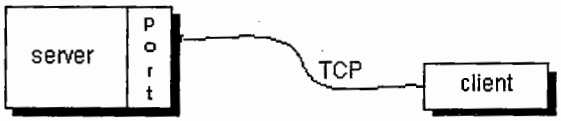
Another example of a service that doesn't need the guarantee of a reliable channel is the ping command. The purpose of the ping command is to test the communication between two programs over the network. In fact, ping needs to know about dropped or out-of-order packets to determine how good or bad the connection is. A reliable channel would invalidate this service altogether.

The UDP protocol provides for communication that is not guaranteed between two applications on the network. UDP is not connection-based like TCP. Rather, it sends independent packets of data from one application to another. Sending datagrams is much like sending a letter through the mail service: The order of delivery is not important and is not guaranteed, and each message is independent of any others.

Note:

Many firewalls and routers have been configured not to allow UDP packets. If you're having trouble connecting to a service outside your firewall, or if clients are having trouble connecting to your service, ask your system administrator if UDP is permitted.

Understanding Ports –



Generally speaking, a computer has a single physical connection to the network. All data destined for a particular computer arrives through that connection. However, the data may be intended for different applications running on the computer. So how does the computer know to which application to forward the data? Through the use of ports.

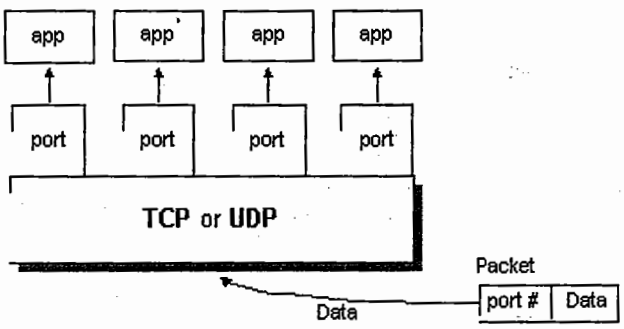
Data transmitted over the Internet is accompanied by addressing information that identifies the computer and the port for which it is destined. The computer is identified by its 32-bit IP address, which IP uses to deliver data to the right computer on the network. Ports are identified by a 16-bit number, which TCP and UDP use to deliver the data to the right application.

In connection-based communication such as TCP, a server application binds a socket to a specific port number. This has the effect of registering the server with the system to receive all data destined for that port. A client can then rendezvous with the server at the server's port, as illustrated here:

Definition:

The TCP and UDP protocols use ports to map incoming data to a particular process running on a computer.

In datagram-based communication such as UDP, the datagram packet contains the port number of its destination and UDP routes the packet to the appropriate application, as illustrated in this figure:



Port numbers range from 0 to 65,535 because ports are represented by 16-bit numbers. The port numbers ranging from 0 - 1023 are restricted; they are reserved for use by well-known services such as HTTP and FTP and other system services. These ports are called well-known ports. Your applications should not attempt to bind to them.

Networking Classes in the JDK

Through the classes in java.net, Java programs can use TCP or UDP to communicate over the Internet. The URL, URLConnection, Socket, and ServerSocketclasses all use TCP to communicate over the network. The DatagramPacket,DatagramSocket, and MulticastSocket classes are for use with UDP.

What Is a URL?

If you've been surfing the Web, you have undoubtedly heard the term URL and have used URLs to access HTML pages from the Web.

It's often easiest, although not entirely accurate, to think of a URL as the name of a file on the World Wide Web because most URLs refer to a file on some machine on the network. However, remember that URLs also can point to other resources on the network, such as database queries and command output.

URL is an acronym for Uniform Resource Locator and is a reference (an address) to a resource on the Internet.

A URL has two main components:

Protocol identifier: For the URL http://example.com, the protocol identifier is http.

Resource name: For the URL http://example.com, the resource name is example.com.

Note that the protocol identifier and the resource name are separated by a colon and two forward slashes. The protocol identifier indicates the name of the protocol to be used to fetch the resource. The example uses the Hypertext Transfer Protocol (HTTP), which is typically used to serve up hypertext documents. HTTP is just one of many different protocols used to access different types of resources on the net. Other protocols include File Transfer Protocol (FTP), Gopher, File, and News.

The resource name is the complete address to the resource. The format of the resource name depends entirely on the protocol used, but for many protocols, including HTTP, the resource name contains one or more of the following components:

Host Name

The name of the machine on which the resource lives.

Filename

The pathname to the file on the machine.

Port Number

The port number to which to connect (typically optional).

Reference

A reference to a named anchor within a resource that usually identifies a specific location within a file (typically optional).

For many protocols, the host name and the filename are required, while the port number and reference are optional. For example, the resource name for an HTTP URL must specify a server on the network (Host Name) and the path to the document on that machine (Filename); it also can specify a port number and a reference.

All About Sockets -

Java Notes - Chapter 19 - Socket Programming

A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.

Using socket we can do interprocess communication. Classes declared in java.net package help us to create socket. Some of the classes declared in java.net package are:

1. URL
2. Proxy
3. InetAddress
4. Socket
5. ServerSocket
6. DatagramSocket
7. DatagramPacket

If we want to get host information then we should use InetAddress class. See the following code.

```
InetAddress hostInfo = InetAddress.getLocalHost();
String hostName = hostInfo.getHostName();
String hostAddress = hostInfo.getHostAddress();
```

Socket programming using TCP protocol:

Client:

```
final int portNumber = 5429;
try( Socket socket = new Socket("localhost",portNumber))
{
   try( DataInputStream dis = new DataInputStream(socket.getInputStream());
        DataOutputStream dos = new DataOutputStream(socket.getOutputStream()))
   {
        //TODO : Communication logic
   }
}
catch(Exception ex)
{       ex.printStackTrace(); }
```

Server:

```
final int portNumber = 5429;
try( ServerSocket server = new ServerSocket(portNumber))
{
   Socket socket = server.accept();
   try( DataInputStream dis = new DataInputStream(socket.getInputStream());
        DataOutputStream dos = new DataOutputStream(socket.getOutputStream()))
   {
```

```
                //TODO : Communication logic
            }
        }
        catch(Exception ex)
        {       ex.printStackTrace();  }
```

If you want to create multi client chat application then we should use thread.

Server should look like this:

```
while( true ){
        Socket socket = server.accept();
        Runnable r = new CommunicationHandler( socket );
}
class CommunicationHandler implements Runnable{
        private Thread thread;
        private Socket socket;
        public CommunicationHandler( Socket socket ){
                this.socket = socket;
                this.thread = new Thread( this );
                this.thread.start();
        }
        public void run( )      {
                DataInputStream dis = new DataInputStream( socket.getInputStream());
                DataOutputStream dis = new DataOutputStream( socket.getOutputStream());
                //  Process input and send response
                socket.close();
        }       }
```

+ Remote Method Invocation (RMI):-

- Distributed Architecture:

* If we want to use existing infrastructure without spending cost & time then we should use Distributed architecture.

* The Distributed architecture is the application design in which the programs and actual architecture spread over the network to share the applications and data.

-Distributed Technologies:

1.Microsoft RPC

2.Microsoft DCOM

3.CORBA objects and clients

4.Web server-client on internet

5.Java Remote Method Invocation

+ Java Remote Method Invocation:

- Java is a distributed because Java supports RMI(Remote method Invocation).
- RMI is a pure java solution to Remote Procedure Calls (RPC) and is used to create distributed application in java.
- RMI allows an object running in one JVM(Client side) to invoke methods of object running in another JVM(Server side).
- Other JVM could be on the same machine or a different one.
- RMI enables client and server side communications through server side object(Remote Object)
- RMI performs marshalling/unmarshalling via Java Serialization.
- Any objects that are to be sent over the network must be serializable.
- Serialization : action of encoding of an object into stream of bytes.
- Marshaling : Transfering state of object over internet(n/w) is called marshaling.
- Unmarshaling : reverse of marshalling.
- Thus,objects to be marshalled or unmarshalled must be serializable.

+ Remote Object :

- An object which implements remote interface directly or indirectly is called remote object.
- java.rmi.Remote is a tagging/marker interface .
- Remote interface specifies the interface whose methods can be invoked from a remote JVM.
- User defined interface should extend this interface.
- Each method of a remote interface, an interface that extends java.rmi.Remote, must list RemoteException in its throws clause.
- The client will look into the Naming service i.e RmiRegistry for Remote object registration and if registered, will get a remote reference for the object.

+ RMI Architecture:

-the communication between client and server side is handled by two intermidiate objects

1. Stub Object(Client side)
2. Skeleton Object(Server side)

+ Stub Object :

- It is a client side Object,acts as a proxy to remote server object.
- Stub is a local representative of remote object,which take care name of method to be invoked & number of argument pass to that remote method.
- Communicates with real object over the internet(N/W) by marshalling and unmarshalling the data to and from the server to the client.
- The stub objects use the invoke() method in RemoteRef to forward the method call

+ Skeleton Object:

- It is a Server side Object.
- The skeleton object calls method from remote link,reads the number of parameters and passes the request from stub object to the remote,and return value from the server back to the stub.
- It returns result or RemoteException.

+ RMI Registry:

- A naming service which maps to remote object.
- Essential operations: bind/rebind,unbind,lookup
- bind adds a service entry to the registry

```
public static void bind(String name, Remote obj)
```

- unbind removes a service entry from the registry

```
public static void unbind(String name)
```

- rebind registers/binds the object obj with already registered object with the new name the NewName in the registry.

```
public static void rebind(String NewName, Remote obj)
```

- lookup allows client to find the objects address using object name.

```
public static Remote lookup(String name)
```

- The java.rmi.Naming class provides methods for storing and obtaining references to remote objects in the remote object registry.

+ Steps to create RMI Application:

- server side-

1. Create Remote Object
2. Make it exportable object using unicastRemoteObject class.

3. Register that interface using getRegistery method by using local registry.
   - like, Registry registry = LocalRegistry.getRegistry();
4. rebind that registry,through this we are going to create allias of that object

- Client side-

1. Make Registry object using LocalRegistry.getRegistry() method.
2. through registry object lookup the remote object.

+ Steps to exeute RMI on console :

1. Compile all server side .java files.
2. copy interface .class file into clients bin folder.
3. Compile client side .java files
4. Start RMI Registry
5. Run server
6. Run client

+ RMI Exception (RemoteException) :

The remote method invoked by the client is always executed at server side,if the method throws any exception at server side and if not caught properly they will be sent to the client as the result of execution.

+ Dynamic Client

- For a normal RMI client application it needs the common interface and stub classes at client side.

- But it is not always possible to have these classes at client side,in these cases a dynamic client application provided by another server can be downloaded at client side and invoked the same way as an applet is executed in the browser. The client application when downloaded by RMIClassLoader, will lookup in the remote registry and call the methods. This is called as bootstrapped rmi client.

+ Dynamic Server

- Similarly there can be a dynamic rmi implementation class which can be downloaded at server and registered with the rmiregistry.

- This dynamic class will provide the remote reference and execution by remote client applications. This is called as bootstrapped server.

- In both of these bootstrapped cases all the downloaded classes are subjected to security restrictions imposed by the policy file permissions.

Java Notes - Chapter 21 - Inside the Java Virtual Machine

The Java Virtual Machine: The Java virtual machine is called "virtual" because it is an abstract computer defined by a specification.

The Lifetime of a Java Virtual Machine: A runtime instance of the Java virtual machine has a clear mission in life: to run one Java application. When a Java application starts, a runtime instance is born. When the application completes, the instance dies. If you start three Java applications at the same time, on the same computer, using the same concrete implementation, you'll get three Java virtual machine instances. Each Java application runs inside its own Java virtual machine.

Inside the Java virtual machine, threads come in two flavors: daemon and non- daemon. A daemon thread is ordinarily a thread used by the virtual machine itself, such as a thread that performs garbage collection. The application, however, can mark any threads it creates as daemon threads. The initial thread of an application--the one that begins at main()--is a non- daemon thread.

A Java application continues to execute (the virtual machine instance continues to live) as long as any non-daemon threads are still running. When all non-daemon threads of a Java application terminate, the virtual machine instance will exit. If permitted by the security manager, the application can also cause its own demise by invoking the exit() method of class Runtime or System.

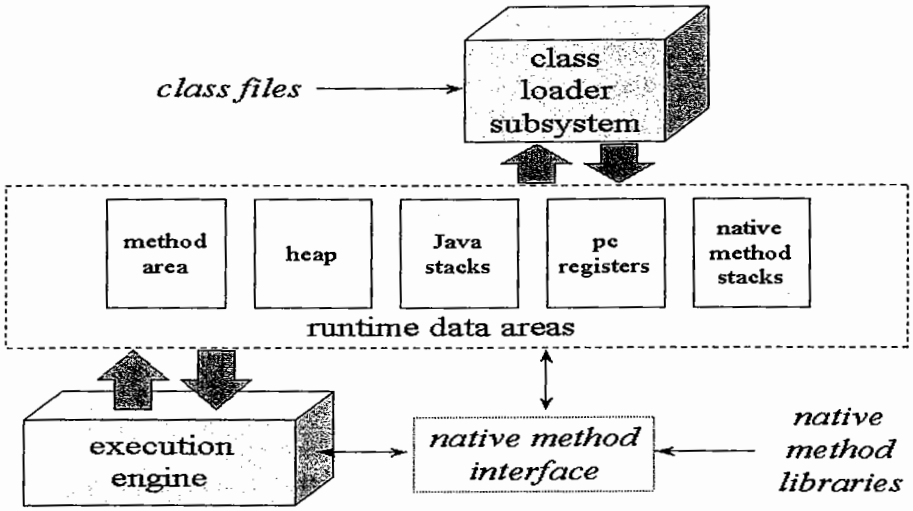
The Architecture of the Java Virtual Machine:

Block diagram of the Java virtual machine that includes the major

subsystems and memory areas described in the specification. Each Java virtual machine has a class loader subsystem: a mechanism for loading

types (classes and interfaces) given fully qualified names. Each Java virtual machine also has an execution engine: a mechanism responsible for executing the instructions contained in the methods of loaded classes.

When a Java virtual machine runs a program, it needs memory to store many things, including bytecodes and other information it extracts from loaded class files, objects the program instantiates, parameters to methods, return values, local variables, and intermediate results of computations. The Java virtual machine organizes the memory it needs to execute a program into several runtime data areas.

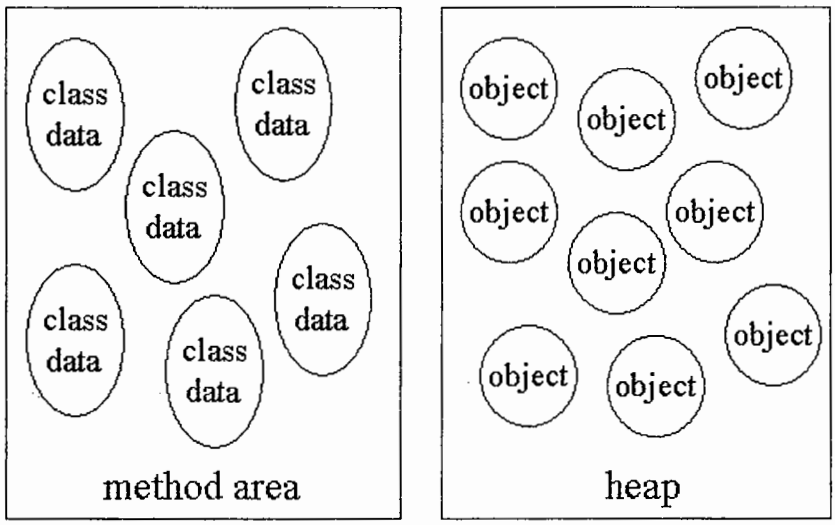


Although the same runtime data areas exist in some form in every Java virtual machine implementation, their specification is quite abstract. Many decisions about the structural details of the runtime data areas are left to the designers of individual implementations.

Different implementations of the virtual machine can have very different memory constraints. Some implementations may have a lot of memory in which to work, others may have very little. Some implementations may be able to take advantage of virtual memory, others may not. The abstract nature of the specification of the runtime data areas helps make it easier to implement the Java virtual machine on a wide variety of computers and devices.

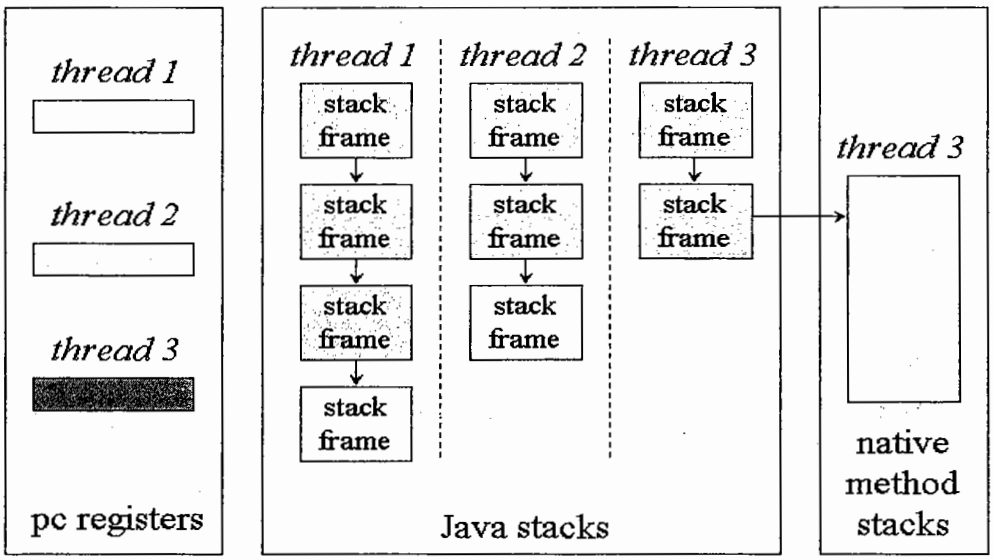
Some runtime data areas are shared among all of an application's threads and others are unique to individual threads. Each instance of the Java virtual machine has one method area and one heap. These areas are shared by all threads running inside the virtual machine. When the virtual machine loads a class file, it parses information about a type from the binary data contained in the class file. It places this type information into the method area. As the program runs, the virtual machine places all objects the program instantiates onto the heap.

As each new thread comes into existence, it gets its own pc register (program counter) and Java stack. If the thread is executing a Java method (not a native method), the value of the pc register indicates the next instruction to execute. A thread's Java stack stores the state of Java (not native) method invocations for the thread. The state of a Java method invocation includes its local variables, the parameters with which it was invoked, its return value (if any), and intermediate calculations. The state of native method invocations is stored in an implementation-dependent way in native method stacks, as well as possibly in registers or other implementation-dependent memory areas.



The Java stack is composed of stack frames (or frames). A stack frame contains the state of one Java method invocation. When a thread invokes a method, the Java virtual machine pushes a new frame onto that thread's Java stack. When the method completes, the virtual machine pops and discards the frame for that method.

The Java virtual machine has no registers to hold intermediate data values. The instruction set uses the Java stack for storage of intermediate data values. This approach was taken by Java's designers to keep the Java virtual machine's instruction set compact and to facilitate implementation on architectures with few or irregular general purpose registers. In addition, the stack-based architecture of the Java virtual machine's instruction set facilitates the code optimization work done by just-in-time and dynamic compilers that operate at run-time in some virtual machine implementations.



Operating System Concepts - Notes

✦ **Operating System Concepts & Linux Programming:**

✦ **Operating System Concepts**

- Introduction to OS
- Introduction to Comp Hardware
- Process Mgmt
- CPU scheduling
- Memory Mgmt
- File & IO Mgmt

✦ **Linux Programming**

- Linux commands
- Shell Scripts
- System Calls

✦ **What is Operating System?**

- OS is intermediary between application programs and computer hardware.
- OS is a resource allocator, which allocates resources (like CPU, RAM, disk, etc) to the running programs as per their requirements.
- OS is a control program, which controls execution of other programs.
- OS CD/DVD = Core OS + Utilities & Applications
- The part of OS which performs basic minimal functionality is called as "Core OS" or "KERNEL".

✦ **OS Functionalities:**

✦ **Process management:**

- Program is a set of instruction given to the machine.
- It is an executable file.
- This file contains exe header, text section, data section, symbol table, etc.
- The exe header contains:
  - magic number : to identify OS in which program can be executed.
  - address of entry point function.
  - info about all the sections in the file.
- When program is executed, the loader (unit of OS) will load it from the disk into the RAM.
- The loader also verifies magic number and file format and if found invalid, program will be terminated.
- For execution of the program OS allocate some memory in RAM which is logically divided into Text, Data, Stack and Heap sections.
- This is called as a "Process". It is formally defined as "Program in Execution".
- To keep info about the process, OS create one more structure, called as "Process Control Block". This struct contains lot of details like:
  - Process Id
  - Exit status
  - Scheduling Info : state, priority, ...
  - Memory Info : page table, ...
  - File Info : Open file desc table, ...
  - IPC info : signal table, ...
  - Execution context (Depends on OS design)
  - Kernel Stack

- etc.

✦ **CPU Scheduling:**

- CPU scheduler is unit of OS which decides the next process to be executed on the CPU using certain algorithms like FCFS, SJF, RR, etc.

✦ **Memory Management:**

- During compilation, compiler assumes a machine with minimal config (e.g. TC compiler assumes a machine with 8086 CPU and 1 MB RAM); this machine is called as "Virtual Machine".

- Assuming this machine compiler & linker generates addresses for each instruction and data. These addresses are known as "logical addresses" or "virtual addresses".

- During process creation additional virtual addresses are added for runtime section i.e. stack and heap.

- The set of virtual addresses of any process are called as "virtual address space".

- While loading the program into memory, actual locations in the RAM are used, which are called as "physical addresses" or "real addresses".

- The set of physical addresses of any process in the RAM is called as "physical address space".

- CPU always execute a process with its virtual addresses.

- MMU is a hardware unit that converts virtual address of a process (requested by the CPU) into physical address of the process (in the RAM).

[CPU] > [MMU] > [RAM]

- The base and limit of each process is stored in the PCB of that process and will be loaded into the MMU when the process is loaded in CPU (context switch).

- The virtual address requested by the CPU is converted into physical address as shown in diag.

✦ **File & IO Management:**

- File in collection of data or information on secondary storage device.

- File system is used to manage multiple files on the single partition.

- In UNIX/Linux, each device is also treated as file.

✦ **Hardware Abstraction:**

- OS hides the hardware details from the application and end users.

- The changes done in hardware are mostly accepted by the OS without intervention of end users.

✦ **Networking:**

- Data can be transferred from one machine to another machine in some network.

- LAN, MAN, WAN are types of networks.

✦ **Protection & Security:**

- Protecting the system from the internal threats is known as "protection".

- One process cannot access data of another process.

- One user cannot access data of another user.

- Securing the system from external threats like viruses, trojans, worms, etc is known as "security".

- Typical antivirus applications are used to secure the system.

✦ **User Interfacing:**

- User interfacing is optional feature of OS (some embedded OS do not have user interfacing).

- User interfacing is provided by the OS with help of a special program called as "shell".

- This program takes input/commands from the user and get them executed from core OS/kernel. This program is also known as "command interpreter".

- There are two types of user interface/shell:

1. Graphical User Interface:
   - Windows: explorer.exe
2. Command Line Interface:
   - DOS: command.com
   - UNIX/Linux: bsh, bash, csh, ksh, etc.
   - In UNIX/Linux shell runs within terminal program.

✦ **Computer Hardware:**

- CPU executes machine level instruction (in each process).
- Each IO device has its own internal dedicated processing unit called as "IO device controller".
- CPU is connected to the IO devices and memory via "BUS".

✦ **IO Structure:**

- The IO device controller communicates with CPU using special signal called as "Interrupt".
- When interrupt occur, CPU pause current task/process and jumps to IVT.
- In low memory area (1st 1KB of RAM) Interrupt vector table is kept, which contains addresses of interrupt service routines. The ISR contains logic for handling interrupts.
- Then CPU executes appropriate ISR (after getting address from IVT) and handles the interrupt.
- After completion of ISR, CPU resume the pause task/process.

✦ **Types of Interrupts:**

1. Non-maskable Interrupt:

- Interrupts generated from the hardware, which cannot be disabled are called as "NMI".
- Typical error interrupts and other critical interrupts fall in this category.

2. Maskable Interrupt:

- When interrupt arrives, it is possible that CPU is busy in executing some OS critical code. If such code is interrupted, it may cause inconsistency in execution of the OS.
- So during execution of such critical tasks, OS disables interrupts using special assembly language instructions.
- Such interrupts are called as "maskable" interrupts.
- When interrupts are enabled again (after completion of critical task), the arrived interrupts will be visible to CPU and will be handled.
- Such interrupt can generated by hw as well as sw.
- Special software (assembly) instructions can be used to generate software interrupt (TRAP).

  e.g. x86 -> INT instruction
  ARM -> SWI instruction

✦ **Types of IO:**

✦ **Synchronous IO:**

- CPU initiates IO by giving instructions to IO device.
- IO device performs IO and CPU will keep checking whether IO is completed or not. This continuous checking is known as "polling".
- This type of IO is called as "Sync IO" and is easier to implement.
- However in such IO, the CPU is no utilized properly.

✦ **Asynchronous IO:**

- CPU initiate IO by giving instrucion to IO device and continue with execution of some another task.
- When IO device completes IO, it send interrupt to the CPU.
- When interrupt is received CPU can continue to execute the earlier task.
- Such type of IO is called as "Async IO".
- This type of IO increases CPU utilization.
- However in such case, OS need to maintain "Device Status Table".
- This table contains list of all IO devices, their status (busy/idle) and set of processes waiting for each IO device (Waiting queue of IO device).

✦ **Storage Structure:**

- Memory/Storage:

|- Primary Memory [Directly accessible by CPU] (Volatile)
  |- CPU registers
  | Cache
  |- RAM (Electronic)
|- Secondary Memory [Not accessible to CPU direct] (Non-volatile)
  |- Hard disk (Magnetic)
  |- Optical disk (CD/DVD)

- Data Flow:

CPU Regr <--> Cache <--> RAM <--> Hard disk

- Comparision of storage devices can be done on basis of speed, size and cost.

- Cache:

- Cache is used avoid speed mismatch between CPU and RAM.
- Data requested by CPU, if not present in cache, it is first copied from RAM to Cache and then from Cache to CPU.

If Data requested by CPU, is present in the cache, it is directly taken from there without informing RAM (about data access). This will speed up the data access.

- If cache is full, the oldest data will be replaced by newer data. Thus cache always contains recent data accessed.

✦ **Booting:**

- If first sector of the storage device/partition contains bootstrap program, then it is called as "bootable device/partition".
- Bootstrap program is different for each OS and can load kernel of that system into the memory.
- Bootloader program displays options for which OS to be started (in case of multiple OS) and depending on user selection starts appropriate bootstrap program.

✦ **Booting Steps:**

- When computer is powered on, programs from the Base ROM (BIOS) are fetched into the RAM automatically.
- The first program from BIOS i.e. POST is executed, which checks whether all devices are functioning properly.
- Then another BIOS program i.e. bootstrap loader is executed, which find the bootable device and starts bootloader program.
- As explained above, bootloader program starts appropriate bootstrap program, which in turn load OS kernel. Thus OS boots.

✦ **Classfication of OS:**

1. Mainframe systems:

✦ **Resident Monitor:**

- Early (oldest) OS resides in memory and monitor execution of the programs. If it fails, error is reported.

✦ **Batch Systems:**

- The batch/group of similar programs is loaded in the computer, from which OS loads one program in the memory and execute it.
- Thus programs are executed one after another.
- In this case, if any process is performing IO, CPU will wait for that process and hence not utilized efficiently.

✦ **Multi-Programming:**

- In such systems, multiple program can be loaded in the memory.
- The number of program that can be loaded in the memory at the same time, is called as "degree of multi-programming".
- In these systems, if one of the process is performing IO, CPU can continue execution of another program. This will increase CPU utilization.
- Each process will spend some time for CPU computation (CPU burst) and some time for IO (IO burst).
- If CPU burst > IO burst, then process is called as "CPU bound".
- If IO burst > CPU burst, then process is called as "IO bound".
- To efficiently use CPU, a good mix of CPU bound and IO bound processes should be loaded into memory. This task is performed by an unit of OS called as "Job scheduler" OR "Long term scheduler".
- If multiple programs are loaded into the RAM by job scheduler, then one of process need to be executed (dispatched) on the CPU. This selection is done by another unit of OS called as "CPU scheduler" OR "Short term scheduler".

✦ **Multi-tasking OR time-sharing:**

- CPU time is shared amoung multiple processes in the main memory is called as "multi-tasking".
- In such system, a small amount of CPU time is given to each process repeatedly, so that response time for any process < 1 sec.
- With this mechanism, multiple processes (ready for execution) can execute concurrently.
- There are two types of multi-tasking:

1. Process based multitasking: Multiple independent processes are executing concurrently. Sometimes (i.e. on multiple processors) also called as "multi-processing".
2. Thread based multi-tasking OR multi-threading: Multiple parts/functions in a process are executing concurrently.

✦ **Multi-user:**

- Multiple users can execute multiple tasks concurrently on the same systems. e.g. IBM 360, UNIX, Windows Servers, etc.
- Each user can access system via different terminal.
- There are many UNIX commands to track users and terminals.

E.g. tty, who, who am i, whoami, w, etc.

2. Desktop Systems:

- User convinence and Responsiveness
- Windows, Mac OS X, Linux, etc.

3. Multiprocessor Systems:

- The systems in which multiple processors are connected in a close circuit is called as "multiprocessor computer".
- The programs/OS take advantage of multiple processors in the computer are called as "Multi-procssing" programs/OS.
- Since multiple tasks can be exeuted on these processors simultaneously, such systems are also called as "parallel systems".

- There are two types of multiprocessor systems:

A. Asymmetric MP:

- OS treats one of the processor as master processor and schedule task for it. The task is in turn divided into smaller tasks and get them done from other processors.

B. Symmetric MP:

- OS considers all processors at equal level and schedule tasks on each processor individually.
- All modern desktop systems use SMP.

4. Distributed Systems:

- Multiple computers connected together in a close network is called as "distributed system".
- It advantages are high availability, high scalability, fault tolerance.
- The requests are redirected to the computer having less load using "load balancing" techniques.
- The set of computers connected together is called as "cluster".

5. Real Time Systems:

- The OS in which accuracy of results depends on accuracy of the computation as well as time duration in which results are produced, is called as "RTOS".
- If results are not produced within certain time (deadline), catastrophic effects may occur.
  These OS ensure that tasks will be completed in a definite time duration.
- Time from the arrival of interrupt till handling of the interrupt is called as "Interrupt Latency". RTOS have very small and fixed interrupt latencies.
- Examples: uC-OS, VxWorks, pSOS, RTLinux, etc.
- Soft real time systems: Not much time critical
- Usually have some secondary storage device.
- Applications: Consumer electronics, ...
- Hard real time systems: Highly time critical
- Usually do not have some secondary storage device.
- Applications: Defence systems, Medical instruments, Space ships, ...

6. Handheld Systems:

- OS installed on handheld devices like mobiles, PDAs, iPODs, etc.
- Challenges:
  - Small screen size
  - Low end processors
  - Less RAM size
  - Battery powered

✦ **OS Design (Implementation):**

1. Simple structure: MS-DOS

```
----------------------------
COMMAND.COM <- command interpreter
----------------------------
```

MSDOS.SYS <- kernel

IO.SYS <- device drivers

---

2. Layered Structure:

- OS is divided into multiple layers, so that each layer depends on the lower layer and provide functionality to upper layer.

---

applications

---

system call APIs

---

system call implmenetation

Kernel Executive : File Mgr, Memory Mgr,
Process Mgr, Scheduler, Thread Mgr, etc.

IO Subsystem

Device Drivers

Hardware Abstraction Layer

- Windows, UNIX, etc.

3. Monolithic Kernel:

- Multiple kernel source files are compiled into single kernel binary image. Such kernels are "mono-lithic" kernels.
- Since all functionalities present in single binary image, execution is faster.
- If any functionality fails at runtime, entire kernel may crash.
- Any modification in any component of OS, needs recompilation of the entire OS.
- BSD Unix, Windows (ntoskrnl.exe), Linux (vmlinuz), etc.

4. Microkernel:

- Kernel is having minimal functionalities and remaining functionalities are implemented as independent processes called as "servers".
- e.g. File management is done by a program called as "file server".
- These servers communicate with each other using IPC mechanism (message passing) and hence execution is little slower.
- If any component fails at runtime, only that process is terminated and rest kernel may keep functioning.
- Any modification in any component need to recompile only that component.
- e.g. Symbian, MACH, etc.

5. Modular Kernel:

- Dynamically loadable modules (e.g. .dll / .so files) are loaded into calling process at runtime.
- In modular systems, kernel has minimal functionalities are rest of the functionalities are implemented as dynamically loadable modules.
- These modules get loaded into the kernel whenever they are called.
- As single kernel process is running, no need of IPC for the execution and thus improves performance of the system.
    - e.g. Windows, Linux, etc.

6. Hybrid Kernel:

- Mac OS X kernel is made by combination of two different kernels.
- BSD UNIX ✦ MACH = Darwin

✦ System Calls: [Dual Mode Protection]

- System calls are the functions exposed by the kernel so that user programs can access kernel functionalities.
- Kernel may have number of syscalls. E.g. UNIX OS have 64 syscalls.
- CPU have a mode bit indicating whether user program is running or system program is running.
    mode=0 => kernel/system/privileged mode
    mode=1 => user/non privileged
- When interrupt occurs, mode is switched from user to kernel mode. And when interrupt is handled (ISR completes) mode is switched back from kernel to user mode.
- System call wrappers/APIs use software interrupt to switch to kernel mode. Then SW intr ISR select appropriate syscall implementation function from System call table and it is invoked.
- In kernel mode the complete access to the system is available and hence all kernel functions/device drivers can directly deal with hardware.
- If any user program try to access hardware directly, CPU will not execute those instructions (mode=1) and such program will be terminated forcefully.
- e.g. open(), close(), write(), read(), lseek(), fork(), exec(), _exit(), ...

✦ CPU Protection:

- To ensure that one process should not consume whole CPU time (e.g. if process stuck in infinite loop), timer hardware is used.
- Timer hardware is configured to generate interrupt periodically, which will cause execution of timer ISR.
- At the end of ISR, CPU scheduler is invoked, which decide the next task to be executed. And control (CPU time) is given to that process.

✦ Process Management:

- Process is program in execution.
- When a process makes system call, functions within kernel are executed. Function activation records of these kernel functions is created on the kernel stack of the process.
- Pointer to kernel stack is maintained in the PCB of the process.

✦ Context Switch:

- If interrupt occured while P1 is executing, values of all CPU registers (execution context) is copied into PCB of the P1 process.

- Then ISR is invoked, which will call scheduler and it will decide the next process to be executed

- Then dispatcher (another unit of OS) will copy the values of CPU registers from the PCB of P2 process into the CPU. Thus CPU will begin/resume execution of P2 process.

- This is called as "context switch".

- During context switch CPU cannot perform any useful task (as CPU registers are getting copied). So very freqent context switch will reduce CPU utilization.

- If context switch is done after long time, then system will not be responsive.

✦ **Process Life Cycle:**

✦ OS Data Structures:

- Job queue / Process table: PCBs of all processes in the system are maintained here.

- Ready queue: PCBs of all processes ready for the CPU execution and kept here.

- Waiting queue: Each IO device is associated with its waiting queue and processes waiting for that IO device will be kept in that queue.

- **Process states:**

  - New, Ready, Running, Waiting, Terminated.

✦ **Inter-process Commmunication:**

✦ Shared Memory:

- A process can transfer data to another process using shared memory.

- It is a common memory area created by OS, in which multiple processes can read/write.

- To keep info about shared memory areas OS maintains structure i.e. shared memory object, which contains: KEY (uniqe id), size, permissions, address of shared memory, number of processes attached to shared memory, etc.

- Addresses of all shared memory objects are kept in a shared memory table and index to shared memory table is called as "shared memory id".

- Imp syscalls:
  - shmget() - create shm region
  - shmat() - get pointer to shm region
  - shmdt() - free pointer to shm region
  - shmctl() - to get info about shm or to delete shm.
  - Commands:

    ipcs -m

    ipcrm -m key
  - Shared memory is fastest IPC mechanism.

✦ **Pipe**

- Pipe is used to communicate betn two processes.
- It is stream based uni-directional communication
- Pipe is internally implemented as a kernel buffer, in which data can be written/read.
- If pipe (buffer) is empty, reading process will be blocked.
- If pipe (buffer) is full, writing process will be blocked.
- If reading process is terminated, writing process will receive SIGPIPE signal.

- There are two types of pipe:

- Unnamed Pipe:

- Used to communicate betn related processes.
  - e.g. who | wc
- Internally pipe is created using pipe() syscall.

- Named pipe / FIFO:

- Used to communicate betn unrelated processes.
- Internally creates an inode and directory entry for a special pipe file on the disk.
- Created using mkfifo command or mkfifo() syscall.
- Opened using open() syscall by reader and writer process.
- write(), read(), close() operations can be performed on both types of pipe.

✦ Message Queue / Mailbox:

- Used to transfer packets of data from one process to another.
- It is bidirectional IPC mechanism.
- Internally OS maintains list of messages called as "message queue" or "mailbox".
- The info about msg que is stored in a object. It contains unique KEY, permissions, message list, number of messages in list, processes waiting for a message to receive (waiting queue).

- IMP Syscalls:

- msgget() - create msg que
- msgctl() - get info or destroy msg que
- msgsnd() - send message into que
- msgrcv() - receive message from que

✦ Signals

- OS have a set of predefined signals, which can be displayed using command

```
kill -l
```

- A process can send signal to another process or OS can send signal to a process.
- kill command is used to send signal to another process, which internally use kill() syscall.
- pkill command is used to send signal to multiple processes/instances of the same program.

```
pkill -SIG programname
```

- IMP Signals:

1. SIGINT: When CTRL✦C is pressed, INT signal is sent to the foreground process. If process does not handle it, process will be terminated.

2. SIGTERM: During system shutdown, OS send this signal to all processes. Process can handle this signal to close resources and get terminated. If process does not handle it, process will be terminated.

3. SIGKILL: During system shutdown, OS send this signal to all processes to forcefully kill them. Any process cannot handle this signal.

4. SIGSTOP: Pressing CTRL✦S, generate this signal which suspend the foreground process.

5. SIGCONT: Pressing CTRL✦Q, generate this signal which resume suspended the process.

6. SIGSEGV: If process access invalid memory address (dangling pointer), OS send this signal to process causing process to get terminated after giving an error message "Segmentation Fault".

7. SIGCHLD: When child process is terminated, this signal is sent to the parent process.

- signal() syscall is used to install signal handler. When signal is received OS calls appropriate signal handler.

✦ **Sockets**

- Socket is defined as communication endpoint.
- Using socket one process can communicate with another process on same machine or different machine in the network.
- Socket = IP address ✦ PORT
- PORT number is 16-bit logical number (0-65535) to identify socket uniquely on a system.
- Two types: TCP sockets / UDP sockets

✦ **Remote Procedure Call**

- ✦ Used to call method from another process on the same machine or different machine in the netwok.
- ✦ Stub and Skeleton are helper objects which transfer data between server and client using sockets.

✦ **Process Creation:**

✦ System Calls:

- Windows : CreateProcess()
- Linux : clone() / fork()
- UNIX : fork()

✦ fork():

- Example:

```
#include <stdio.h>
#include <unistd.h>
int main()
{
        int ret;
        printf("start!\n");
        ret = fork();
        printf("return val : %d\n", ret);
        printf("end!\n");
        return 0;
}
```

- To execute certain task concurrently we can create a new process (using fork() on UNIX).
- fork() creates a new process by duplicating calling process. The new process is called as "child" and calling process is known as "parent".
- fork() returns 0 to child process and pid of the child process to the parent process.
- Even if child is copy of the parent process, after its creation it is independent of parent and both these processes will be scheduled sepeately by the scheduler.
- Based on CPU time given for each process, both processes will execute concurrently.
- In UNIX/Linux each process have some parent process.
- The first process (process 1) is "init" process, which is created by a hardcoded "process 0".
- If parent of any process is terminated, that child process is known as orphan process.
- The ownership of such orphan process will be taken by "init" process.
- Command: ps -e -o pid,ppid,cmd

- When a process terminates, its exit status is returned to the OS. OS store it into PCB of that process (so that its parent can read it).
- If process is terminated before its parent process and parent process is not reading its exit status, then even if process's memory/resources is released, its PCB will be maintained. This state is known as "zombie state".
- To avoid zombie state parent process should read exit status of the child process. It can be done using wait() syscall (which performs 3 steps):
  - pause execution parent until child process is terminated.
  - read exit status & other info from PCB of child process & return to parent process (as out param).
  - release PCB of the child process.

✦ **exec() syscall:**

- exec() syscall "loads a new program" in the calling process's memory (address space) and replaces the older one.
- There are six functions in the family of exec():
  - execl(), execlp(), execle(),
  - execv(), execvp(), execve()
- Example:

```
rct = fork();
if(ret==0)
{
        err = execl("/bin/ls", "ls", ..., NULL);
        if(err < 0)
        {
                perror("execl() failed");
                _exit(1);
        }
}
wait(&staus);
```

- If exec() succeed, it does not return (rather new program is executed).

✦ **Program/Command Execution:**

✦ Synchronous Execution: Parent waits for the child process to complete its execution. Internally calls wait function. By default shell executes each command synchronously.

e.g. ./kcalc

✦ Asynchronous Execution: Parent does not wait for the child process to complete its execution. Shell can execute a command asynchronously by suffixing & to the command.

e.g. ./kcalc &

✦ **CPU Scheduling:**

✦ There are four scenarios when CPU scheduler is invoked:

A. Running -> Terminated (exit)

B. Running -> Waiting (io request)

C. Running -> Ready (interrupt)

D. Waiting -> Ready (io completes)

✦ **Two Types of scheduling:**

1. Non-Preemptive Scheduling:

- Scheduler is invoked only in case A & B (where process give up CPU on its own).
- Also called as "co-operative scheduling".

2. Preemptive Scheduling: Scheduler is invoked in all above cases (Specially in case C & D, running process is forced to leave the CPU).

✦ **CPU Scheduling Criterias:**

- CPU Utilization: MAX : Server:90%, Workstation:60%
- Throughput: MAX : Amount of work completed in unit time.
- Waiting Time: MIN : Time spent by the process in the ready queue waiting for the CPU.
- Turn around Time: MIN : Time from the arrival of process to termination of the process.
- Response Time: MIN : Time from the arrival of the process to first CPU allocation time for that process.

✦ **CPU Scheduling Algos:**

✦ **FCFS**

- Process arrived first will be scheduled first.
- Non-preemptive scheduling
- Avg waiting time depends on the order of arrival of the process.

✦ **SJF**

- Process having minimum burst time will be executed first.
- This algo gives minimum waiting time.
- Can be implemented as Non-premptive as well as Preemptive
- Preemptive SJF is also known as "Shortest Remaining Time First".

✦ **Priority**

- Each process is associated with a number called as priority of the process. Lower is the number, higher is the priority.
- Can be implemented as Non-premptive as well as Preemptive
- Process having lower priority may not get sufficient CPU timer for execution; this is called as "Starvation".
- To resolve this priotity of such processes can be incremented periodically; this is called as "Aging".

✦ **Round Robin**

- Each process is given a small CPU time (called as time quantum) repeatedly.
- Pre-emptive algorithm
- This algo gives minimum response time.

✦ **Fair Share:**

- CPU time is divided into epoch times.
- Each process will get fair amount of CPU time in each epoch based on its priority [nice value].
- By default a process inherit nice value from the parent process. It can be altered programmatically

using nice() system call.

- Root user can change priority of the process using "nice" or "renice" command.
- Lower is the nice value, higher is the CPU time share given to the process.

✦ **RTOS Scheduling Algorithms:**

✦ **Rate Monotonic Scheduling:**

- Static priority based scheduling algorithm i.e. priorities of the processes are not changed at run time.
- Does not utilize CPU fully as number of processes increase.

✦ **Earliest Deadline First:**

- The priorities of processes are dynamically changed to complete the processes within deadline.

✦ **Proportional Share Scheduling:**

- Based on priority of the process, it is given some amount of CPU shares.

✦ **Multi-Level Queue:**

- Depending on nature of the processes, the ready queue is splitted into multiple sub-queues and each queue can have different scheduling algo. This is known as "multi-level queue".
- If a process is not getting sufficient CPU time, in its current queue, then it can be shifted into another queue by OS. This enhanced concept is known as "multi-level feedback queue".

✦ **Process Synchronization:**

- Multiple processes accessing same resource at the same time, is known as "race condition".
- When race condition occurs, resource may get corrupted (unexpected results).
- Peterson's problem, if two processes are trying to modify same variable at the same time, it can produce unexpected results.
- To resolve this problem, one process can access resource at a time. This can be done using sync objects/primitives given by OS.
- Well known sync objects are:
  - Semaphore, Mutex, Event, Critical section, Waitable Timer, Monitor

✦ **Semaphore:**

- Semaphore is a sync primitive given by OS.
- Internally semaphore is a counter. On semaphore two operations are supported:

1. wait operation: dec op: P :

- semaphore count is decremented by 1.
- if cnt < 0, then calling process is blocked.
- typically wait operation is performed before accessing the resource.

2. signal operation: inc op: V :

- semaphore count is incremented by 1.
- if one or more processes are blocked on the semaphore, then one of the process will be resumed.
- typically signal operation is performed after releasing the resource.

Q. If sema count = -n, how many processes are waiting on that semaphore?

Ans: "n" processes waiting

Q. If sema count = 5 and 3 P & 4 V operations are performed, then what will be final count of sema?

Ans: 5 - 3 ✦ 4 = 6

✦ There are two types of semaphores:

1. Counting Semaphore:

- Allows "n" number of processes to access resource at a time.

2. Binary Semaphore:

- Allows only 1 process to access resource at a time.

✦ **IMP Syscalls:**

- sem_init() : init sem with some count
- sem_destroy() : destroy sem
- sem_wait() : wait/dec op
- sem_post() : signal/inc op

✦ **Mutex:**

- Mutex is used to ensure that only one process can access the resource at a time.
- Functionally it is same as "binary semaphore".
- However mutex is more efficient than binary semaphore.
- Mutex can be unlocked by the same process/thread, which had locked it.
- IMP syscalls:
  - pthread_mutex_create() : create mutex
  - pthread_mutex_destroy() : destroy mutex
  - pthread_mutex_lock() : lock mutex
  - pthread_mutex_unlock() : unlock mutex

✦ **Deadlock**

✦ Deadlock occurs when four conditions/characteristics hold true at the same time:

1. No preemption: A resource should not be released until task is completed.
2. Mutual exclusion: Resources is not sharable.
3. Hold & Wait : Process holds a resource and wait for another resource.
4. Circular wait: Process P1 holds a resource needed for P2, P2 holds a resource needed for P3 and P3 holds a resource needed for P1.

✦ **Deadlock Prevention:**

- OS syscalls are designed so that at least one deadlock condition does not hold true.

✦ **Deadlock Avoidance:**

- Processes declare the required resources in advanced, based on which OS decides whether resource should be given to the process or not.

- Algorithms used for this are :

1. Resource allocation graph: OS maintains graph of resources and processes. A cycle in graph indicate circular wait will occur. In this case OS can deny a resource to a process.

2. Banker's algorithm: A bank always manage its cash so that they can satisfy all customers.

3. Safe state algorithm: OS maintains statistics of number of resources and number processes. Based on stats it decides whether giving resource to a process is safe or not (using a formula):

Max num of resources required < Num of resources ✦ Num of processes

- If condition is true, deadlock will never occur.
- If condition is false, deadlock may occur.

Q. A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____

Ans:

3 * 2 < 4 ✦ 3

6 < 7 -> TRUE => Deadlock will not occur

✦ Multi-Threading:

- Multiple parts/functions in a process can be executed concurrently using "threads". It is called as "thread based multi-tasking" or "multi-threading".
- When OS creates a new thread, a new stack is created for the thread within process memory and also a controlling block is created for the thread (TCB).
- Each thread is associated with a function called as "thread function" or "thread procedure". When thread is created, it begins execution of that function and when function completes, thread is terminated.
- TCB of a thread contains info required to execute thread, which includes:
  - Thread Id, Scheduling Info (state, sched algo), User Stack address, Execution context, Kernel stack, etc.
- In modern OS, process is like a container and threads are executed within processes. They are executable units.
- In modern OS, scheduler allocated CPU time to the threads (not the process).
- In modern OS, for each process at least one thread is created by the OS, which execute the main() function. This thread is called as "main" thread.
- Programmer can create additional threads using syscalls or library functions:
  - Windows: CreateThread()
  - Linux: clone()
- It is standard programming to create threads using libraries instead of syscalls.
- POSIX thread library is used to create thread on UNIX/Linux/Mac OS X.

```
pthread_t th;
ret = pthread_create(&th, NULL, func, NULL);
```

arg1: out param contains thread id.

arg2: thread attributes e.g. stack size, priority, scheduling policy

arg3: address of thread func void* func(void*);

arg4: argument of the thread func.

- Thread is a lightweight process.
  - When a thread is created a new TCB and stack is created; while other sections are shared from the parent process.
  - Threads can communicate with each other directly via global/dynamic variables (data/heap section). Their communication is much faster than IPC.
  - Context switch between two threads of the same process is much faster.
- If parent process is terminated (container), all threads in that process are terminated.

✦ **Threading Model:**

- Threads created by thread libraries are used to execute functions in user program. They are called as "user threads".

- Threads created by the syscalls (or internally into the kernel) are scheduled by kernel scheduler. They are called as "kernel threads".

- User threads are dependent on the kernel threads. Their dependency/relation (managed by thread library) is called as "threading model".

- There are four threading models:

1. Many to One:

- Many user threads depends on single kernel thread.
- If one of the user thread is blocked, remaining user threads cannot function.

2. Many to Many:

- Many user threads depend on equal or less number of kernel threads.
- If one of the user thread is blocked, other user thread keep executing (based on remaining kernel threads).

3. One To One:

- One user thread depends on one kernel thread.

4. Two Level Model:

- OS/Thread library supports both one to one and many to many model.

✦ **Memory Management:**

- In multi-programming OS, multiple programs are loaded in memory.
- RAM memory should be divided for multiple processes running concurrently.
- Memory Mgmt scheme used by any OS depends on the MMU hardware used in the machine.
- There are three memory mgmt schemes are available:
  1. Contiguous Allocation:
  2. Segmentation:
  3. Paging:

✦ **Contiguous Allocation:**

A. Fixed Partition:

- RAM is divided into fixed sized partitions.
- This method is easy to implement.
- Number of processes are limited to number of partitions.
- Size of process is limited to size of partition.
- If process is not utilizing entire partition allocated to it, the remaining memory is wasted. This is called as "internal fragmentation".

B. Dynamic Partition:

- Memory is allocated to each process as per its availability in the RAM. After allocation and deallocation of few processes, RAM will have few used slots and few free slots.
- OS keep track of free slots in form of a table.
- For any new process, OS use one of the following mechanism to allocate the free slot.

1. First Fit: Allocate first free slot which can accommodate the process.
2. Best Fit: Allocate that free slot to the process in which minimum free space will remain.

3. Worst Fit: Allocate that free slot to the process in whic maximum free space will remain.

- Statistically it is proven that First fit is faster algo; while best fit provides better memory utilization.

- Memory info (physical base address and size) of each process is stored in its PCB and will be loaded into MMU registers (base & limit) during context switch.

- CPU request virtual address (address of the process) and is converted into physical address by MMU as shown in diag.

- If invalid virtual address is requested by the CPU, process will be terminated.

- If amount of memory required for a process is available but not contiguous, then it is called as "external fragmentation".

- To resolve this problem, processes in memory can be shifted/moved so that max contiguous free space will be available. This is called as "compaction".

✦ Virtual Memory:

- The portion of the hard disk which is used by OS as an extension of RAM, is called as "virtual memory".

- If sufficient RAM is not available to execute a new program or grow existing process, then some of the inactive process is shifted from main memory (RAM), so that new program can execute in RAM (or existing process can grow).

- It is also called as "swap area" or "swap space".

- Shifting a process from RAM to swap area is called as "swap out" and shifting a process from swap to RAM is called as "swap in".

- In few OS, swap area is created in form of a partition. E.g. UNIX, Linux, ...

- In few OS, swap area is created in form of a file E.g. Windows (pagefile.sys), ...

- Virtual memory advantages:

  - Can execute more number of programs

  - Can execute bigger sized programs

✦ Memory Management (continued):

✦ Segmentation:

- Instead of allocating contiguous memory for the whole process, contiguous memory for each segment can be allocated. This scheme is known as "segmentation".

- Since process does not need contiguous memory for entire process, external fragmentation will be reduced.

- In this scheme, PCB is associated with a segment table which contains base and limit (size) of each segment of the process.

- During context switch these values will be loaded into MMU segment table.

- CPU request virtual address in form of segment address and offset address.

- Based on segment address appripriate base-limit pair from MMU is used to calculate physical address as shown in diag.

- MMU also contains STBR register which contains address of process's segment table in the RAM.

✦ Demand Segmentation:

- If virtual memory concept is used along with segmentation scheme, in case low memory, OS may swap out a segment of inactive process.

- When that process again start executing and ask for same segment (swapped out), the segment will be loaded back in the RAM. This is called as "demand segmentation".

- Each entry of the segment table contains base & limit of a segment. It also contains additional bits like segment permissions, valid bit, dirty bit, etc.

- If segment is present in main memory, its entry in seg table is said to be valid (v=1). If segment is swapped out, its entry in seg table is said to be invalid (v=0).

✦ **Paging:**

- RAM is divided into small equal sized partitions called as "frames" / "physical pages".
- Process is divided into small equal sized parts called as "pages" or "logical/virtual pages".
- page size = frame size.
- One page is allocated to one empty frame.
- OS keep track of free frames in form of a linked list.
- Each PCB is associated with a table storing mapping of page address to frame address. This table is called as "page table".
- During context switch this table is loaded into MMU.
- CPU requests a virtual address in form of page address and offset address. It will be converted into physical address as shown in diag.
- MMU also contains a PTBR, which keeps address of page table in RAM.
- If a page is not utilizing entire frame allocated to it (i.e. page contents are less than frame size), then it is called as "internal fragmentation".
- Frame size can be configured in the hardware. It can be 1KB, 2KB or 4KB, ...
- Typical Linux and Windows OS use page size = 4KB.

✦ **TLB (Translation Look-Aside Buffer) Cache:**

- TLB is high-speed associative cache memory used for address translation in paging MMU.
- TLB has limited entries (e.g. in P6 arch TLB has 32 entries) storing recently translated page address and frame address mappings.
- The page address given by CPU, will be compared at once with all the entries in TLB and corresponding frame address is found.
- If frame address is found (TLB hit), then it is used to calculate actual physical address in RAM (as shown in diag).
- If frame address is not found (TLB miss), then PTBR is used to access actual page table of the process in the RAM (associated with PCB). Then page-frame address mapping is copied into TLB and thus physical address is calculated.
- If CPU requests for the same page again, its address will be found in the TLB and translation will be faster.

- Context switch can be handled in one of the following ways:

1. During context switch TLB is flushed i.e. all entries used by previous process will be cleared. Thus new entries page addresses will not conflict with prev process page addresses.

2. TLB can store page addresses along with PID of the process. In this case there is no need to flush TLB during context switch. The comparison of the page address will not conflict with another process, because it is associated with PID of the process. This combination PID and Page Address is called as "ASID".

✦ **Two Level Paging:**

- Primary page table has number of entries and each entry point to the secondary page table page.
- Secondary page table has number of entries and each entry point to the frame allocated for the process.
- Virtual address requested by a process is 32 bits including

  p1 (10 bits) -> Primary page table index/addr

  p2 (10 bits) -> Secondary page table index/addr

  d (12 bits) -> Frame offset

- If frame size is 4KB, 12 bits are sufficient to speficy any offset in the frame. This will also ensure that "d" will not contain any invalid frame offset.

- If virtual address of a process is of 32 bits [p1|p2|d], then maximum address of the process can be
= 1024 * 1024 * 4 * 1024 = 4 GB.

✦ **Page Fault:**

- Each page table entry contains frame address, permissions, dirty bit, valid bit, etc.
- If page is present in main memory its page table entry is valid (valid bit = 1).
- If page is not present in main memory, its page table entry is not valid (valid bit = 0).
- This is possible due to one of the following reasons:
  - Page address is not valid (dangling pointer)
  - Page is swapped out.
- If CPU requests a page that is not present in main memory (i.e. page table entry valid bit=0), then "page fault" occurs.
- Then OS's page fault exception handler is invoked, which handles page faults as follows:

1. Check virtual address due to which page fault occured. If it is not valid (i.e. dangling pointer), terminate the program.
2. If virtual address is valid (i.e. page is swapped out), then locate one empty frame in the RAM.
3. If page is on swap device, swap in the page in that frame.
4. Update page table entry i.e. add new frame address and valid bit = 1.
5. Restart the instruction for which page fault occurred.

✦ **Page Replacement Algorithms:**

- While handling page fault if no empty frame found (step 2), then some page of any process need to be swapped out. This page is called as "victim" page.
- The algorithm used to decide the victim page is called as "page replacement algorithm".
- There are three important page replacement algos:

1. FIFO:

- The page brought in memory first, will be swapped out first.
- Sometimes in this algorithm, if number of frames are increased, number of page faults also increase.
- This abnormal behaviour is called as "Belady's Anomaly".

2. OPTIMAL:

- The page not required in near future is swapped out.
- This algorithm gives minimum number of page faults.
- This algorithm is not practically implementable.

3. LRU:

- The page which not used for longer duration will be swapped out.
- This algorithm is used in most OS like Linux, Windows, ...
- LRU mechanism is implemented using "stack based approach" or "counter based approach".

✦ **Thrashing:**

- If number of programs are running in comparatively smaller RAM, a lot of system time will be spent into page swapping (paging) activity.
- Due to this system performance is reduced.
- The problem can be solved by increasing RAM size in the machine.

✦ **Copy On Write:**

- fork() syscall creates a logical copy of the calling process.
- Initially, child and parent both processes share the same pages in the memory pages.
- When one of the process try to modify contents of a page, the page is copied first; so that parent and child both will have seperate physical copies of that page. This will avoid modification of a process by another process.
- This concept is known as "copy on write".
- The primary advantage of this mechanism is to speed up process creation (fork()).

✦ **Dirty Bit:**

- Each entry in page table has a dirty bit.
- When page is swapped in, dirty bit is set to 0.
- When write operation is performed on any page, its dirty bit is set to 1. It indicate that copy of the page in RAM differ from the copy in swap area.
- When such page need to be swapped out again, OS check its dirty bit. If bit=0 (page is not modified) actual disk IO is skipped and improves performance of paging operation.
    - If bit=1 (page is modified), page is physically overwritten on its older copy in the swap area.

✦ **Demand Paging:**

- When virtual memory is used with paging memory management, pages can be swapped out in case of low memory.
- These pages will be loaded into main memory, when they are requested by the CPU. This is called as "demand paging".

✦ **File Management:**

- File is collection of data/info on secondary storage device.
- Any file has
    - data (contents of the file)
    - meta-data (info about the file).
        - size of file
        - permissions of file
        - user/group info
        - time stamps : creation, accessed, modified
        - type of file
        - info about data blocks of that file.

✦ **Hard Disk:**

- Hard disk is a block device. The data can be read/written in the disk block by block.
- Number of bytes that be read/written on the disk at a time is called as "physical block size" (sector). Typically it is 512 bytes and is decided by the manufacturer.
- OS/Filesystem may read/write disk in terms of larger block size, which is called as "logical block size".
- Logical block size is always multiple of physical block size.

✦ **File System:**

- File system is way of organizing the contents on the disk.
- During formatting file system is created.

- Formatting commands:
  - Windows : format
  - UNIX/Linux : mkfs
- Any file system is logically divided into four parts:
  - Boot block / Boot sector:
    - Contains bootstrap and boot loader program is partition is bootable.
  - Super block / Volume Control block:
    - Contains info about the partition
    - Partition size, label
    - Number of data blocks, number of free data blocks and info about free data blocks.
  - i-node list / Mater file table:
    - The info/metadata of a file is stored in a structure called as "file control block" or "i-node".
    - Collection i-nodes in a partition is called as "i-node list".
  - Data blocks:
    - The data/contents of a file are stored in data blocks.

✦ **Types of File:**

- From (UNIX) OS perspective there are 7 types of files:
  - regular file
  - d directory file
  - l link file
  - p pipe/fifo file
  - s socket file
  - c char device file
  - b block device file

✦ **Directory:**

- From end user perspective, directory is a container which contains sub-directories and files.
- However, OS treats directory as a special file. The directory file contains one entry for each subdirectory or file in it.
- Each directory entry contains i-node number and name of sub-dir / file.

✦ **Links:**

- There are two types of links in Linux/UNIX.

1. Symbolic Link:

- ln -s /path/of/target/file linkpath
- Internally use symlink() syscall.
- A new link file is created (new inode and new data block is allocated), which contains info about the target file.
  - Link count is not incremented.
    - If target file is deleted, link becomes useless.
    - Can create symlinks for directories.

2. Hard Link:

- ln targetfilepath linkpath
- Internally use link() syscall.
- A new directory entry is created, which has a new name and same inode number. No new file (inode and data blocks) is created.
- Link count in the inode of the file is incremented.
    - If directory entry of target file is deleted (rm command), file can be still accessed by link directory entry.
    - Cannot create link for directories.

✦ **rm command:**

- The rm command in Linux, internally calls unlink() system call.
- It deletes directory entry of the file.
- It decrements link count in the inode by 1.
- If link count = 0, the inode is considered to be deleted/free. It can be reused for any new file.
- When inode is marked free, data blocks are also made free, so that they can also be reused for some new file.

✦ **File System Architecture:**

- Virtual File System:
    - This layer redirect file system request to the appropriate file system manager.
- File system manager:
    - File system manager enables access to repective file system on the disk.
    - OS can see all partitions whose file system managers are installed in that OS.
- IO subsystem:
    - Implement buffer cache and other mechanisms to speed up disk IO.

✦ **open() syscall:**

fd = open("/home/nilesh/abc.txt", O_RDONLY);

1. Convert given file path into its inode number. This is called as path name translation and is done by a kernel internal function namei().

2. Read inode of the file from the disk into inode table in memory. Inodes of all recently accessed files are kept in this table.

3. A file position is initialized to 0 and is stored in the open file table. Info of all files opened in the system, is maintained in this table.

4. Each process is associated with a file descriptor table. It keeps info of all files open by that process. For any process, three files are opened by default: 0-stdin, 1-stdout, 2-stderr. This entry stores mode in which file is opened and pointer to OFT entry.

5. Finally index to file desc table entry is returned, which is called as "file descriptor". All further read(), write(), lseek(), close() operations will be using this file desc.

✦ **Disk Allocation Mechanisms:**

Each file system allocate data blocks to the file in different ways:

1. Contiguous Allocation:

- Number of blocks required for the file are allocated contiguously.
- inode of the file contains starting block address and number of data blocks.

- Faster sequential and random access
- Number of blocks required for the file may not be available contiguously. This is called as "External Fragmentation".
- To solve this problem, data blocks of the files can be shifted/moved so that max contiguous free space will be available. This is called as "defragmentation".

2. Linked Allocation:

- Each data block of the file contains data/contents and address of next data block of that file.
- inode contains address of starting and ending data block.
- No external fragmentation, faster sequential access
- Slower random access.
- e.g. FAT

3. Indexed Allocation:

- A special data block contain addresses of data blocks of the file. This block is called as "index block".
- The address of index block is stored in the inode of the file.
- No external fragmentation, faster random and sequential access.
- File cannot grow beyond certain limit.

✦ Free Space Management Mechanisms:

- To allocate free blocks to any file, they should be located quickly. For this reason, info of free blocks is maintained in the super block using one of the following algorithm:

1. Bit Vector:

- Array of bits is used to keep info of used/free blocks.
- Number of bits = number of blocks.
- If nth bit=1, it means nth block is used.
- If nth bit=0, it means nth block is free.

2. Linked List:

- Super block contains address of first free block.
- Each free block contains address of next free block.

3. Grouping:

- One free block contains addresses of remaining free blocks.

4. Counting:

- Each entry in a free block keep starting free block address and number of free blocks after that.

✦ Disk Scheduling:

✦ Hard disk structure:

- Time required to perform read/write operation on particular sector of the disk, is called as "disk access time".
- Disk access time includes two components = seek time and rotational latency.
- Seek time is time required to move head to desired cylinder (track).
- Rotational latency is time required to rotate the platters so that desired sector is reached to the head.

✦ **Disk Scheduling Algorithms:**

- When number of requests are pending for accessing disk cylinders, magnetic head is moved using certain algo. They are called as "disk scheduling algorithms".

1. FCFS:

- Requests are handled in the order in which they arrived.

2. SSTF - Shortest Seek Time First:

- Request of nearest (to current position of magnetic head) cylinder is handled first.

3. SCAN or Elevator:

- Magnetic head keep moving from 0 to max cylinder and in reverse order continuously serving cylinder requests.

4. C-SCAN:

- Magnetic head keep moving from 0 to max cyliner serving the requests and then jump back to 0 directly.

5. LOOK:

- Implementation policy of SCAN or C-SCAN.
- If no requests pending magnetic head is stopped.

****

✦ **Linux File System:**

- Hierarchy:

/ (root of filesystem)

|- bin (programs and commands)
|- sbin (system programs and commands)
|- lib (library files used by various applns *.so)
|- boot (bootloader, kernel image & booting files)
|- home (for each user one directory is created here).
|- root (home dir of root user)
|- usr (user installed programs)
|- var (for system logs)
|- dev (files representing hw devices)
|- proc (info about the system)
|- sys (info about the devices)
|- etc (system config files)

- If username is "dac", then its home directory is "/home/dac". Any user must save all data/files into his/her home directory.
- The "root" user in Linux have full access to all system files. The "root" user's home directory is "/root".
- config files:
  - /etc/passwd (username and passwd info)
  - /etc/inittab (username and shell info)
  - /etc/fstab (info of mounted partitions)
- proc filesystem:
  - /proc/cpuinfo : complete hardware detail of CPU.

✦ **Linux Commands:**

- Linux is a multi-user system i.e. multiple users can login and work concurrently on the single Linux machine.
- If Linux is installed on your machine [SuSE Linux], then the terminals can be accessed using keys as follows:

  CTRL ✦ ALT ✦ F1 -> tty1

  to

  CTRL ✦ ALT ✦ F7 -> tty7 -> :0 -> GUI
- Within GUI terminal, multiple terminal windows can be opened, called as psuedo terminals. e.g. pts/0, pts/1, ...
- The current terminal name can be found using "tty" command.
- Command for knowing current user name: whoami
- Command for knowing current user, current terminal and login time is: who am i
- To know all users connected to the server machine: who
- To know detailed info about uses connected to server: w
- To close the terminal: exit

✦ **Path:**

✦ **absolute path:**

- full path of the file or directory
- always starts from "/" (root of file system)

✦ **relative path:**

- path of file or directory with respect to current directory
- does not start with "/"

✦ **Special Directory Symbols:**

.. -> represent parent directory

. -> represent current directory

~ -> represent user's home directory

✦ **Linux Basic Commands:**

1. pwd
2. mkdir dirpath
3. ls

   ls dirpath

   ls -l
4. cd dirpath
5. rmdir dirpath
6. cat > filepath

   ....

   ....

   CTRL + D
7. cat filepath
8. rm filepath
9. rm -r dirpath
10. cp command:

    cp filepath dirpath => copy given file in given dir

cp filepath destfilepath=> copy given file with another name

cp -r dirpath destdirpath=> copy given dir to given dest dir

11. mv command:

mv filepath dirpath => move given file into given dir

mv dirpath destdirpath => move given dir into given dest dir

mv filename newfilename => rename a file.

12. man commandname

13. touch filepath

- If file already exists, its "modification time" will be changed.
- If file does not exists, a new empty file is created.

14. echo command:

echo "to display a message"

echo -e "hi\tall\nhow was 1st day with linux?\ngood??"

echo -n "no newline after the text"

echo "can display variable value using $varname"

15. Environment variables keep important info about the system.

Command: env

16. grep -> GNU Reguar Expression Parser

- Used to search a pattern in single/multiple file(s).
- grep "word" filepath
- grep -R "word" dirpath
- egrep "fibonac+i" filepath

+ -> char is repeated 1 or more times

* -> char is repeated 0 or more times

? -> char is repeated 0 or 1 time

{n} -> char is repeated n times

{m,n} -> char is repeated minumum m and maximum n times

^ -> match at beginning of line

$ -> match at the end of line

[char1|char2] -> search a from options

[c1-c2] -> search a char from given range

. -> any one character

grep -> to search basic regular expression

egrep -> to search extended regular expression -> grep -E

fgrep -> to search a fixed word -> grep -F

17. chmod command:

- Used to give permissions to a file.
- chmod +x filepath
  - will add execute permission to all [ugo].
  - +x, +w, +r to add execute, write and read perms respectively.
- chmod -w filepath
  - will remove write permission from all [ugo].

- -x, -w, -r to remove execute, write and read perms respectively.

- chmod u+x filepath
  - will add execute permission for the user (owner of file)
- chmod g-w filepath
  - will remove write permission for the group.
- u+w, u+r, u+x, u-w, u-r, u-x
  g+w, g+r, g+x, g-w, g-r, g-x
  o+w, o+r, o+x, o-w, o-r, o-x
- to give permission to file octal form
  PERM -> rwx r-x r--
  111 101 100
  7 5 4
  chmod 754 filepath
- to change permission of all files in a directory
  chmod -R perm dirpath

18. chown command:
    - chown newuser filepath
    - chown newuser:group filepath
    - chown -R newuser:group dirpath

✦ **Redirection:**

1. output redirection
   command > filepath
   e.g. ls -l /home > output.txt
   command >> filepath
2. input redirection
   command < filepath
   e.g. wc < input.txt
   e.g. wc < input.txt > output.txt
3. error redirection
   command 2> filepath
   e.g. ls -l /ho 2> error.txt
   * command < input.txt > output.txt 2> error.txt

✦ **Pipe**

- Used to communicate between two processes.
- Output of a command can be redirected to another command.
- command1 | command2
- e.g. cat -n names.txt | head -11 | tail 7

✦ **VI editor**

- Developed by Bill Joy

- Editor works in two modes
  - Insert (Edit) mode : Press "i"
  - Command mode : Press "Esc"
- To open/create a file using VI editor
  vim filepath
- VI editor basic commands
  :w -> write
  :q -> quit
  :wq -> write and quit
  :q! -> quit without saving
- VI editor advanced command (to customize VI)
  :set number
  :set autoindent
  :set tabstop=4
  :set shiftwidth=4
  :set autowriteall
- VI editor copy/paste related commands:
  yy -> copy current line
  n yy -> copy n lines from cursor
  :m,ny -> copy from mth line to nth line
  p -> paste
  dd -> cut current line
  n dd -> cut n lines from cursor
  :m,nd -> cut from mth line to nth line
  u -> undo
  CTRL R -> redo
  y$ -> copy from cursor to end of line
  y^ -> copy from cursor to start of line
  yw -> copy current word
  n yw -> copy n words after the cursor
  d$ -> cut from cursor to end of line
  d^ -> cut from cursor to start of line
  dw -> cut current word
  n dw -> cut n words after the cursor

✦ **C Programming on Linux:**

step 1: vim filename.c
  Write and save the C source code.
step 2: gcc filename.c -o filename.out
  Compile .c file and create executable (.out) file.
step 3: ./filename.out
  To run the program.

✦ **C++ programming on Linux:**

step 1: vim filename.cpp

Write and save the C++ source code.

step 2: g++ filename.cpp -o filename.out

Compile .cpp file and create executable (.out) file.

step 3: ./filename.out

To run the program.

✦ **C/C++ Program Debugging:**

- While compilation use -g flag

gcc filename.c -g -o filename.out

- While debugging

gdb ./filename.out

- Set break points wherever appropriate

gdb> break funname

gdb> break linenum

gdb> run

Important gdb commands:

cont -> run till next break point

step -> step into the function

next -> step over the function

print varname -> watch variable value

quit -> stop debugging

help -> to see help about gdb commands

✦ **BASH Shell Scripts:**

- BASH reference manual - ebook

- #!/bin/bash -> shebang line
- comments starts with #

- do not use $ sign while assigning value to var

var=123

- to read value of var use $ sign

echo $var

newvar=$var

- to execute command and assign output to a variable

var=$(command ...)

var=`command ...`

- to execute command

command ...

- relational operator

-eq -ne -lt -le -gt -ge

- logical operators

-a -o !

- if [condition]
  then
    ...
  fi

- if [condition]
  then
    ....
  else
    ....
  fi

- Can use "elif" as combination as else-if.

- case expression in
  c1)
    ...
    ;;
  c2)
    ...
    ;;
  *)
    ...
  esac

- while [condition]
  do
    ....
  done

- break & continue can be used with all types of loops (while, until, for loop) like C/C++

- until [condition]
  do
    ....
  done

- for var in collection
  do
    ....
  done

  The for loop is used to access elements one by one from the collection.

- Special Operators to use in a condition.

  -e check if path exists
  -d check if path is of directory

-f check if path is of file
-r check if path is readable
-w check if path is writable
-x check if path is executable

- All conditions in script [...] are executed internally with "test" command.
- man test

✦ **Positional Parameters:**

- Extra info passed to the script while executing it on command line is called as "positional parameters".
- ./scriptname arg1 arg2 arg3 arg4
- They can be accessed in script as : $1, $2, ..., $9
- Number of positional parameters : $#
- List of positional parameters : $*
- Name of current shell script : $0
- "shift n" command discard first n args and next arguments can be accessed
- Example:

```
#!/bin/bash
echo $1
echo $2
shift 2
echo $1
echo $2
```

- If above script runs as: ./script arg1 arg2 arg3 arg4

The output will be:

```
arg1
arg2
arg3
arg4
```

- Using shift command we can access parameter 10 and onwards.

- Functions:

```
function funname()
{
        # args are accessed as $1, $2, ...
        ....
}
res=$(funname ...)
```

- Functions must be defined at the start of the script
- All results of echo statements will be kept in temp string and will be returned at the end of the function, which can be collected in another variable as shown above.

Q.1 **An OS uses a paging system with 1K pages. A given process uses a virtual address space of 128K and is assigned 16K of physical memory. How many entries does its page table contain?**

Answer: **128**

Explain:

virtual address space =	128KB		131072 Bytes
number of frames/pages =	1K		1024
size of one page =	virtual addr space / no of pages		128 Bytes
physical memory of process =	16KB		16384 Bytes
number of pages for process =	physical memory / page size		128
number of page table entries =	number of pages for process		128

Q.2 **An OS uses the elevator algorithm to schedule the disk-arm. I/O requests are currently pending for blocks on tracks 1, 3, 8, 11, 15, and 16. The disk arm is currently at track 9 and moving upwards. In what order will these requests be handled?**

Answer: **11, 15, 16, 8, 3, 1**

Explain: elevator algorithm = SCAN algorithm

Q.3 **A particular system uses a page size of 1K bytes. A page table for a particular process begins as follows: [3, 4, *, 1, *, 8 ...]**
*** A. What physical address corresponds to the virtual address of 50?**
*** B. Name a virtual address that will generate a page fault.**

Answer: **A: 3122**
B: Any Addr betn 2048 to 3071 OR betn 4096 to 5119

Explain:

Logical Address	Physical Address
0	3072
1024	4096
2048	INVALID
3072	1024
4096	INVALID
5120	8192

Page Table

Logical Page	Physical Frame	Valid/Invalid
0	3	valid
1	4	valid
2	*	invalid
3	1	valid
4	*	invalid
5	8	valid

A: Address 50 = Page 0 address => Physical Frame 3 i.e. Frame 3 address + 50 (offset) => 3072 + 50 = 3122
B: Range of addresses for which physical addresses are not present : Any virtual address between 2048 and 3071 or between 4096 and 5119

Q.4 **With a segmentation, if there are 64 Segments and the maximum segment size is 512 words, the length of the logical address in bits is_____**

Answer: 16

Explain: In segmentation scheme, Logical address contains "segment address" and "offset address".

64 segments :	segment address bits =	6
512 words = 512 * 2 = 1024 bytes :	offset address bits =	10
	Total Address Bits =	16

Q.5 **In an operating system using paging , if each 32-bit address is viewed as a 20-bit page identifier plus aS 12 bit offset, what is the size of each page?**

Answer: **4KB**

Explain: Page offset = 12 bits

So Page size = 2 ^ 12 = 4096 bytes

Q.6 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The largest allocation request in Kbytes that could be denied is for _____.**

Answer: **541 KB**

Explain: Assuming that both allocations are done contiguously at the bottom or top of the memory, the rest of the memory will be empty

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB

So cannot allocate the block greater than it...

Q.7 **A 1000 Kbytes memory is managed using variable partitions but no compaction. It currently has two partitions of sizes 200 kbytes and 260 Kbytes respectively. The smallest allocation request in Kbytes that could be denied is for _____.**

Answer: **181 KB**

Explain: For the smallest allocation of the memory, we need to assume that memory is allocated in the middle as follows so that X, Y, Z almost equal ...

Total Memory :		1000 KB
Total Allocation:	200 KB + 260 KB =	460 KB
Remaining Free Memory :	1000 KB - 460 KB =	540 KB
Divide remaining memory so that X, Y, Z will be equal i.e. Remaining Memory Divided by 3		
Smallest Available Slot =	540 KB / 3	180 KB

So we cannot allocate any slot larger than it

X
200
Y
260
Z

Q.8 **If Semaphore values have 12, then 6V and 4P operations the resultant value is _____**

Answer: 14

Explain:
Each P operation decrements semaphore count by 1
Each V operation increments semaphore count by 1
Original Count = 12
Incrementing 6 Times = 18
Decrementing 4 Times = 14
Final Count = 14

Q.9 **Disk request come to a disk driver for cylinders in the order 10,22,20,2,40,6 and 38, at time when the disk drive is reading from cylinder 20. The seek time is 6 ms per cylinder. the total seek time, if the disk arm sheduling algorithm is first come first served is _____.**

Answer: 876 ms

Explain:
Seek Sequence for FCFS will be :
20, 10, 22, 20, 2, 40, 6, 38

		20	10
		10	12
		22	2
		20	18
Total Seek Distance =	146	2	38
Seek time per cylinder =	6 ms	40	34
Total Seek Time =	876 ms	6	32
		38	
		Total Seek Dist :	146

Q.10 **Each process p(i)=1,2,3,.....,9 is coded as follows**
repeat
p(mutex)
{critical section}
v{mutex}
forever
The code for p10 is identical except that it uses V(mutex) instated of p(mutex).What is the largest no of processes that can be the inside the critical section at any moment?

Answer: 3

Default behaviour of P() / wait operation locks mutex, while V() / signal operation unlocks mutex.
Due to this nature, only one process out of P1 to P9, will be inside the mutex.(Now there is 1 process)
The P10 process calls V() operation and enters into critical section (Now threre are 2 processes)
However, since P10 unlock the mutex, one more process from P1 to P9 can enter into critical section. (Now threre are 2 processes)
Thus there can be at max 3 processes in the critical section.

Q.11 **At a particular time of computation the value of counting semaphore is 7. Then 20 `p` operation and x `v` operation were completed on this semaphore. If the final value of the semaphore is 5, x will be _____**

Answer : **18**

P() operation decrement count, while V() operation increment the count.

Current Count =	7
P() Operations =	20
New Count =	-13
Final Count =	5
V Operations =	18

Q.12 **A semaphore count of negative n means (s=-n) that the queue contains _____ waiting processes.**

Answer : **n**

If semaphore count is zero and any process perform decrement operation the process goes to waiting state,
Thus the first process doing decrement operation will be waiting and semaphore count -1.
So n waiting processes will make semaphore count = -n

Q.13 **The OS uses a round robin scheduler. The FIFO queue of ready processes holds three processes A,B,C in that order. The time quantum is 18 msec. A context switch takes 2 msec. After running for 13 msec, B will block to do a disk read, which will take 30 msec to complete. Trace what will happen over the first 100 msec. What is the CPU efficiency over the first 100 msec?**

Answer : **90%**

A FIFO Queue indicate that processes are added at the end of the queue and removed from the beginning (it is not FCFS algo).
During context switch CPU is not doing some fruitful work and hence is not utilized at that time.
When a process goes for IO, it is removed from READY queue and is added to IO WAITING queue of particular device.
Considering all these points the seqence of scheduling will be as follows (including context switch):
A (18) => SW (2) => B (13 io) => SW (2) => C (18) => SW (2) => A (18) => SW (2) => C (18) => SW (2) => B (5)

Total Milliseconds =	100 ms
Unused time (switching) =	10 ms
CPU utilization / efficiency =	90 ms
CPU utilization =	90 %

156

Q.14 **In a page memory the page hit ratio is 0.35. The time required to access the page in secondary memory is equal to 100 nanosec. the time required to access page in primary memory is 10 ns . the avg time reqd to access page is _____.**

Answer : 68.5 ns

Page hit ratio represent probability of getting the page in main memory.

Hit ratio = 0.35 i.e. out of 100, only 35 pages will be available in main memory.

i.e. out of 100, only 65 pages will be available taken from secondary memory.

Time for accessing page in memory = 10 ns

Total time for accessing 35 pages = 350 ns

Time for accessing page in disk = 100 ns

Total time for accessing 65 pages = 6500 ns

Total time for accessing 100 pages = 6850 ns

Average time for accessing 1 page = 68.5 ns

Q.15 **A 1000 MB hard disk has 512-byte sectors. Each track on the disk has 1000 sectors. The number of tracks on the disk is _____.**

Answer : 2048

disk size = number of tracks * sectors per track * sizeof one sector

number of tracks = disk size / (sectors per track * sizeof one sector)

number of tracks = 1000 MB / (1000 * 512)

2048

Q.16 **The address sequence generated by tracing a particular program executing in a pure demand paging system with 100 records per page, with 1 free main memory frame is recorded as follows. what is the number of page faults?**

0100, 0200, 0430, 0499, 0510, 0530,0560, 0120, 0220, 0240, 0260, 0320,0370

Answer : 7

Here no of records per page is 100, so the page no corresponding to these address will be

1,2,4,4,5,5,5,1,2,2,2,3,3

Also only one frame is available along with pure demand paging

So, Page fault will occur for frame nos. 1,2,4,5,1,2,3

Hence no of page fault is =7

Q.17 **Peak bandwidth of 64 bits, 33 MHZ based PCI bus would be _____**
Answer : 266 MB / sec

Peak Bancwidth =	Bus Width (in Bytes) * Bus Freq (in Hz)	
Bus Width =	64 Bits	8 Bytes
Bus Freqnecy =	33 MHz	33.3 MHz
Peak Bandwidth =		266.4 MB/sec

Q.18 **Cosider a system having m resourses of same type. The resourse are shared by 3 processes A,B,C which have peak time demands of 3,4,6 respectively. The minimum value of m that ensures that deadlock will never occur is _____.**
Answer : 11

Formula for resources of the same type :
Total of max needs < nc. of resource instances + no. of processes
13 < m + 3
So, m >= 11

Q.19 **A certain moving arm disk storage with one head has following specification:**
Number of tracks/recording surface = 200
Disk rotation speed = 2400 rpm
Track storage capacity = 62500bits
the avg latency time (assume that the head can move from one track to another only by traversing the entire block)
Answer : 2.5 sec

	Number of tracks =		200
	Number of rev per sec =	2400 / 60	40
	Time required for 1 revolution =	1 / 40	0.025 sec
Correct	Average Latency =	0.5 * time for 1 rev	0.0125 sec
But Ans	Average rotational latency =	0.5 * tracks * time for 1 rev	2.5 sec
	Track storage capacity =		62500 bits
	Data transfer rate =	number of rev (tracks) per sec * track storage capacity	
		i.e. 40 * 62500 bits / sec	
			2500000 bits/sec
			2.5 mb/sec

Q.20 **A system has 3 processes sharing 4 resourses. If each process needs a maximum of 2 units, deadlock ____**
Answer: **Can Never Occur**

Formula for resources of the same type :
Total of max needs < no. of resource instances + no. of processes
6 < 4 + 3
This holds true, indicating that deadlock never occur

Q.21 **In a multi-user OS 20 requests are made to particular resource per hour on an avg, the probability that no request are made in 45 minute is ___**
Answer: **e ^ -20**

Avg requests per unit time (1 hr) = 20
Num of req per 45 mins i.e. L = 15

Probability is given by = (L ^ x * e ^ -L) / x!
x is num of expected reqs i.e. 0
So probability will be = (15 ^ 0 * e ^ -15) / 0! = e ^ -15

159