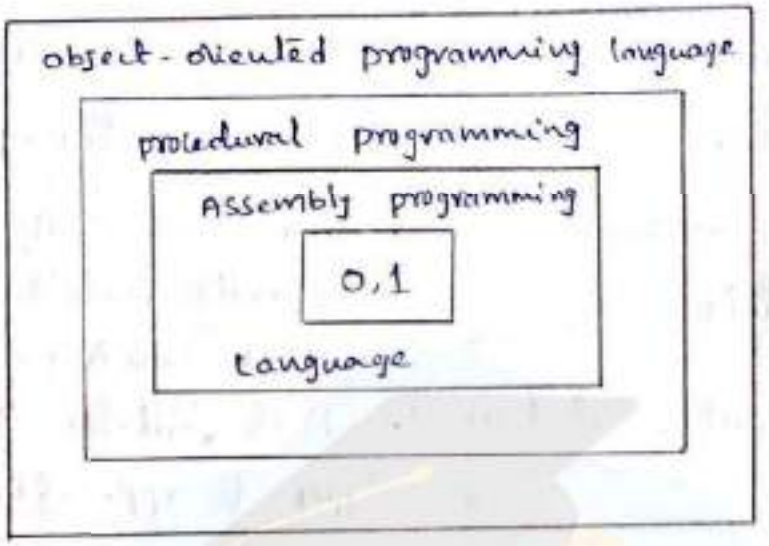
* Introduction *

* Language: It is a communication barrier, used by us to communicate with each other. In terms of programming Language is a barrier between human & a system or application (software).



→ Languages broadly classified as

- Low level programming language
- Middle level programming language
- High-level programming language

→ In low level programming language, to give instructions to a machine, we use Assembly language. It comprises of Mnemonics.

→ Next, we used middle level programming language for communication. The best known example for this is 'C'. It comes under procedure oriented programming language. It uses well structured steps and procedures to build a program. In simple terms, it is a collection of functions or procedures. Mostly, it uses english words as their identifiers.

→ Next, We started using high level programming languages for communication, the best example for this JAVA. It comes under object oriented programming language.

* Differences between procedure oriented / structure oriented and object oriented programming

procedure oriented programming language	object oriented programming language
• It mainly focus on "process".	• It mainly focus on "data".
• It uses top-down Approach	• It uses bottom-up approach.
• Each function considered as a separate module.	• Each class considered as a separate module, where class is collection of methods.
• It doesn't support real-time applications.	• It is suitable for all types of applications.
• It is difficult to debug an application and extend any application.	• It is easy to debug and extend any application.
• It doesn't provide any security Ex:- 'C'	• It provides security Ex:- 'Java'
• Less reusability	• More reusability

→ Top-down approach is also called as step-wise approach. In point of 'C', this approach first programmer has to write a code for main function, In that, they will call sub-functions.

→ Bottom-up approach starts with low-level system, then it looks for high level system. In this, first programmer has to write code for modules, then they look for integration of modules.

* Java OOPs Concepts :-

→ OOPs stands for object oriented programming system.

→ object means a real world entity such as pen, chair, table etc.

→ object oriented programming is a methoddogy & paradigm to design a program using classes and objects.

→ It simplifies the software development and maintenance by providing some features & concepts.

→ The programming language where everything is represented as an object, is known as truly object-oriented programming language.

→ Smalltalk and java are considered as the truely object-oriented programming language.

→ The following are the concepts (&) features (&) principals of object-oriented programming.

- object
- class
- Abstraction
- Encapsulation
- Inheritance
- polymorphism

• object :-

→ An object is a real world entity such as pen, chair, table, car, dog and etc.

→ objects are key to understanding object-oriented technology.

In general, a real world objects have state and behavior. for example, a pen has state (color, company name, model) and behavior (writing, drawing). Ex: Motorbikes, dogs.

In software, the object's state is represented by variables and behavior is represented by methods.

Def^n:- "An object is a software bundle of variables and related Methods."

Example:- ① Object: car
State : color, make
Behavior : climb hill, slowDown, Accelerate etc.

② Object : House
State : current location, color
Behavior : close/open main door.

• Class :-

A class is a collection of similar objects. In the real world, you often have many objects of the same kind. for example, your bicycle is just one of many bicycles in the world.

In terms of object-oriented, we say that your bicycle object is an instance of the class of objects known as bicycles.

Def^n:- " A class is a blueprint or prototype, that defines the variables and the methods common to all objects of a same kind.

Example: ① object : Byke
class : Bykes that same characterics.
② object : Dog
class : Dogs
• Each dog have same variables and Methods like barking(), hungry()

• Abstraction :-

Abstraction is the concept of hiding the internal details and describing things in simple terms. For example: phone call, we don't know the internal processing.

Def^n: "Hiding internal details and showing functionality is known as abstraction". Let us take the example of a car, we know that if accelerator pressed, speed will increase but don't know the internal process how speed will be increased.

• Encapsulation :-

Encapsulation is the technique used to implement abstraction in object-oriented programming.

Def^n:- "Binding code and data together into a single unit is known as encapsulation"

For example, capsule, it is wrapped with different medicines.

→ A java class is the example of encapsulation.

→ In simple words, Encapsulation is a process of wrapping code and data into single unit. Let us take an example of a HR in a company. We communicate through HR not directly with the departments. HR is acting as public interface here.

• Inheritance :-

The process by which one class acquires the properties and functionalities of another class is known as inheritance.

Def^n:- "When one object acquires all the properties and behaviors of another (parent) object is known as inheritance".

→ It provides code reusability. It is used to achieve runtime polymorphism.

For example, A child inherits the properties of its parent.

Example: In object-oriented terminology, Mountain bikes, racing bikes and tandem bikes are all sub classes of the Bicycle super class.

→ Each subclass inherits the properties and functionalities of super class 'Bicycle' (eg: speed, cadance, braking ...).

- polymorphism:-

polymorphism is the concept where an object behaves differently in different situations. There are two types of polymorphism - compile-time and runtime polymorphism.

Defn:- "When one task is performed by different ways is known as polymorphism".

For example, To draw something e.g. Shape a rectangle etc..

Example: The same message 'Move', the man walks, fish swim and birds fly.

* What is Java:-

Java is a programming language and a platform.

→ Java is a high level, robust, secured and object-oriented programming language.

→ Java is a high level modern programming language. And it was introduced by "Sun MicroSystems" in 1995. It was developed by a "team under James Gosling".

→ platform is nothing but any hardware & software environment in which a program runs. Since Java has its own runtime environment i.e JRE(Java Runtime Environment)

* Where Java is used?

According to Sun, 3 billion devices run Java. There are many type of applications that can be created using Java programming. Some of them are as follows:

- Standalone Application:

It is also known as desktop application (or) window based application. An application that we need to install on every machine such as media player, antivirus and etc.

→ AWT and swing are used in Java for creating this Appln.

- Web Application:

An application that runs on the server side and creates dynamic page, is called web application. Eg: irctc.com

→ Servelt, JSP, Struts, JSF technologies are used for creating web applications in Java.

- Enterprise Application:

An application that is distributed in nature, such as banking applications etc. "EJB" is used for this application

• Mobile Application :

An application that is created for mobiles.

→ currently Android and Java ME are used for creating Mobile applications.

* History of Java :-

→ Java team members (James Gosling, Mike Sheridan and Patrick Naughton), initiated the Java language project in June 1991 for digital devices such as set-top boxes, televisions etc.

→ The small team of sun engineers called as "Green Team"

→ Firstly, it was called "Greentalk" by James Gosling and file extension was ".gt".

→ After that, it was called "Oak". Why Oak? Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A, France, Germany and etc.

→ In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

→ The team gathered to choose a new name. The suggested words were "dynamic", "revolutionary", "silk", "Jolt", "DNA" etc.

→ According to James Gosling "Java was one of the top choices along with silk". since Java was so unique, most of team members preferred Java.

→ Java is an island of Indonesia where first coffee was produced (called Java coffee).

→ Notice that Java is just a name.

→ originally developed by James Gosling at sun Microsystems and released in 1995.

* Features of Java (or) Java buzzwords :-

There are many features of Java. They are also known as Java buzzwords.

The features of Java given below

- Simple
- object-oriented
- platform independent
- Secured
- Robust
- Architecture neutral
- portable
- Dynamic
- Interpreted
- High performance
- Multithreaded
- Distributed

• Simple:

According to Sun, Java language is Simple because

→ Syntax is based on C and C++.

→ removed many confusing and rarely-used features e.g. Explicit Pointers, operator overloading etc.

→ No need to remove unreferenced objects because there is automatic garbage collection in Java.

→ It eliminates the complexities of C and C++, therefore Java has been made simple.

• Object-Oriented:

→ object-oriented means we organize our software as a combination of different types of objects that contains both data and behaviour.

→ object-oriented programming (OOPs) is a methodology that simplify software development and maintenance by providing some concepts & rules.

→ The basic concepts of OOPs are:

* Object
* class
* Inheritance
* polymorphism
* Abstraction
* Encapsulation

• platform independent:

→ A platform is the hardware & software environ-ment in which a program runs.

→ There are two types of platforms: software-based and hardware-based. Java provides software-based platform.

→ Java code can be run on multiple platforms e.g. Windows, Linux, Mac os and etc.

→ Java code is compiled by the compiler and converted into byte code. This byte code is a platform inde-pendent code.

→ It is achieved by JVM (Java virtual Machine). The philosophy of Java is "Write Once, Run Anywhere (WORA).

• Secured:

→ Java is secured because Java does not use explict pointers and All java programs runs inside the virtual machine sandbox.

→ Java uses the public key encryption system for providing security.

• Robust:

→ Robust simply means strong. Java is robust programming language because

* Java uses strong Memory Management.
* There are lack of pointers that avoids security problem.
* There is automatic garbage collection in Java.
* There is exception handling and type checking Mechanism in Java.

→ All these points makes Java robust.

- Architecture - neutral:

→ There is no implementation dependent features e.g. size of primitive types is fixed.

→ In 'C' programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in 'Java', it occupies 4 bytes of Memory for both 32 and 64 bit architectures.

- portable:

→ Java is portable because we may carry the Java bytecode to any platform.

→ Java compiler is written in ANSI C with clean portability boundary.

→ The Java programs can run on any hard-ware environment.

- Dynamic:

→ Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment.

→ Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

- Interpreted:

→ Java byte code is translated on the-fly to native machine instructions and is not stored anywhere.

→ Java byte code can be interpreted on any system that provides a Java Virtual Machine (JVM).

• High performance:

→ With the use of Just-In-Time compilers, Java enables high performance.

→ Java is faster than traditional interpretation.

→ Just-In-Time (JIT) compiler translates Java byte code directly into native machine code for very high speed performance.

• Multithreaded:

→ Java was designed to meet the real-world requirements. To accomplish this, Java supports multi-threaded programming, which allows you to write programs that do many things simultaneously.

→ A thread is like a separate program, executing concurrently.

→ The main advantage of multi-threading is that it doesn't occupy memory for each thread, it shares a common memory area.

• Distributed:

→ Java is designed for the distributed environment of the internet. We can create distributed applications in Java.

→ We may access files by calling the methods from any machine on the internet.

→ Java's remote method invocation (RMI) make distributed programs possible.

* Simple Java program:-

→ In this section, we can discuss how to execute a Java program and what are the requirements to execute a Java program.

→ For executing any Java program, you need to

- Install the JDK (Java Development Kit).
- Set path of the jdk/bin directory.
- Create the Java program.
- Compile and run the Java program.

* Install the JDK:

JDK - Java Development Kit

→ It is a software development environment for Java applications and applets.

→ JDK includes

- Java compiler (Javac)
  It translates the source code to byte code.
- Java debugging tool (Jdb)
  It is used to run Java program
- Java Runtime Environment (JRE)
  It provides an environment to run any Java program at any platform.
- Java Archiving tool (jar)
  It is used to distribute the Java appln through network with .jar extension.

→ JDK is available for free at www.oracle.com under Java SDKs and tools → Java SE.

→ Download the latest JDK and install it.

→ On Windows, the JDK will be installed by default directory ie " C:\ program files \ Java \ jdk 1.8.xx ".

* Set path of the JDK/bin directory:

The path is required to be set for using tools such as Javac, Java etc.

For setting the permanent path of JDK, you need to follow these steps:

- Goto My computer properties → advanced tab → environment variables → new tab of user variable → Write 'path' in variable name → Write 'path of bin folder' in variable value → OK → OK → OK.

→ The path of bin folder is look like

"C:\Program Files\Java\jdk1.x.x\bin".

→ To verify that JDK is properly installed, open the command prompt type "javac" and press "Enter".

* Create the Java program:

→ To create Java program open any editor such as notepad.

→ Type the source code

Ex:-

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

→ Save this file as "Sample.Java".

* Compile and run the Java program:

→ To compile 'javac' command is used

for ex: Javac Sample.Java
→ we get "Sample.class" file

→ To execute 'Java' command is used

for ex: Java Sample
→ we get o/p 'Hello Java'.

In the above Java program, let's see what is the meaning of class, public, static, void, main, string[], System.out.println().

- class is a keyword is used to declare a class in Java.
- public keyword is an access modifier which represents visibility, it means it is visible to all.
- static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method.
- void is the return type of the method, it means it doesn't return any value.
- main is a method, it represents startup of the program. The main method is executed by the JVM.
- String[] args is used for command line arguments.
- System.out.println() is used print statement.
  - System: It is a class, which belongs to java.lang package.
  - out: It is an output stream object, which is a member of System class.
  - println(): It is a method supported by the output stream object "out". It is used to display any kind of output on the screen.

→ At compile time, Java file is compiled by Java compiler and converts the Java code into bytecode (class file).

→ At runtime, class file is converted into machine understandible Instructions.

* Java comments :-

→ The comments are statements that are not executed by the compiler and interpreter.

→ The comments can be used to provide information, explanation and hide program code.

There are three types of comments in Java.

1. Single Line Comment
2. Multi Line Comment
3. Documenation Comment

1. Single Line comment:

This is used to comment only one line.

Syntax: // This is single line comment

2. Multi Line comment:

This is used to comment multiple lines of code.

Syntax:
```
/*
This
is
multi line
comment
*/
```

3. Document comment:

This is used to create documentation API.

To create documentation API, you need to use "javadoc tool".

Syntax:
```
/**
This
is
document
comment
*/
```

→ The javadoc tool creates HTML files for your program with explanation.

* Datatypes in Java:-

→ Datatypes represents the different values to be stored in the variable.

→ Java defines eight data types, those are

- byte
- short
- int
- long

} Integer group

- float
- double

} floating-point group

- char } character group
- Boolean } Boolean group

Datatype	Default Size	Range
byte	1 byte	-128 to 127
short	2 byte	-32,768 to 32,767
int	4 byte	-2147483648 to 2147483647
long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 byte	$-1.7 * e^{38}$ to $1.7 * e^{38}$
double	8 byte	$-3.4 * e^{308}$ to $3.4 * e^{308}$
char	2 byte	0 to 65,536
Boolean	1 bit	0 or 1.

* variables:-

Variable is a name of memory location, in that we can able to store the value for the particular program.

There are three types of variables

→ Local variable

→ Instance variable

→ Static variable

Ex: `int x = 10;` // where x is variable name.

→ Local variable:

A variable which is declared inside the method is called local variable.

→ Instance variable:

A variable which is declared inside the class but outside the method, is called instance variable. It is not declared as static.

→ Static variable:

A variable that is declared as static is called static variable. It cannot be local.

Example:-

```
class Vardemo
{
  int x = 50; // instance variable
  static int m=10; // static variable
  void mains()
  {
    int n=9; // local variable
  }
}
```

* Constant:-

→ There are several values in the real world which will never change, those are called as constants.

eg:- PI (π) values is 3.142 and a day will always have 24 hrs.

→ A constant in Java is used to map or assign an exact and unchanging value to a variable.

→ Java does not directly support constants, However, a static final variable is effectively a constant.

Example: public static final int MAX_VALUE =25;

* The scope and Lifetime of variables :-

→ Each variable in modern programming language has:
- a name
- an address
- a type

→ In addition to above properties, each variable also has:
- a scope
- a lifetime

• scope:

→ The scope of a variable is the locations/places/range in a program where the variable is accessible/visible.

→ We can declare variables within any block.

→ Block is begun with an opening curly brace and ended by a closing curly brace.

→ one block equal to one new scope in Java.

→ A scope determines what variables are visiable to other parts of your program and also determines the lifetime of those objects.

• Lifetime:

→ The lifetime of a variable is the location (i.e place) where the variable exists.

→ The lifetime refers to the amount of time a variable exists.

→ variables are created when their scope is entered, and destroyed when their scope is left. This means that a variable declared with in a method will not hold their values outside the method.

→ variables are created and destroyed while the program is running.

* operators :-

An operator is a symbol that is used to perform operations. There are many types of operators in java such as

- Arithmetic operators

  +, -, *, /, %., ++, --

- Relational operators

  ==, !=, >, <, >=, <=

- Logical operators

  &&, ||, !

- Bitwise operators

  & - Bitwise AND
  | - Bitwise OR
  ^ - Bitwise exclusive OR

Ex:

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

<< - Left shift. Ex: a = 0001000, b = 2
>> - Right shift.

a << b	0100000
a >> b	0000010

- Assignment operators

  =, +=, -=, *=, /=, %=

- conditional operator

  exp? value1 : value2

* operator precedence & hierarchy :-

operators	precedence
postfix	expr++, expr--
prefix	++expr, --expr
multiplicative	*, /, %.
additive	+, -
shift	<<, >>
relational	<, >, <=, >=
equality	==, !=
bitwise AND	&

bitwise exclusive OR	^
bitwise OR	|
logical AND	&&
logical OR	||
conditional	? :
Assignment	=, +=, -=, *=, /=, %=.

* Expression:-

An expression is a construct made up of variables, operators and method invocations, which are constructed according to the syntax of the language, that evaluates to a single value.

Examples:

```
int marks = 25;
int Extmarks = 75;
int Total = 0;
Total = marks + Extmarks;
```

* Type conversion and casting:-

• Type conversion:-

It converts the one data type into another. If both are compatiable, then Java compiler will perform the type conversion automatically.

for example converting a int into float,
converting a float into double.

Ex:-

```
int i = 100;
Long l = i;
float f = l;
```

o/p:

```
i value 100
l value 100
f value 100.0
```

→ Type conversion is done by Java compiler, but remember we can store a large data type into the other.

• Type casting:-

When a user can convert the one higher data type into lower data type then it is called as the type casting.

If both are incompatiable types, we must type casting.

Example :-

```
double d = 10.04;
long l = (long)d;
int i = (int)l;
```

o/p: d value 10.04
l value 10
i value 10

Note:- Type conversion is done by compiler and Type casting is done by user.

Example :-

```
class TypeConversion
{
  public static void main(String[] args)
  {
     int x = 1024;
     float y;
     y = x;
     System.out.println("y value is "+y);
  }
}
```

o/p: y value is 1024.0

Example :-

```
class TypeCasting
{
  public static void main(String[] args)
  {
     double d = 10.04;
     long l = (long)d;
     int i = (int)l;
     System.out.println("d value is "+d);
     System.out.println("l value is"+l);
     System.out.println("i value is"+i);
  }
}
```

o/p:- d value is 10.04
l value is 10
i value is 10

* Enumerated Types:-

An enum type is a special datatype that contains fixed set of constants.

In java programming, you define an enum type by using the 'enum' keyword.

Example:

```
public enum Day {
        Sunday, Monday, Tuesday, Wednesday, Thursday,
        Friday, Saturday
        }
```

You should use enum types any time you need to repre-sent a fixed set of constants.

program:-

```
public class EnumExample {
    enum Day {
    Sunday, Monday, Tuesday, Wednesday, Thursday, Friday,
     Saturday
     }
    public static void main (String [] args)
     {
        Day yesterday = Day.Thursday;
        Day today = Day.Friday;
        Day tomorrow = Day.Saturday;
     System.out.println ("Today is " + today);
     System.out.println ("Tomorrow will be" + tomorrow);
     System.out.println ("yesterday was" + yesterday);
     }
}
```

o/p: Today is Friday
Tomorrow will be Saturday
yesterday was Thursday

* Conditional statements:-

These are used to check the condition and execute the set of statements based on condition.

The Java supports following conditional statements

→ If-else statements

→ switch statement

→ If-else statement:-

If statement is used to test the condition. It checks boolean condition: true or false.

There are various types of if statement in Java.

- If statement
- If-else statement
- If-else-if ladder.

• If statement:

Syntax:-

```
if (condition)
{
  // code to be executed
}
```

• If-else statement:

Syntax:-

```
if (condition)
{
// code if condition is true
}
else
{
// code if cond'n is false
}
```

• If-else-if ladder:

Syntax:

```
if (condition1)
  {
// code if condition1 is true
  }
  else if (condition2)
  {
// code if condition2 is true
  }
  ...
  else
  {
  // code if all conditions are false
  }
```

Example: Demonstrate if-else statements.

```
public class IfElseDemo
{
  public static void main (String [] args)
  [
    int marks = 76;
    char grade;
    if (marks >= 90)
     {
       grade = 'A';
     }
    else if (marks >= 80)
       {
         grade = 'B';
       }
    else if (marks >= 70)
     {
       grade = 'C';
     }
    else if (marks >= 60)
     [
       grade = 'D';
     }
     else if (marks >= 50)
     {
       grade = 'E';
     }
     else
     [
       grade = 'F';
     ]
    System.out.println ("Grade is " +grade);
  }
}
```

output:- Javac IfElseDemo.Java
Java IfElseDemo
Grade is C.

→ Switch-case statement :-

The switch-case statement tests the value of given variable against a list of case values and when a match is found, a block of statements associated with that case is executed.

Syntax:-

```
switch (expression)
{
    case value1 : statements;
                  break;
    case value2 : statements;
                  break;
                  :
    default : statements;
}
```

Example:- Demonstrate switch-case statement.

```
public class SwitchDemo
{
 public static void main (String[] args)
 {
  int day = 2;
  switch (day)
  {
  case 1 : System.out.println ("sunday"); break;
  case 2 : System.out.println ("Monday"); break;
  case 3 : System.out.println ("Tuesday"); break;
  case 4 : System.out.println ("Wednesday"); break;
  case 5 : System.out.println ("Thursday"); break;
  case 6 : System.out.println ("Friday"); break;
  case 7 : System.out.println ("saturday"); break;
  default : System.out.println ("Invalid choice");
  }
}
}
```

O/P: Monday

* Loop statements :-

The Java supports following looping statements.

Those are
- While loop
- Do-while loop
- for loop

• While loop:

The java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:-

```
while (condition)
  {
   // code to be executed
  }
```

• Do-while loop:

If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

syntax:

```
do
 {
  // code to be executed
 } while (condition);
```

• for loop:

If the number of iteration is fixed, it is recommended to use for loop.

There are three types of for loop in java.

- Simple for loop
- For-each loop
- Labeled for loop

- Simple For loop:

Syntax:

```
for (initialization; condition; incr/decr)
{
 //code to be executed
}
```

- Foreach loop:

Syntax:-

```
for (Type var: array)
{
 //code to be executed
}
```

- Labeled For loop:

Syntax:

```
Labelname:
for(initialization; condition; incr/decr)
{
 //code to be executed
}
```

Examples:- While

```
public class WhileExample
{
public static void main(String[] args)
{
  int i=1;
  While (i<=10)
  {
    System.out.println(i);
    i++;
  }
}
}
```

o/p:- 1
2
3
4
:
:
10

Example: Do-While

```
public class DoWhileExample
{
public static void main (String[] args)
{
 int i=1;
 do{
    System.out.println(i);
    i++;
    }while(i<=10);
  }
}
```

o/p: 1 2 3 ... 10

Example: For

```
public class ForExample
{
public static void main (String[] args)
{
 for(int i=1; i<=10; i++)
  {
   System.out.println(i);
  }
}
}
```

o/p: 1 2 3 ... 10

* Break and continue statements :-

→ The statements break and continue alter the normal control flow of compound statements.

→ The break statement immediately jumps to the end of the appropriate compound statement (loop).

→ The continue statements immediately jumps to the next iteration (if any) of the appropriate loop.

• break:-

When a break statement is encountered inside a loop, the loop is terminated and program control resumes at the next statement following the loop.

Syntax:- break;

Example:

```
public class BreakDemo
{
public static void main(String[] args)
{
 for(int i=1; i<=10; i++)
 {
   if (i==5)
   {
     break; // Terminate loop if i is 5
   }
   System.out.println(i);
 }
 System.out.println("Loop is over.");
}
}
```

output: C:\>Javac BreakDemo.Java

C:\>Java BreakDemo

1
2
3
4
Loop is over.

→ In simple words, the break keyword is used to breaks (stopping) a loop execution.

• Continue:-

When a continue statement is encountered inside the body of a loop, remaining statements are skipped and loop proceeds with the next iteration.

Syntax: continue;

Example:

```
public class ContinueDemo
{
  public static void main(String[] args)
  {
   for(int i=1; i<=10; i++)
   {
     if(i%2==0)
     {
       continue; // skip next statement if i is even
     }
     System.out.println(i);
   }
  }
}
```

output:

```
C:\> javac ContinueDemo.java
C:\> java ContinueDemo
1
3
5
7
9
```

→ In simple words, the continue keyword is used to skip the particular recursion only in a loop execution.

* Simple java standalone programs :-

1. Fibonacci Series in java

```
class Fibonacci
{
public static void main(String args[])
{
 int n1=0, n2=1, n3, i, count=10;
 System.out.print(n1+" "+n2); // printing 0 and 1
 for(i=2; i<count; i++)
 {
  n3=n1+n2;
  System.out.print(" "+n3);
  n1=n2;
  n2=n3;
 }
}
}
```

output: 0 1 1 2 3 5 8 13 21 34

2. prime Number

```
class primeNum {
    public static void main(String args[])
    {
      int num=17; // int num=Integer.parseInt(args[0]);
      int flag=0;
      for(int i=2; i<num; i++)
      {
        if(num%i==0)
        {
          System.out.println(num + " is not a prime");
          flag=1;
          break;
        }
      }
      if(flag==0)
        System.out.println(num + " is a prime Number");
    }
}
```

output: 17 is a prime Number

3. palindrome Number:

```
class palindrome
 {
 public static void main(String args[])
 {
  int r, sum=0, temp;
  int n=454; // n= Integer.parseInt(args[0]);
  temp=n;
  while(n>0)
  {
   r=n%10;
   sum=(sum*10)+r;
   n=n/10;
  }
  if(temp==sum)
  {
   System.out.println(" palindrome Number");
  }
  else
  {
   System.out.println(" Not palindrome");
  }
 }
}
```

output:-
palindrome Number

4. Factorial

```
class factorial
 {public static void main(String args[])
  {
   int i,fact=1;
   int num=5; // n= Integer.parseInt(args[0]);
   for(i=1;i<=num;i++)
    {
     fact=fact*i;
    }
   System.out.println("Factorial of "+num+" is: "+fact);
  }
 }
```

output:- Factorial of 5 is 120

5. Armstrong Number

```
class ArmstrongNum
{
public static void main (String args[])
{
 int sum=0, r, temp;
 int n=153; // n=Integer.parseInt(args[0]);
 temp = n;
 while (n>0)
 {
  r = n%10;
  sum = sum + (r*r*r);
   n= n/10;
 }
 if (temp == sum)
 {
   System.out.println ("Armstrong number");
 }
 else
  {
   System.out.println ("Not a Armstrong Number");
  }
}
}
```

Output: Armstrong Number

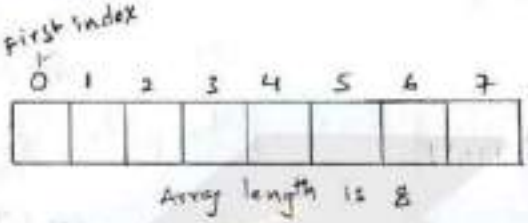
* Arrays:-

→ Till now, we have discussed how to declare variables of a particular datatype, which can store a single value. There are the situations where we might wish to store a group of similar type of values in a variable.

→ It can be achieved by a special kind of data structure known as arrays.

* An array is a collection of similar data elements and it is a data structure where we store similar elements. We can store only fixed set of elements in an array.

→ Array in java is index based, first element of the array is stored at 0(zero) index.



• Advantages of array:

→ code optimization: It makes the code optimized, we can retrive & sort the data easily.

→ Random access: We can get any data located at any index position.

• Disadvantage of array:

→ Size limit: We can store only fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in java.

There are two types of arrays:

- Single dimensional array
- Multi dimensional array

• Single dimensional array:

It is an array with only one dimension (or) index. It can be visualized as a single row or a column.

Declaration:

```
Datatype variable [] = new datatype [size];
(or)
Datatype variable[];
variable = new datatype [size];
```

// Declaration & Initialization

(or) datatype variable [] = {element1, element2, ... ,n}

Ex:- int marks [] = new int [5];

(or) int marks [] = {45, 60, 70, 80, 9}

```
int marks [];
marks = new int [5];
```

Example:-

```
class SingleDarray
{
 public static void main (String args [])
 {
  int marks [] = new int [6]; // Declaration
  marks [0] = 60; // Initialization
  marks [1] = 58;
  marks [2] = 70;
  marks [3] = 80;
  marks [4] = 78;
  marks [5] = 89;
  // printing values in array
  for (int i=0; i < marks.length; i++)
  {
    System.out.println (marks [i]);
  }
 }
}
```

(or)

```
int marks [] =
  {60, 58, 70, 80, 78, 89};
```

output:
60
58
70
80
78
89

• Multi dimensional array:

It is an array with two or more dimensions to indexes. It can be visualized as a matrix of rows and columns.

Lets take a two-dimensional array, In this case, data is stored in row and column based index (also known as matrix-form).

Syntax for array declaration (two)

```
Datatype variable [][] = new datatype [rows] [col];
```

(or)

```
Datatype variable[][];
variable = new datatype [rows] [col];
```

(or) `datatype variable[][] = {{e11, e12 ..}{e21, e22 ..}..}`

Ex:-

```
int marks[][] = new int [3][6];
```

(or)

```
int marks[][];
marks = new int [3][6];
```

(or) `int marks[][] = {{48, 65 ..}{78, 62 ..}{}..}`

Example:-

```
class TwoDarray
{
 public static void main (String args[])
 {
  int marks[][] = new int[3][6];
  marks[0][0] = 48;
  marks[0][1] = 52;
  marks[0][2] = 63;
    :
  marks[2][4] = 75;
  marks[2][5] = 82;

// printing 2-D array
for (int i=0; i < marks.length; i++) {
for (int j=0; j < marks[i].length; j++) {
      system.out.print (Marks[i][j] + " ");
    }
    system.out.println ();
} } }
```

```
// Declaring & Initializing 2D array.
int marks[][] =
(or) {{48,52,63,65,66,71},{67,78,87,61,64,
   76},{78,67,65,77,75,82}};
```

output:-

```
48 52 63 65 66 71
67 78 87 65 64 76
78 67 65 77 75 82
```

* Console input and output (or) console class :-

→ In this, We learn about java.io.Console class. This class provides convenient methods for reading input and writing output to standard streams (Keyboard and display) in command-line (console) programs. Note: Console class comes under "java.io" package

→ The Console class provides following methods,

those are

- printf() - Writes a formatted string to console's output stream.
- readLine() - Reads a single line of text from console's input stream.
- readPassword() - Reads a password from console input stream with echoing disabled.

Example :-

```
import java.io.*; //package
class ConsoleioDemo
{
public static void main(String args[])
{
Console c= System.console();
c.printf("Enter your name: "); // console output
String name = c.readLine(); // console input
c.printf("Enter your company name : ");
String cname = c.readLine();
c.printf("congrats %s \n", name);
c.printf("you are the Employee of company : %s", cname);
}
}
```

output :-

```
javac ConsoleioDemo.java
java ConsoleioDemo
Enter your name : Madhu
Enter your company name : TKRCET
congrats Madhu
you are the Employee of company : TKRCET
```

* Scanner class :-

→ The Scanner class comes under java.util package. And it is used to getting input from user.

→ System is a class in the java.lang package. This class has three predefined variables : in, out and err.

- in refers to standard input stream (Keyboard)
- out refers to standard output stream (Monitor)
- err refers to standard error output stream (monitor).

→ The Scanner class uses System.in object & variable to get input from Keyboard (Standard input stream).

The Scanner class have following Methods

- nextLine() - It returns the input as a String.
- nextInt() - It returns the input as an integer.
- nextFloat() - It returns the input as a Float.
- nextLong() - It returns the input as a long.
- nextShort() - It returns the input as a short.
- nextDouble() - It returns the input as a double.

Example :-

```
import java.util.*;
class ScannerDemo {
public static void main(String args[]) {
Scanner sc = new Scanner(System.in);
System.out.println("Enter your name: ");
String name = sc.nextLine();
System.out.println("Enter your age : ");
int age = sc.nextInt();
System.out.println(" Hai " +name);
System.out.println(" your age is " +age);
}
```

output :-

```
javac ScannerDemo.java
java ScannerDemo
Enter your name
Madhu
Enter your age
30
Hai, your age is 30
```

* Classes and objects :-

Java is an object-oriented programming language. classes and objects are basic building blocks of OOP.

* classes :-

A class is a blueprint or prototype that defines the variables and methods common to all objects of a certain kind.

(or) A class is a group of objects that has common properties.

→ A class in java can contain: data member, method, constructor, block, class and interface.

Syntax to declare a class

```
class <class_name>
{
 //variables declaration
 // Methods declaration
}
```

→ A class can be declared using the keyword 'class' followed by the name of the class that you want to define.

→ The class body contains two different sections: variable declaration and Methods declaration.

* Objects :-

An object is a software bundle of variables and related Methods. An object is an instance of a class. i.e by using object, we can able access variables and methods in that class.

Syntax to declare an object

```
class_name object_name = new class_name();
```

(or)

In general, Syntax is type objectname;

where type is name of class. because a class is an user-defined data type.

→ The keyword 'new' is used to allocate memory at runtime.

• Instance variable:

A variable that is created inside the class but outside the method, is known as instance variable. Instance variable doesn't get memory at compile time. It gets memory at runtime when object is created. That is why, it is known as instance variable.

Example of object and class:

```
class Student
{
  int id = 521;
  String name = "Madhu";   } //data members (also instance variables)
  public static void main (String args[]) → //method.
  {
    Student s1 = new Student(); //creating an object of Student
    System.out.println ("your rno is:" s1.id);
    System.out.println ("your name is:" s1.name);
    int age = 30;
    String branch = "CSE";   } //local variables
    System.out.println ("your age is : " + age);
    System.out.println ("your branch is :" + branch);
  }
}
```

o/p:

```
javac Student.java
java Student
your rno is : 521
your name is : Madhu
your age is : 30
your branch is : CSE
```

In this example, we have created a Student class that have two data members id and name. We are creating the object of the Student class by new keyword and printing the object's value.

* Methods:-

Defⁿ: A java method is a collection of statements that are grouped together to perform an operation.

→ None of the methods can be declared outside the class.

→ All methods have a name that starts with a lowercase character.

→ In java, Methods are used to,

- Make the code reusable.
- Simplify the code.
- Top-down programming.

→ Java supports two types of methods, those are

. Instance methods:- These are used to access/manipulate the instance variables and also access class variables.

. Class methods:- These are used to access class variables but cannot access the instance variables unless and until they use an object for that purpose.

Syntax for Method Declaration

```
[modifiers] return_type method_name (parameter-list)
{
    statements list // Method body
}
```

In the above syntax,

* modifiers (optional) defines the scope (public, protected, default or private).
* return_type - It can be either void (if no value is returned) or if a value is returned.
* method_name - The method name must be a valid java identifier. (Method name starts with lower case letter).

* Parameter List : you can pass one or more values to a method by listing the values in parenthesis following method name.

* Method body : The method body defines what the methods does with the statements.

Example :-

The following example to demonstrate how to define a method and how to call it-

```
import java.io.*;
public class MaxNumber
{
public static void main (String args[])
{
  Console c = System.console();
  c.printf("Enter first number \n");
  int a = Integer.parseInt(c.readLine());
  c.printf("Enter second number \n");
  int b = Integer.parseInt(c.readline());
  int maxc = MaxFunction(a,b);
  System.out.println("Maximum Number is = " + maxc);
 }
public static int maxFunction (int n1, int n2)
{
   int max;
   if ( n1 > n2)
      max = n1;
   else
      max = n2;
   return max;
 }
}
```

O/p: javac MaxNumber.java
java MaxNumber
Enter first number
56
Enter second Number
51
Maximum Number is = 56.

Example: A method with void return type.

```
import java.io.*;
Public class Grade
{
 public static void main (String args[])
 {
  Console c = System.consde();
  c.printf("Enter your marks in blw 0 to 100 \n");
  int marks = Integer.parseInt(c.readLine());
  grade(marks);
 }
 public static void grade (int tmarks)
 {
   if (tmarks >= 90)
     System.out.println(" Grade A");
   else if(tmarks >= 70)
     System.out.println(" Grade B");
   else if (tmarks >= 50)
     System.out.println(" Grade C");
   else
     System.out.println(" Grade D");
   }
}
```

```
o/p: javac Grade.java
     java Grade
     Enter your marks in blw 0 to 100
     92
     Grade A.
```

Note:- If a method is declared using static keyword, then no need to create an object to access it. otherwise, we can create object and then access methods by using that object.

Example: The following example shows method calling with an object creation.

```
import java.io.*;
class Calc
{
public static void main(String args[])
{
 Calc obj = new Calc(); //object creation for class Calc.
 Console c = System.console();
 c.printf("Enter first number");
 int x = Integer.parseInt(c.readLine());
 c.printf("Enter second number");
 int y = Integer.parseInt(c.readLine());
 obj.sum(x,y);
 obj.sub(x,y);
}
void sum(int a, int b)
 {
  System.out.println("sum is :" +(a+b));
 }
void sub(int a, int b)
 {
  System.out.println("sub is :" +(a-b));
 }
}
```

```
O/P:- javac Calc.java
      java Calc
      Enter first number
      20
      Enter second number
      5
      Sum is : 25
      sub is : 15
```

* Method overloading :-

→ If a class have multiple methods by same name but different parameters, it is known as Method overloading.

→ In java whenever a method is being called, first the name of the method is matched and then, the number and type of arguments passed to that methods are matched.

→ Method overloading is a feature that allows a class to have two or more methods having same name but different parameters.

There are two different ways of method overloading

- Method overloading by changing data type of arguments
- Method overloading by changing no. of arguments.

Example :-
Method overloading by changing datatype of arguments.

```
class Calculate
{
 void sum(int a, int b)
 {
  System.out.println("sum is :" + (a+b));
 }
 void sum(float a, float b)
 {
  System.out.println("sum is :" + (a+b));
 }
 public static void main(String args[])
 {
  Calculate cal = new Calculate();
  cal.sum(8, 5); // sum(int a, int b) is method is called.
  cal.sum(4.6f, 3.8f); // sum(float a, float b) is called
 }
}
```

```
// javac calculate.java
  java Calculate
  sum is 13
  sum is 8.4
```

Example :-

Method overloading by changing no. of argument.

```
class Calsum
{
 void sum(int a, int b)
 {
  System.out.println(a+b);
 }
 void sum(int a, int b, int c)
 {
  System.out.println(a+b+c);
 }
 public static void main(String args[])
 {
  Calsum obj = new Calsum();
  obj.sum(10,10,10); // sum() with 3 parameters
  obj.sum(20,20); // sum() with 2 parameters
 }
}
```

o/p: javac Calsum.java
java Calsum
30
40

* Constructor :-

→ constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation.

→ It constructs the values i.e provides data for the object that is why it is known as constructor.

→ Java constructors are the methods which are used to initialized objects.

There are basically two rules defined for the constructor

→ Constructor name must be same as its class name.

→ Constructor must have no explicit return type.

There are two types of constructors

- Default constructor
- parameterized constructor

- **Default constructor :-**

A constructor that have no parameter is known as default constructor. This is used to provide default values to an object.

Example :-

```
class DefaultConstr
{
DefaultConstr() // constructor method
{
System.out.println(" Default Constructor method called.");
}
public static void main(String args[])
{
DefaultConstr dc = new DefaultConstr();
}
}
```

output: javac DefaultConstr.java
java DefaultConstr
Default Constructor method called.

- **parameterized constructor :-**

→ A parameterized constructor is a constructor, that has parameters. (or) A constructor that have parameters is known as parameterized constructor.

→ This is used to provide different values to the distinct objects.

Example :-

In the following example, we have created the constructor of student class that have two parameters.

```
class students
{
 int id;
 String name;
 Students (int i, String n) // parameterized constructor.
 {
  id = i;
  name = n;
 }
 void display() // Method
 {
  System.out.println( id + "  " + name);
 }
 public static void main(String args[])
 {
  Student  s1 = new Students (521, "Madhu");
  Student  s2 = new students (512, "Hari");
  s1.display();
  s2.display();
 }
 }
```

output:-
```
javac students.java
java students
521  Madhu
512  Hari
```

* Constructor overloading :-

Constructor overloading is a technique in java in which a class can have any number of constructors that differ in parameter lists.

The compiler differentiates those constructors by taking into account the number of parameters in the list and their type.

Example

```
class Sum
{
  int a,b,c;
Sum(int x, int y) // with 2 parameters
{
  a=x;
  b=y;
}
Sum(int x, int y, int z) // constructor with 3 parameters
{
  a=x;
  b=y;
  c=z;
}
void display()
{
 System.out.println("sum is"+(a+b+c));
}
public static void main(String args[])
{
  Sum s1 = new Sum(10, 11); // with 2 parameters
  Sum s2 = new Sum(10,20,30); // with 3 parameters
  s1.display();
  s2.display();
}
}
```

output: javac Sum.java

java Sum

sum is 21

sum is 60

There are some differences between constructors and methods. These are,

Constructor	Method
* Constructor is used to initia-lize values to an object.	* Method is used to expose behavior of an object.
* Constructor must not have return type.	* Method must have return type.
* Constructor is invoked implicitly	* Method is invoked explicity
* constructor name must be same as the class name.	* Method name may or may not be same as class name.

* Java Garbage collection :-

In Java, garbage means unreferenced objects. Garbage collection is process of destroyed the unused memory automatically.

In other words, it is a way to destroy the unused objects.

To do so, we were using free() function in C language and delete() in C++. But, in Java it is performed automatically. So Java provides better Memory Management.

An object can be unreferenced in following ways

→ By nulling the reference

Example :- Employee e = new Employee();
e = null;

→ By assigning a reference to another

Example :- Employee e1 = new Employee();
Employee e2 = new Employee();
e1 = e2

In java, the garbage collector of JVM collects only those objects that are created by new keyword. so if you have created any object without new, you can use finalize method to perform cleanup processing (destroying remaining objects).

finalize() method:

This method invoked each time before the object is garbage collection. This method is used to destroying the objects which are not created by "new" keyword.

gc() method:

The gc() method is used to invoke the garbage collector to perform cleanup processing. But this method does not guarantee that JVM will perform the garbage collection. It is only request to the JVM for garbage collection.

Example:-

```
class GarbageDemo
{
public void finalize()
  {
    System.out.println("object is garbage collected");
  }
  public static void main(String args[])
  {
    GarbageDemo g1 = new GarbageDemo();
    GarbageDemo g2 = new GarbageDemo();
     g1 = null;
     g2 = null;
     System.gc();
  }
}
```

o/p: javac GarbageDemo.java
java Garbage
object is garbage collected
object is garbage collected

* String class :-

Generally, String is a sequence of characters. But in java, String is an object that represents a sequence of characters.

→ The java.lang.String class is used to create string object.

→ In Java, An array of characters works same as java String.

for example:

```
char[] ch = {'M', 'a', 'd', 'h', 'u'};
String s = new String(ch);
```

is same as:

```
String s = "Madhu";
```

→ In Java, string is an object, it can be created by using java.lang.String class.

There are two ways to create String object

* By String literal :-

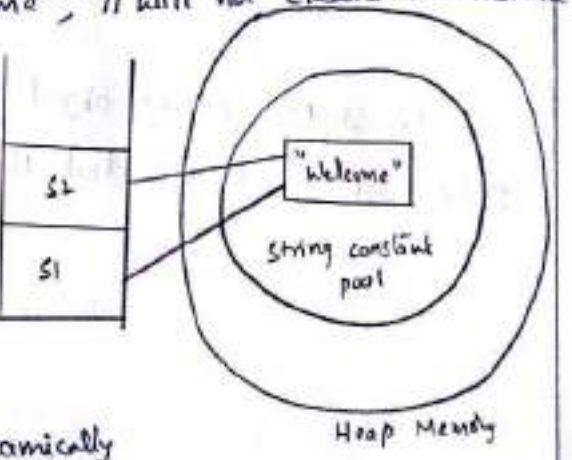
Java String literal is created by using double quotes

Ex:- `String s = "Welcome";`

When we create a string literal, the JVM checks the string constant pool first, if the string already exists in the pool, a reference to the pooled instance is returned.
If String doesn't exist in the pool, a new string instance is created and placed in the pool.

for example:

```
String s1 = "Welcome";
String s2 = "Welcome"; // will not create new instance.
```



Note:- String objects are stored in a special memory area known as string constant pool. It is an constant table to store String literals.

Note:- A heap is a general term used to (to) any memory that is allocated dynamically and randomly. (It is allocated by O.S).

* By new Keyword :-

In This, We can string object by using "new" Keyword.

for example :

String s = new String ("Welcome");

In this case, JVM will create a new string object in heap memory and the literal "Welcome" will be placed in the String constant pool.

Example :-

```
class StringExample
{
public static void main (String args[])
{
 String s1 = "java"; // creating a string by java string literal
 char ch[] = {'s', 't', 'r', 'i', 'n', 'g', 's'};
 String s2 = new String(ch); // converting char array to string
 String s3 = new String ("example"); // creating string by new keyword
 System.out.println(s1);
 System.out.println(s2);
 System.out.println(s3);
 }
}
```

output :- javac StringExample.java
java StringExample
java
Strings
example

→ In Java, string object is immutable That means once a string object is created it cannot be altered.

In java, String class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace() and etc.

* charAt():

This method returns a char value at the given index number. The index starts from 0 (zero).

Syntax:- charAt (int index)

→ It returns char value.

Example:
```
String name = "Madhu";
Char ch = name.charAt(4);
S.o.p(ch);
```
o/p:- u

* concat():

This method is used to combine two strings. And it returns combined string.

Syntax:- concat (String str2)

Example:
```
String s1 = "java string";
s1 = s1.concat(" is immutable");
S.o.p (s1);
```
o/p:- java string is immutable

* contains():

This method searches the sequence of characters in this string. It returns true if sequence of char values are found in the given string otherwise returns false.

Syntax:- contains (char sequence)

Example:
```
String name = "Madhu Naidu Tattikota";
S.o.p (name.contains ("Naidu"));
S.o.p(name.contains ("Reddy"));
```
o/p:- True
false

* equals():

This method is used to compares the two given strings. If both are equal it returns true otherwise it returns false.

Syntax:- equals(anotherstring)

Example:
```
String s1 = "madhu";
String s2= "madhu";
String s3 = "MADHU";
S.O.P(s1.equals(s2));
S.O.P(s1.equals(s3));
```
O/P: true
false.

* length():

This method is used to find the length of string. It returns count of total numbers of characters.

Syntax:- int length();

Example:
```
String s1 = "Madhu";
String s2= "Tatkkota";
S.O.P(s1.length());
S.O.P(s2.length());
```
O/P: 5
9

* substring():

This method returns a part of the string. It is used to extract some part of given string.

Syntax:- substring(int startIndex)
and
substring(int startIndex, int endIndex)

Example:-
```
String s1 = "Madhu";
S.O.P(s1.substring(2,4));
S.O.P(s1.substring(3));
```
O/P: dhu
hu

* startsWith():

This method checks if this string starts with given prefix. It returns true if this starts with given prefix else returns false.

Syntax:- startsWith(String prefix)

Where prefix is sequence of characters.

Example:- String s1 = "Madhu";
s.o.p(s1.startsWith("Ma"));
s.o.p(s1.startsWith("dh"));

o/p: true
false

* endsWith():

This method checks if this string ends with given prefix. It returns true if this ends with given prefix else returns false.

Syntax:- endsWith(String prefix)

Where prefix is sequence of characters.

Example:- String s1 = "Madhu";
s.o.p(s1.endsWith("hu"));
s.o.p(s1.endsWith("dh"));

o/p: true
false.

* replace():

This method is used to replace old characters with new characters in a given string.

Syntax:- replace(char oldchar, char newchar)

Example:- String s1 = "Java is an OOP";
s.o.p(s1.replace('a','i'));
s.o.p(s1.replace("is","was"));

o/p:- Jivi is in OOP
Java was an OOP

* indexOf():-

This method returns index of given character value or substring. If it is not found, it returns -1, The index counter starts from zero.

Syntax:- indexOf (char ch)
indexOf (char ch, int fromindex)
indexOf (String substring)
indexOf (String substring, int fromindex)

Example:
```
String s = "this is madhu";
s.o.p (s.indexOf ('s')); // it returns 3
s.o.p (s.indexOf ('s', 4)); // it returns 6
s.o.p (s.indexOf ("is")); // it returns 2
s.o.p (s.indexOf ("is", 4)); // it returns 5
```

* toLowerCase():-

This method is used to convert given string into lower case letter.

Syntax:- toLowerCase()

Example:
```
String name = "MAdhu";
s.o.p (name.toLowerCase());
```
o/p: madhu

* toUpperCase():-

This method is used to convert all characters in given string into upper case letter.

Syntax:- toUpperCase()

Example:
```
String name = "madhu";
s.o.p (name.toUpperCase());
```
o/p: MADHU.

* Inheritance in Java :-

→ Inheritance is one of the key features of object oriented programming. Inheritance can be defined as the process where one class acquires the properties (Methods and fields) of another class.

→ The idea behind inheritance in Java is that you can create new classes that are built upon existing classes.

→ Inheritance in Java can be best understood in terms of parent and child relationship, also known as super class (parent) and sub class (child) in Java language.

→ Inheritance defines 'is-a' relationship between a super class and its sub class.

→ The main use of inheritance is code reusability.

Syntax of Java inheritance :

```
class subclass-name extends superclass-name
{
  // Methods and fields
}
```

→ In the above syntax, the extends keyword indicates that you are making a new class that derives from an existing class.

→ In simple words, the extends keyword is used to perform inheritance in Java.

→ In the terminology of Java, a class that is inherited is called a super class, the new class is called a subclass.

Example :-

→ In the figure, programmer is the sub class and Employee is the super class.

→ Relationship between two classes is programmer IS-A Employee.

programmer.java

```
class Employee
{
  int salary = 4000;
}
class programmer extends Employee
{
  int bonus = 1000;
  public static void main (String args[])
  {
    programmer p = new programmer();
    System.out.println ("programmer salary is :" + p.salary);
    System.out.println (" Bonus of programmer is :" + p.bonus);
  }
}
```

output :-

```
javac programmer.java
java programmer
programmer salary is : 4000
Bonus of programmer is : 1000
```

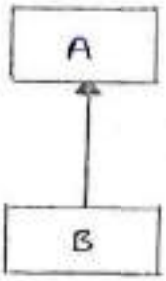
Types of inheritance:

In Java, there can be three types of inheritance those are → Single inheritance
→ Multilevel inheritance
→ Hierarchical inheritance

Note:- Multiple inheritance is not supported in Java.

* Single inheritance:

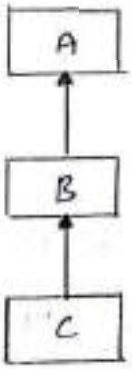
When a class extends another one class only then we call it a single inheritance.



. In the figure, the class B extends the class A. Here A is a parent class and B is a child class.

* Multilevel inheritance:

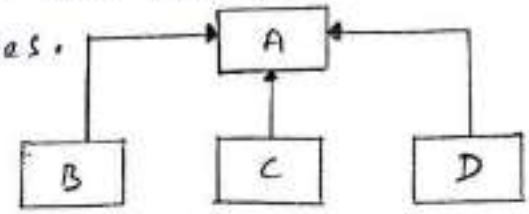
When a class is derived from another class and it acts as the parent class to other class, is known as multilevel inheritance.



. In the figure, the class 'B' inherits properties from class 'A' and again class 'B' acts as a parent to class 'C'.

* Hierarchical inheritance:

In this, one parent class will be inherited by many sub classes.

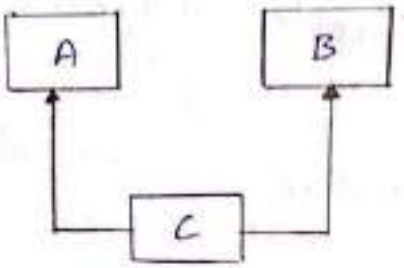


. In the figure, the class 'A' will be inherited by class 'B' class 'C' and class 'D'.

We have two more inheritances: Multiple & Hybrid, which are not directly supported by Java.

* Multiple Inheritance:

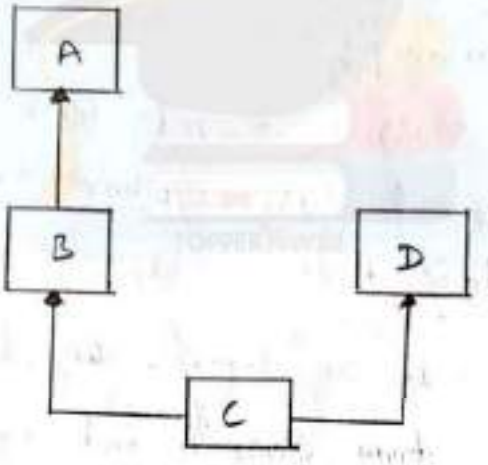
Multiple Inheritance is nothing but one class extending more than one class. It is basically not supported by many Object Oriented programming languages such as Java, Small talk.



Note:- We can achieve multiple inheritance in Java using interfaces.

* Hybrid Inheritance:

Hybrid inheritance is the combination of both Single and Multiple Inheritance. It is nd directly supported in Java only through interfaces we can achieve this.



Note:-

A class member that has been declared as private will remain private to its class. It is not accessible by any code outside its class, including subclasses.

Examples:-

* Single Inheritance Example

```
class A
{
  void dispA()
  {
   System.out.println("disp method of class A");
  }
}
class B extends A
{
 void dispB()
 {
  System.out.println("disp method of class B");
 }
 public static void main(String args[])
 {
   B b = new B();
   b.dispA(); //call dispA() method of class A
   b.dispB(); //call dispB() method of class B
 }
}
```

output:- javac B.java
java B
disp method of class A
disp method of class B

* Multilevel Inheritance Example

```
class A
{
 void dispA()
 {
  System.out.println("disp method of class A");
 }
}
class B extends A
{
```

P.T.O

```
  void dispB()
  {
   System.out.println("disp method of class B");
  }
}
class C extends B
{
  void dispC()
  {
   System.out.println("disp method of class C");
  }
  public static void main (String args[])
  {
    C c = new C();
    c.dispA(); // call dispA() of class A
    c.dispB(); // call dispB() of class B
    c.dispC(); // call dispC() of class C
  }
}
```

output:-

```
javac C.java
java C
disp method of class A
disp method of class B
disp method of class C
```

* Hierarchical Inheritance Example

```
class A
{
 public void dispA()
 {
  System.out.println("disp method of class A");
 }
}
class B extends A
{
 public void dispB()
 {
  System.out.println("disp method of class B");
 }
}
```

```
class C extends A
{
 public void dispC()
 {
  System.out.println("disp method of class C");
 }
}
class D extends A
{
 public void dispD()
 {
  System.out.println(" disp method of class D");
 }
}
class HInheritance
{
 public static void main(String args[])
 {
  B b= new B();
  b.dispB();
  b.dispA();
  C c= new C();
  c.dispC();
  c.dispA();
  D d= new D();
  d.dispD();
  d.dispA();
 }
}
```

output:

```
Javac HInheritance.java
Java HInheritance
disp method of class B
disp method of class A
disp method of class C
disp method of class A
disp method of class D
disp method of class A
```

* Access Modifiers (or) Member access rules in Java :-

The access modifiers in Java specifies accessibility (scope) of a data member, method, constructor or class.

There are 4 types of access modifiers in Java

1. private
2. default
3. protected
4. public

1. private :-

The private access modifier is accessible only within class.

Example :-

```
class A
{
private int data = 40;
private void msg()
{
 System.out.println("Hello Java");
}
}
public class Simple
{
 public static void main (String args[])
 {
   A obj = new A();
   System.out.println(" obj.data);
   obj.msg();
 }
}
```

output: Javac Simple.Java
compile Time Error
compile Time Error

In the above example, We have created two classes 'A' and 'Simple'. Class 'A' contains private data member and private method. We are accessing these private members from outside the class, so there is compile time error.

2. default:-

If you don't use any modifier, it is treated as default by default. The default modifier is accessible only within package.

Example:-

```
// Save by Adef.java
package pack1;
class Adef
{
 void msg()
 {
  System.out.println("Hello");
 }
}
```

- In This example, we have created two packages pack1 and pack2. We are accessing the Adef class from outside it's package, so it cannot be possible to access from outside the package with default access modifier.

```
// Save by Bdef.java
package pack2;
import pack.*;
class Bdef
{
 public static void main(String args[])
 {
  Adef obj = new Adef(); // Compile Time Error
  obj.msg(); // Compile Time Error
 }
}
```

output: Javac Bdef.java
Compile Time Error.

Example:-

```
class Adef {
  void msg() { System.out.println("Hello"); }
 }
class Bdef {
 public static void main(String args[])
 {
   Adef obj = new Adef();
   obj.msg();
 }
}
```

output:- Javac Bdef.java
Java Bdef
Hello.

3. protected :-

The protected access modifier is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.

Example :-

// Save by ProtectA.java

```
package pack;
public class protectA
{
 protected void msg()
 {
  System.out.println("Hello");
 }
}
```

Note :- In compilation, where -d specifies the destination where to put the generated class file.

* We can use . (dot) to keep the package within the same directory.

// Save by ProtectB.java

```
package mypack;
import pack.*;
class protectB extends protectA
{
 public static void main(String args[])
 {
  protectB obj = new protectB();
  obj.msg();
 }
}
```

output :- javac -d . protectA.java
javac -d . protectB.java
java mypack.protectB
Hello.

In the above example, we have created the two packages pack and mypack. In the package pack, the msg method is declared as protected, so it can be accessed from outside the class only through inheritance.

4. public :-

The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.

Example :-

```
Package pack;                    // save by X.java
public class X
{
 public void msg()
 {
  System.out.println(" Hello");
 }
}
```

```
package mypack;
import pack.*;          // save by Y.java
class Y
{
public static void main (String args[])
{
 X obj = new X();
 obj.msg();
}
}
```

output:- javac -d . X.java
javac -d . Y.java
java mypack.Y
Hello

Let's understand the access modifiers by simple table

Access Modifier	Within class	Within package	outside package by subclass only	outside package
private	Y	N	N	N
Default	Y	Y	N	N
protected	Y	Y	Y	N
public	Y	Y	Y	Y

note:- Y - Yes, N - No.

* Super Keyword :-

The super keyword in java is a reference variable that is used to refer immediate parent class.

The super keyword is used for

- → Accessing the variables of the parent class
- → calling the methods of the parent (or) super class
- → Invoking the constructors of the parent class.

Example :-

Bike.java

```
class Vehicle
{
  int speed = 50;
  void message()
  {
    System.out.println("Welcome to Vehicle class");
  }
}
class Bike extends Vehicle
{
  int speed = 100;
  void message()
  {
    System.out.println("Welcome to Bike class");
  }
  void display()
  {
    System.out.println("Bike speed is :" + speed); // call variable of local
    System.out.println("Vehical Avg Speed is :" + super.speed);
                                                  // variable of parent class
    message(); // invoke local method
    super.message(); //invoke parent class method
  }
  public static void main(String args[])
  {
    Bike b = new Bike();
    b.display()
  }
}
```

output:- javac Bike.java

java Bike

Bike speed is : 100

Vehical Avg speed is : 50

Welcome to Bike class

Welcome to Vehicle class

→ The super() is used to invoke the parent class constructor

Example :-

Car.java

```
class Vehicle
{
 Vehicle()
 {
  System.out.println("Vehicle is created");
 }
}
class Car extends Vehicle
{
 Car()
 {
  super(); // invoke parent class constructor.
  System.out.println("Car is created");
 }
 public static void main(String args[])
 {
   Car c = new Car();
 }
}
}
```

output:-

```
javac Car.java
java Car
Vehicle is created
Car is created
```

* Final Keyword :-

The final Keyword in java is used to restrict the user. The final Keyword can be applied on variables, methods and classes.

The Keyword final is used for the following reasons,

→ The final Keyword can be applied on variables, to declare constants.

→ The final Keyword can be applied on methods to disallow method overriding

→ The final Keyword can be applied on classes to prevent (or) disallow inheritance.

* final variable:

If you declare any variable as final, you cannot change the value of final variable (It will be constant).

Example:- Glamour.java

```
class Glamour
{
final int speedlimit=100;// final variable
void run()
{
 Speedlimit = 400;
}
public static void main(String args[])
{
 Glamour obj = new Glamour();
 obj.run();
}
}
```

output:- javac Glamour.java

Compile Time Error:
cannot assign a value to final variable

In the above example, we cannot able change the value of variable speedlimit, because it is declared as final.

* final method:

If you declare any method as final, you cannot override that method. It is called as final method.

Example:- Hero.java

```
class Bike
{
final void run() // final method
 {
  System.out.println("running");
 }
}
class Hero extends Bike
{
 void run()
 {
  System.out.println("running safely with 60mph");
 }
```

```
public static void main(String args[])
{
 Hero obj = new Hero();
 obj.run();
}
}
```

output: javac Hero.java

Compile Time Error:
overridden method is final

* final class:

If you declare any class as final, you cannot extend it. i.e we cannot inherit that class. It is called as final class.

Example:-

Honda.java

```
final class Bike // final class
{
 void run()
 {
  System.out.println("running safely with 60kmph");
 }
}
class Honda extends Bike
{
public static void main(String args[])
{
 Honda obj = new Honda();
 obj.run();
}
}
```

output: javac Honda.java

Compile Time Error:
Cannot Inherit from final Bike

* In simple words, The final keyword in Java

→ Stop value change.

→ Stop Method overridding.

→ Stop Inheritance.

* Object class :-

In java, there is one special class i.e "Object" class. The object class is the parent (or) super class of all the classes in java by default. In other words, All other classes are sub-classes of object class.

→ It is the topmost class of java.

→ The object class is helpful if you want to refer any object of any class.

The object class defines following methods, which means that they are available in every object.

* object clone():
creates a new object that is the same as the object being cloned.

* boolean equals (object obj):
Determines whether one object is equal to another.

* void finalize():
called before an unused object is recycled.

* class getClass():
obtains the class of an object at run-time.

* void wait():
Waits on another thread of execution.

* String toString():
Returns a string that describes the object.

* int hashCode():
Returns the hashcode number for this object.

* void notify():
Resumes execution of thread waiting on the invoking object.

* Polymorphism :-

Polymorphism is a concept by which we can perform a single action by different ways.

Polymorphism is derived from two greek words : 'poly' and 'morphs'. The word "poly" means many and "Morphs" means forms. So polymorphism means many forms.

There are two types of polymorphism in Java, those are compile time polymorphism and runtime polymorphism. We can perform polymorphism in java by method overloading and method overriding.

The run time polymorphism can be done by method overriding in java.

* Method Overriding :-

If child class has the same method as declared in the parent class, it is known as method overriding.

In other words, if sub class provides the specific implemenation of the method that has been provided by one of its parent class, it is known as method overriding.

→ Method overriding is used to provide specific implemenation of a method that is already provided by its super class.

→ Method overriding is used for runtime polymorphism.

Rules for method overriding

→ Method must have same name as it's in the parent class.

→ Method must have same parameters as in the parent class.

→ There must be Is-A relationship (Inheritance) between parent and child classes.

Example :-

Bike2.java

```
class Vehicle
{
 void run()
 {
  System.out.println("vehicle is running");
 }
}
class Bike2 extends vehicle
{
 void run()
 {
  System.out.println("Bike is running safely");
 }
 public static void main(String args[])
 {
   Bike2 obj = new Bike2();
   obj.run();
 }
}
```

output: javac Bike2.java
java Bike2
Bike is running safely

In the above example, we have defined the run method in the sub class as defined in the parent class but it has some specific implemenation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

Note:- static or final method cannot be overridden.

Note:- Remember when you write two or more classes in a single file, the file will be named upon the name of the class that contains the main method.

* Dynamic Binding :-

Binding is the process of connecting a method call to its body.

There are two types of binding

1. static binding (early binding)
2. Dynamic binding (late binding)

1. static binding :

When binding is performed before a program is executed i.e at compile time is called as static binding (or) early binding.

The static binding is performed by compiler when we overload methods.

i.e When multiple methods with the same name exists with in a class (i.e Method overloading) which method will be executed depends upon the arguments passed to the method. So, This binding can be resolved by the compiler.

Example :-

```
class Animal
{
 void eat() //Method without arguments
 { System.out.println("animal is eating");
 }
 void eat(String food) // method with string argument
 { System.out.println("dog is eating..." + food);
 }
 public static void main(String args[])
 {
   Animal a = new Animal();
   a.eat();
   a.eat("Biscuits");
 }
}
```

o/p: javac Animal.java
java Animal
animal is eating
dog is eating... Biscuits

2. Dynamic Binding:

When binding is performed at the time of execution i.e at runtime is called as dynamic binding (or) late binding.

The dynamic binding is performed by JVM (Java virtual Machine) when we overridden methods.

i.e When a method with the same name and signature exists in superclass as well as subclass (i.e Method overriding). Which method will be executed (superclass version or sub class version) will be determined by the type of object. The objects exists at runtime. So this binding is done by JVM.

Example:

```
class Animal
{
 void eat()
 {
  System.out.println("Animal is eating...");
 }
}
class Dog extends Animal
{
 void eat()
 {
  System.out.println("dog is eating...");
 }
 public static void main(String args[])
 {
  Animal a = new Dog();
  a.eat(); //call method in child class
  Animal a1 = new Animal();
  a1.eat(); //call Method in parent class
  Dog d= new Dog();
  d.eat(); //call Method in child
 }
}
```

O/P: javac Dog.java
java Dog

```
dog is eating...
Animal is eating...
dog is eating...
```

* Abstract class :-

Abstraction is a process of hiding the implementation details and showing only functionality to the user.

In another way, it shows only important things to the user and hides the internal details for example sending sms, you just type the text and send the message. you don't know the internal processing about the message delivery.

Def":- A class that is declared with abstract keyword, is known as abstract class in Java. It contains one or more abstract methods.

→ An abstract class can have abstract and non-abstract methods.

Abstract Method:

A method that is declared as abstract and does not have implementation is known as abstract method.

(or)

An abstract method is a method that is declared with no body or implementation.

example:- abstract void run();
abstract void display();

Example for abstract class:-

Ex1:-
```
abstract class Bike
{
  abstract void run();
}
```

Ex2:-
```
abstract class Animal
{
  abstract void sound();
  abstract void eat();
```

Example :-

```
abstract class Animal //abstract class
{
abstract void sound(); //abstract method
void eat(String food) // normal method
{
System.out.println("this animal likes "+food);
}
}
class Lion extends Animal
{
void sound()
{
System.out.println("Lions Roar! Roar!");
}
public static void main(String args[])
{
Lion l=new Lion();
l.sound();
l.eat("flesh");
}
}
```

output:- javac Lion.java
java Lion
Lion Roar! Roar!
this animal likes flesh

→ The abstract classes cannot be instantiated, and they require subclasses to provide implementation for their abstract methods by overriding them and then the subclasses can be instantiated.

→ Abstract classes contain one or more abstract methods. It does not make any sense to create an abstract class without abstract methods, but if done, the Java compiler does not complain about it.

→ An abstract class can have data member, abstract method, method body, constructor and even main() method.

* Package :-

A package is a group of classes, interfaces and sub-packages.

packages are used in java, in-order to avoid name conflicts and to control access of class, interface and enumeration and etc. using package it becomes easier to locate the related classes.

package are categorized into two forms

→ Built-in package

→ user-defined package.

* There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql and etc.

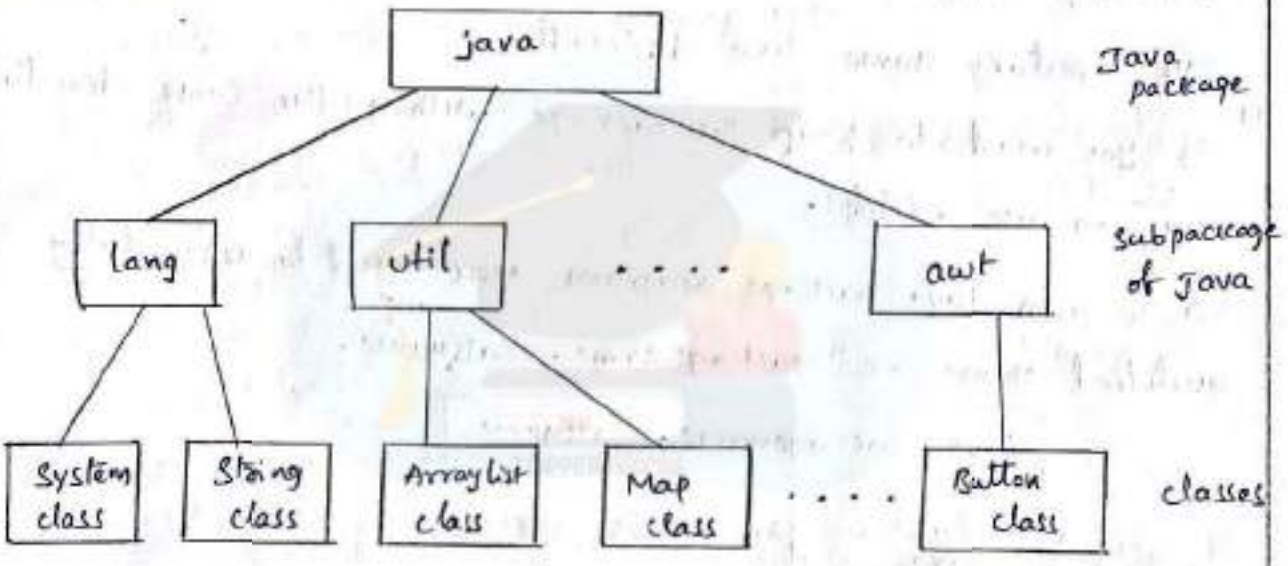


fig:- Built-in package

In the above figure, java is main package and it has many sub packages (lang, util, io, net, ... awt). And each subpackage has several classes (system, string, map ..... Button class).

* In java, the user can able to create package by using a keyword i.e "package". The package keyword is used to create a package in java.

```
package <package name>;
```

example :- Simple.java

```
package mypack;
public class Simple
{
public static void main(String args[])
{
System.out.println("Welcome to package");
}
}
```

↳ To compile java package, you need to follow the syntax

```
javac -d directory javafilename
```

In the above syntax, the -d specifies the destination (or) directory where to put the generated class file. you can say any directory name like d:/madhu.

If you want to keep the package within the same directory, you can use . (dot).

↳ To run java package program, you need to use fully qualified name i.e packagename.classname.

```
java packagename.classname
```

output:- javac -d . Simple.java

java mypack.Simple

Welcome to package

Note:- If we declare package in source file, the package declaration must be the first statement in the source file.

Since the package creates a new namespace there won't be any name conflicts with names in other packages. using packages, it is easier to provide access control and it is also easier to locate the related classes.

* Importing packages :-

The import Keyword is used to import built-in and user-defined packages into your java source file. So that your class can import to a class that is in another package by directly using its name.

There are three ways to access the package from outside the package.

1. using fully qualified name
2. import the only class you want to use.
3. import all the classes from the particular package.

1. using fully qualified name:

If you use fully qualified name then only declared class of this package will be accessible. Now there is no need to import. But you need to use fully qualified name every time when you are accessing the class or interface.

It is generally used when two packages have same class name e.g java.util and java.sql packages contains 'Date' class.

Example:-

A.java

```
package pack;
public class A
{
public void msg()
{
 System.out.println("Hello");
}
}
```

B.java

```
package mypack;
class B
{
public static void main(String args[])
{
pack.A obj = new pack.A(); // using fully qualified name
obj.msg();
}
}
```

output:- javac -d . A.java
javac -d . B.java
java mypack.B
Hello.

2. Using packagename.classname (or) import the only class you want to use:

If you import package.classname then only declared class of this package will be accessible.

* In this, we can import only class that you want to use. i.e we can able to access that class in another package.

Example:-

X.java

```
package pack;
public class X
{
public void msg()
{
System.out.println("Hello");
}
}
```

Y.java

```
package mypack;
import pack.X;
class Y
{
public static void main(String args[])
{
```

```
    X obj = new X();
    obj.msg();
  }
}
```

output:- javac -d . x.java
javac -d . y.java
java mypack.Y
Hello.

3. Import all the classes from the particular package (or) using packagename.* :-

If you use package.* then all the classes and interfaces of this package will be accessible but not subpackages.

The import keyword is used to make the classes and interface of another package accessible to the current package.

Example:-

Humans.java

```
package animals;
class Animals
{
public void display()
{
 System.out.println(" All are the animals in the world");
}
}
public class Humans extends Animals
{
public void msg()
{
 System.out.println (" class A animals are Humans");
}
public void msgBC)
{
 System.out.println(" Humans are animals with intelligence");
}
}
```

World.java

```
package world;
import animals.*;
class World
{
public static void main(String args[])
{
 Humans obj = new Humans();
 obj.display();
 obj.msg();
 obj.msgB();
}
}
```

output:- javac -d . Humans.java
javac -d . World.java
java world.World
All are the animals in the World
Class A animals are Humans
Humans are animals with intelligence.

Note:- If you import a package, subpackages will not be imported. i.e If you import a package, all the classes and interfaces of that package will be imported excluding the classes and interfaces of subpackages. Hence, you need to import the subpackages as well.

Note:- Sequence of the program must be package then import then class. i.e import statement is kept after the package statement.

Ex
package statement
import statement
class statement
;

} This is sequence in java program.

Rule:- There can be only one public class in java source file and it must be saved by the public class name.

* Subpackage :-

package inside the package is called the subpackage. It should be created to categorize the package further.

The standard of defining subpackage is packagename.subpackagename. for example "pack.subpack".

Example :-

Simple.java

```
package pack.subpack;
class Simple
{
public static void main(String args[])
{
System.out.println("Hello Subpackage");
}
}
```

output:- javac -d . Simple.java

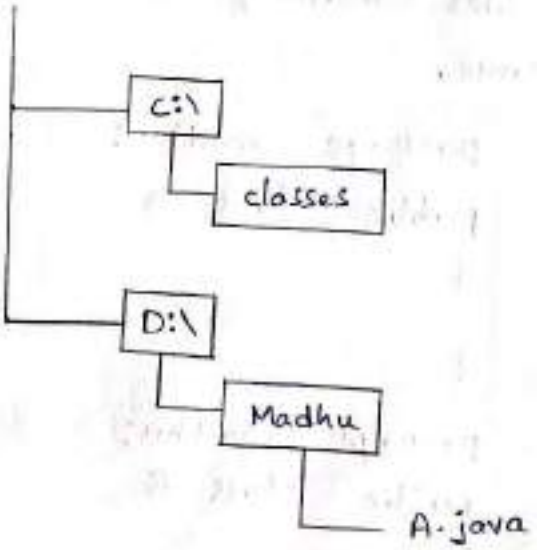
java pack.subpack.Simple

Hello Subpackage

* Setting CLASSPATH :-

If you want to save (or) put Java source-files and class files in different directories & drives then we need to set classpath to run or execute those class files.

for example:



Now, I want to put the class file of A.java source file in the folder of C: drive.

Example:-

S.java

```
package pack;
public class S
{
public static void main(String args[])
{
System.out.println("This is example of setting classpath");
}
}
```

To compile:

```
D:\Madhu > javac -d c:\classes S.java
```

To Run:

To run this program from D:\Madhu directory, you need to set classpath of the directory where the class file resides.

```
D:\Madhu > set classpath = c:\classes;.;
D:\Madhu > java pack.S
This is example of setting classpath.
```

* How to put two public classes in a package :-

In java, There can be only one public class in a package. If you want to put two public classes in a single package, have two java source files containing one public class, but keep the package name same.

for example:

```
package madhu;
public class A
{
  =
}
```

// Save as A.java

```
package madhu;
public class B
{
  =
}
```

// Save as B.java

* Exception Handling :-

The exception handling is one of the powerful Mechanism to handle the runtime errors so that normal flow of the application can be maintained.

* Exception :-

In general, Exception is an abnormal condition. In java, exception is an event that disrupts the normal flow of the program. (or)

An exception is a problem that arises during the execution of a program. When an Exception occurs the normal flow of the program is disrupted and the program terminates abnormally.

An exception can occur for many different reasons. The following are some scenarios where an exception occurs.

- → A user has entered an invalid data.
- → A file that needs to be opened cannot be found.
- → A network connection has been lost in the middle of comm-unication.

Some of these exceptions are caused by user error, others by programmer error, and others by physical resources that have failed in some manner.

Based on these, the exceptions are categories into three types. Those are

- → checked exceptions
- → unchecked exceptions
- → Errors.

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception.

→ Checked exceptions:

A checked exception is an exception that occurs at the compile time, these are also called as compile time exceptions. These exceptions cannot simply be ignored at the time of compilation, the programmer should take care of these exceptions.

Examples:

File Not Found Exception
Class Not Found Exception
NoSuchFieldException
EOFException, etc.

The checked exceptions can be handle (checked) at the time of compilation.

→ Unchecked exceptions:

An unchecked exceptions is an exception that occurs at the time of execution. These are also called as Runtime exceptions. These include programming bugs, such as logic errors or improper use of API. Runtime exceptions are ignored at the time of compilation.

Examples:

ArithmeticException
ArrayIndexOutOfBoundsException
NullPointerException
NegativeArraySizeException, etc.

The unchecked exceptions can be checked at runtime.

→ Errors:

These are not exceptions at all, but problems that arise beyond the control of the user or the programmer. Error defines problems that are not excepted by programmer or program.

Examples: Memory Error,
Hardware error, JVM error etc.

→ An exception is nothing but runtime error. It can be handled to provide a suitable message to user.

→ Java uses a mechanism or model to handle exceptions, This mechanism is known as Exception Handling Mechanism.

→ Exception Handling is a mechanism to handle runtime errors such as classNotFound, ArithmeticException, IO, etc.

→ The core advantage of exception handling is to maintain the normal flow of the application. Normally exception interrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

```
statement 1;
statement 2;
statement 3;
statement 4;
statement 5; // exception occurs
statement 6;
statement 7;
statement 8;
statement 9;
statement 10;
```

Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of code will not be executed i.e statement 6 to 10 will not run.

If we perform exception handling, rest of the statements will be executed. That is why we use exception handling in java.

→ There are 5 keywords used in java exception handling.

- try
- finally
- Throws
- catch
- Throw

* Exception class :-

The exception class can handle any kind of exceptions by using built-in exception classes. There are many built-in classes in Exception class. Some of them are.

- ArithmeticException : Arithmetic error
- ArrayIndexOutOfBounds Exception : Array index out of bounds
- NullPointerException : Accessing an object through null pointer
- NumberFormatException : Given no. is not number
- ClassNotFoundException : class not found
- IOException : Input/output exception
- FileNotFoundException : unable to locate a file
- SQLException : SQL statement is not well.

→ Exception Handling requires the following four steps

1. Finding the problem (Identify the statements whose execution may result in exception. put all those statements in a try {...} block.)
2. Inform that an exception is thrown (Throw the exception)
3. Receive the exception (Catch the exception using catch{..} block).
4. provide exception handling code in catch block.

* Try and catch block :-

• Try block:

It is used to enclose the code that might throw an exception. i.e all statements that are likely to raise an exception at run time are kept in try block. This block will detect an exception and throws it. It will be handled by catch block.

Try block must be followed by either catch or finally block.

Syntax:- Try - catch

```
try
{
 // code that may throw exception
}
catch (Exception_classname ref)
{
 // statements
}
```

• catch block:

It is used to handle the exception. It must be used after the try block only. It provides a suitable message to the user, then user will be able to proceed with the appli -cation.

→ A try block can follow multiple catch blocks, i.e we can use multiple catch blocks with a single try.

→ try { } block may have one or multiple statements.

→ try { } block may throw a single type of exception or

multiple exceptions. But at a time it can throw only single type of exception.

Syntax:-

```
try
{
 statements that may throw exceptions
}
catch (ExceptionType 1  e1) {...}
catch (ExceptionType 2  e2) {...}
   :
catch (ExceptionType n  en) {...}
```

Example:-

SampleException.java

```
class SampleException
{
public static void main (String args [])
{
 int a=0;
 int b= 10;
 try
 {
  int c = b/a;
  System.out.println("The result is " + c);
 }
 catch (ArithmeticException e)
 {
  System.out.println("Divided by zero");
 }                          (or)
                        e.toString()
  int sum= a+b;
  System.out.println("Sum is" + sum);
 }
}
```

o/p:- javac SampleException.java
java SampleException
Divided by zero (or) java.lang.ArithmeticException: / by zero
Sum is 10.

* Multiple catch blocks :-

A try block can be followed by multiple catch blocks. you can have any number of catch blocks after a single try block.

If an exception occurs in try block then the exception is passed to the first catch block in the list. If the exception type matches with the first catch block it gets caught, if not the exception is passed down to next catch block.

Rule:- At a time only one Exception is occured and at a time only one catch block is executed.

Rule: All catch blocks must be ordered from most specific to most general. i.e. catch for ArithmeticException must come before catch for Exception.

Example:-

MultipleException.java

```
class MultipleException
{
public static void main(String args[])
{
 try
 {
  int arr[] = {1,2};
   arr[4] = 3/0;
 }
 catch (ArithmeticException ae)
 {
  System.out.println(" Divide by zero");
 }
 catch (ArrayIndexoutofBoundsException e)
 {
  System.out.println(" array index out of bound");
 }
}
```

o/p:- javac MultipleException.java
java MultipleException
Divide by zero.

Example for unreachable catch block

While using multiple catch statements, it is important to remember that all catchs blocks must be ordered from most specific to most general. i.e we have to write general exception class as last catch block.

URException.java

```
class URException
{
public static void main (String args[])
{
 try
 {
  int a[] = {1,2};
    a[3] = 5/0;
 }
 catch (Exception e) // This block handles all exceptions
 {
  System.out.println(" Generic Exception");
 }
 catch (ArithmeticException ae) // This block is unreachable
 {
  System.out.println(" Divided by zero");
 }
}
}
```

output:- javac URException.java

compile time Error

The above program aviases compile time error, because you write general exception catch block as first catch block. so we need to write most specific exceptions first and then general exception catch block.

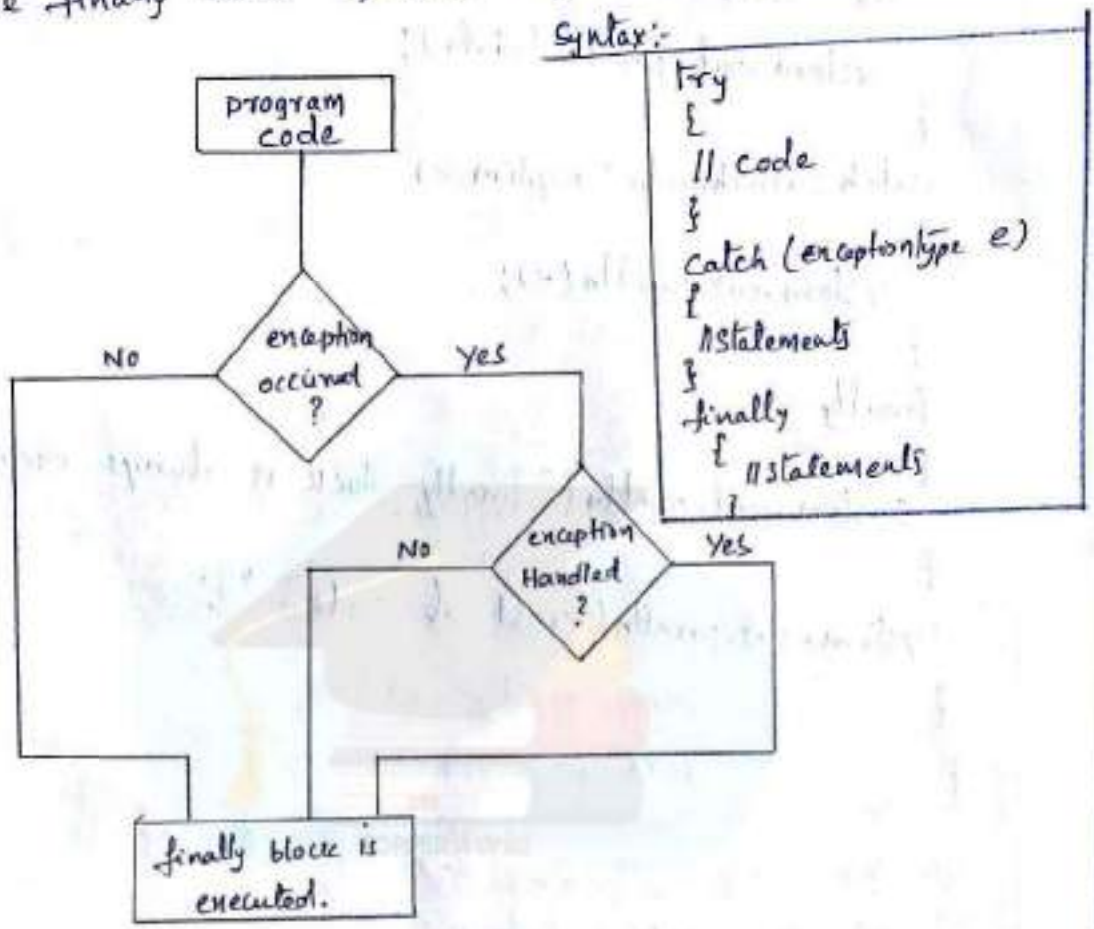
The general Exception class handles all exceptions if no other specific exception catch blocks.

*finally block:-

The finally block is used to execute important code such as closing connection, closing a file etc.

finally block is always executed whether exception is handled or not.

The finally block follows try or catch block.



Syntax:-

```
try
{
// code
}
catch (exceptiontype e)
{
//statements
}
finally
{ //statements
}
```

Note:- If you don't handle exception, before terminating the program JVM executes finally block (if any).

→ Finally block in Java can be used to put "clean up" code such as closing a file, closing connection etc.

→ Let's see the different cases where finally block can be used/executed.

The finally block can be executed when/where exception doesn't occur, exception occurs and not handled, exception occurs and handled.

Rule:- For each try block there can be zero or more catch blocks, but only one finally block.

Example:- Where exception doesn't occur

Finallycase1.java

```
class FinallyCase1
{
public static void main(String args[])
{
 try
 {
  int data = 25/5;
  System.out.println(data);
 }
 catch (ArithmeticException e)
 {
  System.out.println(e);
 }
 finally
 {
  System.out.println("finally block is always executed");
 }
 System.out.println("rest of code...");
}
}
```

```
// javac FinallyCase1.java
   java FinallyCase1
   5
   finally block is always executed
   rest of code...
```

Example where exception occurs and not handled

Finallycase2.java

```
class Finallycase2
{
 public static void main(String args[])
 {
  try
  {
   int data = 25/0;
   System.out.println(data);
  }
```

```
catch (NullPointerException e)
{
 System.out.println(e);
}
finally
{
 System.out.println("finally block is always executed");
}
System.out.println("rest of the code...");
}
}
```

O/P:- javac FinallyCase2.java
java FinallyCase2
finally block is always executed
Exception in thread "main" java.lang.ArithmeticE
xception: /by zero

Example: Where exception occurs and handled.

FinallyCase3.java

```
class FinallyCase3
{
public static void main(String args[])
{
 try
 {
  int data=25/0;
  System.out.println(data);
 }
 catch (ArithmeticException e)
 {
  System.out.println("Divided by zero");
 }
 finally
 {
  System.out.println("finally block is always executed");
 }
 System.out.println("rest of the code...");
 }
}
```

O/P: javac FinallyCase3.java
java FinallyCase3
Divided by zero
finally block is always execute
rest of code...

* throw Keyword :-

The java throw Keyword is used to explicitly throw an exception. We can throw either checked or unchecked exception in java by throw Keyword.

program execution stops on encountering throw statement.

Syntax:- throw throwableinstance

(or)

throw exception

Example:

In this example, We have created the validate method that takes integer value as parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message "welcome to vote".

ThrowExp.java

```
class ThrowExp
{
void validate (int age)
{
  if (age<18)
    throw new ArithmeticException ("not valid");
  else
    System.out.println ("welcome to vote");
}
public static void main(String args[])
{
    ThrowExp t = new ThrowExp();
     t.validate (13);
     System.out.println("rest of code");
}
}
```

o/p: javac ThrowExp.java

java ThrowExp

java.lang.ArithmeticException: not valid

* Throws Keyword :-

The throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

The throws keyword mainly applied on methods to provide information to caller of the method about the exception.

Syntax:-

```
return_type method_name() throws exception_class_name
{
 // method code
}
```

Example :-

Let's see the example of java throws keyword which describes that exceptions.

```
public class ThrowsDemo {
void validate (int age) throws ArithmeticException
{
  if (age <18)
    throw new ArithmeticException ("not valid");
  else
    System.out.println (" welcome to vote");
}
public static void main (String args[])
{ try
  {
    ThrowsDemo td = new ThrowsDemo();
    td.validate(12);
  }
  catch (ArithmeticException e)
  { System.out.println(e);
  }
  System.out.println (" rest of code..");
}}
```

```
o/p: javac ThrowsDemo.java
     java ThrowsDemo
     java.lang.ArithmeticException
     : not valid.
     rest of code..
```

* Multithreading :-

* Introduction :-

First, we need to know about Multitasking.

→ Multitasking is a process of executing multiple tasks simultaneously. We use multitasking to utilize the CPU.

Example: class room student
1. listening
2. writing

→ Multitasking can be achieved by two ways (or) Multitasking is classified into two types

- process-Based Multitasking (Multiprocessing)
- Thread-Based Multitasking (Multithreading)

* process based Multitasking :-

Executing multiple tasks simultaneously, where each task is separate independent process (or) program is called as process based Multitasking.

Example :-

1. Typing a Java program in notepad.
2. Listen audio songs.
3. Download a file from internet.

The above three tasks are performed simultaneously in a system, but there is no dependency between one task & another task.

→ process based Multitasking is best suitable at "operating system" level not at programming level.

* Thread-based Multitasking :-

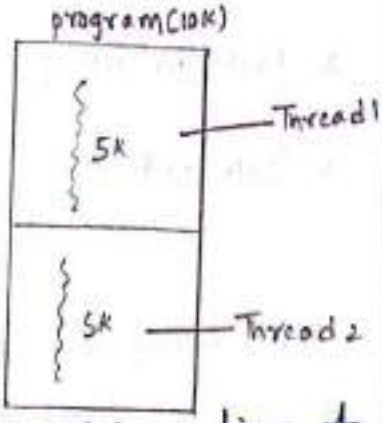
Executing multiple tasks simultaneously, where each task is a separate independent part of the same program (or) process is called Thread-based Multitasking.

→ The each independent part is called a thread.

→ Thread-based Multitasking is best suitable at programming level.

Example:-

Let a program has 10K lines of code, where last 5K lines of code doesn't dependent on first 5K lines of code. Then both are starts the execution simultaneously. so it takes less time to complete execution.



Note:- Any type of Multitasking is used to reduce response time of System and improves performance.

* Multithreading:-

→ A thread is a lightweight sub process, a smallest unit of processing. It is a separate path of execution.

→ Threads are independent, if there occurs exception in on thread, it doesn't affect other threads.

→ It shares a common memory area.

→ As shown in figure, thread is executed inside the process. There can be Multiple process inside the O.S and one process can have multiple threads.

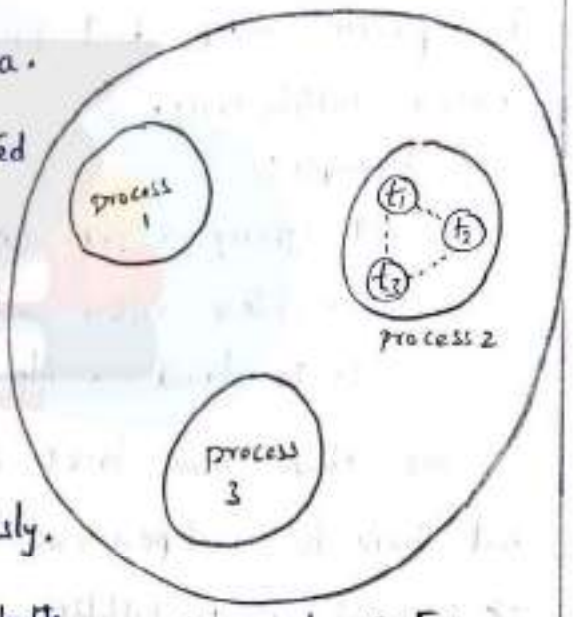


fig: operating System.

Defn:- Multithreading is a process of executing multiple threads simultaneously.

↳ Multiprocessing and Multithreading, both are used to achieve multitasking.

But we use multithreading than multiprocessing because threads shares a common memory area and context-switching between threads takes less time than process.

↳ The main application areas of Multithreading are

→ To develop Multi-media movies
→ TO develop video games
→ To develop web servers & Application Servers, etc.

* Life Cycle of a Thread (or) A Thread Model :-

The Life cycle of the thread is controlled by JVM. And it has five states as follows

→ New
→ Runnable
→ Running
→ Non-Runnable (blocked)
→ Terminated

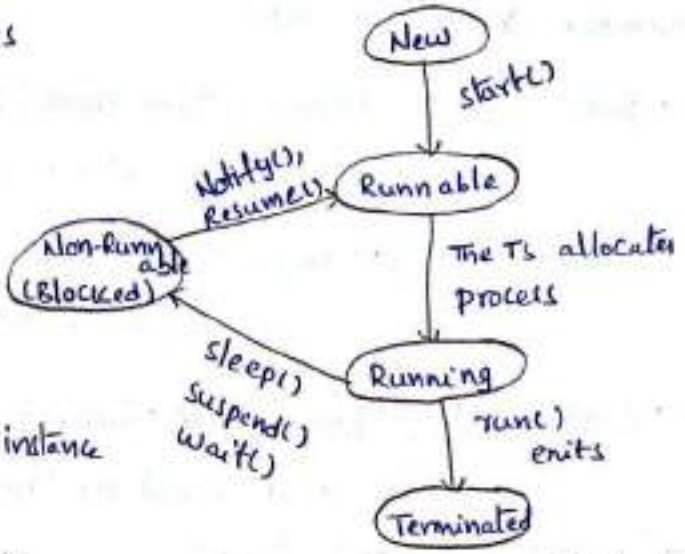


fig:- Life cycle of a Thread.

* New:- In this state, the new instance of thread class is created.
Eg:- Mythread t = new Mythread();

* Runnable :- In this, the thread is ready to execute after invocation of start() method. Eg: t.start();

* Running :- In this, the thread is running by using run() method.

* Non-Runnable (Blocked):-

In this, the thread is in blocked state, i.e the thread is still alive, but is currently not eligible to run.

* Terminated (Dead):-

In this, the thread is in dead state, when run() method exits (or) completed the process.

* Thread class :-

The Thread class provides methods and constructors to create and perform operations on a thread.

↳ Commonly used methods of Thread class:

1. public void run() : It is used to perform action for a thread.
2. public void start() : It is used to starts the execution of the thread. JVM calls the run() method on the thread.

3. public void sleep (long milliseconds):
To stops the execution of thread for the specified number of milliseconds.

4. public void setName (String name):
It is used to set (or) changes the name of a thread.

5. public void getName():
It is used to get the name of a thread.

6. public int setpriority (int priority):
It is used to set(or) change the priority of the thread.

7. public int getpriority():
It returns the priority of the thread.

8. public boolean isAlive():
It checks if the thread is alive or not.

9. public void yield():
It used to pause the currently executing thread and allow other threads to execute.

10. public void suspend():
It is used to suspend the thread.

11. public void resume():
It is used to resume the suspended Thread.

12. public void stop():
It is used to stop the execution of Thread.

↳ Constructors:

1. Thread() - without parameters/arguments.
2. Thread(String name) - with one string argument.
3. Thread(Runnable r) - with runnable argument.
4. Thread(Runnable r, String name) - with runnable and string arguments.

* creating Thread :-

There are two ways to create a thread

↳ By extending Thread class.

↳ By implementing Runnable interface.

↳ By extending Thread class :-

In This, By extending Thread class we can able to create a thread and able to start a thread by calling start() method of Thread class.

Syntax:

```
class classname extends Thread
{
  =
  =
}
```

In thread class, we can mainly use two methods run() and start().

Example:- SimpleThread.java

```
class A extends Thread
{
 public void run()
 {
  for(int i=1; i<=5; i++)
  {
   System.out.println("Thread A value = "+i);
  }
 }
}
class SimpleThread
{
public static void main (String arg[])
{
  A a = new A();
  a.start();
}
}
```

Output: javac SimpleThread.java
java SimpleThread
Thread A value = 1
" " 2
" " 3
" " 4
" " 5

↳ By implementing Runnable interface :-

In this, By implementing runnable interface, we can able to create & execute Thread.

→ The Runnable interface should be implemented by any class.

→ Runnable interface have only one Thread method named run(), where run() method is abstract method.

→ The programmer should declare object for sub class, that object can be used as agument in Thread class constructor from Thread object.

Syntax:

```
class class_name implements Runnable
{
 public void run()
 {
  =
 }
}
```

Example:- ThreadRunDemo.java

```
class A implements Runnable
{
 public void run()
 {
  for(int i=1; i<=5; i++)
  {
   System.out.println("Thread A value = "+i);
  }
 }
}
class ThreadRunDemo
{
public static void main(String a[])
{
  A a = new A();
  Thread t = new Thread(a);
  t.start();
}
}
```

output:-

```
javac ThreadRunDemo.java
java ThreadRunDemo
Thread A value = 1
"          2
"          3
"          4
"          5
```

* Thread priority:-

→ In java, each thread has a priority. priorities are represented by a number between 1 and 10.

→ In most cases, thread schedular schedules the threads according to their priority.

→ But it is not guaranteed because it depends on JVM specification that which scheduling it chooses.

→ There are three constant thread priorities defined in Thread class.

1. public static int MIN_PRIORITY
2. public static int NORM_PRIORITY
3. public static int MAX_PRIORITY

→ The value of MIN_PRIORITY is 1,
The value of NORM_PRIORITY is 5,
and The value of MAX_PRIORITY is 10.

Example:

```
class Thread1 extends Thread
{
public void run()
{
for(int i=1; i<=5; i++)
{
System.out.println("Thread1 i="+i);
}
System.out.println("Thread1 is in terminated state");
}
}
class Thread2 extends Thread
{
public void run()
{
```

```
for (int j=1; j<=5; j++)
{
System.out.println ("Thread2 j=" +j);
}
System.out.println ("Thread2 is in Terminated state");
}
}
class Thread3 extends Thread
{
public void run()
{
for (int k=1; k<=5; k++)
{
System.out.println ("Thread3 k=" +k);
}
System.out.println ("Thread3 is in terminated state");
}
}
class ThreadpriorityDemo
{
public static void main (String arg[])
{
Thread1 a = new Thread1();
Thread2 b= new Thread2();
Thread3 c = new Thread3();
System.out.println ("Default priority for Thread1 is:" +a.getpriority());
System.out.println ("Default priority for Thread2 is:" +b.getpriority());
System.out.println ("Default priority for Thread3 is:" +c.getpriority());
a.setpriority (Thread.MIN_PRIORITY);
b.setpriority (Thread.NORM_PRIORITY);
c.setpriority (Thread.MAX_PRIORITY);
```

```
System.out.println("New priority of Thread1 is:" + a.getPriority());
System.out.println("New priority of Thread2 is:" + b.getPriority());
System.out.println("New priority of Thread3 is:" + c.getPriority());
a.start();
b.start();
c.start();
}
}
```

output:

```
javac ThreadPriorityDemo.java
java ThreadPriorityDemo
Default priority of Thread1 is: 5
"  "  Thread2 is: 5
"  "  Thread3 is: 5
New priority of Thread1 is: 1
"  "  Thread2 is: 6
"  "  Thread3 is: 10
Thread3 k = 1
"  k = 2
"  k = 3
"  k = 4
"  k = 5
Thread3 is in Terminated state
Thread2 j = 1
"  j = 2
"  j = 3
"  j = 4
"  j = 5
Thread2 is in Terminated state
Thread1 i = 1
"  i = 2
"  i = 3
"  i = 4
"  i = 5
Thread1 is in Terminated state
```

* Methods of Thread class :-

→ sleep() :-

The sleep() method of Thread class is used to stop the execution of thread for the specified amount of time.

Syntax: sleep(long milliseconds)

The sleep() may throws an "InterruptedException". i.e we need to use try & catch statements while using sleep() method.

Example 1:

```
class sleepMethod extends Thread
{
 public void run()
 {
  for(int i=1; i<=5; i++)
  {
   try
   {
    Thread.sleep(500);
   }
   catch (InterruptedException e)
   {
    System.out.println(e);
   }
   System.out.println(i);
  }
 }
}
class ThreadSleepDemo
{
 public static void main(String arg[])
 {
  SleepMethod t1 = new SleepMethod();
  SleepMethod t2 = new SleepMethod();
  t1.start();
  t2.start();
 }
}
```

Output:-

```
javac ThreadSleepDemo.java
java ThreadSleepDemo
1
1
2
2
3
3
4
4
5
5
```

Note:- If you sleep a thread for the specified time, the thread scheduler picks up another thread and so on.

→ Naming Thread:-

→ The Thread class provides methods to change and get the name of a thread, those are setName() and getName() methods.

→ By default, each thread has a name i.e Thread-0, Thread-1 and so on.

getName(): It is used to get the name of thread.

Syntax: getName()

setName(): It is used to set (or) change the name of thread

Syntax: setName(String name)

Example:

```
class A extends Thread
{
public void run()
{
System.out.println("Thread A is running...");
}
}
class B extends Thread
{
public void run()
{
System.out.println("Thread B is running...");
}
}
class ThreadNameDemo
{
public static void main(String args[])
{
  A a=new A();
  B b=new B();
  System.out.println("Default Name of Thread A is"
                                    +a.getName());
  System.out.println("Default Name of Thread B is"
                                    +b.getName());
  a.setName("Madhu");
  b.setName("Hari");
```

```
System.out.println("New Name of Thread A is"
                    + a.getName());

System.out.println("New Name of Thread B is"
                    + b.getName());

a.start();
b.start();
}
}
```

```
output: javac ThreadNameDemo.java
        java ThreadNameDemo
Default name of Thread A is Thread-0
    "      "    B is Thread-1

New name of Thread A is Madhu
   "    "      "  B is Hari

Thread A is running
Thread B is running
```

→ Joining Threads:-

Sometimes one thread need to know when other thread is terminating (or not.

In Java, isAlive() and join() are two different methods that are used to check whether a thread has finished its execution or not.

* isAlive(): This method returns true if the thread still running otherwise it returns false.

Syntax: isAlive()

Example: ThreadIsAliveDemo.java

```
class MyThread extends Thread
{
public void run()
{
 System.out.println("r1");
 try
 {
  Thread.sleep(500);
 }
 catch (InterruptedException e)
 {
   System.out.println(e);
 }
} } System.out.println("r2");
```

```
class ThreadisAliveDemo
{
 public static void main(String arg[])
 [
   MyThread t1 = new MyThread();
   MyThread t2 = new MyThread();
   t1.start();
  //t2.start();
   System.out.println(t1.isAlive());
   System.out.println(t2.isAlive());
 }
}
```

output: javac ThreadisAliveDemo.java
java ThreadisAliveDemo
True
false
r1
r2

* join():- This method waits for a specified thread completes its execution. It allows us to specify the time for which you want to wait for the specified thread to terminate.

Syntax: join()
(or)
join(long milliseconds)

→ join() method throw InterruptedException, i.e we need to use try & catch statements while using join() method.

Example:

```
class Thread1 extends Thread
{
 public void run()
{
for(int i=1; i<=5; i++)
{
 try
 {
 Thread.sleep(500);
```

```
 catch (InterruptedException e)
 {
  System.out.println(e);
 }
 System.out.println("Thread1 value is: "+i);
}
}
}
class Thread2 extends Thread
{
public void run()
{
for(int i=1; i<=5; i++)
{
 System.out.println("Thread2 value is :" + i);
}
}
}
class ThreadjoinDemo
{
public static void main(String arg[])
{
 Thread1 t1 = new Thread1();
 Thread2 t2 = new Thread2();
 t1.start();
 try
 {
  t1.join();
 }
 catch (InterruptedException e) { }
  t2.start();
}
}
```

output:-

```
javac ThreadjoinDemo.java
java ThreadjoinDemo
Thread1 value is : 1
"       "    2
"       "    3
"       "    4
"       "    5
Thread2 value is : 1
"       "    2
"       "    3
"       "    4
"       "    5
```

Example :- Example for MultiThreading, Race between Hare & Tortoise story.

```
class Tortoise extends Thread
{
 public void run()
 {
 for (int i=1; i<=101; i++)
 {
  System.out.println ("Distance covered by Tortoise = " +i);
 }
  System.out.println ("Tortoise has completed the Race...");
 }
}
class Hare extends Thread
{
 public void run()
 {
 for(int j=1; j<=101; j++)
 {
  System.out.println ("Distance covered by Hare = " +j);
 }
 System.out.println ("Hare has completed the Race...");
 }
}
class Race
{
public static void main (String args[])
{
  Tortoise t = new Tortoise();
  Hare h = new Hare();
  t.start();
  h.start();
 }
}
```

output :-

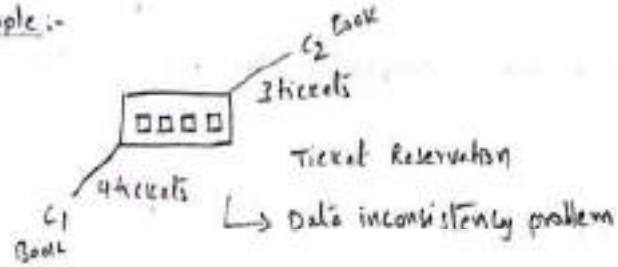
```
javac Race.java
java Race
Distance covered by Tortoise = 1
                .  .  .  2
Distance covered by Hare = 1
                .  .  .  2
Tortoise has completed the Race..
Hare has completed the Race..
```

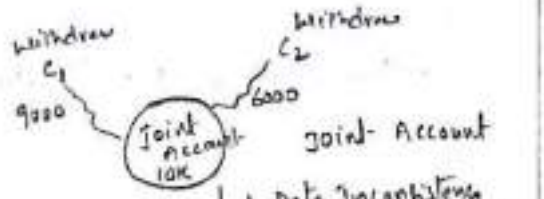
* Synchronization in Threads :-

Synchronization is a process, that allows only one thread to access shared resource at a time, if multiple threads trying to access shared resource.

→ If multiple threads are trying to operate simultaneously on same java object, then there may be a chance of data inconsistency problem.

Example :-





→ To overcome this problem, we should go for Synchronized Keyword.

→ Synchronized is a modifier applicable only for methods and blocks but not for classes & variables.

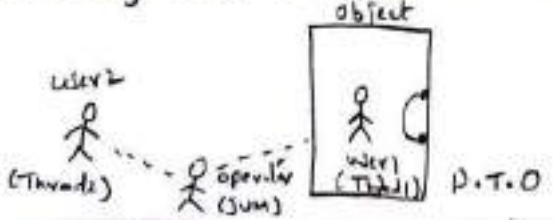
→ If a method declared as Synchronized then at a time only one thread is allowed to execute that method on given object so that data inconsistency problem will be resolved.

→ The main advantage of Synchronized Keyword is we can resolve data inconsistency problems, but the main disadvantage of Synchronized Keyword is it increases waiting time of threads and creates performance problems.

→ It increases response time if there are many users.

Note :- " Hence if there is no specific requirements, then it is not recomended to use Synchronized Keyword.

↳ Internally, Synchronization is implemented by using lock. Every object in java has an unique lock in Java, whenever we are using Synchronized Keyword then only lock concept will come into the picture.

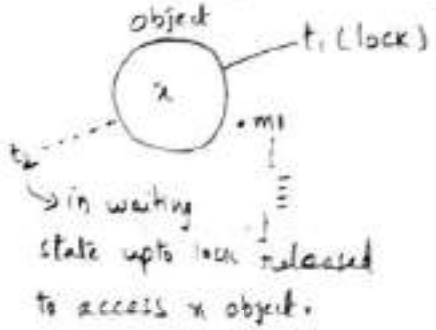


P.T.O

If a thread, wants to execute synchronized method on the given object, first it has to get lock of that object, one thread got the lock then it is allowed to execute any synchronized method on that object.

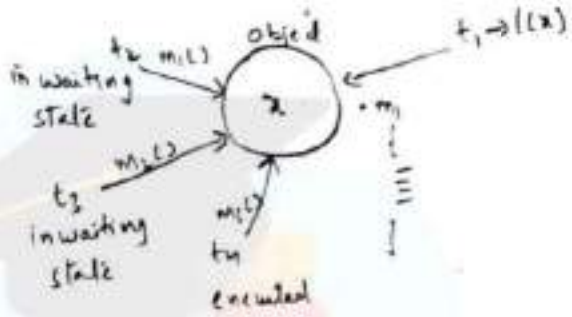
→ once method execution completes automatically thread releases the lock.

→ Acquiring & releasing locks internally takes cares by JVM not by programmer (or) Thread.



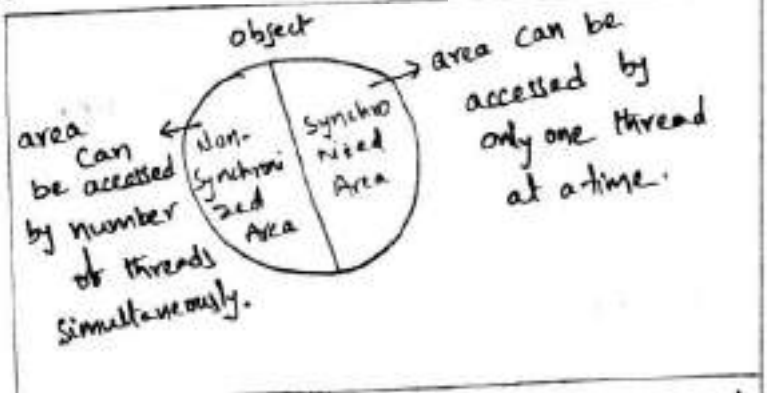
Example

```
class X
{
 Synchronized m1();
 Synchronized m2();
 m3();
}
```



→ While a thread executing synchronized method on the given object, the remaining threads are not allowed to execute any synchronized methods simultaneously on the same object, but remaining threads allowed to execute non-synchronized methods simultaneously.

Note:
Lock concept is implemented based on object but not based on method.



→ In java, any object can have two areas, those are synchronized area and non-synchronized area.

→ Synchronized area allows a thread to update (or) delete data.

→ Non-Synchronized area allows a thread to read the data (or) content.

* We should know, which method is declare as Synchronized (or) which method is declare as non-Synchronized method.

→ where ever, we are performing update operations (add/remove/replace) that methods can be declared as Synchronized (or) declared in Synchronized area.

→ where ever, we performing read operation, that methods can be declared as non-Synchronized/area.

Ex:-
```
class x
{
 Synchronized Area
 {
  //object state changing
 }
 Non-Synchronized Area
 {
  //object state won't be changed
 }
}
```

Example
```
class Banking
 {
   Non-Synchronized BalanceEnquiry()
                    {
                     = just read operation
                    }
     Synchronized withdraw()
                    {
                     = update
                    }
 }
```

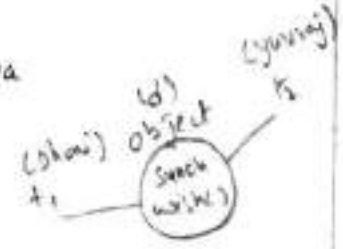
Example
```
class TicketReservation
{
 Non-Synchronized seatavaliability()
                    {
                     = just read operation
                    }
     Synchronized Booking()
                    {
                     = update
                    }
}
```

* Example :- SynchronizedDemo.java

```
class Display
{
public Synchronized void wish(String name)
{
for (int i=1; i<=5; i++)
{
System.out.println("Good Morning :");
try
{
Thread.sleep(2000);
}
catch(InterruptedException e){}
System.out.println(name);
}
}
}
class MyThread extends Thread
{
Display d;
String name;
MyThread(Display d, String name)
{
this.d = d;
This.name = name;
}
public void run()
{
d.wish(name);
}
}
class SynchronizedDemo
{
public static void main(String args[])
{
Display d = new Display();
MyThread t1 = new MyThread(d, "Dhoni");
```

```
MyThread t2 = new MyThread (d, "yuvraj");
 t1.start();
 t2.start();
}
}
```

output:-
```
javac SynchronizedDemo.java
java SynchronizedDemo
Good Morning : Dhoni
   "      "  : Dhoni
   "      "  : Dhoni
   "      "  : Dhoni
   "      "  : Dhoni
Good Morning : yuvraj
   "      "  : yuvraj
   "      "  : yuvraj
   "      "  : yuvraj
   "      "  : yuvraj
```



Example :-

TSynchronization Demo.java

```
class printTable
{
public Synchronized void printtable (int n)
{
 System.out.println ("Table of" +n);
 for (int i=1; i<=10; i++)
 {
  System.out.println (n * i);
  try
  {
   Thread.sleep (2000);
  }
  catch (InterruptedException e) { }
 }
 }
}
class MyThread1 extends Thread
{
 printTable pt;
 MyThread1 (printTable pt)
 {
  this.pt = pt;
 }
```

```
    public void run()
    {
     pt.printTable(2);
    }
   }
class MyThread2 extends Thread
{
 PrintTable pt;
 MyThread2(PrintTable pt)
 {
  this.pt = pt;
 }
 public void run()
 {
  pt.printTable(5);
 }
}
class TSynchronizationDemo
{
public static void main(String args[])
{
 PrintTable obj=new PrintTable();
 MyThread1 t1=new MyThread1(obj);
 MyThread2 t2=new MyThread2(obj);
 t1.start();
 t2.start();
}
}
```

output:-

```
javac TSynchronizationDemo.java
java TSynchronizationDemo
Table of 2
2
4
6
:
20
Table of 5
5
10
15
:
50
```

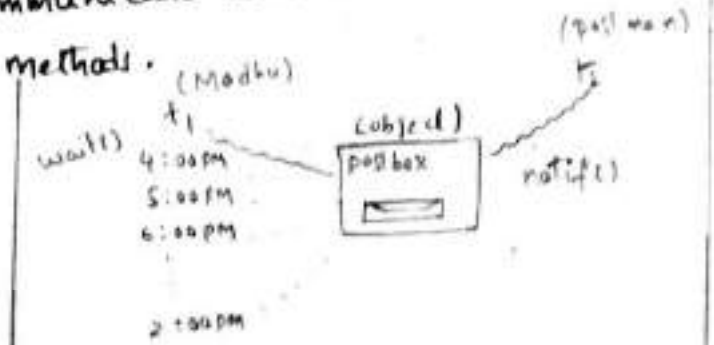
* Interthread communication :-

→ Interthread communication is a mechanism in which one thread is communicate with another thread by using wait(), notify() and notifyAll() methods.

(or)

→ Interthread communication is all about allowing Synchronized threads to communicate with each other.

→ Two threads can communicate with each other by using wait(), notify() and notifyAll() methods.

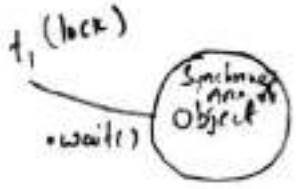


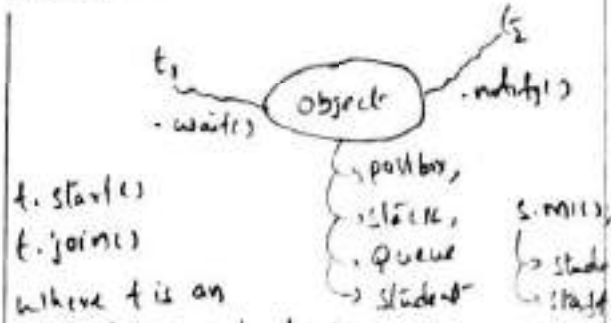
In this, t1 (Madhu) needs updation, so t1 is responsible to call wait(), and t2 (postman) needs to perform updation, so t2 is responsible to call notify().

→ The thread, which is expecting updation is responsible to call wait() method, then immediately entered into waiting state

→ The thread, which is responsible to perform updation, after performing updation, it is responsible to call notify(), then waiting Thread will get that notification and continue its execution with those updated information.

Note:- wait(), notify() and notifyAll() methods present in object class, but not in Thread class. Because a thread call wait(), notify() and notifyAll() on any object.





→ To call wait(), notify() (or) notifyAll() methods on any object, Thread should be owner of that object. i.e the thread should has lock of that object, i.e the thread should be inside Synchronized Area.

"Hence, we can call wait(), notify() and notifyAll() methods from synchronized area otherwise we will get runtime Exception i.e "IllegalMonitorStateException".

→ If a thread calls wait() method of any object, it immediately releases lock of that particular object and entered into waiting state.

→ If a thread calls notify() of any object, it releases lock of that object, but may not immediately.

Note:- Except wait(), notify() & notifyAll(), There is no other method where thread releases the lock.

Methods :-

* wait()
  → wait() waits for until get notification (or) specified Milliseconds.

Syntax:- public final void wait() throws InterruptedException

(or)

public final void wait(long ms) throws InterruptedException

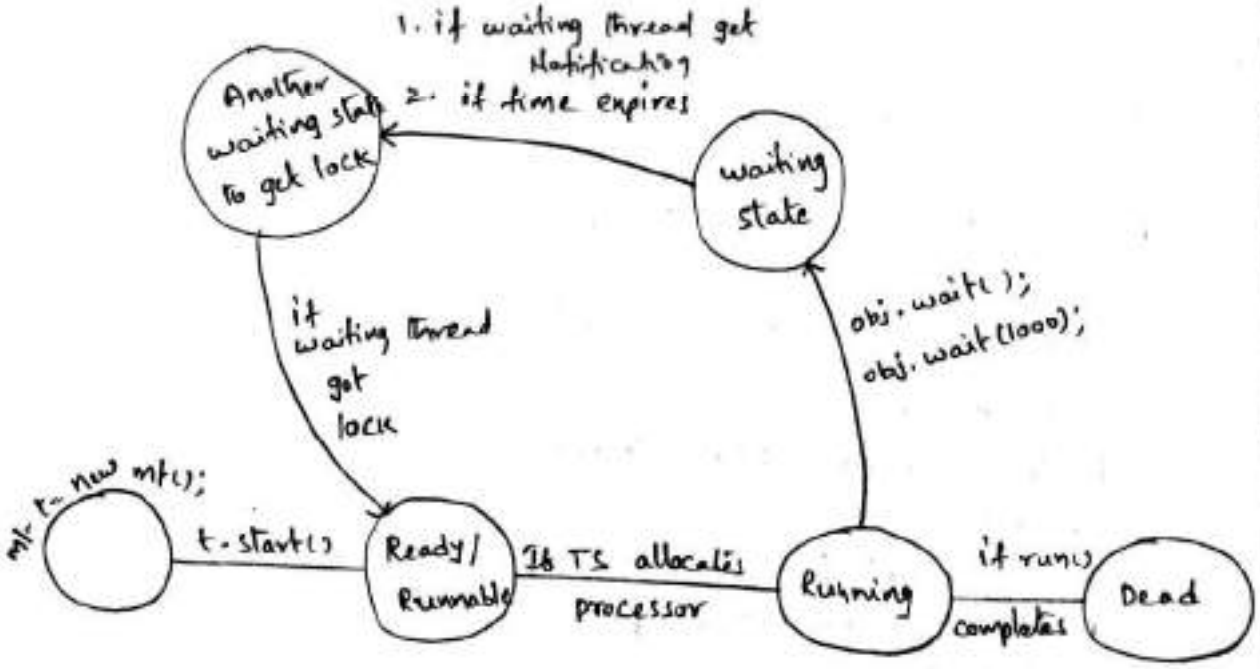
notify()
→ notify() gives notification to another thread

Syntax:- public final void notify()

(or)

public final void notifyAll()

→ understanding wait() & notify() methods in Thread Lifecycle



Example :-

```
class Customer
{
 int amount=10000;
 Synchronized void withdraw (int amount)
 {
  System.out.println("going to withdraw...");
  if (this.amount < amount)
  {
   System.out.println("Less Balance ; waiting for deposit...");
   try
   {
    wait();
   }
   catch (Exception c) { }
  }
  this.amount -= amount;
  System.out.println("withdraw completed");
 }
 Synchronized void deposit(int amount)
 {
  System.out.println("going to deposit...");
  this.amount += amount;
  System.out.println("Deposit completed...");
  notify();
 }
}
class ITCDemo
{
 public static void main(String args[])
 {
  Customer c = new Customer();
  new Thread() {
   public void run() { c.withdraw(15000); }
  }.start();
  new Thread(){
   public void run(){ c.deposit(10,000);}
  }.start();
 }
}
```

output:- javac ITCDemo.java
java ITCDemo
going to withdraw
Less Balance; waiting for deposit...
going to deposit
deposit completed
withdraw completed.

* Introduction to JDBC :-

→ In today's scenario, many enterprise level applications need to intract with databases for storing information.

→ For this purpose, we used an API (Application programming Interface) i.e ODBC (open Database Connectivity).

→ The ODBC API was the database API to connect and execute query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e platform dependent and unsecured).

→ That is why java has defined its own API, called JDBC (Java Database connectivity). That uses JDBC drivers (written in Java language).

→ The JDBC drivers are more compatible with Java Applns to provide database communication.

→ JDBC is a java API to connect and execute query with the database. JDBC API uses jdbc drivers to connect with the database.

→ JDBC supports a wide level of portability and JDBC is simple and easy to use.

→ In JDBC API, a programmer needs a specific driver to connect to specific database.

RDBMS	Driver
Oracle	oracle.jdbc.driver.OracleDriver
MySQL	com.mysql.jdbc.Driver
SyBase	com.sybase.jdbc.SybDriver
SQLServer	com.microsoft.jdbc.SqlServer
DB2	com.ibm.db2.jdbc.net.DB2Driver

* List of some popular Drivers.*

* JDBC Architecture :-

The main function of the JDBC is to provide a standard abstraction for java applications to communication with databases.

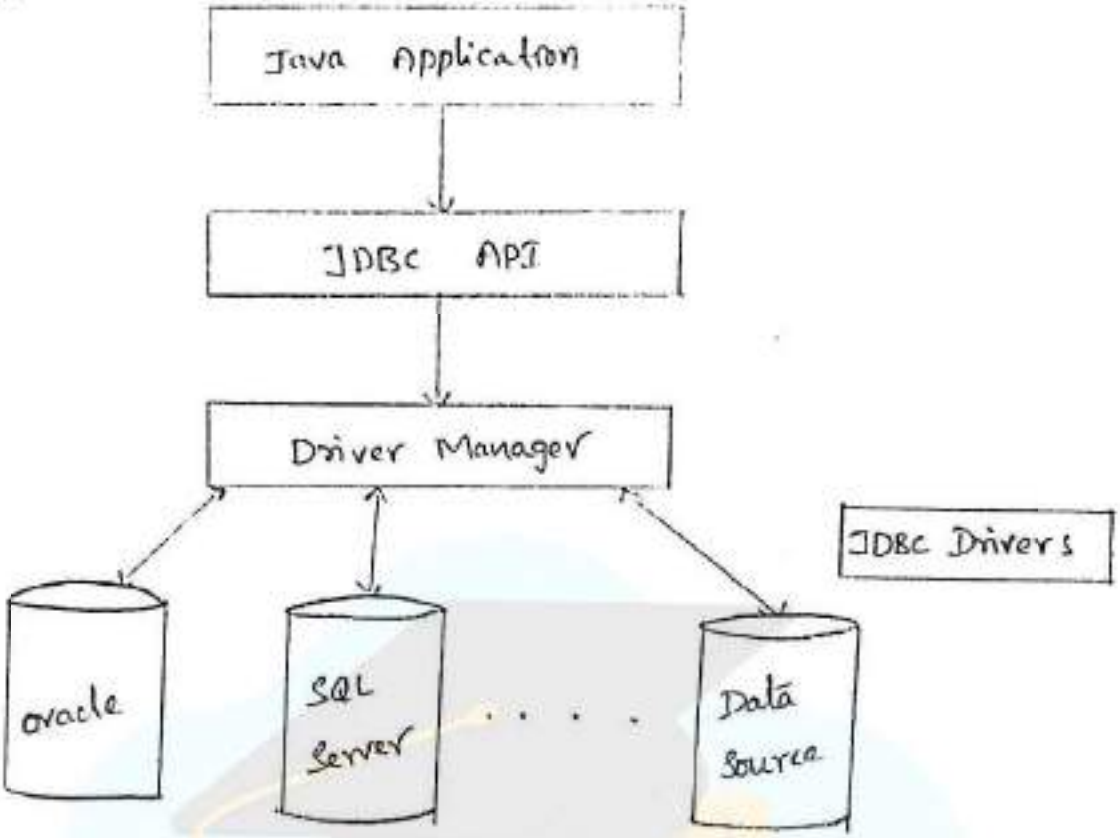


Fig:- The JDBC Architecture

As shown in figure, The Java application that wants to communicate with a database has to be programmed using JDBC API.

The JDBC Driver is required to process the SQL requests and generate the results.

The JDBC driver has to be plays an important role in the JDBC architecture. The Driver Manager uses some specific drivers to effectively connect with specific databases.

→ JDBC Driver is a software component that enables java application to intract with the database.

There are 4 types of JDBC drivers. Those are

→ Type-1 Driver (JDBC-ODBC bridge driver)
→ Type-2 Driver (partial JDBC driver)
→ Type-3 Driver (pure java driver for middleware)
→ Type-4 Driver (pure java driver with direct database connection)

* Type-1 Driver (JDBC-ODBC bridge driver) :-

The Type-1 driver acts as a bridge between JDBC and other database connectivity mechanisms such as ODBC.

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC method calls.

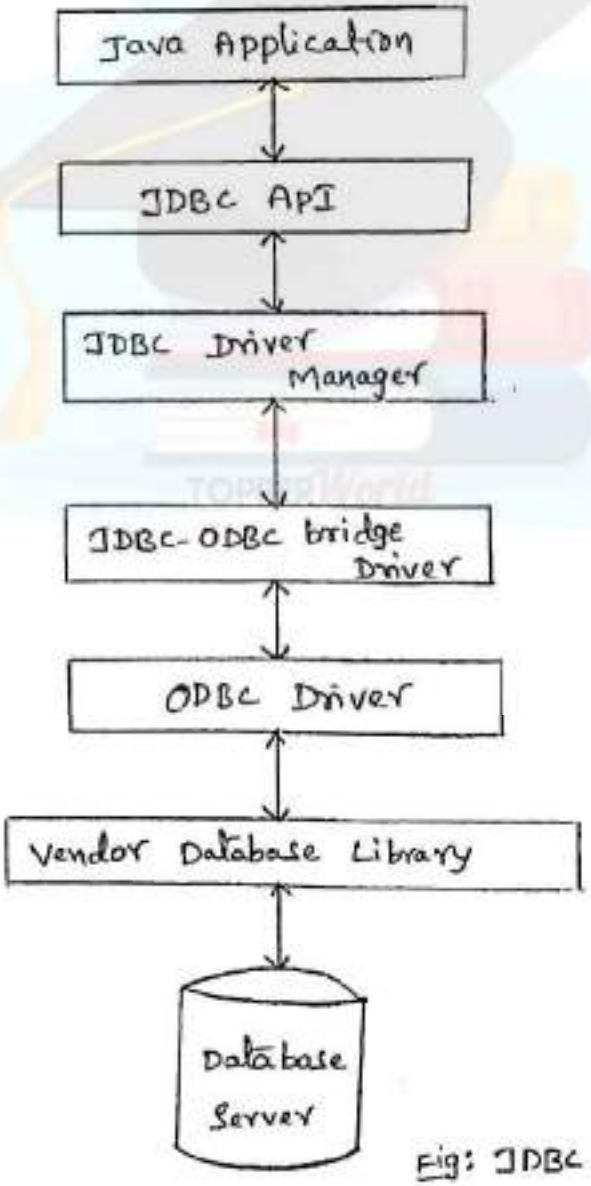


Fig: JDBC-ODBC Bridge Driver

Advantages:

* Easy to use.
* Can be easily connected to any database.

Disadvantages:

* performance degraded because large number of trans-lations (i.e JDBC calls to ODBC calls).
* The ODBC driver needs to be installed on the client machine.

* Type-2 Driver (partial JDBC driver) :-

The Type-2 driver uses the client-side libraries of the database. so This driver is also called as Native-API driver.

This driver converts JDBC method calls into native calls of the database API. It is not written entirely in Java, so it is called as partial JDBC driver.

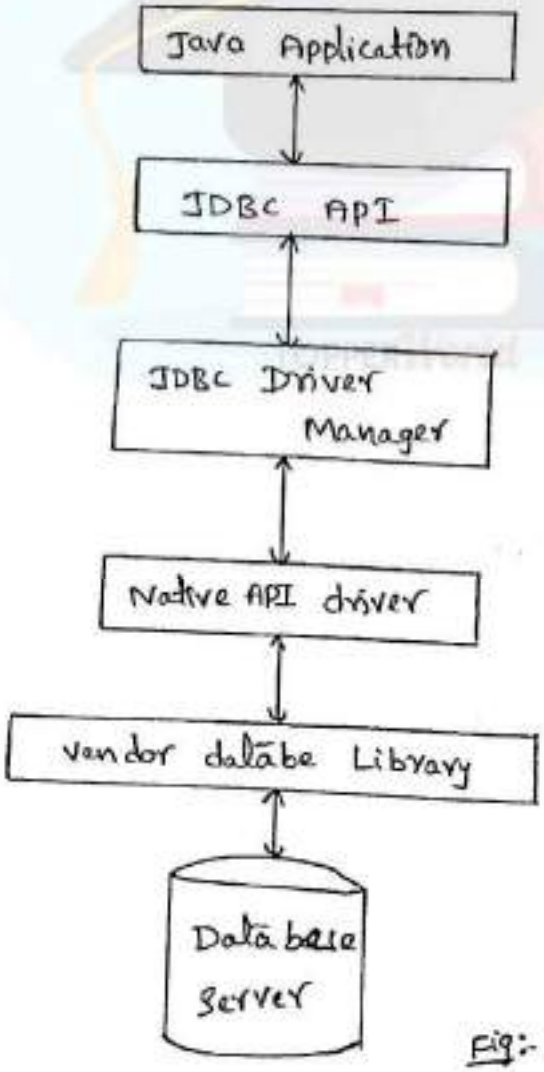


Fig:- Native API driver

Advantages:

* performance upgraded than JDBC-ODBC bridge driver.
* suitable to use with server-side applications.

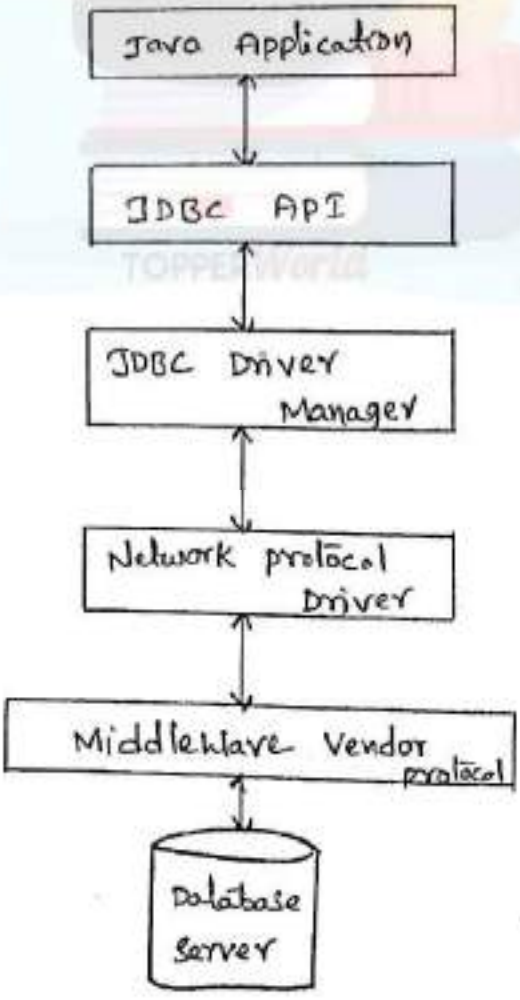
Disadvantages:

* This native driver needs to installed on the each client machine
* The vendor client library needs to be installed on client machine.
* It may increase the cost of the application if the application needs to run on different platforms.

* Type-3 Driver (pure Java driver for middleware) :-

The type-3 driver is completely implemented in Java, hence it is a pure Java JDBC driver.

The type-3 driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. so it is called as Network protocol driver.



Advantages:

* No client side library is required on client side.
* pure Java drivers and auto downloadable.

Disadvantages:

* Network support is required on client machine.
* This driver is costly compared to other drivers.

* Type-4 Driver (pure java driver with direct database connection):-

The type-4 driver is a pure java driver, which converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver.

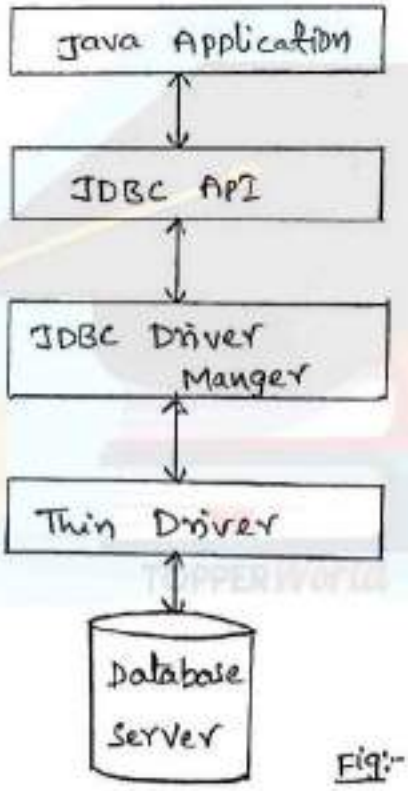


Fig:- Thin Driver.

Advantages:

* This driver is pure java driver and auto downloadable.
* Better performance than all other drivers
* No software is required at client side or server side.

Disadvantages:

* Drivers depends on the Database.

* Database programming using JDBC :-

JDBC APIs are used by a Java application to communi-cate with a database.

In otherwords, we use JDBC connectivity code in Java application to communicate with a database.

There are 5 steps to connect any Java application with the database in Java using JDBC. They are as follows:

Step 1 : Register the driver class

Step 2 : Creating connection

Step 3 : Creating statement

Step 4 : Executing SQL statements

Step 5 : closing connection.

* Step 1 :- (Register the driver class)

In this step, We register the driver class with driver Manager by using forName() method of Class class.

Syntax: Class.forName(Driver class Name)

Example: Class.forName("oracle.jdbc.driver.OracleDriver");

* Step 2 :- (Creating connation)

In this step, We can create a connection with database Server by using getConnection() method of DriverManager class.

Syntax: getConnection(String url, String name, String pwd)

Example:

```
Connection con = DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:1521:xe",
                                "System", "admin");
```

* Step 3 :- (Creating statement)

After the connection made, We need to create the statement object to execute the SQL statements.

The createStatement() method of Connection interface is used to create statement. This statement object is responsible to execute SQL statements with the database.

Syntax: create.Statement()

Example:

```
Statement stmt = con.Statement();
```

* Step 4:- (Executing SQL statements)

After the statement object is created, it can be used to execute the SQL statements by using executeUpdate() (or) executeQuery() method of statement interface.

The executeQuery() method is only used to execute SELECT statements.

The executeUpdate() method is used to execute all SQL statements except SELECT statements.

Syntax: executeQuery(String query)
executeUpdate(String query)

Example:

```
// using executeQuery()
String query = "Select * from emp";
Resultset rs = stmt.executeQuery(query);
// using executeUpdate()
String query = "insert into emp values(504,'Madhu',29);
stmt.executeUpdate(query);
```

* Step 5:- (closing the connection)

After executing all the SQL statements and obtaing the results, we need to close the connection and release the session.

The close() method of Connection interface is used to close the connection.

Syntax:- close()
Example:- con.close();

* Example :- (connectivity with oracle database)

For connecting java application with the oracle database, we need to know following information to perform database connectivity with oracle.

In this example we are using oracle 10g as the database, so we need to know following information for the oracle database.

* Driver class : The driver class for oracle database is "oracle.jdbc.driver.OracleDriver".
* Connection URL : The connection URL for the oracle 10G database is "jdbc:oracle:thin:@localhost:1521:xe".

Where jdbc is the API, oracle is the database, thin is the driver, Localhost is the servername on which oracle is running, 1521 is the port number and xe is the oracle service name.

* username : The default username for the oracle database is "system".
* password : password is given by the user at the time of installing the oracle database.

→ To connect java application with the oracle database ojdbc14.jar file is required to be loaded.

→ There are two ways to load the ojdbc14.jar file, we need to follow any one of two ways.

1. paste the ojdbc14.jar file in "java/jre/lib/ext" folder
2. set classpath

Firstly, search the ojdbc14.jar file then go to "java/jre/lib/ext" folder and paste the jar file here.

(or)

Set class path : To set classpath, goto environment variable then click on new tab. In variable name write classpath and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as

"c:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;".

* Example:

Let's first create a table and insert two or more records in oracle database.

```
SQL> create table emp(id number(10), name varchar2(40),
                                    age number(3));

SQL> insert into emp values(501,'Madhu',30);
SQL> insert into emp values(502,'Hari',32);
SQL> insert into emp values(503,'Satti',33);
```

* Program: connect java application with oracle database for selecting or retriving data.

SelectData.java

```
import java.sql.*;
import java.util.*;
class SelectData
{
public static void main(String args[])
{
 try
 {
 // Step 1: load the driver class
 Class.forName("oracle.jdbc.driver.OracleDriver");
  // Step 2: create the connection object
 Connection con = DriverManager.getConnection(
    "jdbc:oracle:thin:@localhost:1521:xe","System","admin");
 // Step 3: create the statement object
  Statement stmt = con.createStatement();
 // Step 4: execute query
  ResultSet rs = stmt.executeQuery("select * from emp");
  while(rs.next())
  {
   System.out.println(rs.getInt(1)+"    "+rs.getString(2)+"    "
                      +rs.getString(3));
  }
 // Step 5: close the connection object
  con.close();
 }catch(Exception e){System.out.println(e);}
}
}
```

output:

```
D:\> javac SelectData.java
D:\> java SelectData
501     Madhu     30
502     Ravi      32
503     Salli     33
```

* program: connect java application with oracle database -for inserting data.

InsertData.java

```
import java.sql.*;
import java.util.*;
class InsertData
{
public static void main (String args[])
{
 try
 {
 Class.forName ("oracle.jdbc.driver.OracleDriver");
 Connection con = DriverManager.getConnection ("jdbc : oracle : thin
                  :@localhost : 1521 : xe ", "System", "admin");

 Statement stmt = con.createStatement();
 stmt.executeUpdate ("insert into emp values (504, 'Ganesh', 28)");
 System.out.println ("Inserted ... ");
 con.close();
 }
 catch (Exception e)
 {
  System.out.println (e);
 }
}
}
```

output:

```
D:\> javac InsertData.java
D:\> java InsertData
Inserted ...
```

* program: java application with oracle database for update data.

Updatedata.java

```
import java.sql.*;
import java.util.*;
class Updatedata
{
public static void main (String args[])
{
 try
 {
  Class.forName ("oracle.jdbc.driver.OracleDriver");
  Connection con = DriverManager.getConnection ("jdbc:oracle:
                  thin:@localhost:1521:xe", "system", "admin");
  Statement stmt = con.createStatement();
  stmt.executeUpdate ("update emp set age=38 where id=503");
  System.out.println ("updated....");
  con.close();
 }
 catch (Exception e)
 {
   System.out.println("Exception is :" +e);
 }
 }
 }
```

output:

```
D:/> javac Updatedata.java
D:/> java Updatedata
 Updated....
```

* DriverManager class :-

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropri-ate driver.