Java

Introduction

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

[Platform: Any hardware or software environment in which a program runs is known as a platform]

- Java is based on the concept "Write once, run anywhere"

History

- Java was Developed by Sun Microsystems in the year of 1995. James Gosling is known as the father of Java.
- Before Java, its names was Oak. Since Oak was Already a registered company, so James Gosling and his team changed the names from Oak to Java.

→ Development and Early Naming (1991-1995)

1) Java was Developed by James Gosling, Patrick Naughton at Sun Microsystem in 1991-

It was initially called 'Oak' but was renamed in "Java" 1995.

3) Oak is a symbol of strength and chosen as a national tree of many countries like U.S.A, France etc. In 1995, Oak was renamed as "Java" because it was already a trademark by Oak Technologies.

→ Significance of Java name

1) Java is not an acronym, it's simply a name inspired by the coffee produced on Java island.

→ Motivation and Design Goals

1) Java was created to be a platform-independent, architecture-neutral language

2) The goal was to develop software for various consumer electronic devices like TV, and microwaves.

3) The problem with other languages such as C, C++, was that their compilers were expensive and time-consuming to create., an easier and cost-efficient sol^{n} need, Gosling work on portable, platform-independent language develop and it suitable for different environments.

→ Evolution and Popularity

- The rise of the WWW significantly boosted java popularity
- Without the development of the web, Java might not have become as popular.
- Over the year java continued to grow and evolve

→ Java Versions

- Java JK Alpha and Beta (1995)
- Java JDK 1.0 (1996)
- Java JDK 1.1 (1997)
- . . .

Features of Java

- The feature of java are also known as Java Buzzwords.

---

1) Simple:- Java is very easy to learn and its syntax is simple, clean and easy to understand.

→ Java has removed many complicated and rarely-used feature ex: operator overloading, explicit pointers

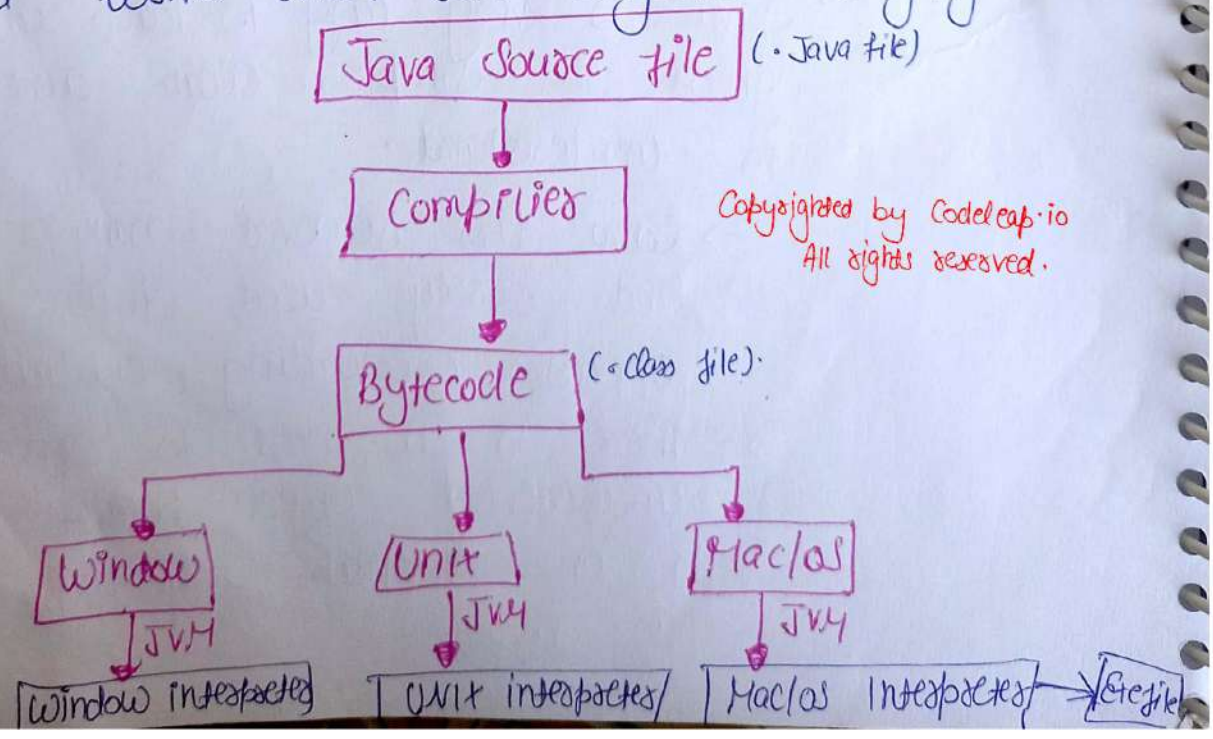
→ There is no need to remove unreferenced object because there is an Automatic garbage collection in Java.

2) Object - Oriented: Java is fully object oriented language which follow all the rules of OOPS like inheritance, abstraction, polymorphism etc and every thing can be treated in the form of objects.

- Object - oriented means we organize our software as a combination of different type of object that incorporate both data and behaviour.

---

3) Platform - Independent: Java is platform independent because it is different from other languages like C, C++ etc, which are compiled into platform spectic machines [~~which~~ i.e JVM has been installed] [JVM provide runtime environment in which java bytecode can be executed] and Java is a "write once, run anywhere language".

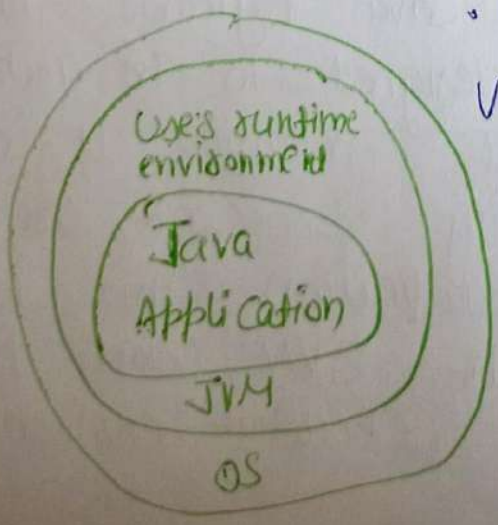


- Java code can be run on multiple platform
  eg: Window, Linux, Mac/os etc.
- Java code is compiled by the compiles and converted into bytecode.
- Bytecode is a platform - independent code due to its can be executed on any System where JVM was installed.

→ JVM reads the bytecode file & interprete it into machine code.
→ JVM includes an interpreter for specific operating System that translate the bytecode into executable code in Operating System

It can run on any specific operating System

4) Secured: Java is best known for its Security because
- No explicit pointer
- Bytecode Verifies
- Program run inside java Virtual machine sandbox



4) Robust: (Strong)

- It uses strong memory management
- Java provides automatic garbage collection which run on JVM
- These are exception handling and the type checking mechanism in Java. All these point make Java robust

5) Java is Distributed: Java is distributed because it facilitates users to create distributed application in Java.
- RMI & EJB are used for creating distributed application.

6) High performance: ~~Java is faster than other~~ It performance is is enhanced by JIT Compilers, efficient memory management and multithreading capabilities.

7) Case Sensitive: Dynamic.

8) Architecture Neutral: Java Bytecode is designed to be architecture neutral, meaning it can run on any processor architecture without modification. This is achieved through JVM which provide the necessary execution environment [and allowing concurrent execution of two or more threads for Max CPU utilization.

→ Portable

→ Multi-threaded

It not occupy memory for each thread.
It shares a common memory area.

Java Application

- **Standalone application**
  - ↳ also known as desktop or window-based application and need to install on every machine.
  - ↳ eg: Media player, antivirus.
- **Web application**
  - ↳ that run on server side and creates a dynamic page.
  - ↳ eg: Servlet, Spring etc. technologies are used to create web application
- **Enterprise application**
  - ↳ An application that is used in distributed system such as banking application.
  - ↳ EJB is used.

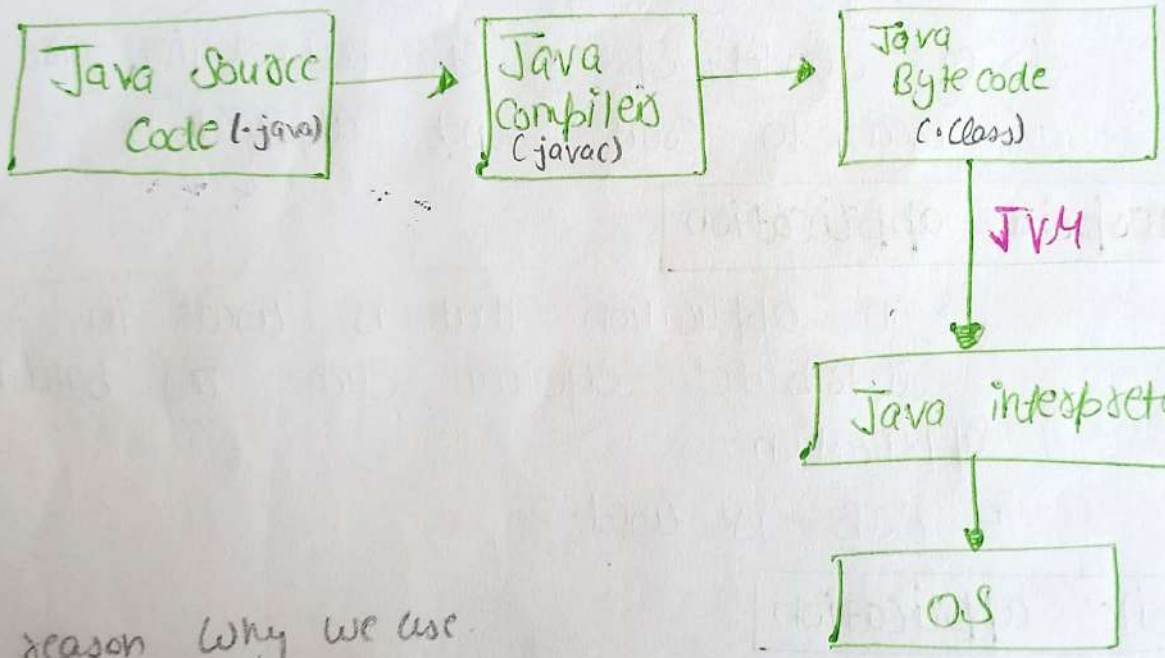
- **Mobile application**
  - ↳ An application that is created for mobile device
  - ↳ Android and Java ME used

Java platform / Editions

1) Java SE (Standard edition)
2) Java EE (Enterprise edition)
3) Java ME (Micro edition)
4) JavaFX.

JVM → called a virtual Machine because it doesn't physically exist, used to provide runtime environment in which java Bytecode code can be executed

- Stand for "Java Virtual Machine"
- Java Virtual Machine is an abstract machine responsible for Compiling and executing java byte-code, potentially using JIT compilation to improve performa-
- It is a part of Java Runtime Environment, which calls the main function of a program.
- It converts the bytecode into machine language



reason why we use

- JVM facilitates a platform-independent way of executing Java source code. Its basis on WORA (Write Once Run Anywhere).
- JVM comes with JIT (just-in-Time) Compiler that converts java source code into machine code.
- It has numerous libraries, tools and frameworks.

→ JVM performs following operations

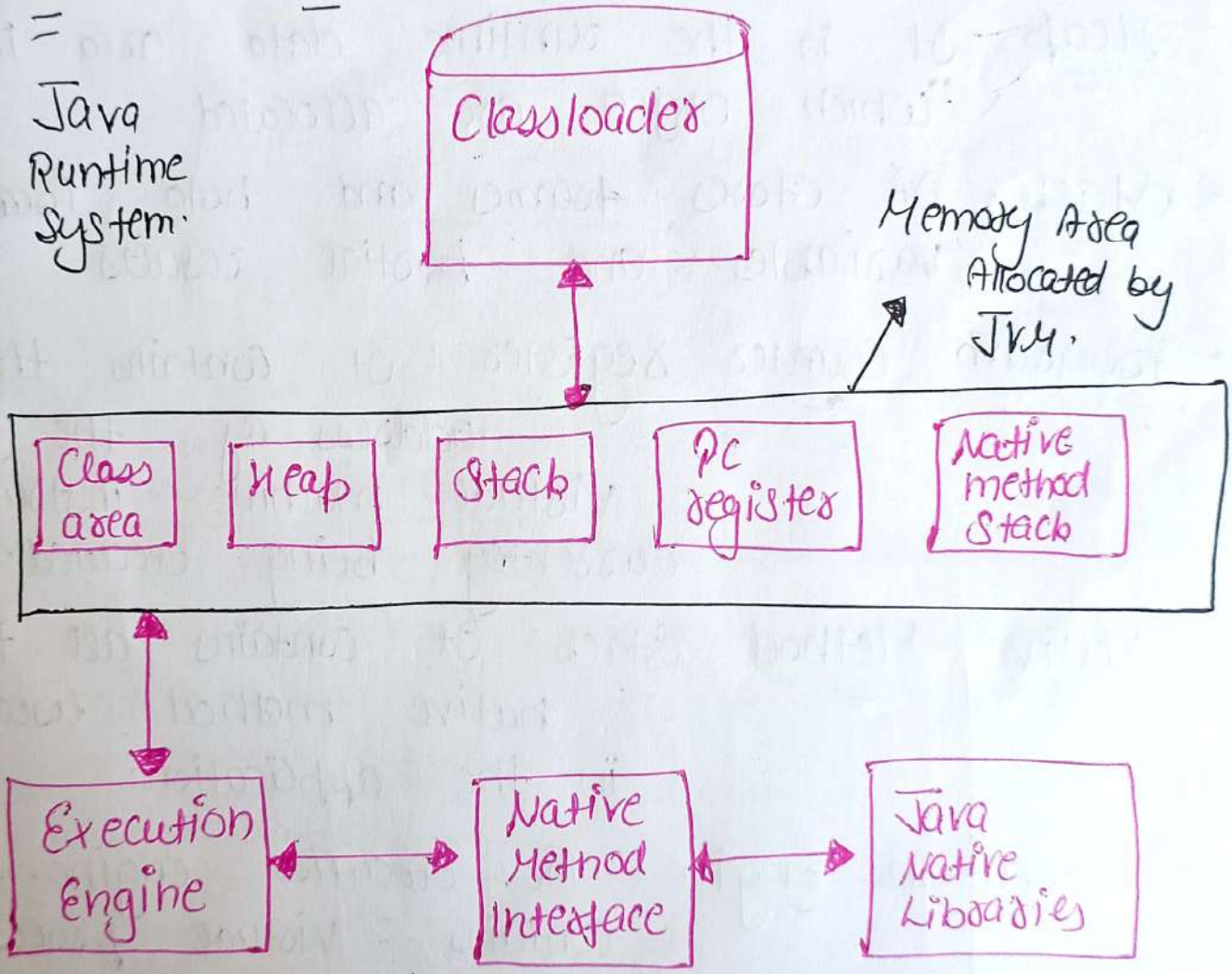
- loads code
- verifies code
- Executes code
- Provide runtime environment

[Note : read bytecode and execute it as native machine code on host machine JVM acts a bridge btw the bytecode and underlying hardware.

→ JVM Architecture

=

Java Runtime System.



- Classloader : It is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the Classloader.

→ It perform three major function.
Loading, Linking, Initialization.

JVM

- Bootstrap Classloader
- Extension Classloader
- System

} → built - in Classloader.

- **Class area**: It stores per-Class structures such as the runtime constant pool, field and method data.
- **Heap**: It is the runtime data area in which object are allocated.
- **Stack**: It stores frames and hold local variables and partial results.
- **Program Counter register**: It contains the address of the java virtual machine instruction currently being executed.
- **Native Method Stack**: It contains all the native method used in the application.
- **Execution Engine**: The Execution engine contains: Virtual processor
  : Interpreter
  : JIT compiler to improve the perform.
- **Java Method interface**: It is a programming framework, allow java code, which is running in JVM to call by libraries.

- Native Libraries: Collection of Native libaaries (C, C++), which are needed by the execution engine.

Components of JVM (Tasks)

- Class loader
  - loading (.class)
  - linking (verification)
  - Initialization

- Bytecode Verifier
  - verify the bytecode, Check vaild or not
- JIT Compiler [Execution ~~engine~~ code]
- Garbge Collector
  - Automatically manage memory by declaiming memory occupied by object that are no longer in use.
- Security manager (Run time enviroment)

JRE

- Stand for "Java runtime Environment"
- It is used to provide the runtime environments and implementation of JVM
- It is physically exist.

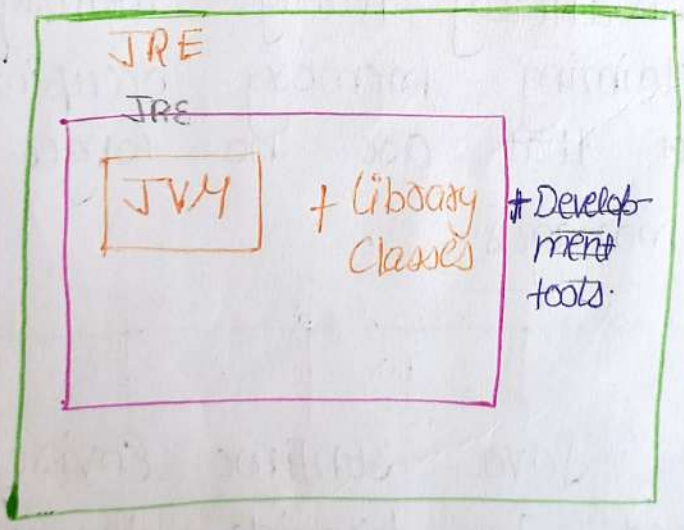
→ JRE = JVM + Class libaaries → that JVM uses at runtime.

→ Components of JRE

- Deployment technologies
  - → such as java plug-in
  - → Java web start
- Uses interface toolkits
  - → Drag & Drop
  - → Swing
- Integration libraries
  - → Interface definition lang.
  - → Java Database Connectivity.

[Advantages
- Platform indepent.
- Automatic Memory Management
- Rich Standard Libraries
- security

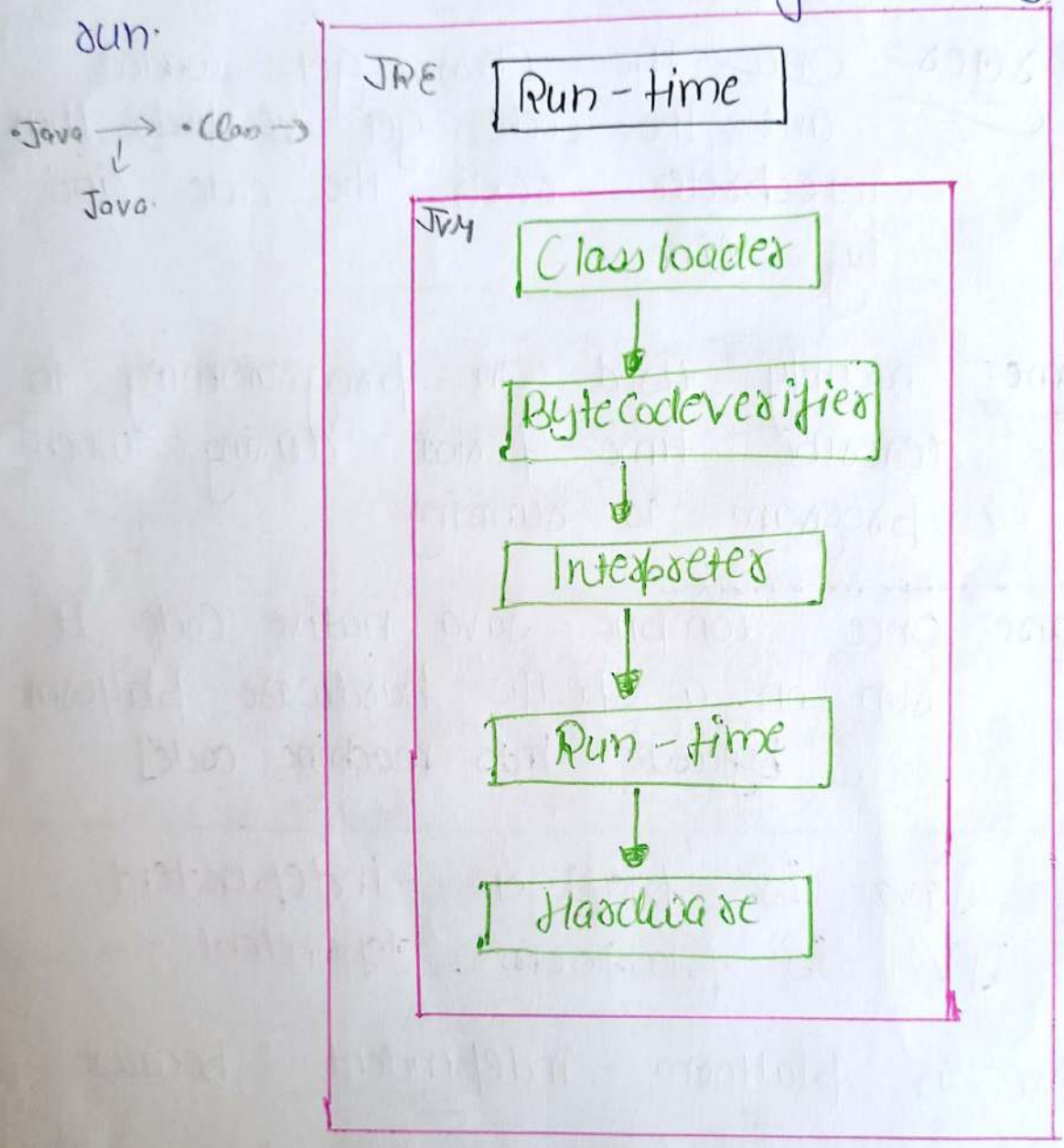
→ How JRE work with JVM



JRE acts as a layer on the top as.

- JRE has an instance of JVM with it, library classes.
- To understand the working of JRE let see an ex. [Program save with .java extn

- firstly Compiles program into bytecode, compiler generates a .class file - work of JRE start - To run any java program, need JRE - The flow of the bytecode to run:



→ Classloader : required to run a program
= =
- uses 3 Classloaders, when JVM is started
  - Bootstrap
  - Extensions
  - System.

→ Byte-Code Verifier: It can be considered as a gatekeeper. It verifies the bytecode so the code doesn't disturb interpreter.

→ Interpreter: Once the Class get loaded and the code get verified, then interpreter reads the code line by line.

→ Run-time: mainly used in programming to describe time period during when a program is running

→ Hardware: Once compile Java native code, it run on a specific hardware platform. [bytecode into machine code]

Why Java is platform-independent but JVM is platform dependent.

→ Java is platform-independent: because it run on any platform that has a JVM installed. mainly based on "Write once, run anywhere".

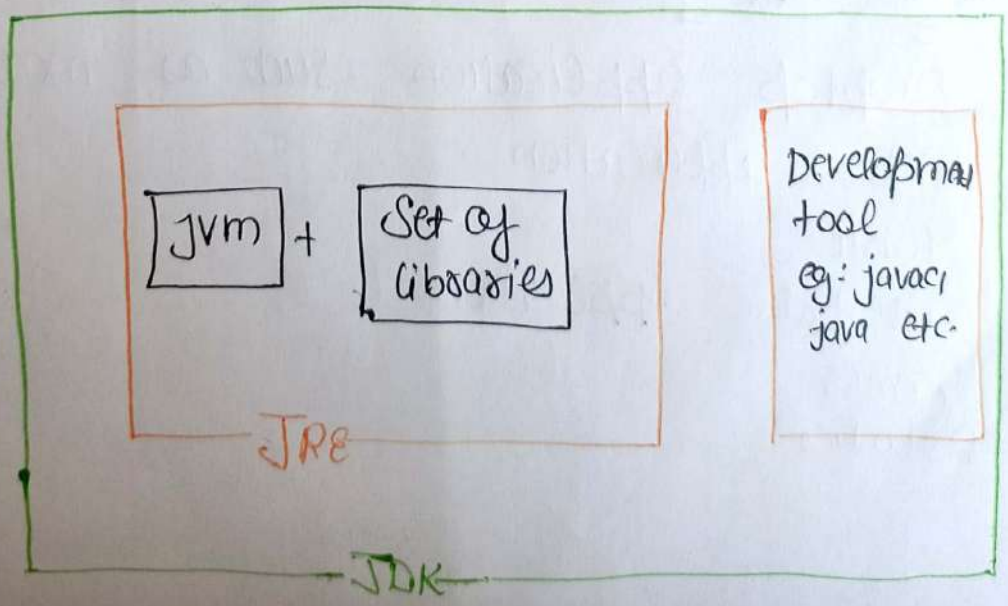
→ JVM is platform dependent: because it must be implemented separately for each platform to Translate Java bytecode into native machine code for specific platform.

JDK

- Stand for "Java Development kit".
- It is a software development environment which is used to develop java application and applets.
- It is physically exists

JDK = JRE + Development Tools

- JDK is an implementation of any one of the Java platforms released by Oracle Corporation:
  - → Standard Edition Java Platform
  - → Enterprise Edition Java platform
  - → Micro Edition Java platform.
- JDK contains JVM and few other resources such as interpreter (java), a compilies (javac), archiver (jar) etc to complete the development of a java application



features of JDK

- It includes all the functionalities of JRE and JVM
- JDK help developers to handle the exception using multiple extensions in a single catch block.
- It has various other development tools like the debuggers, compiles etc.

⇒ The JDK comes with the a collection of tools that are used to develop and run Java program

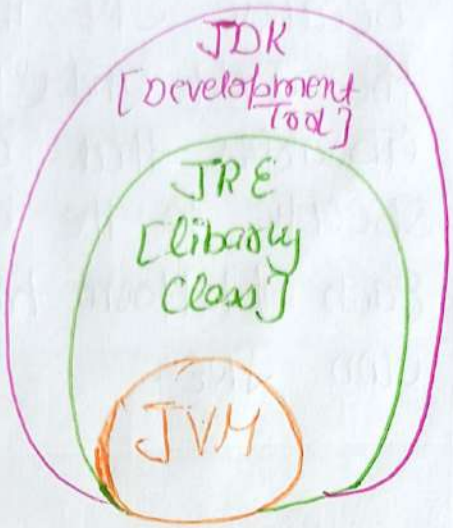
- appletviewer
- javac (Java Compiler)
- java (Java interpreter)
- javap (Java Dissembler)
- jdb (Java debugger)
- javadoc (HTML document)

JComponent.

Java Applications

- Desktop application such as media player.
- Web application.
- Mobile
- Enterprise Application
- Games
- Robotics.

Difference b/w JDK / JRE / JVM



JDK	JRE	JVM
• JDK Stand for Java Development kit	• JRE Stand for Java runtime environment	• JVM Stand for java Virtual Machine
• It include JRE, development tool like compilers and debuggers	• It inclue JVM, core libraries to run Java application	• Include a Specification, implemen. and runtime instance (Class loader, bytecode verifier, runtime interpreter)
• purpose: To develop and compile Java Application	• To provide a runtime enviro-ment to run java application	• To execute java bytecode and provide an environment for java program execution
• JRE + development Tools ← contains	• JVM + libraries	• Core component of both JRE & JDK. (JIT compiler, Garbage collector).
• JDK is the superset of JRE	• JRE is the subset of JDK	• JVM is a subset of JRE
• JDK installation size is larger	• Medium	• Small (only the execution engine.

JDK	JRE	JVM
• Platform-dependent because JDK include tool and compilers that are specific to the os and hardware. Each platform has its own JDK (Window/Mac)	• Platform-dependent because JRE includes the JVM and standard libraries that are specific to the OS. Each platform has its own JRE.	• Platform-independent because JVM execute Java bytecode, which is platform-independent, JVM itself is implemented for specific OS & hardware, so each platform has its own JVM
• Tool provided Compiler (javac), debugger (jdb) etc	• Libraries, Java plug-in for web browsers	• JIT, compiler, Garbage Collector
• JDK comes with the installer	• JRE only contains environment to execute source code	• JVM bundled in both software JDK & JRE
• Used by developers to write, compile and debug java application	• Used by end-user to run java application	• Used to execute Java application by converting bytecode into machine code.

Java Garbage Collector

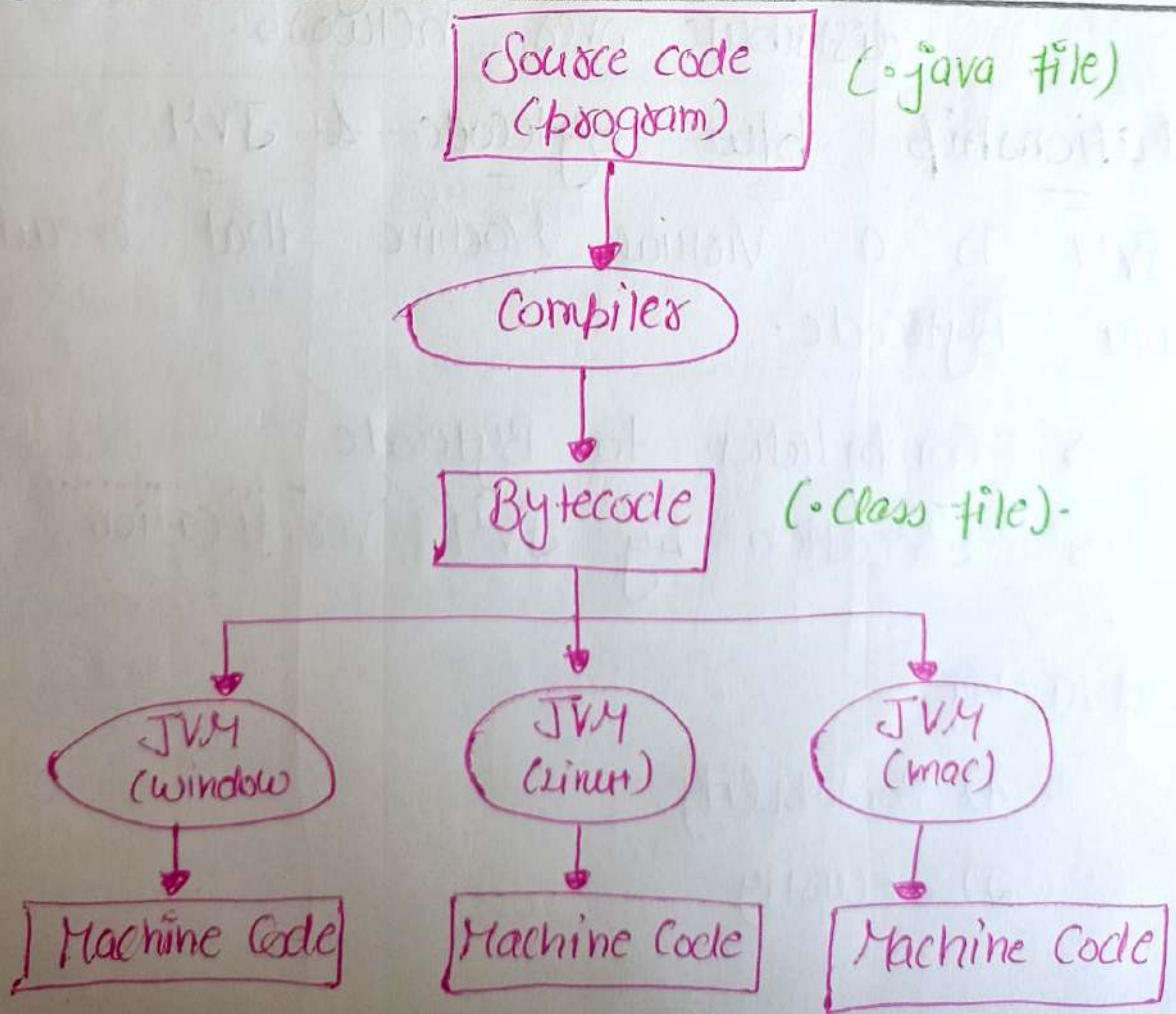
= that automatically freeup memory by re-collecting item that are no longer referenced in the code

• This make life easier for programmer because they no longer have to manage memory manually.

- The allocation and deallocation of memory for object is done by garbage collector is an automated way by Java.

Java Bytecode

- Java Bytecode is the instruction set for the java virtual Machine.
- Bytecode can be defined as an intermediate code generated by the compiler after the compilation of source code. This intermediate code make java is a platform-independent language.
- Bytecode is not directly executed by hardware, but is designed to be executed by the JVM.



→ Characteristics

1) **Platform - independence**

↳ Bytecode is platform - independent, meaning the same bytecode can run on any platform that has compatible JVM.

2) **High - level Abstraction**

↳ Bytecode operate at higher level of abstraction compared to machine code.

3) **Compact and efficient**

↳ Bytecode is more compact than source code, making it easier to distribute over network.

Relationship blw Bytecode & JVM

- JVM is a Virtual Machine that executes java Bytecode.
  - → Compilation to Bytecode
  - → Execution by JVM / verification

→ Advantages

1) Portability
2) Security

Difference b/w C, C++, Java

[Java, C++] [object] [has access specifiers]

programs divide into small part called function

C
- Developed by Dennis ritchies in 1970.

[No] [less secure] [based on unsecure world] [C, fortanan]

- Structure oriented Based

C++
- Developed by Bjarne Stroustrup in 1980
- Based upon the object oriented principle

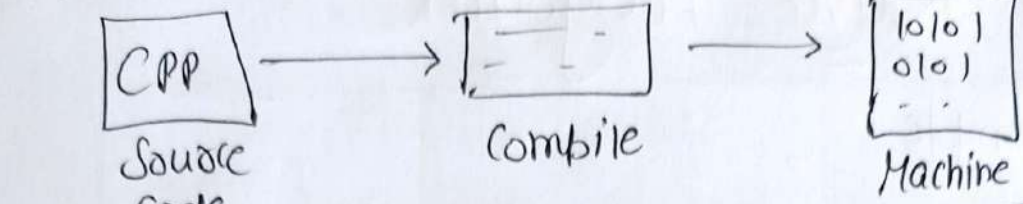
Inherit Syntax

Inherits OOP concept

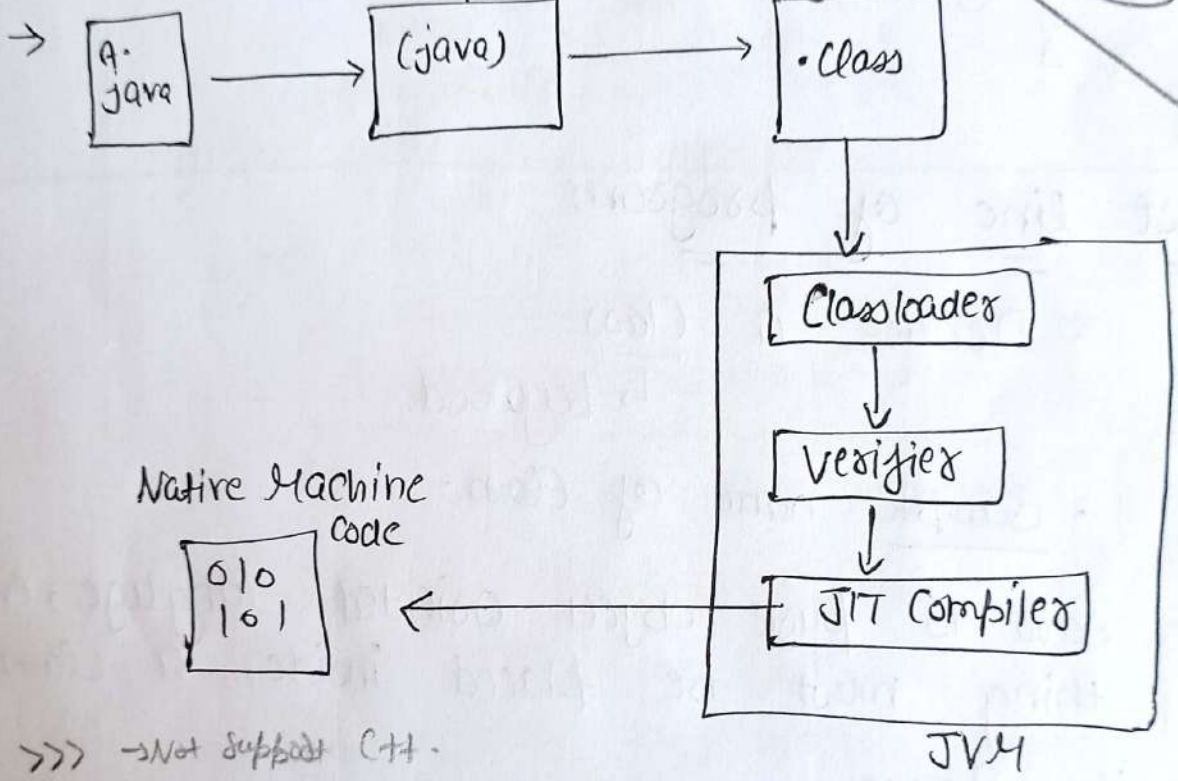
Java
- Develop by James Gosling in 1995
- Purely object oriented

C	C#	Java
• Procedural language	• object-oriented programming language	• Pure object-oriented Programming language
• based on BCPL (origin)	• based on C language	• based on C & C#.
• It use the top-down approach.	• It uses the bottom-up approach	• It also uses the bottom up approach.
• It is a static programming language	• It is also a static programming language.	• It is a dynamic programming language.
• The code is executed directly	• Same	• The code is executed by JVM.

C	C++	Java
• It is platform dependent	• platform dependent	• platform independent because of byte-code.
• It use a compiler only to translate the code into machine language	• Same as C	• It use both compiles and interpreter and it is also known as interpreted language
• The source file has .c extension	• The source file has .cpp extension	• The source file has a .java extension
• It support pointer	• It also support pointer	• Not support due to it high security.
• It use the calloc(), malloc(), free(), realloc() method to mange the Memory.	• It use new and delete operator to manage the memory.	• It uses a gaobage collector to manage the memory.
• It is widely use to develop driver.	• It is widely used for system programming	• It is used to develop web and mobile applicatio
• Support	• It support goto Statement	• Not Support
• Not support the overloading concept	• Method and operator overloading support	• only Method overloading support



Program execution in C++



>>> →Not support C++.

Program execution in java

C

```
# include <stdio.h>
int main (void)
  {
  printf ("---");
 }
```

C++

```
#include <iostream>
using namespace std;
int main()
  {
    cout<< "Hello";
    return 0;
  }
```

First Java Program

```
public class Sample
{
    public static void main (String args[])
    { ← indicates the beginning of method main().
        System.out.println("Hello java");
    } ← indicates the end of method
}
```

→ first line of program

- Declare a Class
  - ↳ keyword
- Sample name of Class.

∴ Java is pure object-oriented language, every thing must be placed inside a Class.

→ Opening Brace

- Every Class defination begin with an opening Brace "{" and ends with a matching Closing Brace "}" in the last line of program.

→ Third line of program

`public static void main (String args[]).`

- This line declare 'main' method.

→ public: keyword is an access modifier which represent visibility

→ static: keyword

→ void: return type of method, it means it doesn't return any value.

→ main (): represent Java application begins

→ String args[]: declares a parameter names args.

- fifth line

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

Line output

→ System : predefined Class

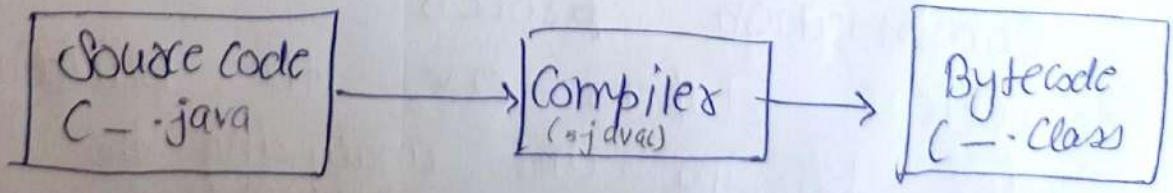
→ out : is the object

→ println : is the method, display the string on screen.

Compiling and Running the program

↳ same step in JRE work run-time.

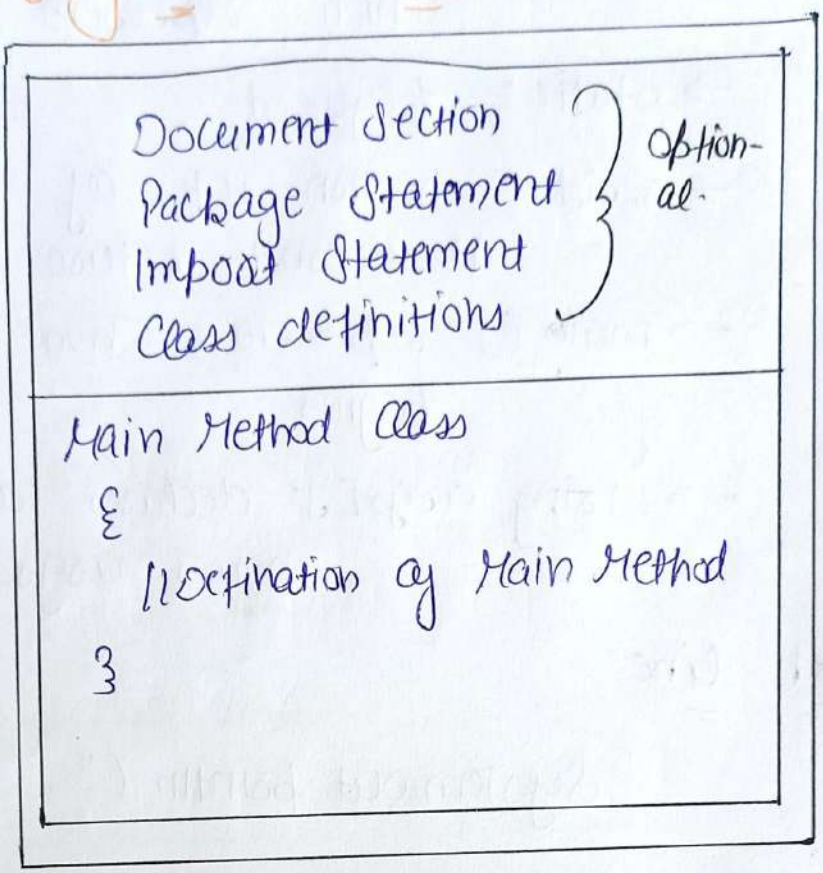
a)



- Compiling the program.

Tannu.java —javac→ Tannu.class —run→ Tannu

Java Program Structure



Tokens

- Tokens are the smallest elements of a program that is meaningful to the compiler.
- The process of breaking down the source code into token is called Lexical analysis, and it is the first phase of the compilation process.
- Lexical Tokens are the building blocks for parsing and analyzing the syntax of the code.
- A java program can be written using tokens, white Space.

→ Type of Tokens

- keywords
- Identifiers
- Literal
- operators
- Comments
- Separators

→ These token are separted by the delimiters

White - Space

↳ defined as blank space, tab, newlines and form feeds.

- The white space are ignored by the compiler.
- White space are used to separate tokens.

⇒ keywords

↳ keywords are pre-defined or reserved words in a programming language.

- each keyword is meant to perform a specific function in a program.
- They cannot be used as a variable or object name or Class name. because these are pre-defined words by java.

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- default
- enum
- extends
- float
- do
- import
- private
- protected
- public
- this
- void
- try

⇒ Identifiers

↳ are used to name of variables, methods, array, class etc.
- usually defined by user.

eg: A-Z and a-z

- Rules for identifiers

1) An identifier may consist of underscore (_), dollar sign ($), characters (A-Z), (a-z)
2) Blank space are not allowed
3) Identifier are case sensitive
4) first letter of identifier must be contain underscore, characters, dollar sign.
5) keyword or reserved word are not allowed.

- Valid identifiers

_myvariable myvariable
$myvariable variable123
Myvariable

- Invalid Identifiers

```
My Variable     // Contain a space
123my           // Begin with a digit
char            // keyword
```

⇒ Literals

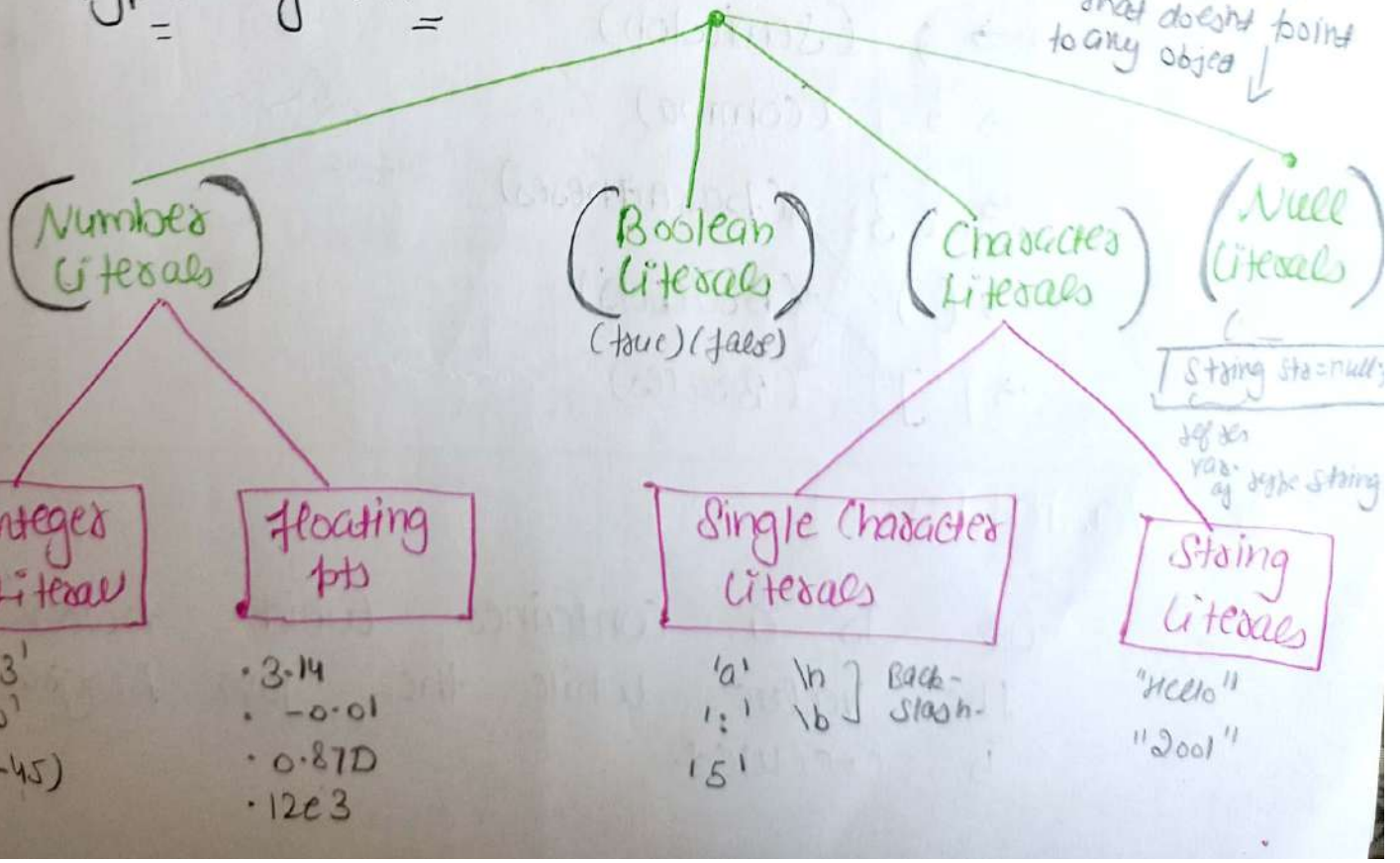
- also known as Constant.
- Literal is a notation that represents a fixed value in the source code, remains unchanged during the execution of a program.

[Symbolic Const
[final datatype const = val;
final float PI = 3.141 ;

eg:

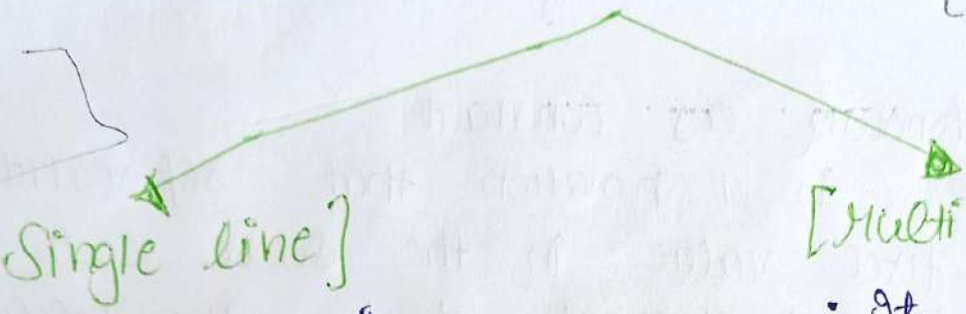
```
int a = 20;
    ↓   ↓
 variable  Literal
```

→ Types of Literals



⇒ Comments

- Comments are ignored by java compiler.
- Comments are used to enhance readability and understanding of code.



[Single line]

- It begin with double Slash (//)

[Multi-line]

- It specified with the use of /* */.

⇒ Separators (or Delimiters)

- used to indicate where the group of codes are divided and arranged.

→ ; (Semicolon)
→ , (Comma)
→ { } (Parentheses)
→ ( ) (Brackets)
→ [ ] (Braces)

⇒ Variables

- It is a container which holds the value while the java program is executed.

- A variable can be used to store a value of any datatype.

→ `datatype variablename;`

→ **Scope of variables**

↳ area of program where the variable is accessible is called scope of variable.

→ local variable
→ instance variable
→ Class variable

---

eg:

```
Class A
{
  int data;  //instance variable.
  static int m = 100;  //Class variable
  void method()
  {
    int n = 60;  //local variable.
  }
}
```

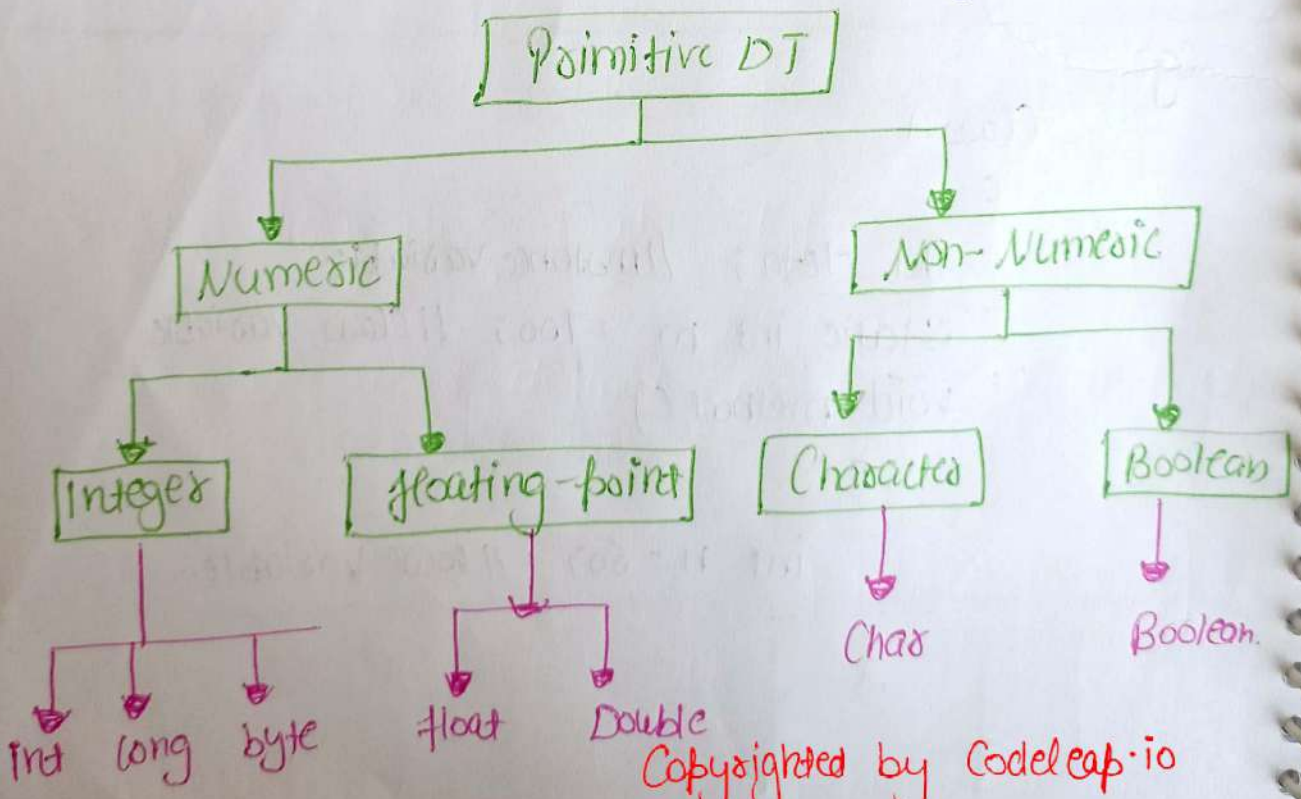
- Main Diff b/w local & instance variable →

[accessed by their name] [accessed by using the object of class]

Data Type

- Data type specify the size and type of values that can be stored in the variables.
- Data type are divided into two categories
  - Primitive Data types
  - Non-Primitive Data types.

→ Primitive Data types :- are the building block of data manipulation
- It consist of single value.



1) Integer types

- hold whole numbers
- eg: 101, 88, -115 etc.

- **Byte:** used to save memory in large arrays where the memory saving is required.

[default value is 0] of all integer type

→ Size: 8-bit
→ range: -128 to 127.

- **Short:** It is also used to save memory just like byte data type.

→ Singe: 16-bit
→ range: -32768 to 32767

- **int:** used as a default data type for integral values.

→ Size: 32-bit
→ range: -2147 to 2147 (-2^{31} to $2^{31}-1$)

- **long:** when need a range of value more than provided by int, it is use.

→ Size: 64-bit
→ range: (-2^{63}) to ($2^{63}-1$)

---

2) Floating types

- hold a fractional part
- eg: 47.25, -142.75.

- **Flood:** If need to save memory in large array of floating point no

- Single - precision of 32-bit.
- default value is 0.0f

Double: used for decimal value just like float value.
- Double-precision of 64-bit
- default value is 0.0d.

```
double PI = 3.14;
```

2) **Char:** → Used to store single Character
- Single 16-bit Unicode Characters.
- Range : 0 to 65536.

('\u0000' default value)

```
Char letter = 'A';
```

3) **Boolean:** → used to store only two value
- → True
- → false
- → It specfices one bit of information.

```
boolean one = true;
```

→ Non-Primitive Data Types ← used to access object

- **Class** → uses defined datatype which holds both data and methods.
  - data → internal data of Class
  - methods → function in Java
- **Interfaces:** Interface are implemented by Classes.

→ Array: It hold multiple variable of same data types

Eg:-

```
public Class Datatype Example
{
  public static void main (String [] args)
  {
    int i = 7;          //int
    char a = 'G';       //char
    byte b = 4;         //byte
    double d = 4.3567;  //double
    float f = 4.767F;   //float
    String greeting = "Hello"; //String
    int [] numbers = {1,2,3}; //Array

  System.out.println ("Char : " +a );
  System.out.println ("integer : " +i);
  System.out.println ("double : " +d);
  System.out.println ("String : " +greeting);
  System.out.println ("Array : " +number [0]);
  }
}
```

Java use Unicode

- It hold 2 bytes for characters
  - → lowest value = \u0000
  - → highest value = \uFFFF

→ • Robust,
• Versatile
• Internalization
• It represent characters, allow to support wide range of Char & symbol from various language.

Operators

- An operator is a symbol used to indicate a spectic operation on variable in a program.

eg: a + b → operands.
↓
operators

- Arithmetic operators
  → are use to perform arithmetic operation on variable and data.

operator	Symbol
Addition	+
Substraction	-
multiplication	*
division	/
modulo	%

Ex:
```
Class GFG {
    public static void main (String [] args)
    {
        int a = 10;
        int b = 3;
        System.out.println ("a+b =" + (a+b));
        System.out.println ("a/b =" + (a/b));
    }
}
```

- Assignment operator: used to assign value to variable.

eg: $x = 5;$

$x += 7$ → Shorthan operator.

assignment operator

operator	Description
= (assignment)	assign the right operand value to left operand. $x =$
+= (addition assignment)	It add the right operand to the left operand and assign the result to left operand $x += 7$ $x = x + 7$
-= (Subtraction assignment)	It subtract the right operand from the left operand and assign the result to left operand.
*= (Multiplication assignment)	It multiply the right operand with the left operand and assign the result to left operand
/= (Division assignment)	It divide a varible by left operand with the right operand and assign the result of left operand

Ex:-

```
import java.io.*;
Class GFG
{
  public Static Void main (String[] args)
  {
    int a=7;
  System.out.println("a+=3: " + (7+=3));
  System.out.println("a-=2: " + (a-=3));
                          // f= f+3; 7+3 =10
  }
}
```

[a+=3]
a= a+3

output:

a+=3 : 10
a-=2 = 8

• Relational operators

• are used to compare two numbers and return a boolean value.

Operator	Description
< (less than)	It return True, if value of left operand less than right operand, else false
> (Greater than)	It return True, if value of left operand greater than to right operand.
== (equal to)	It return True, if two operand are equal. otherwise false.
!= (Not equal to	It return true if left operand not equal to right operand.

→ Ex:
```
import java.io.*;
class GFG {
    public static void main (String [] args)
    {
        int a = 10;
        int b = 20;
        System.out.println ("a==b: " + (a==b));
        System.out.println ("a!=b: " + (a!=b));
    }
}
```

output: a==b : false
a!=b : True.

• logical operators:

Used to check two or more expression is true or false.

operators	Description
&& (Logical AND)	return true when both condn are true.
|| (Logical OR)	return true if at least one condn is true.
! (Logical NOT)	return true when a condn is false and vice-versa.

Ex:

```
import java.io.*;
Class GFG {
    public static void main (String [] args)
    {
    boolean x = true;
    boolean y = false;
System.out.println ("x && y : " + (x && y));
System.out.println ("x || y : " + (x || y));
System.out.println ("! x : " + (! x));
}
}
```

output:

x && y : false
x || y : true
! x : false

- Bitwise operators :- Manipulate the data at bit level.

→ '&' (bitwise AND) ["p&q: " + (p&q));
→ '|' (bitwise OR)
→ '^' (bitwise XOR)
→ '~' (bitwise NOT)
→ "<<" (left Shift)
→ ">>" (right Shift)
→ ">>>" (unsigned right Shift).

- Ternary operators

Condition ? expr 1 : expr 2 ;

→ (x > y) ? x : y ;

Precedence and Associativity

- Precedence and associativity rules are used when dealing with hybrid eqn involving more than one type of operator.
- In such cases, these rules determine which part of eqn consider first.

- Precedence:

→ Determine which operation is performed first in an expression with more than one operation.

[Higher precedence are evaluated first]

eg → 10 + 20 * 30

10 + 600

610

[✓]

→ 10 + 20 * 30

30 * 30

[+]

- Operator associativity :

→ It is used when two operator of same precedence appear in an expression

→ Associativity can be : left to right
Right to left

Higher

Category	Operator	Associativity
Postfix	() []	left to right
Unary	+ ++ ~ -- !	right to left
Multiplicative	/ * %	left to right
Additive	+ -	left to right
Relational	<< >> >=	left to right
Equality	== !=	left to right
Bitwise AND	&	left to right
" XOR	^	left to right
" OR	|	left to right
Logical AND	&&	left to right
" OR	||	left to right
Conditional	?:	Right to left
Assignment	= += -= *= /=	Right to left

Lower precedence

Assignment

'=' is used to assign value to variable.

```
variable-name = expression;
```

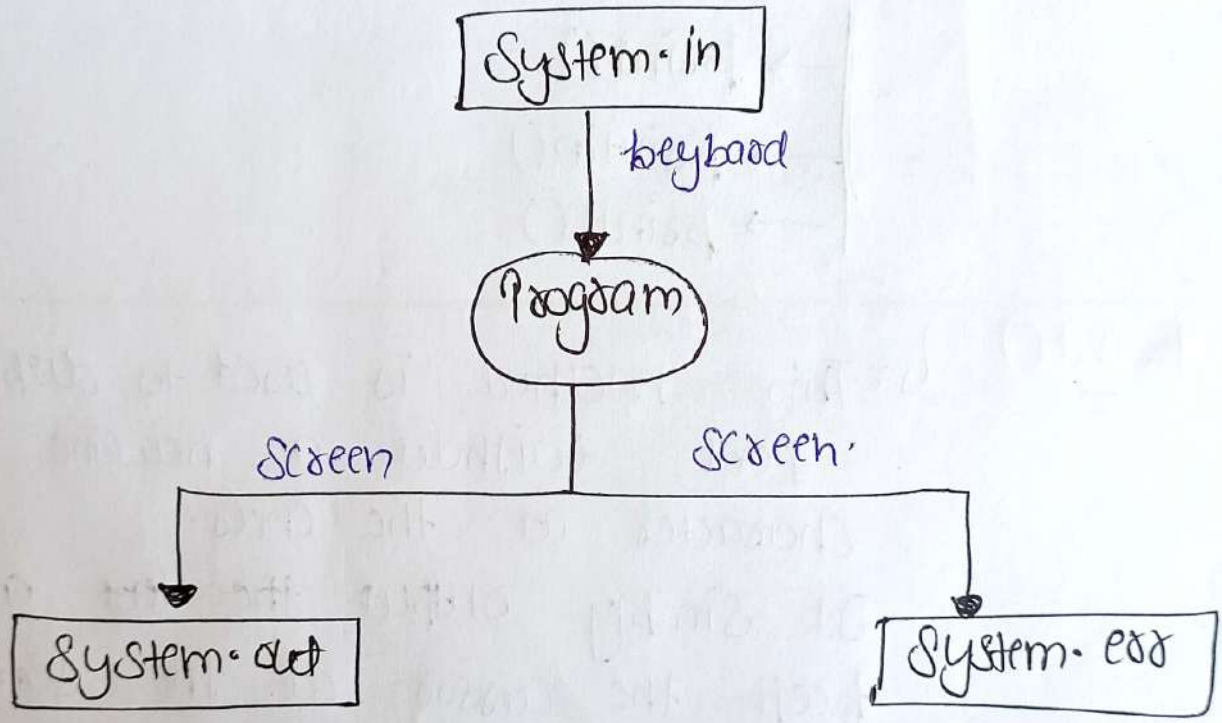
eg: x = x + 7

↓

x += 7 is equivalent to x = x + 7

I/O in Java

- Used to process the input and produce the output.
- Java uses the concept of a Stream to make I/O operation fast.
- The java.io package contains all the classes required for input and output operations.



- Stream: sequence of data
  - composed of bytes.
  - all Stream are attached with the console.

→ System.out : Standard output Stream
→ System.in : Standard Input Stream
→ System.err : Standard error Stream

Standard Output

- This is the standard output stream that is used to produce the result of a program on an output device like the computer screen.
- This is done using the System.out object.
- There are three method that can be used to display output

→ print()
→ println()
→ printf()

- print() :→ This method is used to display text without a newline character at the ends.
  - It simply output the text and keep the cursor at the same line.

→ Syntax: System.out.print(parameter);

→ Ex: import java.io.*;

```
Class Demo {
    public static void main (String [] args)
    {
        System.out.print ("GfG!");
        System.out.print ("GfG!");
    }
}
```

Output: GFG! GFG!

- println(): This method is used to print text to the standard output with a newline character at the ends.
- It print the text and move the cursor to next line.

→ Syntax: System.out.println(parameters);

→ Example:

```
import java.io.*;
Class Demo
  {
    public static void main (String[] args)
   {
   System.out.println ("GFG ! ");
   System.out.println (" GFG ! ");
  }
}
```

[both take single argument

→ Output:

GFG!
GFG!

- printf(): This method is used for formatted output. It allow you to specify a format string similar to those used in C language 'printf()' function.
- It take multiple argument as compare two to other method.

format specifier
%d → integer
%f → floating
%s = String
%b = boolean

Eg: int num = 10;
System.out.printf ("value of num is : %d %n", num);

output → value of num is : 10

- %d is a placeholder for an integer value and %n is newline character

Standard Input

- It is referred as 'System.in'
- There are various way for reading input data from the users in the command line environment (Console).

→ Scanner Class
→ Buffer Reader Class
→ Command line argument

- Scanner Class :→ The 'Scanner' Class in Java is part of the 'java.util' package.
  - It is a convenient way to read input of various type from different sources such as Standard input, file.
  - It provide method to parse primitive types and String. make it easy to handle user input. using regular expression.

formatted input = is done by Scanner Class

→ hasNext()
→ nextInt()
→ nextDouble()
→ next().
→ nextLine().

bas: ← primitive DT

· This is done with the keyword import

```
import java.util.Scanner;
```

import keyword — Package — Class

---

eg:

```
import java.util.Scanner;
public class inputExample {
    public static void main (String [] args)
    {
        // create an object of Scanner
        Scanner scanner = new Scanner (System.in);
        System.out.println (" Enter name! ");
        String name = scanner.nextLine ();
        System.out.println ("Hello, " + name);
        scanner.close ();
    }
}
```

create new scanner object.

Name :

output:

```
Enter name : tannu
Hello, tannu.
```

→ new keyword is use to create new object of class. The object we are creating is of Scanner Class.

· BufferReader Class

→ used to read text from a character - input stream. It buffers the input to provide efficient reading of characters, arrays. and reducing the no of I/o operation

→ BufferedReader reader = new BufferedReader(
new InputStreamReader(System.in));

String name = reader.readline();
System.out.println(name);

- 'BufferedReader' is created by wrapping 'System.in' with an 'inputStreamReader', which converts bytes from the input stream into characters.

- Command line arguments
  - Command line arguments are parameters passed to a program when it's executed via the command line.
  - In java, these argument are stored as strings in the 'arg' parameter of the 'main' () method.

['String[] args']

→ The first argument is stored in arg[0], second in arg[1] .....

→ public Class Main {
public static void main (String[]
{
System.out.println("no of arg :" + arg.length);
for(int i = 0; i < args.length ; i++)
{

```
System.out.println("Argument " + i + ": " + arg[i]);
}
}
```



```
3
3
```

→ No. of arg : 3
Argument 0 : arg1
" 1 : arg2
" 2 : arg3

↓

[Array Contain the command-line argument passed to program

Classes

- A class in java is a blueprint for objects that define the structure and behavior (properties and methods).
- It hold both data and method.

→ Class is not a real-world entity, it just a blueprint from which object are created.

→ Class does not occupy memory.

Defining a Class

```
access_modifies Class <class_name>
{
    [data member ;]
    [ method ;]
    [constructor ;]
    [nested Class ;]
    [interface ;]
}
```

→ A Class can create using Class keyword.

→ Class_name is the user define name of Class.

→ Inside the Class, these identifies are optional.

Ex:

```
Class Student {
    int rollno;
    String name;
    void display() {
        System.out.println("roll no is " + rollno);
        System.out.println("name is " + name);
    }
}
```

Object

- An object is an instance of a Class. When a Class is defined, no memory is allocated until an object of that Class is created.

→ Creating an object

To create an object, use the 'new' keyword.

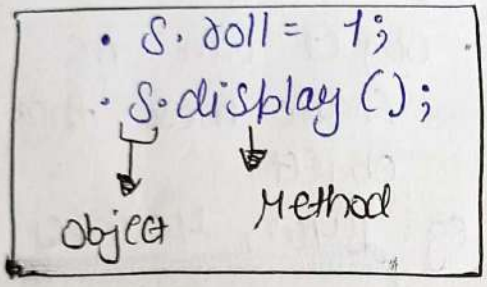
Ex:

```
public Class C_ClassandObject {
    public Static void main (String [] args)
    {  // Create an object of Student Class.
        Student a = new Student();
        a.rollno = 1;
        a.name = "Tannu";
        // call the method on object
        a.display();
    }
}
```

→ It saves as named [C_ClassandObject]

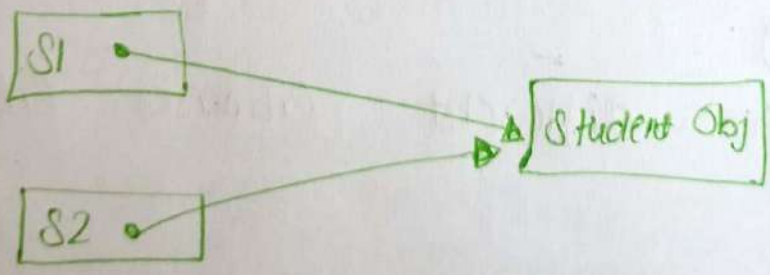
- An object consist of
  - State (represent the data of an object)
  - Behaviour (represent the behavior of an object)
  - Identity: [identify each object uniquely]

- Accessing Class Members
  a) objectname.variablename = value;
  b) objectname.methodname (actual arguments);



- Assigning object reference variable
  - possible to create two or more reference to the same object

  a) Student S1 = new Student();
     Student S2 = S1;



- By using Constructor
  - Student S1 = new Student("A", "B", "C");

Class

- Class is the blueprint of an object
- A Class does not allocate memory space when it is created
- Class is a logical entity
- A Calss can be only declared once
- eg: Car

Object

- An Object is an instance of the Class.
- Object allocates memory space whenever they are created.
- Class object is a physical entity.
- object can be create more than one object

eg: BMW, Mercedes etc.

Anonymous Object

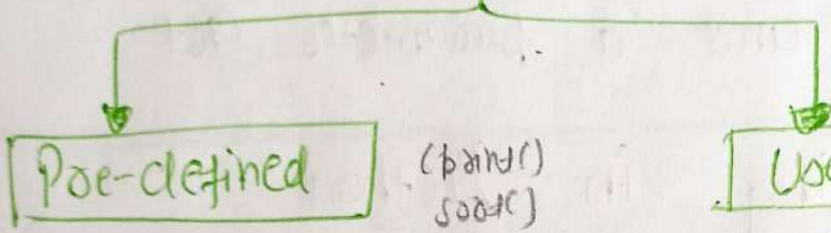
- These are Object that are instantiated but are not stored in a reference variable.
- They will be destroyed after method calling.
- used in different libraries like AWT.

Method → Static method, Instance method } way to create

- A method is a block of code or collection of statement or a set of code grouped together to perform a certain task or operation.

- It is used to achieve the reusability of code (code optimization)
- The method is executed only when we call it
- The most method in Java is **main() method**

→ Type of Method



Pre-defined (print() sqrt() equals() length())
- that is already defined in java class libraries → built-in method
- we can directly use these method just by calling them in program at any point

User-defined (add() sum() sub())
- The method written by the user or programmer is known as user-defined method.
- These method are modified according to requirement.

→ Method declaration

```
accessmodifier returntype methodname (list)
{
    // method body
}
```

Accessmodifier : public, private, protected.
returntype : 'int', 'void', 'String'.
Methodname : Should be start with lowercase letter

eg:

```
                Method name
                    ↓
public int sum (int a, int b)
{                   ↑
                    parameter list
    //method body
}
```

→ Method signature :- It contain method name and a parameter list.

Argument passing in Method

- Parameter passing in Java refers to the mechanism of transferring data b/w method or functions.
- Java support two types of parameter
  - → Call by value
  - → Call by reference.

→ Type of parameters

a) Formal parameters

- A variable and its corresponding data type are referred to as formal parameters when they exist in the definition or prototype of function.

Syntax:

```
returntype functionName (datatype)
{
    // function body
}
```

b) Actual parameters

The value or expression that corresponds to a formal parameter and is supplies to a function or method during call is refered actual parameter

Syntax:

function name (argument)

Call - by - Value

- The copy of the value of the actual parameter is passed to the formal parameter of the method.
- Any of the modification made to the formal parameter within the method do not affect the actual parameter
- Java passes all primitive data type by value.

---

eg:

```
Class demo
{
    Void inc (int x)
    {
        x = x + 3;
    }
}
Class CallbyValue
{
```

```
public static void main (String args[])
{
    demo obj = new demo();
    int i = 20;
    System.out.println("Before Calling");
    System.out.println(" value of i = " +i);

    obj.inc(i)          // Method Called.
    System.out.println(" After Calling");
    System.out.println("value of i =" + i);
}
}
```

- Output:
```
Before Calling
value of i = 20
After Calling
value of i = 20
```

→ i remain unchanged after calling the 'increment' method.

Call by reference

[(copy) of value]

→ receive a reference of variable passed as an argument

- A reference to the actual parameter memory location is passed to the formal parameter of the method.
- Any modification made to the parameter inside the method will affect the original variable.
- Java passes all non-primitive data types.

Ex:-

```
class demo
{
    int i;
    demo(int i)          // Constructor
    {
        this.i = i;
    }
    void inc(demo x)     //object as argument
    {
        x.i = x.i + 3;
    }
}
class Call by reference
{
    public static void main(String[] args)
    {
        demo obj = new demo(20);
        System.out.println("Before Calling");
        System.out.println("value of i =" + obj.i);
        object.inc(obj); //method called
        System.out.println("After Calling");
        System.out.println("Value of i =" + obj.i);
    }
}
```


Output:

```
Before Calling
value of i = 20
after Calling
Value of i = 23.
```

- Value of 'i' change after the calling 'increment' because the method directly modifies the object referenced by 'obj'.
- 'obj' inside the 'increment' method affect the original object referenced by 'obj'.

Constructor

- In Java, a constructor is a special type of method that is automatically called when an instance of a Class is created.
- It is a special method because its name is same as the Class name
- The constructor is invoked whenever an object of its associated Class is created.

Characteristics

→ Constructor do not have a return type, not even 'void'.

→ Constructor can be overloaded.

Syntax:

```
Public Class Classname {
    //Constructor
    Public Classname () {
        // initialization code
    }
}
```

Types of Java Constructor

1.) Default constructor
2) Parameterized constructor

- Every time an object is created using the new() keyword, at least one constructor is called.

```
Class Classname
{
    ...
    // constructor
    Classname() {
        ...
    }
}
// create an object of above Class
// Call above constructor
Classname obj = new Classname();
```

Rules (When java constructor is called)

- A constructor can not be abstract, final, static
- Access modifies can be used in constructor declaration.
- The constructor of a Class must have the same name as the Class name.

Default Constructor

→ Cannot be overloaded as it has no parameter

- A constructor that has no parameters is known as Default constructor.
- Purpose: It is used to provide the default value to the object like 0, null, etc, depending on the type.

- Example:

```
Class Student
{
    int rollno, marks;
    // Creating a Default Constructor
    Student()
    {
        rollno = 1;
        marks = 12;
    }
// method to display the value
    void display()
    {
        System.out.println("Roll no:" + rollno);
        System.out.println("marks:" + marks);
    }
public static void main(String[] args)
    {
    // Creating object and passing value
        Student s = new Student();
    // Calling Method to display the value obj
        s.display();
```

```
    }
}
```

Output: Rollno : 1
Marks : 12

Parameterized Constructor

→ Can be overloaded with different parameter set.

- A constructor that has parameter is known as parameterized Constructor
- It is used to provide different value to distinct object.

Ex:

```
class Student
{
  int roll;
  String name;
  // Parameterized constructor
  Student(int r, String n)
  {
    rollno = r;
    name = n;
  }
  // Method to display the value.
  void display()
  {
    System.out.println("Rollno : " + rollno);
    System.out.println("Name : " + name);
  }
  public static void main(String[] args)
  {
```

(Class)

(Parameterized Constructor)

(Display Method)

(CSI)

(Main Method)

```
// Creating object using parametrized constructor
Student S1 = new Student (1001, "Tannu");
Student S2 = new Student (1002, "Annu");
// Displaying the detail of object (S1)
S1.display();
S2.display();
}
}
```

Output: Roll no : 1001
Name : Tannu
Rollno : 1002
Name : Annu.

Constructor Overloading

- Constructor Overloading allow a Class to have more than one constructor with different parameters.
- They are arranged in a way that each constructor perform a different task.

Ex:

```
Class area
{
    area (double side)   // Constructor I defined.
    {
        double sqareaea = side *side;
        System.out.println("Area of Square = " +sq);
    }
```

```
area(double length, double Breadth)   // Constru 2 defined.
  {
    double rectarea = length * Breadth;
    System.out.println("Area of Rectange = " +rectarea);
  }
public static void main (String args[])
  {
    area a1 = new area (2.5);          // Const 1 invoked
    area a1 = new area (3.5, 4.5);     // Cons2 invoked
  }
}
```

output: Area of square = 6.25
Area of rectangle = 15.75

→ Class Student

```
{
  int id; int age;
  String name;
  // Creating Constructor 1
  Student (int i, String n)
  {
    id = i;
    name = n;
  }
  // creating Constructor 2 with three arg.
  Student (int i, String n, int a)
  {
    id = i;
    name = n;
```

→ void display()
System
(id + " " + name + " " + age);

Java Constructor

- A constructor is used to initialize the state of an object
- A constructor must not have a return type
- The constructor name must be same as the Class name
- The constructor is invoked implicitly.
- The Java compiler provide a default constructor if you don't have any constructor in a Class

Java Method

- A method is used to expose the behaviour of an object.
- A method must have a return type.
- The Method name may or may not be same as the Class name.
- The method is invoked explicitly.
- The method is not provided by the compiler in any case

Default Constructor

- No parameter
- Automatically provided by Java if no constructor are defined
- Class myClass {
  myClass ()
  {
  //
  }

Parameterized

- Can have parameter
- Defined explicitly by the programmer with parameters
- Class myClass {
  myClass (int a, int b)
  {
  //
  }

Default Constructor

- Instantiated when an object is created without argument

Parameterized

- Used when an object is created with specific value passed as arguments

---

Copy Constructor

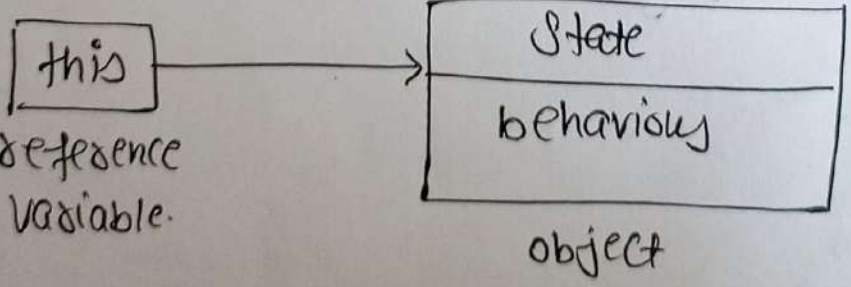
- There is no copy constructor in Java; we can copy the value of one object to another by using constructor like copy constructor in C++.

→ There are many ways to copy the value of one object into another

- By Constructor
- By clone() method of object class

This keyword

- These can be a lot of usage of java this keyword. In java, this is a reference variable that refers to the current object.



→ Way to uses of 'this' keyword.

- to refer the current Class instance variable
- Using this() to invoke the current Class constructor

Eg:

```
Class Student {
        int a;
        int b;
    Student (int a, int b)
      {
        this.a = a;
        this.b = b;
      }
    void display()
      {
        System.out.println("a = " +a+ " b = " +b);
      }
    public static void main (String [] args)
      {
        Student obj1 = new Student (1001, 80);
        Student obj2 = new Student (1002, 90);
        obj1.display();
        obj2.display();
      }
}
```

Student()
{ System.out.println("Create") invoked
}

a = a;
b = b;

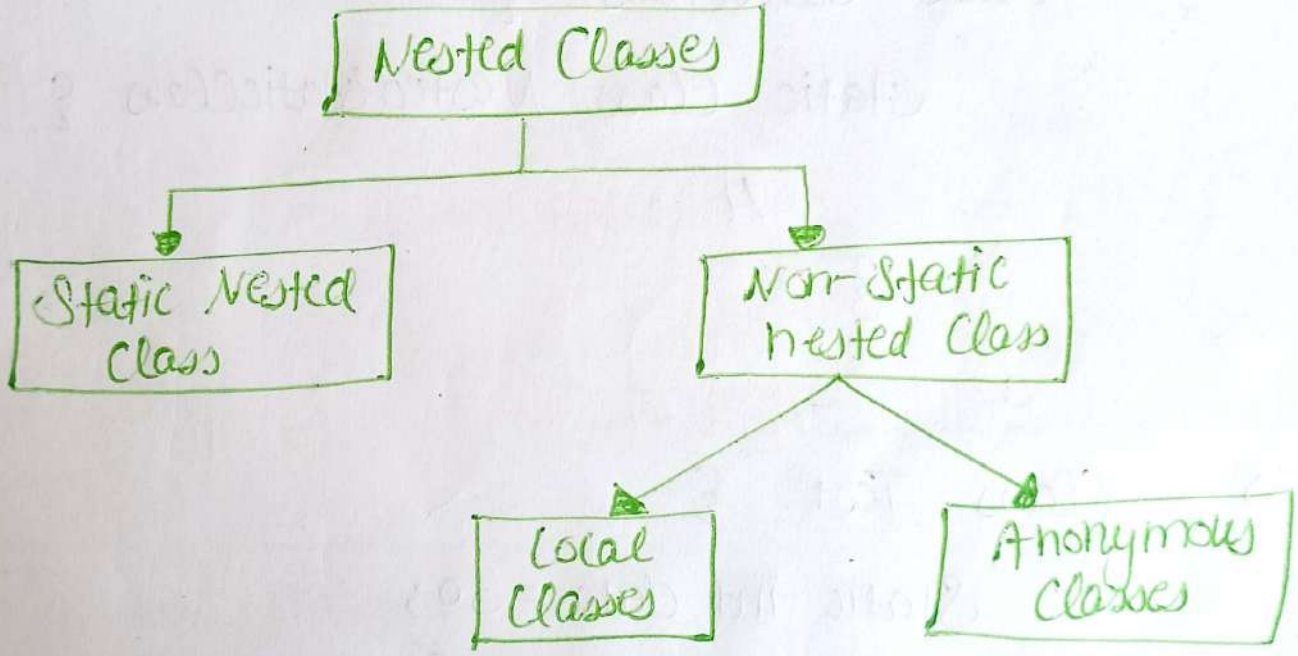
name of parameters & instance variable are same and get result wrong, so this problem using this keyword.

output: a = 1001, 80 (b2)
= a = 1002, 90 (b=)

Nested Class

- A class that is defined within another class is known as nested classes.
- They enable to logically group classes that are only used in one place, thus this increases the use of encapsulation and creates more readable and maintainable code.

→ Nested Classes are divided into two categories



Syntax:

```
Class OuterClass
{
...
    Class NestedClass
    {
    ...
    }
}
```

• Static Nested Class

[outerClass.StaticNestedClass nestedObject = new outerClass.StaticCla();

→ A Static Class is a Class that is created inside a Class is called Static nested Class

→ It Cannot access non-Static data member and methods

→ It can be accessed by outer Class name; including ~~private~~.

Ex:
```
Class outerClass {
    Static Class NestedStaticClass {
        // . . .
    }
}
```

→ outerClass
```
Class Test {
    Static int data = 30;
    Static Class inner {
        void msg()
        {
            System.out.println("data is" + data);
        }
    }
    public Static void main (String args[])
    {
        Test.innerobj = new Test.inner();
        obj.msg();
    }
}
```

Static nested Class

method of nested Class 'inner'

Main() Method.

• Non-Static Nested Class (Inner Class)

- It is most important type of nested class. It is also known as Inner Class.
- We use inner classes to logically group classes and interfaces in one place so that it can be more readable and Maintainable, Code optimization (require less code to write)

→ Syntax:

```
Class OuterClass
{
    // code
    Class innerclass
    {
        // code
    }
}
```

- It not declared with the 'Static' keyword
- It have access to all members of enclosing (outer) Class including private
- It can be declared with access modifiers

eg:-

```
Class Test → outerClass
{
    private int data = 30;   → instance variable of outer Class
    Class inner → inner Class
    {
        void msg()
        {
            System.out.println("data is "+data);
```

```
    }
}
public static void main (String [] args)
    {
        Test . obj = new Test ();   ← instance of outer class
        Test . inner in = obj . new inner();   ← instance of inner class
         in . msg();
    }
}
```

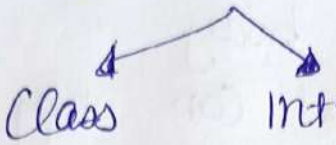
output = Data is 30 because it's accessing the instance variabe 'data' from the outer class within the inner class.

→ Local inner Class — This is a nested class that is defined within a method or block.

```
Class outerClass {
    void SomeMethod ();
  {
    Class Local innerClass  {
        // . .
    }
}
}
```

→ Anonymous inner Class — A Class without any name.

→ two ways



- It commonly used when need to create a small class for single-use scenario.

Abstract Class

- A Class which is declared with the abstract keyword is known as abstract Class in Java.
- It can have abstract and non-abstract method.

→ Syntax declaration of an abstract Class

```
public abstract Class Class-name
{
...
...
}
```

- It not used to create object. It is only based as base Class for other Classes.
- Abstract Class can't be Static.

→ Abstract Method

- Abstract method have no implementation in the abstract class, also does not have any method body
- Abstract method can only be declared in abstract classes.

→
```
abstract Class Class_name
    {
        abstract void method_name ();
    }
```

---

eg:→
```
//abstract Class
abstract Class Bike {
        abstract void run();
    }
Class Honda extends Bike
    {
        void run()
            {
                System.out.println("running safely");
            }
    }
Class abstractdemo
    {
        public static void main (String args[])
            {
                Bike obj = new Honda();
                obj.run();
            }
    }
```

↙ Bike is abstract class with abstract method run.
- honda is subclass of Bike.

Garbage Collection

- The technique to deallocate the memory is called garbage collection.
- It performed automatically.

→ In Java, the memory is allocated to the object using 'new' operator. In C++, the memory is deallocated using 'delete operator' but java deallocates the memory itself.

→ JVM is responsible for reclaiming memory from object that are no longer in use.

Static Members

- Static members are those which belong to the class and can access these members without instantiating the class.
- The static keyword use by declaring static members. → used for memory management.
- The static can be:
  - → variable
  - → method
  - → block.

- Static variable: → declare any variable as static.
  - → static variable get memory only once.

- It can be used to refer the common property of a Classes

eg:

```
Class Student
{
    int rollno;          //instance variable
    String name;         // instance variable.
    //Same as Constructor
```

- Static Method :- If apply static keyword with any method, it is known as static method.

Property

→ Static method belong to the Class rather than the object of a Class.

→ A static method can access static data members and can change the value of it.

→ It can be invoked without the need for creating an instance of a Class.

→ Restriction

- this and supper keyword not be used
- not use non-static data members
- only call static method.

eg:

```
Class Calculate
  {
    Static int cube (int x)
      {
        return x*x*x;
      }
    public static void main (String arg[])
    {
      int result = Calculate.cube(5);
      System.out.println (result);
    }
  }
```

[int a = 40;
↓
Error shown

→ 125

---

why Java main() Method is Static?

- because object is not required to call a static method.

- Java Static Block :→ We can declare a static block which get executed exactly once, when the class first loaded
  - → used to initialize the static data members
  - → executed before the main method at the time of classloading

eg:

```
Class A
  {
    static { System.out.println(" block ");
    }
public static void main(String arg[])
  {
    System.out.println("Hello");
  }
}
```


Array

- An array is a collection of similar type of element which has contiguous memory location.
- In Java, all array are dynamically allocated.

- Advantage : Code optimization , fast access
- Disadvantage: Size limit , Lack of flexibility

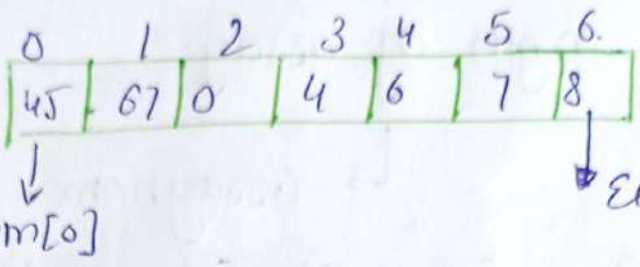
→ Type of Array

- Single Dimensional array
- Multi Dimensional array.

- Single Dimensional array

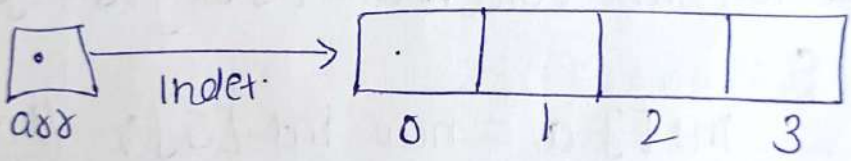
Only one subscript are used in one-D array.

- Array in Java are declared with a specific data type, followed by square bracked "[ ]" indicating the size of the array.



→ array length = 7
first index = 0
last index = 6

- One-Dimensional array



→ **Declaration**

datatype[ ] arr;

or

datatype arrayname[ ];

→ Eg: int marks[ ];
float[ ] numbers;

→ **Initialization of array**

- Data type arrayname[ ] = { value1, ... };

eg: int marks[ ] = { 40, 30, 60 };
// declare arr

// initialize array

→ marks [0] = 40;
marks [1] = 30;
marks [2] = 36;
marks [3] = 20;

→ **finding length of array:**

↳ arrayname.length

eg: int marks [] = { 40, 35, 36 };
int sizeofmarks = marks.length.

= 3

---

eg:
```
Class Testarray {
public static void main (String [] args)
{   int a[]
    int[] a = new int [5]; // declaration.
    // initialization
    a [0] = 10;
    a [1] = 20;
    a [2] = 70;
    a [3] = 40;
    a [4] = 50;
    // Traversing array // accessing array element
for (int i = 0; i<a.length ; i++)
    {
        System.out.println (a [i]);
    }
}
}
```

System.out.println ("Element ");
" " " ("Marks of Sub1
= " + Ma[2]);

- MultiDimensional array

→ It can be define by two subscripts or more.

→ In MultiDimensional array, data is stored in row and column based index.

$$x = \begin{bmatrix} 1 & 5 & 9 \\ 4 & 6 & 6 \\ 7 & 7 & 1 \end{bmatrix}$$

Column. (↓)

3*3 (→ row)

	Column 0	Column 1	Column 2
Row 0	1 x[0][0]	5 x[0][1]	9 x[0][2]
Row 1	4 x[1][0]	6 x[1][1]	6 x[1][2]
Row 2	7 x[2][0]	7 x[2][1]	1 x[2][2]

→ **Syntax:**

Datatype arrayname [ ] [ ] = new Datatype [row] [colu

→ **Declare**

Datatype arrayname [ ] [ ];
arrayname = new datatype [row][column];

eg: int matrix [ ][ ];
matrix = new int [2] [3];

Q: Class Main {

```
public static void main(String[] args)
{
  // initialize a 2D array with value
  int [][] arr = { {1,2,3}, {4,5,6}, {7,8,9} };
  System.out.println("No of rows : " + arr.length);
  System.out.println("No of column : " + arr[0].len);
  // Print 2D array value.
  System.out.println("Matrix value:");
  for (int j = 0; i < arr.length; i++)
  {
    for (int j = 0; j < arr[i].length; j++)
    {
      System.out.print(arr[i][j] + " ");
    }
    System.out.println(); // Newline for new row
  }
}
```

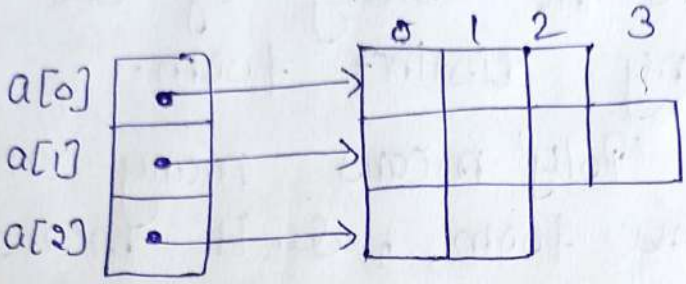
}

output:

```
No. of row : 3
No. of Column : 3
Matrix value :
1 2 3
4 5 6
7 8 9
```

Jagged array

- A jagged array is an array of array such that member array can be of different size



→ int a[][] = new int [3][]
a[0] = new int [3];
a[1] = new int [4];
a[2] = new int [2];

---

final keyword

- final is a modifier which provides restriction in java
- final keyword can be used as
  - final variables
  - final Classes
  - final Method

Syntax: final Class fClass
{
..
}

Polymorphism in Java

- Polymorphism is an important concept of OOPS
- It refers to the ability of one thing to take many distinct form.
- The word 'Poly' means many and 'morphs' means form. So, it means many forms.

→ Type of Polymorphism

1) Compile-time polymorphism
2) Run-time polymorphism

• Compile - Time polymorphism

- also known as Static Binding (early binding).
- It can be achieved by using method overloading.
- Static Binding means that the code associated with the method call is linked at compile time.

Benefit

- Code reusability
- Flexibility
- Improved readability.

```
Class Dog {
  void eat ()
    { System.out.println("...");
    }
  public static void main
    Dog d1 = new dog ();
    d1.eat ();
  }
}
```

Method Overloading

- Method overloading refers to defining multiple method in the same class with the same name but with different parameters.
- Method overloading increases the readability of the program.

→ Different way of Method overloading in Java.

- →Changing the no of Parameter
- → Changing Data type of Arguments.

Syntax:

```
Class Sum {
    public int sum (int x, int y)
    {
        return (x+y);
    }
    (double
public int sum (int x, int y, int z)
    {
        return (x+y+z);
    }
public Static void main (String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10,20));
        System.out.println(s.sum(10,20,30));
    }
```

→ Class main{

```
public static void main(String[] args)
{
// Declare, creation, initialization of Integer array
int[] myarray = {10, 20, 30};
// Accessing array element
for(int i = 0; i < myarray.length; i++)
{
System.out.println("Element " + i + ":" +
myarray[i]);
}
}
```

Output Element 0 : 10
" 1 = 20
" 1 = 30.

Run-time polymorphism

- It is also known as Dynamic method dispatch
- It can be achieved by using Method overriding.
- Dynamic binding means that the code associated with the method call is linked at run time
- Dynamic binding also known as late binding

```
class Animal{
  ---
  ---
class dog extend
  animal
  {
  ---
  ---
Animal a = new dog();
  a.eat();
  }
}
```

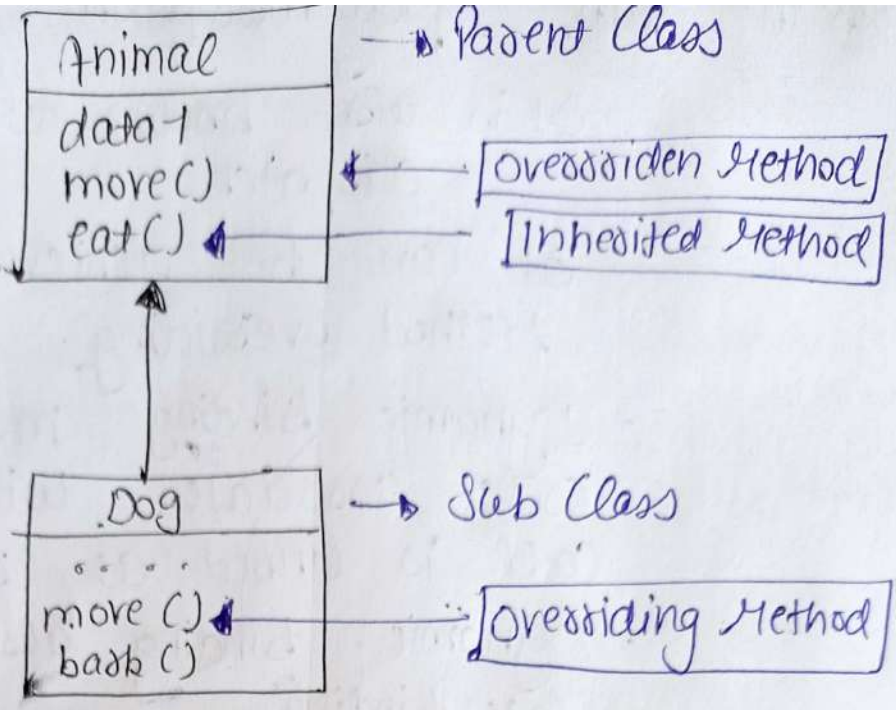
or

It is a process in which a call to an overridden Method is resolved at run time rather than compile-time.

Method Overriding

- Method overriding occurs when a subclass provides a spectic implementatio of a method that is already defined in its superclass.
- The method in the subclass must have the same signature (name & parameter) as the method in the superclass.)

→ Overriding and Access Modifiers
→ final Method can not be overriden
→ Static Method can not be overriden



Implementation

```
class Student {
    void display()
    {
      System.out.println("i m student");
    }
}
Class Studentpiet extends Student {
    void display()
    {
      System.out.println("i am piet stude");
    }
}
public Class C_method overriding {
    public static void main(String [] args) {
        Studentpiet s1 = new Studentpiet();
        s1.display();
    }
}
```

Wrapper Class

- Wrapper Classes are used to convert primitive data type into an object.
- When we create an object to a wrapper class, it contain a field and in this field, we can store primitive data type. i.e wrap a primitive DT value into a wrapper class object

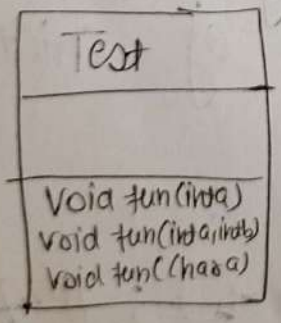
Primitive Datatype	Wrapper Class
char	Character
byte	Byte
Short	Short
int	Integer
long	Long

Upcasting

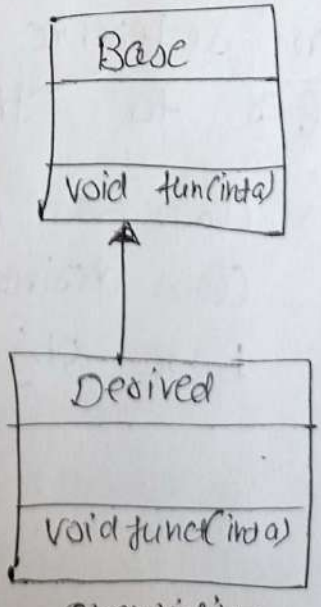
When reference variable of parent class refer to the object of child class.

```
→ Class base (. . )
  Class derived extends base (. . )
  base obj = new derived( ) ; // upcasting.
```

Method overloading	Method overriding
• Method overloading is a compile time polymorphism	• It is a run-time polymorphism.
• It help to increase the readability of the program	• It is used to grant the specific implementation of method which is already provided by its parent class or super class
• Must have different parameters	• Must have the same parameters.
• Can have the same or different return type	• Must have the same parameters.
• Not depend on inheritance	• Depend on inheritance.
• Static binding is being used for overloaded methods	• Dynamic binding is being used for overriding method.



Overloading



Overriding

String

- String is an object that represent seq. of characters.

→ String str = "Hello, World!";
→ String str = new String("Hello, World!");

Way to create a String
- Using String literal
- Using new keyword.

- Str

0	1	2	3	4	5	6	7	8	9
H	E	L	L	O	W	O	L	d	\0

- In java, every character is stored in 16 bits.

---

eg:
```
public class Stringer {
    public static void main(String[] args)
    {
        String str = new String("Tannu");
        System.out.println(str);
    }
}
```

output: Tannu.

String Class

- These are two ways to declare and create a string.
  - → without using constructor of String Class
  - → By using Constructure of String Class.

→ Create a String without Constructors

```
String Str = "Java";
```

- Create a string by name Str and initializes it to value Java.
- String is Class and Str is an object.

→ Create a String using String Constructors

a) To create an empty string.

```
String Str = new String ();
```

b) Constructs a String that is a copy of String Constant

```
String Str = new String ("Java");
```

c) Create String that have initial value.

```
String (Char Array of Class [])
```

Ex: Char chars[] = {'a', 'b', 'c', 'd'};
String str = new String(chars);

d) Specify a subrange of a character array

String(Char arrayofchars[], int Startindex, int numchars)

Ex:
= Char chars[] = {'a', 'b', 'c'};
String str = new String(Char 2, 4);

Ex:
=

```
Class Stringmethods {
    public static void main(String[] args)
    {
    String s1 = "java";
    String s2 = new String("Tannu");
    Char[] Char = {'a','b','c'};
    String s3 = new String(Char);
    Char[] Char2 = {'a','b','c','d','e'};
    String s4 = new String(Char2, 1, 4);
    System.out.println(s1);
    System.out.println(s2);
    "  "  "  (s3);
    "  "  "  (s4);
    }
}
```

→ java
tannu
abc
bcde.

String operation (or String Class Method)

- String operation are performed by using String Class method.

1) Finding String length

- used to find the length of string.

```
String Str = "Hello";
int length = Str.length(); // 5.
```

eg:
```
class lengthExample
{
    public static void main(String args[])
    {
        String Str = "DS";
        System.out.println("String length: " +
                           Str.length());
    }
}
```

Output:
String length : 2

2) String Concatenation

- Combine two string by using 2 Method
  - → + operation
  - → Concat() method. two ↓ string combine into a single one.

eg: Class Concaterample

```
{
    public static void main (String args[])
    {
        String s1 = "Ram"
        String s2 = s1.concat ("Kumar");
        System.out.println (s2);
    }
}
```

String s1 = "Ram"
String s2 = "Kumar"
String result
= s1 + s2;

output: Ramkumar

3) String Comparision

- The String Class includes several method that Compare Strings.

→ CompareTo () Method
→ equal () Method

---

eg: Class StringCompare

```
{
    public static void main (String arg[])
    {
        String s1 = "Aditi";
        String s2 = "Tannu";
        String s3 = "Tannu";
        System.out.println (s1.CompareTo (s2));
        System.out.println (s1.compare to (s3));
    }
}
```

→ Suppose S1 and S2 are two String Object

- S1 == S2 : The method return 0
- S1 > S2 : The method return a positive value
- S1 < S2 : Negative value

Eg:

```
Class equal Example
{
    public static void main (String args[])
    {
        String S1 = "Tannu";
        String S2 = "Tannu";
        String S3 = "Goel";
        System.out.println (S1.equals (S2));
        System.out.println(S1.equals (S3));
    }
}
```

output : true
false

→ It compare the two given String for equality.

→ If not matches, return false

→ If all matches, return True.

3) Character Extraction

- provide to extract characters from a given String object

→ charAt() Method

↳ return a char value at a given index number.

- The index no start from 0.

eg: String str = "Hello";
Char ch = str.charAt(1);

Ans → 'e'.

→ getBytes() Method

↳ return a byte array of String words.

4) Searching in the String

- The indexOf() Method is used to search for the first occurrence of a character string. and returns the index at which the character found.

eg: String s = "Sushil";
System.out.println("Indexof('i') " + s.indexOf('i'));

→ indexof('i')4.

5) **Extracting Substring**

- **Substring()** method return a part of the String.

eg:

```
String S1 = "Tannu";
String.out.println(S1.SubString(2));
```

→ nnu.

6) **Replacement of String**

- to replace certain String with some other String

→ **replace() method**

↳ replace the old char sequence to new char sequence.

eg:

```
String S = "Tannu Dhanda";
String rString = S.replace('a','e');
System.out.println(rString);
}
}
```


output: Tennu Dhende

→ **trim() Method**

↳ eliminates leading spaces.

7) tolowerCase() Method

↳ return the string in lowercase letters.

eg: String s = "JAVA";
String slower = s.toLowerCase();

→ java.

8) toUpperCase() Method

↳ String s = "string";
String supper = s.toUpperCase();

→ STRING

String Buffer Class

- StringBuffer is similar as String but provides some additional functionality of Strings
- Stringbuffer Class is mutable, it means that is growable and writable too.
- Each String Buffer has a fixed capacity.

→ Create a String by using Stringbuffer Constructors.

- StringBuffer()
  ↳ With no character with initial capacity 16 characters.

eg: StringBuffer sb = new StringBuffer();

- StringBuffer(int size)
  - Create a StringBuffer object with no characters but with initial capacity. (40);

→ Method of StringBuffer Class

- length() Method
  - return current length of a StringBuffer

eg → StringBuffer str = new StringBuffer("Ta...
System.out.println(str.length());

⇒ 2.

- Capacity() Method
  - return the current capacity of the buffer i.e total allocated space.

eg: →

- append() Method
  - Concatenates the given argument with this String.

eg:

```
StringBuffer str = new StringBuffer ("Hello");
str.append (" Java");
System.out.println (str);
```

- **reverse () Method**
  - used to reverse the current string

eg:

```
StringBuffer str = new StringBuffer ("Hello");
str.reverse ();
System.out.println (str);
```

→ olleh.

- **insert () method.**
  - insert the given string at a given position in a StringBuffer objeG

eg

```
StringBuffer str = new StringBuffer ("Hello");
str.insert (1, "Bye");
System.out.println (str);
```

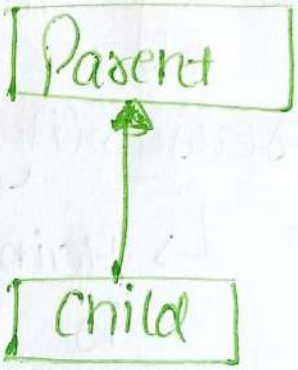
→ HByeello

Program 5

String Class	String Buffer
• String Class is immutable	• String Buffer Class is mutable.
• Once a String object is created, it cannot be changed	• It can be changed
• String Class implement immutable characters strings which are read-only.	• It can be mutable characters implement which are growable and writable too.
• It is slow and consumes more memory	• It is fast and consumes less memory.
• No thread-safe	• Thread safe

Inheritance

- Inheritance in Java is a Mechanism in which one object acquires all the properties and behaviour of a parent object.
- It is an important part of OOPs.
- Inheritance represent the IS-A relationship which is also known as a parent-Child relationship.



→ Derived Class :) The Class which inherits the other Class.
- It is also called SubClass, Child Class.

→ Base Class :) The Class whose features are inherited.
- also known as Super / parent Class.

Syntax of Java inheritance

```
Class Subclass-name extends SuperClass-name
{
    // method and fields
}
```

→ extends keyword indicates that making a new class that derives from an existing class, it means [increase the functionality].

Benefits of Inheritance

- Code reusability
  - ↳ Main advantage of inheritance is reusability
    - → Inheritance allow the reuse the code, minimizing redundancy.
    - → This reuse reduce the need to write similar code multiple time, saving time and effort.
- Extendability
  - ↳ Inheritance make it easier to extend existing code by adding new functionalities to

Subclass without modifying existing code, thus adhering to the open/closed principle.

eg: 'Person' class, extend it to create a 'Student' class with additional attributes like 'grades' and 'school'. thus extending the functionality.

- Method overriding: → Subclasses can provide specific implementation for method defined in the superclass.
  - This allow for polymorphic behaviour where the method get executed is determined at run time.

- Encapsulation

- Polymorphism: → Inheritance support polymorphism enabling a single interface to represent different forms.
  - This allowing for flexibility in code.

- Code organization and Maintenance
  → It help to organize code into a logical hierarchy, making it easier to understand and maintain., reducing duplication.

Access Control

- It define how the member of a class can be accessed.

→ Private
→ Public
→ Protected
→ Default

} Access control in JAVA

- Public Access :- The public access modifier is accessible from everywhere.
  - The public access modifies is specified using the keyword public.
  - There is no restriction on the scope of public data members.

eg:
```
public Class A
{
    public void display ()
    {
        . . .
    }
}
```


- Private Access :- The private acess modifies is specified using the keyword private.

- The method or data members declared as private are accessible only within the class in which they are declared.

eg :-
```
Class A {
    private void display ()
    {
    .
    .
    .
    }
}
```

- Protected access: The protected access modifier is specified using the keyword protected.
  - The method or data members declared as protected are accessible within the same package or subclass in different packages.

eg:-
```
public Class A {
    protected void display ()
    {
    .
    .
    .
    }
}
```

- Default Access: When no access modifies is specified for class, method, data members, it said to be Default Access modifier.

eg: Class A
{
void display()
{
...
}
}

→

AM / AL	Default	Private	Public	Protected
Same Class	Yes	Yes	yes	Yes
Same package subClass	yes	No	Yes	Yes
Same package non-subClass	yes	No	Yes	Yes

→ Show the visibility provided by various access modifier.

Type of Inheritance

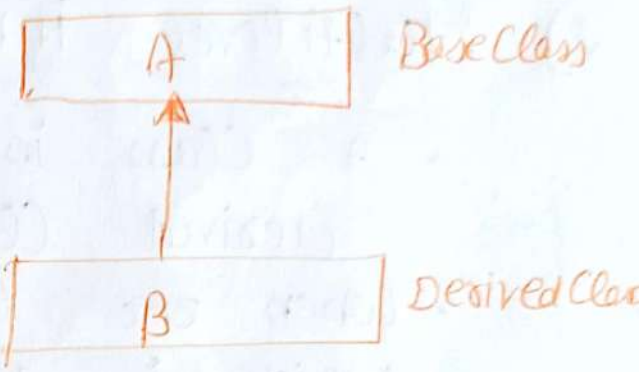
1) Single Inheritance
2) Multilevel inheritance
3) Hierarchical inheritance.

· Java does not support Multiple inheritance directly because multiple inheritance support multiple base Class and in java, the classes cannot be inherited from more than one Class., then.

implement multiple inheritance through interfaces.

1) Single Inheritance

- When a class inherits another class, it is known as single Inheritance
- There is only one base class and one derived class.



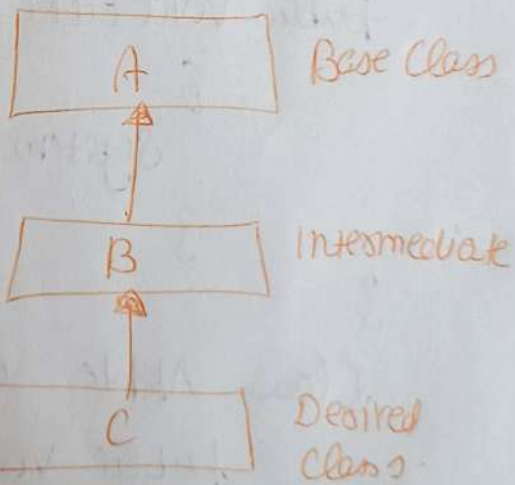
eg:
```
Class fruit {
public void eat()
        {
          System.out.println("eating");
        }
}
Class Apple extends fruit {
      public void smell()
            {
              System.out.println("smell");
            }
      Class Testinheritance {
          public static void main (String args[])
                {
```

```
        Apple d = new Apple();
        d.apple();
        d.eat();
    }
}
```

Output: Smell
Eating

3) Multilevel inheritance

- A class is derived from another derived class.
- When one class inherits another class which is further inherited by another class. It is known as multilevel inheritance.



→ Class A is base class for the derived class B

→ Class B is a base class for derived class C

∴ Class B is an intermediate base class.

Syntax:

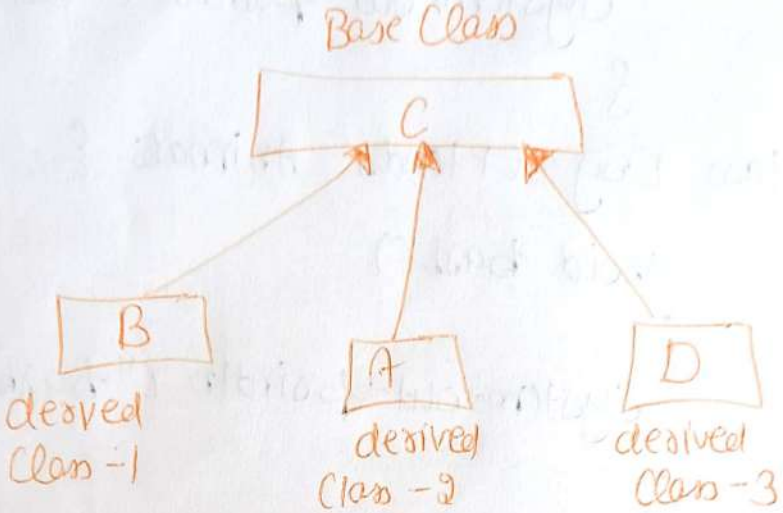
```
Class A
{
}
Class B extent Class A
{
}
Class C extend B
{
}
```

File : Testinherit2.java

eg:-
```
Class Animal {
    void eat()
    {
     System.out.println("eating");
    }
Class Dog extends Animals {
     void bark()
    {
     System.out.println("barking");
    }
Class BabyDog extends Dog {
     void weep()
     {
      System.out.println("weeping");
     }
}
Class Testinhesitance2 {
 public static void main(String args[])
     {
      BabyDog d = new babyDog();
      d.weep();
      d.bark();
      d.eat();
     }
}
```

2) Hierarchical Inheritance

- When two or more classes inherits from a single class.



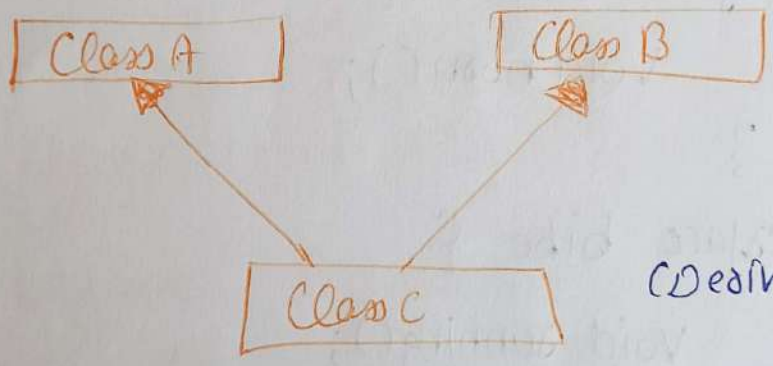
eg:
```
Class Animal {
    Void eat ()
        {
          System.out.println (" eating ");
        }
Class Dog extends Animal {
    Void bark ()
        {
          System.out.println (" barking ");
        }
Class Cat extends Animal {
    Void meow ()
        {
          System.out.println (" meowing");
        }
}
```

```
Class Testinheritance3 {
    public static void main (String args[])
    {
        Cat c = new Cat ();
        c.meow ();
        c.eat ();
```

- Multiple Inheritance → where a class inherit from more than one class.

→ Java does not support multiple inheritance through classes to avoid the complexity and ambiguity that can arise it.

→ Class A {
void msg ()
Class B
void msg {
Class C extends A, B
C obj = new C
ob.msg()



(Base Class)

(Derived Class).

→ implement same method. Call ambiguity

→ Class C inherits A and B Classes.

Sol^{n} :→ Multiple inheritance can only be archieve by Interfaces

Syntax of implementing multiple interfaces

- Class myclass implement interface1, Interface2,

```
{
    //---
}
```

Syntax:

```
interface A {
        void _C();
    }
interface B {
        void C();
    }
Class C implement B, C {
        void C() {
            //
        }
    }
```


Ex:

```
interface Vehicle {
        Void horn();
    }
interface bike {
        void running();
    }
Class Autovehicles implement Vehicle, bike {
        void horn()
        {
            System.out.println("- - -");
        }
        void running()
        {
            - - - -
        }
}
```

```
Class multipleinheritance {
public static void main (String [] args)
        {
            autovehicle C = new autovehicle ();
                C.hoon();
                C.running();
        }
}
```

→ Multiple inheritance is not supported in case of class, But it is supported in case of interface because there is no ambiguity as implementation is provided by the implementation class.

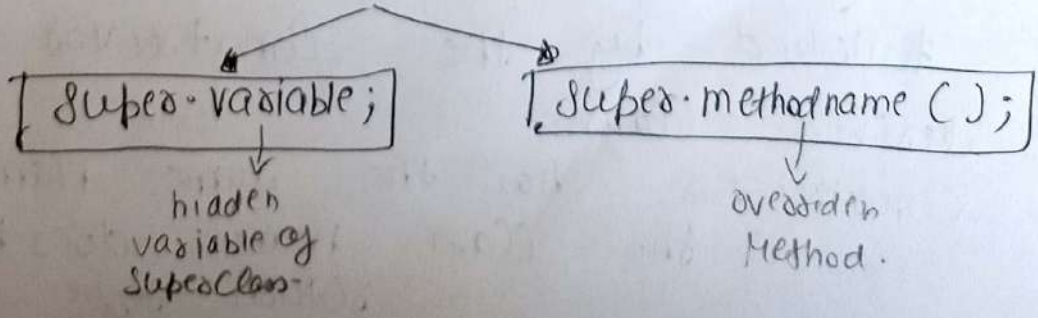
Super keyword

- Super keyword is used to access the member of the super class.

→ There are two purpose of super keyword.

|→ To call superclass constructor in the subclass.

|→ Accessing the superclass hidden member in sub class.



Ex:

```
Class Parent {                                  ← Variable
    int x = 10;
}
Class Child extends parent {
    int x = 20;
    Void display()
    {
        System.out.println(" " + " ");          // 20.
        System.out.println(" super.x);          // 10
                    → super.display()           → Method (Normal program)
    }
}
Class Test super {
    public static void (String args[])
    {
        Child obj = new Child();
        obj.display();
    }
}
```

Role of Constructor

- Constructor are used to initialize objects
- When a derived Class is extended from the base Class, the constructor of the base Class is executed first followed by the Constructor of the derived Class.
- Constructor has the same name as the Class name, does not have a return type.

a) Default Constructor

```
Class base
 {
    base ()
      {
         // default constructor
      }
 }
Class derived extends base
   {
     derived ()
       {
          // ( Derived Class Constructor Called)
       }
   }
Class Defaultc
   {
     public - - - -
     {
        derived d = new derived ();
     }
}
```

→ If the superclass define a default constructor (a constructor with no parameters), it is automatically called when an object of subclass is created.

b) SuperClass parametrized Constructor

If the superClass only define a parameterized constructor (a constructor with parameters) the subClass constructor must explicitly call one of the superClass constructor using 'super()' keyword.

eg:
```
Class base
  {
    int x;
    base(int a)
      {
        x = a;
      }
  } /* child class */
Class devide extend base
  {
    int y;
    devived (int a, int b)
      {
        /* Accessing parent class constructor */
        super(a);
        y = b;
      }
    /* Method to show value of a & b */
    void display()
      {
        System.out.println("x = "+x+", y = "+y);
      }
  }
  // Drives Code.
Class parameterizedC
  {
    - - - .
    {
      devived d = new devived(10, 20);
      d.display();
    }
}
```


→ x = 10 & y = 20

Why Constructor are not inherited in Java

- because they are not considered member of Class, They are special Method used to initialize object of a Class.

→ When a Child Class extends parent Class, it inherits State and behaviour of parent Class but it does not inherit Constructor of super Class.

→ reason :→ discussed previous

- A Constructor Cannot be called as a method.

```
i.e Child obj = new Parent();
```
↳ not allowed.

Ex:
```
Class SuperClass {
        SuperClass()
        {
          - - - -
        }
}
Class SubClass extends SuperClass
    {
        //no Constructor defined explicitly
    }
  - - -
  - - -
      SubClass obj = new SubClass();
}
```

Interface In Java

- Interface looks like Class but it is not a Class. An interface can have methods and variable just like the Class but the method declared in interface are by abstract method (no implementation, only declare the method signature).
- the variable declared automatically in an interface are public, static & final

Reason to use interface

- used to achieve fully abstraction
- By interface, support the functionality of Multiple inheritance
- It can be used to achieve loose coupling

Defining an interface

```
interface interfacename
{
    variable declaration;
    method declaration;
}
```

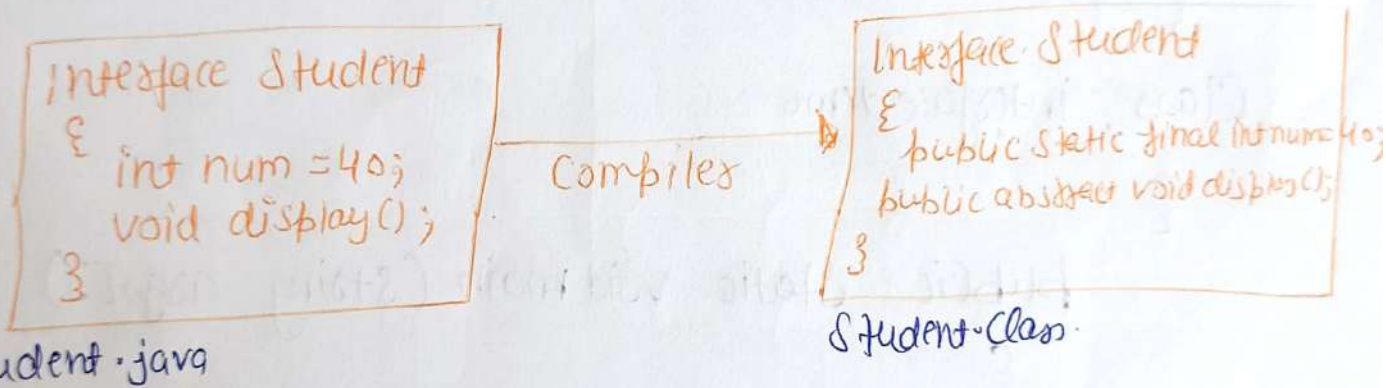
interface is keyword
interfacname → any valid java identifier.

Eg:
```
Interface Student
{
  int num = 40;
  void display();
}
```

Internal addition by Compiler

- Java compiler add public and abstract keywords before the interface method.

→



→ Interface fields : public, static, final by default

→ method : public and Abstract

Implementing Interface

→ by using implements keyword to implement an interface.

Syntax:
```
Class Classname implement interfacename
{
  // ...
}
```

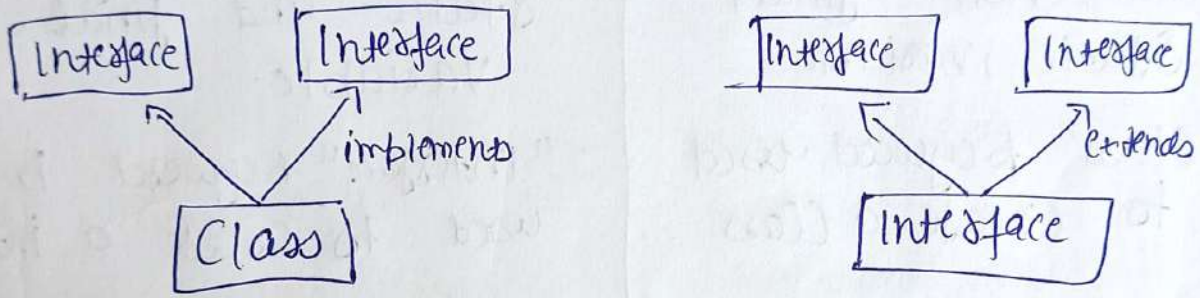
eg:
```
interface message
{
  void display();
}
Class sample implement message
{
  public void display()
  {
    System.out.println("Implementing interface");
  }
}
Class interfacedemo
{
  public static void main (String args[])
  {
    Sample obj = new sample ();
    obj.display ();
  }
}
```


It also use Extend keyword in Interface.

```
interface Child interface Extends parent interface
{
  - -
  - -
}
```

Multiple inheritance in Java

- Java Does not support multiple inheritance, But a large number of real-life application require the use of multiple inheritance.
- Java provide an alternate approach known as interface to support the concept of Multiple inheritance.



Marker or tagged Interface

- An interface that have no member.
- used to provide info to JVM. that JVM may perform some useful operation.

Nested interface

- An Interface can declared within another interface.

→ interface f
{
}
interface s.
{
}.

Class nested implem f.s

Class

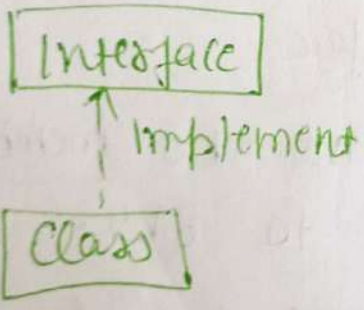
- Class can have abstract and non-abstract Methods
- doesn't support Multiple inheritace
- a Class implemented the interface
- Class can have final, non-final, Static, variable
- "Class" keyword used to create a Class

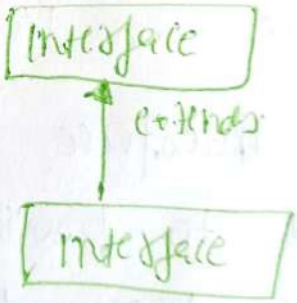
Contain constructor

Interface

- Interface can have only abstract method
- Support Multiple inheritance
- not implement Class
- Interface has only Static and final variable.
- "Interface" keyword is used to create a interface

Not





Interface

- "Interface" keyword is used to declare a interface
- Support Multiple inheritance
- can't implement Class
- Slow

Abstract Class

- "Abstract" keyword.
- Not
- Can implement the interface
- fast.

Abstract Class

- Interface only have abstract method
- only have static & final variable
- An abstract Class can be extended using "extends" keyword

eg: public interface Draw {
    void draw();
}

Interface

→ abstract & non-abstract Method.

: Can have final, non, final, static, non static variable.

- using "implement" keyword.

eg: abstract Class Shape
{
    abstract void draw();
}

// Creating interface of 3 Method

eg: interface one
{
    void m1();
    void m2();
    void m3();
}

// Creat abstract Class

Abstract Class two implement one
{
    void m3()
    {
        -- ("I m Method 3");
    }
}

// subclass of abst

~~Crea~~ Class derived extend two
{
    public void m1()
    {
        —— ("Method1");
    }
    —— ("Method22");

Class Test
{
    one obj = new derived();
    obj.m1();
}

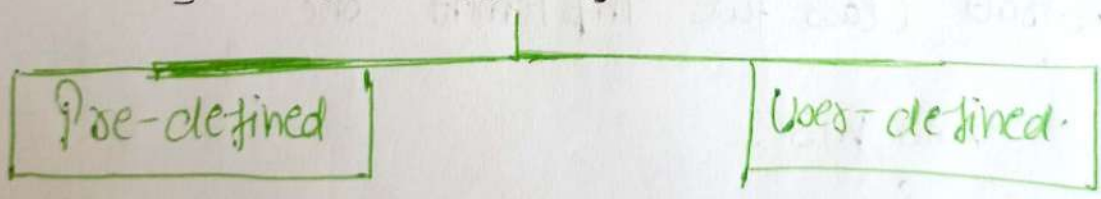
Package

- Java environment provide a powerful means of grouping related Classes and interface together in a single unit called packages.
- A java package is a group of similar type of Classes / interface and sub-packages.

⇒ Advantages

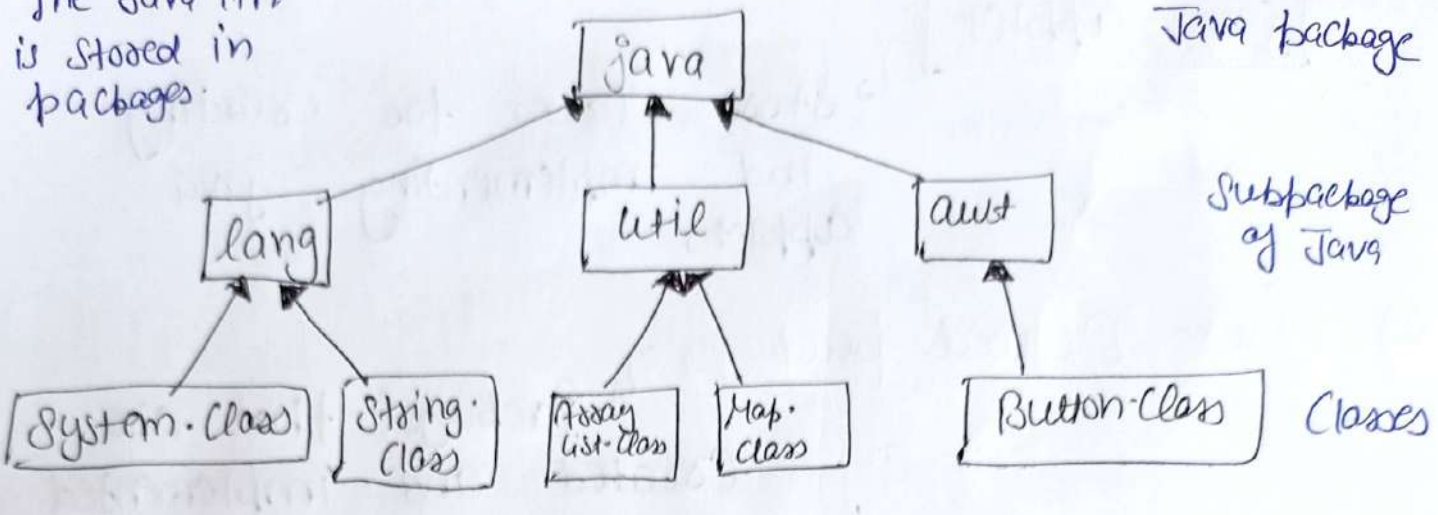
1) remove naming collision
2) provide a access protection.
3) used to categorize the classes and interface so that they can be easily maintained.

⇒ Type of package



1) Pre-defined packages :- These package Consist of a large number of Classes which are a part of JAVA API.

The Java API is stored in packages.



Java package

Subpackage of Java

Syntax:

= import packagename.*;

eg: import java.util.*;

- Most commonly used packages and their Classes.

→ java.lang :→ Store a large no. of general purpose Classes
- such as System Class, String Class, Math Class.

→ java.io :→ Store the I/O Class
- Data input Stream Class., file input Stream

→ java.util :→ - Store language Utility Classes such as vector, date/Time.
→ Scanner Class., 'HashMap'.

→ java.awt :→ It contain Classes for implementing graphical User interface. → frame button.

- java.applet → Store Classes for creating and implementing java applets.

2) User Defined packages : → Package that are created and implemented by the users.

1) Create a package → `package packagename;`

2) Create the directory of package name.
3) define the Class of package and declare all clases public.
4) Store each file of Class as Classname.java
5) Compile the file. This create .Class file in package directory.

eg :-

```
package mypackage;
    public Class myClass
        {
            public void name (Strings)
                {
                    System.out.println(s);
                }
        }.
```

→ import package.*;
→ import package.Classname;

} access public package member from outside the package.

Naming Conventions

- package are written in lowercase letters
- Meaningful name.
- Reverse domain name to avoid naming conflicts.
- They should be period-delimated.

Defining a Package

- package keyword is used to define a package in java.

```
package packagename;
```

↓ keyword — ↓ Name of package.

Adding a Class to package

```
→ package packagename;
  public Class Classname
  {
  - -
  }
```

Step1: Code

```
package mypack;
public class Sample
  {
    public Static void main (String arg[])
    {
     System.out.println(" Welcome");
    }
}
```

Step2: Save this file as Sample.java

Step3: Compile java package using following Syntax

```
Javac -d directory java-filename
```

eg: Javac -d . Sample.java

[If we want to keep the package within the same directory use (.)]

Step4: Run java Package program.

→ use fully qualified name

eg: mypack.Simple to run the Class

· Command to run java package

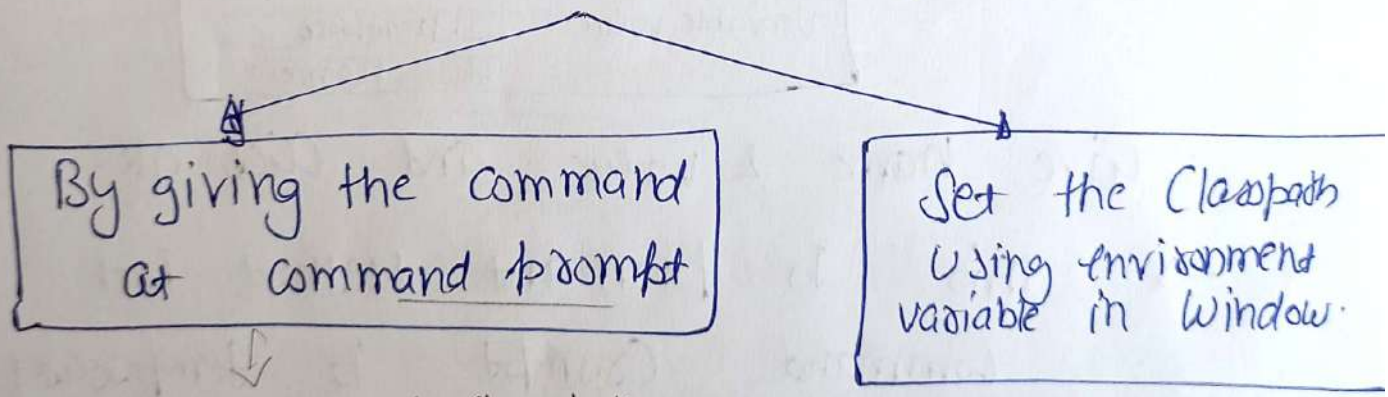
```
java mypack.Sample
```

output: Welcome

CLASSPATH

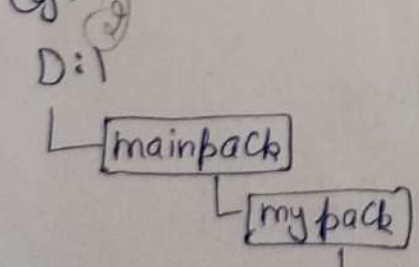
- By default the Java run-time system search the Class file at current location, But if we talk about packages, it must not be stored at the same location.
- Therefore we need to have provide the path, where package are stored
  ↳ and class path.
- The CLASSPATH is a user defined environment variable that is used by Java to determine where pre-defined Classes are located.
- also used by JVM to load Classes.

→ Classpath can be set of two ways



- Set ClassPATH = <packagepath>

eg:



→ mypack package is created in d:\mainpack. and store the Class file of sample.java

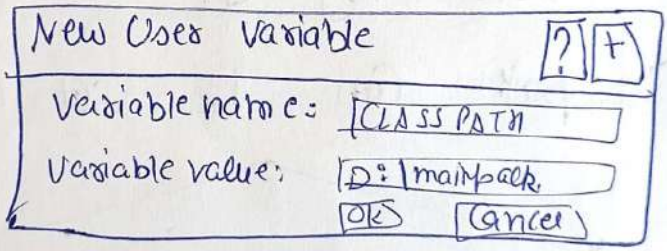
→ Directory structure for this package.

[C:\> set CLASSPATH = .; D:\mainpack;]

- Then Classpath need to set location of ③ package.

2nd Set

- Right Click on My computer icon
- Select property ; System properties dialog bot appears
- Click on Advanced Tab ; Click on Environment variable Button.
  ↳ Dialog bot appear
- Click on new button in user variable for owner area. New User Variable dialog bot appear



- Give name & value and Click OK.

→ This is permant Method but Command Compt is temporary.

Subpackage

→ Package inside the package.

→ package mainpack.subpack;

Package

1) `import package.*;`

↓

used to bring Classes, interface from other package into current class file package.

→ If use package.* then all the Classes and interfaces of this package will be accessible but not subpackage.

2) `packagename.Classname`

→ If import package.Classname then only declared Class of this package will be accessible.

3) `full qualified name`

→ use fully qualified name then only declared class of this package will be accessible. No need is need to import.