# Java

C → 1972 → AT&T bell laboratries → Denies Richi

C++ (CPP) or (C with classes → 1980 → do → Strone Stroub

Java → 1995 → Java β version (user side testing version)

1996 Jan 23rd → Java 1.0 version

→ Sun micro systems → James Osleng

Java symbol → coffee glass or mug

~~Reason~~

Languages

| Function oriented | object oriented. | Both Function, Object oriented |
|---|---|---|
| Ex:- C | Ex:- Java | Ex:- C++. |

Reason:- Introduction of internet in 1990.

For this a language is required. and C, C++ doesnt support internet perfectly. as they are system based languages.

System based languages:- If an application runs properly only on the system in which it was created (or) system with same configurations then it is called as system based language

Ex:- C, C++

For this a new language was developed which is called as Java.

Why CPP was developed ?

In C the major disadvantage was data security. That is If there is a ~~fu~~program with 10 functions to access a variable in multiple functions we will declare it as global variable. But if we want it to be accessed in some particular functions only it fails. To overcome this a concept called classes was introduced which is known as C++ (or) that is C with

Architecture neutral language = JAVA.

1991 → OAK
1993 → OAK (Basic version)
1995 → JAVA → (From coffee)

Introduction :- Java is related to c++, which is a direct descendent of C. Most of the characteristics of java is inherited from these 2 languages.

From C, Java derives its syntax. Many Java object oriented features are influenced by C++.

History :- Initially commeted to develop Java in 1991 it took 18 months to develop the first working version.

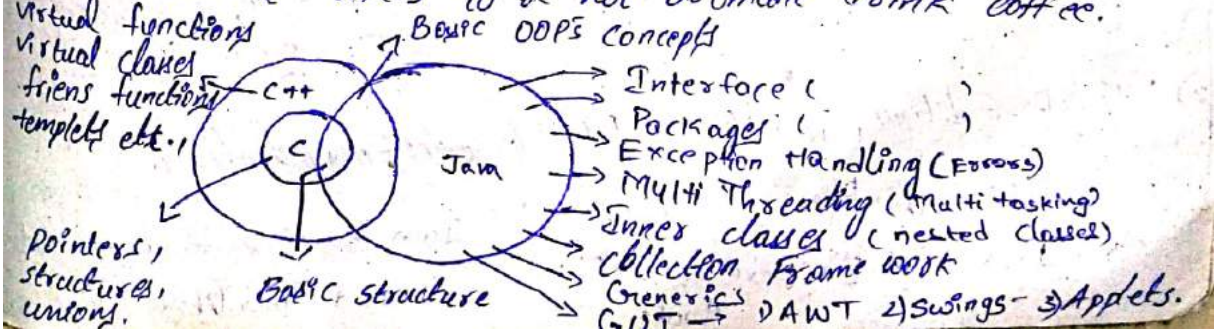
Java was conceived by James Gosling, Patric Naughton, Chris warth, Ed Frank, and Mike sheridan at sun microsystems.

* It is initially called as OAK but was renamed as "Java" in 1995

Why OAK is replaced with "Java"?

"OAK" was just a working project name used, supposedly named after an oak tree out side James Gosling's office window. When it was officially released, they didn't want to keep the current working project name and choose to switch it to "Java".

The name Java ~~doesn't~~ specifically doesn't have meaning. Rather than it refers to a hot aromatic drink coffee.



Java 1.1 19/2/97

1) Process - Thread based multi
2) Process based threading

Exception handeling:-

Exception:-

---

Versions in Java

Java 1.0 :- Basic version or first released version.

1) It was released on 23rd Jan 1996.
2) It has lots of bugs. Aplets are big issue

Java 1.1 :- It was released on Feb 2nd 1997. Major addition included an:

1) Extensive retooling of AWT event model
2) Inner classes added to the language.
3) Java beans
4) JDBC

Java 2 (1.2 to 1.4)

from this version onwards Java come in 3 flavours.

1) Standard edition(J2SE): Used for normal purpose applications
   s/w :- JDK. (Java Development Kit)
2) Enterprise edition(J2EE): Business edition
   s/w :- server service providers (Tomcat, websphere, glass fish)
3) Micro Edition (J2ME) :- Mobile editionapplications.
   s/w :- wireless tool kit

JDK = JRE + Development tools

JRE → Java Runtime environment = JVM + Java libs

Java 5 (version 1.5):-

It was released on sept 30 2004

Added major changes to language itself, making it easie

30% of Java is changed 1.5 version then compare to 1.4 version. new features in Java 1.5 V.

1) for-each loop or enhanced for loop.
2) Var-Argement method
3) Queue
4) Generics
5) enum
6) Auto Boxing and AutounBoxing
7) co-varient return type
8) Annotations.
Imp 9) static import.
10) String Builder.

Java 1.6 (Java 6):-

It was released on December 11 2006. from 2007 onwards Java is takeoves by ORACLE co.

Java 1.7 (Java 7):-

It was released on July 28 2011 released by ORACLE organiszation. Features:-

1) Updations for integeral literal.
2) Extension for exception handling
3) Diamon feature (extension for generics)
4) Update DATE API.

JDK 1.8 (Java 8) Mar 18 2014,

Features

1) Lambda expression
2) Functional programming
3) Extension for interfaces.
4) Streaming
5) Updated Date API.

JDK 9 (Java 9):- sept 21st 2017.

features:- 1) Jshell 2) Extension for interfaces.

remove the Applets Concept.

JDK 10 (Java 10):- Marge Mar 20th 2018

extension for Jshell

JDK 11 (Java 11) Sept 25th 2018.

Java 11 onwards java is paid version.

Open JDK -- Free.

Oracle JDK -- paid.

Java 12 (Java 12) Mar 23rd 2019

Java Buzz words or features of Java.

Sun

Malloc return type is void pointer → generic pointer:- has a capability to store address of any data type.

Classes declaration:-

```
class employee
{
 int eno;
 char ename[80];
 float sal;
 void readdata()
 {
 }
 void calculation()
 {
 }
 void display()
 {
 }
}
```

In Java classes are similar to structures in C. Here all the functions are written in the class itself.

Memory is allocated to ~~eeeploclass~~ through the syntax

classname space variable → key word = new classname(); // to allocate

These methods can be accessed through . syntax

variable.~~fun~~method();

function.

methods are nothing but functions but used in languages that doesn't involve structures.

```
employee e= new employee();
e.readdata();
```

→ object creation.

In c++ semicolon is required at end of class but in Java there is no such requirment.

Java Buzzwords or features of Java

simple:-

Java is a simple programming language. Because of 2 reasons

a) Sun people maintained the same syntax of C and C++ in Java

2) Pointers are eliminated in Java.

a. Why pointers are elemenated in Java?

1) Pointers can crash a program easily.

2) Using pointers, it is possible to develop harmful programs like virus & hacking programs. So pointers are security threat for Java.

Because of this reason pointers are eliminated from Java.

Networking:- connecting systems
robust:- strong language in which app doesn't collapse
JVM:- Java Virtual Machine
Default exception handler -> exception handling
class loader subsystem -> dynamic memory allocation
Garbage collector -> memory deallocation
Execution Engine. -> causes system independence.

2) object oriented

3) Networking:- With this concept one can connect system.

4) Robust:- A language is said to be robust if it is strong and prevents the collapse of apps due to errors.

Error <- Compilation / syntax
Run time
Logical.

| | Java |
|---|---|
| Run time errors | -> Errors |
| Logicalt errors | -> Exception |

Java employees an assitaut to deal with exceptions Called. Default exception handler.

Java mainly uses 4 assistants to handle errors.

1) Default exception handler :- exception handling.
2) Class loader subsystem :- dyamic memory allocation.
3) Garbage collection :- memory deallocation.
4) Execution engine :- system independence.

These 4 assistants work under JVM:-

4) Execution Enginee:- Gives system independence to a application. (.obj .exc)

why system dependence occurs ? :- .obj file { without header files. (code)
.exc file { with header files. i.e machine understa

file..java → Source Code → Compiler → file.class generates byte code → Execution Enginee →

Interpreter + JIT compiler

JIT :- Just in time

-> Provide memory -> execute instruction → Generate output.

Portable How Java turns system independent?

Here 2 files are created 1) .java file 2) .class file. This .class file can be executed on any java run time environment. Hence .class file concept turns it system independent.

interpreter:- Runs a file line by line

Compiler :- Runs a file entire at a time

2) Object - oriented :- Java programs use objects and class

~~Java~~ object :- An object is anything that exists physically in this An object will have properties (variables) and actions (methods

~~It represent~~ Variables :- It represents memory location to store dat

Class :- It is a group name that specifies properties and actions of objects.

1) class also contains variables & methods
2) class is only specification but object exist physically.
3) class is a model or blue print for creating objects.

3) Distributed :- Information is distributed on various Computers on a network.

4) Robust :- Robust means strong. Java programs are strong and they won't crash as easily as C and C++

There are 2 reasons for this.

1) Java has good excelent exception handeling featusc.

2) Memory managment featuse.

Here memory allocation is done by JVMs class loader subsystem.

3) Memory deallocation is done by garbage collector.

5) Secure :- Security problems are eliminated by using java on internet.

6) System independence or archetectiual neutrality :-

Java byte code is not machine dependent. It can be run on any machine on any processor. we can achive system independence by using Java byte codes.

7) Portable :- If a machine yeilds same result on ~~any~~ every machine. Then that program is called portable. This can be achived by javas system independence nature.

8) Interpreted :- In any language only either interpreter or Compiler is used to execute the code. But in java we can use both compiler and interpreter for the execution. (contd.)

Interpreter can understand byte code and converts it into machine understandble code.

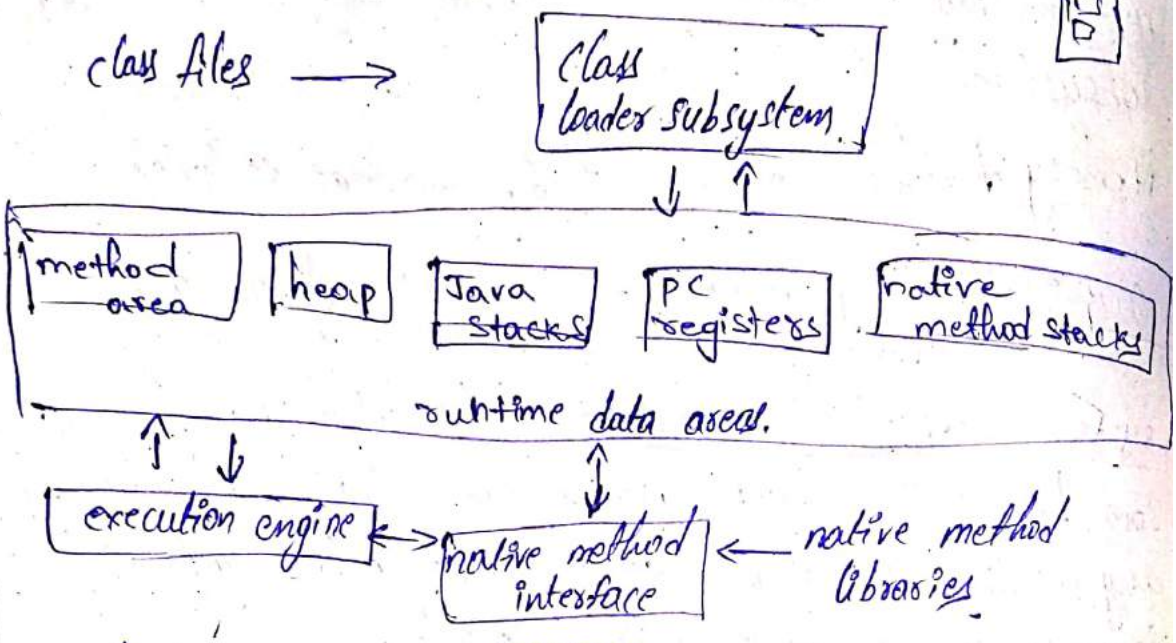
9) Multi threading :- 1) Process based 2) Thread based. | Executing statments are called threads. Each thread can perform number of statments. This is essentia program to server side programs.

Thread is nothing but a light weight processor or a sub individual part of program.

10) High performance :- The problem with the interpreter inside the JVM is that it is slow. Because of this, java programs used ~~to~~ to run slow. To overcome this problem along with the interpreter sun people

introduced JIT (Just In Time) compiler, which can enhance the speed of execution.

JVM architecture :-




method area :- here ~~objects~~ memory is allocated for variables in class

heap :- For all objects memory is allocated in heap area

PC register (program count register) :-
Stores address of next executable instruction

native method stack :- It is used to speed up a process

Hot spot :- Spot where JIT is performing functions

[Documentation section.] → optional.
[Package statment]
import statments.
class or interface or enum

} Global variables are absent in Java.

Documentation section :- Non executable statments that are readble. These are placed in comments.

Java Comments :- Multi line, single line, API Comment.

1) Multi line :- /* ---- */ } → Taken from c
2) Single line :- // } → Taken from cpp
3) API Comment :- /** ---- */

This is developed in Java. These are used for API documentation and are present in .html format.

API :- Bunch of programs used to develop a software

Package statment :- It is a directory which has a set of classes or interfaces that perform similar operation.

Packages — Default packages
Packages — User defined packages

Java library

Standard library (Java)

1) lang 2) io 3) util
4) sql 5) net 6) awt.
7) applet

Extended library (Javax)

1) sql 2) swing
3) servlet.

lang :- Contains basic required classes.
io :- Contains classes related to input & output.
util :- Contains classes to control system utilities.
sql :- Contains classes to perform JDBC operations
net :- Contains classes to provide communication b/w systems in a network
awt (abstract window tool kit) :- One of the mechanism to develop GUI. It contains classes related to GUI
applet :- Contains classes ~~to provide~~ related to applets.

sql :- similar to sql in standard library. but for enterpreneure purpose.
swing :- Extension of awt. It helps in development of paint, calculator etc.
servlet :- It is a program that works on server side.

○ -> not important i.e not used frequently.

User defined classes:- It can be is nothing but creating our own packages to store user generated classes

We can create our own packages with the help of package statment.

Syntax :- package packagename; → key word

↓

package . package name ;

In a program we can write almost single (1) package statment.

Package statment is the 1st executable statment in a program.

Import:- ll to # include.

It is a statment which provides the ability to access classes present in other packages. like current package elements.

Syntax :- 1) default packages

import library . packagename . class name ;

2) User defined packages

import package name . classname ;

Class syntax :-

```
class class name
{
    variables;
    methods;
    blocks;
}
```

Called as members

ll to structure

variables :-

methods :- functions

blocks :- seperate statments

members

Instance / object members.

Static / class members. → Key coord.

```
Class Test
{
    int x ; → Instance variable.
    static int x ; → static variables
}
```

How to access variables/members in classes?

It can be done in 2 ways

1) with the help of object.

object creation:- Allocating memory for all variables in a class

Syntax:- new classname ();

For accessing them we need a pointer

∴ (class name reference variable = new classname();) ← object creation

accessing:- refvar. variables; | refvar. methods;

2) with the help of class name.

Syntax:- Class name. variable;

class name. method;

with the second method we can access only static variables, methods.

Main method

In c ! 1) void main() { } | 2) int main() { return 0; } | 3) main() { return 0; }

But these 3 doesn't work in Java.

There is another main syntax.

void main (int argc, char *argv[])

argc → argument count → no. of variables +(1) → file name.

argv → String[]

Can be used from Command line.

String → char ch[]; or → char *ch; | String → Char ch[][]; (or) → char *ch[];

String → String constraint in java

import. java. lang. string; → import. java.lang. system;

Class First {

access permission → public static void main(String args[])

{ System. out. println ("Hello world"); }

There are 4 types for access permission

1) Default 2) Public 3) Private 4) Protected.

free from syntax.

Default :- It can be accessed with in package

Public :- It can be accessed where ever.

Private :- It can be accessed with in the class.

Protected :- It can be accessed with in directory

System.out.println("Hello world");

System → class located in lang

out → static ref variable for printstream class in java io package

println → Method in PS.

println();
print();
printf(); → not frequently

To develop & execute the applications must contain JDK & JRE after the installation the java tools has to be available at command prompt. For that purpose we have to set path.

We can set the path in 2 different ways.

1) Through command prompt 2) As environment variable

1) Set Command

Key word: Set path=%path%;.;C:\Program files\Java\JDK1.8.0_2

This path will be lost whenever command path is collapsed

Close all command prompts

My computer → properties → System Advance option → environment variable → click new button on user variable → provide variable name as path, → value.

C:\Program files\Java\JDK1.8.0_211\bin;

```
/*This is my first program in JAVA. At BDPS. */
// import the system and string classes from java.lang package.
/** This is my first class in java. language.
    By Tulasi */
Public class first
{
/** our program execution starts from main().
    And it is called by JVM at run time.
    Main() having Arguments as string array type.
*/
  public static void main(String [] args)
  {
  System.out.println("Hello world");
  }
}
class last
{
}
```

To generate the API level documentation. Use Java compiler

1) Syntax :- Javadoc filename.java.
2) Syntax Javadoc Java doc -d drive name\file name.java
2) -> To save it at desired location

bin

Ex:- Java doc -d .\api First java

↓ current running directory.

Identifiers :- Name given to classes, packages, variables.

1) A-Z, a-z, 0-9, _, $ → added in java } These characters can only be used to generate identifiers
2) It should not start with numericals.
3) No limit for identifier length. Recomended (15-16)

lower case alphabets.

1) Ex:- java.lang ; java.util ; java.awt.

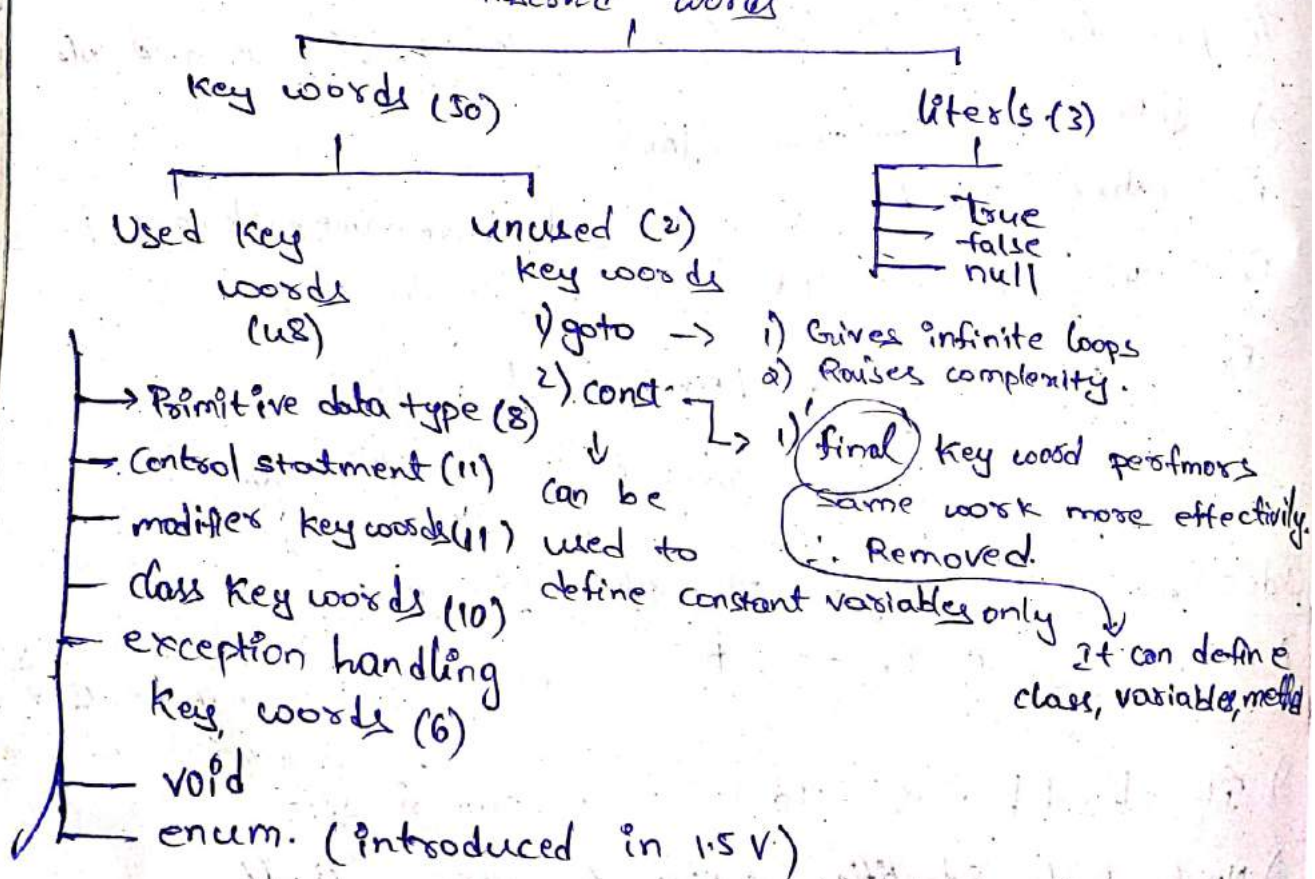
2) Class name (or) interface (or) enum names :- Each and every word will have its first letter capital in the name.
Ex:- String, System, StringBuffer, InputStreamReader, PrintStream

3) Method and variable names:- From second word onwards each and every word first letter will be in upper case
Ex:- getData(), println(), read(), readLine(), studName().

4) Constant variables:- Every character must be upper case. We can seperate words by using underscore (_).
Ex:- PI, MAX_VALUE, MIN_VALUE

5) Reserved words :- All characters are lower case alphabets
Ex:- class, void, static.

No of reserve words = 53.



Primitive data types (8) :- 1) byte 2) short 3) int 4) float 5) double 6) char 7) long

Control Keywords (11) :- 1) if 2) else 3) switch 4) case 5) default 6) break 7) continue 8) while 9) do 10) for 11) assert

modifier key words (11) :- 1) Private 2) Protected 3) Public 4) static 5) strictfp 6) final 7) synchronised 8) native 9) abstract 10) transient 11) volatile

Except these no other modifier can be used on local variables.

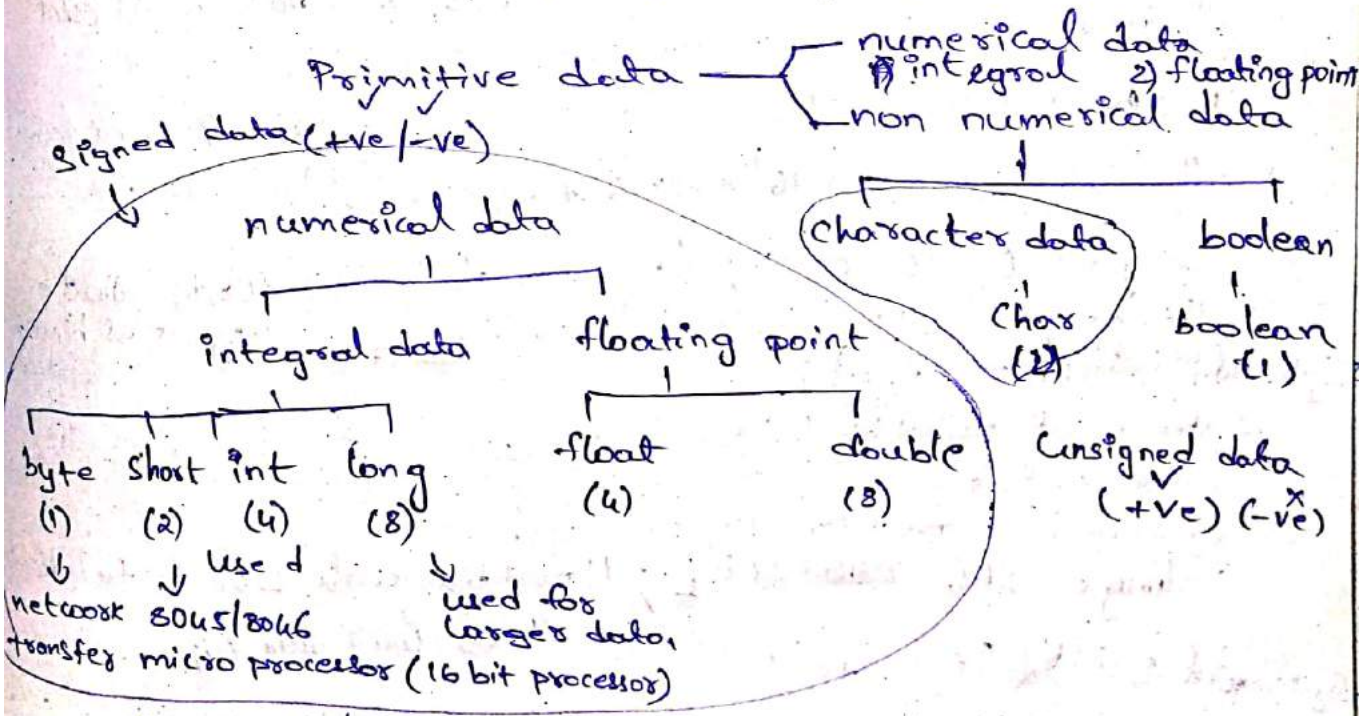
Class key words (10) :- 1) package 2) import 3) class 4) interface 5) extends 6) implements 7) new 8) this 9) super 10) instance of

Exception key words (6) :- 1) try 2) catch 3) throw 4) throws 5) finally 6) return

Datatypes

- Primitive data types
  1) int 2) char 3) float 4) double
- User defined data types
  1) class 2) interface 3) enum.

1) Primitive data types:- Data types that are directly accepts which are provided by user.




Signed and unsigned int concept is absent in java

Formula to find minimum & maximum value for numerical values

n- no of bites  minimum = $-2^{n-1}$

maximum = $2^{n-1}-1$

Uni code :- similar to ASCII (www.unicode.org)

Literal :- Variable assigned to store data.

Literals are of 5 types.

1) Integral literal 2) floating point literals 3) character literals
4) boolean literals 5) string literals

Integral literals :- Int is the default data type for integral data

Note :- Java.lang is the default package for every program in Java. Hence there is no need to import the classes which are present in lang package

Class integral literals.

```
{
  public static void main(String [] args)
  {
    byte a = 124; // Decimal number system
    short b = 0127; // prefix with zero (0) to indicate it as octal
    // Conversion :- 127 => 1x8^2 + 2x8^1 + 7x8^0 = 87
    int e = 0x14; // prefix with zero x => (0x) to indicate as hexa (0x)
    // Conversion :- 1x16^2 + 10x16^1 + 4 = 420.
    long d = 0b 10110011; // prefix with zero b => (0b) to indicate
    // conversion :- 1x2^7 + 1x2^5 + 1x2^4 + 2+1 = 179 (0B). it as binary
System.out.println("A = " + a);
    System.out.println("B = " + b);
    System.out.println("C = " + c);
    System.out.println("D = " + d);
    long e = 866-666222L; // postfix with L or l to indicate
                          it as long data type
System.out.println("E = " + e);
    // from 1.7 onwards _ is also accepted in b/w numbers. ;;
```

## Floating point literals:-

Double is the default data type for floating point data.

```
class floating literals
{
  public static void main (String [] args)
  {
  float a= 122.55 f; // postfix f or F to indicate it as fla
  double b= 104.97 ; // In can accept because *
  double c = 546.954 d; It will also accept d or D postfix
  double d = 0122.45 ;
  // float and double can accept any default integral val
  double e = 0x106 ;
  System. out. println( "A="+a);
  System. out. println ( "B = "+b);
  System. out. println ("C = "+c);
  System. out. println ("D= "+d);
  System. out. println ("E = "+e);
  }
}
```

o/p:- A = 122.55
B = 104.97
C = 546.954
D = 122.45
E = 422

ASCII codes.
Tab - 9
backspack-8
Enter - 13

68 :-
96+8
104

a - 97
b - 98
c - 99
d - 100
e - 101
f - 102
g - 103

## Class literals

```
class Literals
{
public static void main (String [] args)
  {
  char a= 'a' ;
  char b= 79; // ASCII 79= N.
  char c= '\U0068' // Unicode representation.
System. Out. println("A="+a);
System. out. println ("B="+b);
```

Syntax :- '\u ____ 4 digit hexa code

```
System.out.println("C = "+C);
boolean d = false;
System.out.println("D = "+d);
String s = "4456";
String s1 = "45";
String s2 = "a";
String s3 = "string";
System.out.println("S = "+ s);
System.out.println("S1 = "+s1);
System.out.println("S2 = "+s2);
System.out.println("S3 = "+s3);
}
}
```

o/p :- A = a
B = kN
C = 4
D = false

S = 4456
S1 = 45
S2 = a
S3 = string.

Variable :-

Based on the stored data variables can be classified into 2 types 1) primitive variables :- The variables used to store primitive data
2) Reference variables :- The variables used to store objects.

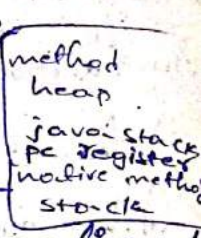
Based on area of declaration and memory location variables can be classified into 3 types.

1) Local variables 2) Instance variables 3) static variables.

Ex :-
```
Class test
{
    int x;          => instance variable
    static int y;   => static variable
    void fun(int w)
    { int z; }      -> local variables.
}
```

Local variables :- 1) Used in particular method or block 2) Use :- swaping etc 3) Time of generation :- when particular method called 4) Place of memory :- Java stack

Time of deallocation :- when particular method / block execution is ~~called~~ completed.




2) If you want to represent data for particular operation only then we should go for local variables

3) Memory allocation is done at the time of method calling as the part of method entry.


4) Memory de allocation is done when ever method execution is compleated.

∴ Scope of local variables is same as method.

5) Local variable (or) Temperory variable (or) method level variables

Imp :- Default values concept is not appliable for local variables. If not assigned compile time error appears. Hence we have to provide initialisation before accessing them.

Except final modifier no other modifiers can be used on local variables

```
class Local variables
{
  public static void main(String[]arg)
  {
    int x=0, b=10;
    final int y;
    if(b==10)
      x=111;
    x=111;
    y=666;
    System.out.println("x= "+x+"\t y= "+y);
    x=555;
    System.out.println("x= "+x);
  }
}
```

Declaration of final variable can be done only once.

// Assigning values to variables in condition statments in not possible unless they have been assigned with default values.

o/p :- x=111 y=666

## Default values

Primitive Variables :-

1) Integral (byte, int, short, long) = 0
2) floating (float, double) = 0.0
3) Character (char) = \u0000
4) boolean = false

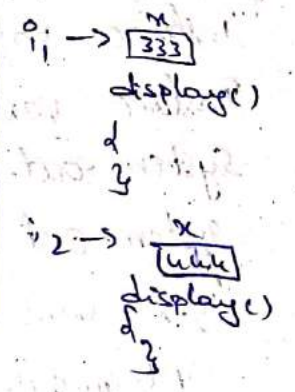
Reference variables = null.

Instance variables :- 1) The variables which are declared inside the class but outside all the methods and blocks are called instance variables.
2) If the value of the variable varies from object to object then we should go for instance variables.
3) For instance variables memory allocation can be done at the time of object creation with in the heap area as the part of
4) Memory can be deallocated when ever object is destroyed. hence the scope of instance variables is same as object.
5) This variables also called as atributes or object level variables
6) There is no need to perform the initialization for instance variables because JVM will provid the default values for
7) We can access the instance variables directly from instance area of the same class.
8) But we are not allowed to instance variables directly from static area
   a) If we want to access it from static area with the help of object of that class.

```
class Instance variables
{
int x;
void display()
{
System.out.println("x = " +x);
}
public static void main (String[] args)
{
Instance variables i1 = new Instance variables();
```

Instance variables = new Instance variables();

```
System.out.println("object 1 x = " + i1.x);
System.out.println("object 2 x = " + i2.x);
i2.x = 445;
System.out.println("object 1 x = " + i1.x);
System.out.println("object 2 x = " + i2.x);
i1.x = 333;
System.out.println("object 1");
i1.display();
i2.x = 445;
System.out.println("object ");
i2.display();
}
}
```




Static variables:-

Object 1) The variables which are declared with static modifiers outside the method or block are static variables.

2) If the value of variable cannot varied from object to object then we should go for static variables.

3) For static members memory allocation can be done at the time of class loading with in method area.

4) Memory can be deallocated whenever class is unloaded hence the scope of static variables exactly same as of class.

5) Static variables are also called as fields or class level variables.

6) For static variables there is no need to perform initialization because JVM provide the default values.

7) we can access the static members both instance & static area of the same class

8) we can access the static members of other class either with the help of object or class name but recomended to access class name.

1) Class loading → static
2) object creation → Instance
3) Method ... → local

class instance variables

```
{
int x;
void display()
{System.out.println("x="+x);
}
public static void main(String[]args)
{
Instance variables i1=new instance variables();
Instance variables i2=new instance variables();
System.out.println("object 1 x="+i1.x);
System.out.println("object 2 x="+i2.x);
i2.x=444;
System.out.println("object 1 x="+i1.x);
System.out.println("object 2 x="+i2.x);
i1.x=333;
System.out.println("object 1");
i1.display();
i2.x=444;
System.out.println("Object 2");
i2.display();
}
}
```

Internal execution flow :-

1) Start JVM
2) JVM creates the thread with the name "Main Thread".
3) Attach the .class file to the main thread
4) Perform the class loading.
   a) Identify and provide memory allocation for all the static mem (methods, blocks, variables) from top to bottom and provide defa values to static variables.
   b) Provide initialization to static variables and execute static blocks from top to bottom.
5) execute the main
6) Perform the class unloading

7) Destroy the main thread
8) Shutdown the JVM

class internal Flow.

```
{static int x;
 public static void main (String [] args);
 {
  System.out.println(" main() started");
  display();
  System.out.println(" main() ended ");
 }
 static
 {
  system.out.println(" static block 1");
  display();   // methods called in static block/method
 }             should must also be static.
 static void display()
 {
  system.out.println( "x = "+x+ "\t Y = "+y);
 }
 static int y=333;
 static
 {
  system.out.println("static block 2");
    display();
    int x=888;
  }
}
```

o/p:- static block 1
x=0 Y=0
static block 2
x=0 Y=333
Main() started
x=888 Y=333
Main() ended.

## Command line arguments

The values which are passed to the main method as arguments are called command line arguments.
All these arguments are stored into Args variable of String array type in the main method.

Note :- In java array is considered as a class. In array class we have variable with the name of "length" which is used to find the number of elements available in array

```
class Command demo
{
  Public static void main (String [] args)
  {
    int n = args.length;
    System.out.println ("No of arguments = " + n);
    if (n == 2)
    {
      System.out.println (" Argument 1 = " + arg[0]);
      System.out.println ("Argument 2 = " + arg[1]);
    }
    else
    System.out.println ("Provide 2 arguments");
  }
}
```

Write a program to perform addition of 2 integer values by reading the values from command prompt.

```
class add_sub
{
  public static void main (String [] args)
  {
    int n = args.length;
    System.out.println("No of arguments = " + n);
    if (n == 2)
    System.out.println (" Addition = " + arg[0]+arg[1]);
    else
    System.out.println ("Provide 2 arguments");
  }
}
```

```
class Additiondemo
{
public static void main (String[] args)
{
if (args.length==2)
{
int a = Integer.parseInt(args[0]);
int b = Integer.parseInt(arg[1]);
System.out.println("sum = " + (a+b));
}
else
System.out.println(" Provide 2 arguments");
}
}
```

**Wrapper class** :- The classes which are used to convert primitive data into object format are called wrapper classes. For every primitive data type 1 associated raper type is present.

| Primitive data type | wrapper class | To convert string into primitive data type. |
|---|---|---|
| byte | Byte | parseByte (string s) |
| short | Short | parseShort (string s) |
| int | Int | parseInt (string s) |
| long | Long | parseLong (string s) |
| float | Float | parseFloat (string s) |
| double | Double | parseDouble (string s) |
| boolean | Boolean | parseBoolean (string s) |
| char | Character. | — |

**Syntax for accessing**:- Byte.parseByte (~~string~~ ) ← argument name

**Operators in Java** :-

1) Arthematic operator
2) Assignment operator
3) Relational operator
4) Logical operator
5) Increment/dec operator
6) Conditional(operator). (ternary operator)
7) Bitwise operator
8) new operator
9) Equality operator
10) instance of operator

11) Types casting operator  12) [] operator (array operator)

1) Arthematic operator :- +, -, *, /, %.

2) Assignment operator

a) simple assignment
```
int a, b=44;
a=33;
```

b) Multiple assignment operator
```
int a,b,c;
a=b=c=45;
```

c) Compound assignment
+=, -=, *=, /=, %=
(short hand assignments)

3) Relational operator :- <, >, <=, >=

4) Logical operator :- Logical And (&&), Logical OR(||), Logical(!) Not

Short circuit operators :- Here we can skip by simply checking single condition

5) Increment / Decrement operator :-

```
Class Incre Decre
{
public static void main (String[] args)
{
int a=45, b,c
b = ++a; // b=46, a=46
c = a--; // c=46, a=45
System.out.println("A= "+a+"\tB= "+b+"\tC= "+c);
b= --a; // b=44 a=44
c = a++; // c=44 b=45
System.out.println("A= "+a+"\tB= "+b+"\tC= "+c);
}
}
```

~~A=45; B=46 C=~~

~~A=45 B=44 C=44~~

operators
- unitary
- binary
- ternary

(Unitary operators will have symbol on left
exception :- increment/decrement)

+=, -=, *=, ++i, i++

o/p :- A= 45 B= 46 C=46
A= 45 B= 44 C= 44

6) Terniary Operator :-

Syntax in C :- expression 1 ? exp-2 : exp-3;

Ex :- int a = 10
int b = a? 45:65;

Syntax in java :- condition ? exp-1 : exp-2;

int a = 10
int b= a? 45:65 ✗

boolean a = true
int b= a? 45:65 ✓

c = 11 > 10 && 56 <= 56 ? 10 : 55.

12 55. 10

case 1 :- int a = 10, b = 56;

int c = ++a > 10 && b-- <= 56 ? a++ : b--;

System.out.println("A = "+a+" B = "+b+" c = "+c);

A = 12 B = 55 C = 11

Case 2 :- int a = 10, b = 56;

int c = ++a > 10 || b-- <= 56 ? a++ : b--;

System.out.println("A = "+a+" B = "+b+" c = "+c);

A = 12 B = 56 C = 11

case 3 :- int a = 10, b = 56;

int c = ++a > 11 && b-- <= 56 ? a++ : b--;

11 > 11 ✗ c = 55

A = 11 B = 55 C = 55

System.out.println("A = "+a+" B = "+b+" c = "+c);

A = 11 B = 55 C = 55.

2) Bitwise operators :-

Bitwise operators

Bitwise logical operators

1) Bitwise logical AND (&)
2) Bitwise logical OR ( | )
3) Bitwise logical XOR (^)

Bitwise shift operators

1) Left Shift (<<)
2) Right shift (>>)
3) Right shift with zero fill (>>>)

byte b1 = 10

= 00001010

Left shift by 2 b1 << 2

00 00 10 10

00 10 10 00

+ve

∴ +40

byte b1 = -10

= 11110110

Left shift by 2 b1 << 2

11 11 01 10

11 01 10 00 => 11011000

-ve

∴ -40 ← 2's complement 01010000

byte b1 = 10 = 00001010

Right shift by 2 b1 >> 2

00 00 10 10 => 0 00 000010

sign bit

+ve

= +2

byte b1 = -10 = 11110110

Right shift by 2 b1 >> 2

11110110 => 11 11 1101

↓ 00000010+1

00000011

= -3

Right shift with zeros:-

byte $b_1 = 10 = 00001010$

$00001010 \Rightarrow 00000010$

$= +2$

byte $b_1 = -10 = 11110110$

$11110110 \Rightarrow 00111101$

$\Rightarrow +61$

Right shift with zero = opposite of left shift.

```
Class shiftdemo
{
 public static void main (String [] args)
 {
  int b1=10 , b2= -10;
  System.out.println (" Left shift operator (<<)");
  system.out.println (" B1 << 2=" + (b1<<2));
  System.out.println (" B2 << 2=" + (b2<<2));
  System.out.println (" Right shift operator (>>)");
  System.out.println (" B1 >> 2 =" + (b1>>2));
  System.out.println (" B2 >> 2 =" + (b2>>2));
  System.out.println (" Right shift with zero(>>>)");
  System.out.println (" B1 >>> 2 =" + (b1>>>2));
  System.out.println (" B2 >>> 2 =" + (b2>>>2));
 }
}
```

o/p:- Left shift operator

$B_1 =$

8) new operator :- It is used to provide the dynamic memory allo

Most of the cases new operator is used to create objects.

Syntax:- new classname ();

If we want to store classname reference variable = new class

9) Equality operator:- (==, !=) used to compare both primitive & object

primitive → Content ~~value~~ comparison

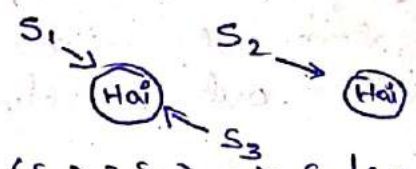
object → reference comparison

Equality operators are used to perform comparision b/w primitive data and objects.

It performs the content comparison b/w primitive variable and it performs reference comparison b/w objects.

int a=40, b=40 → a [40] b [40] if (a==b) ⇒ true.

string s1 = new string("Hai");
string s2 = new string("Hai");
string s3 = s1;




if (s1==s2) ⇒ false
if (s1==s2) ⇒ true

10) Instance of operator :- It is used to check weather the given object or reference variable belongs to specified type or not. Syntax :- (x) instance of (Y) → class/interface/enum
x → refence variable/object

Note :- when ever we use instance of there must be some relation b/w x & Y.

11) Type Casting operator :- Basically type conversions are 2 types.

1) implicit type casting :- Java compiler automatically from 1 type converts data to another if there is no data losage. such type of convers is called implicit type casting or widining or up casting

2) ~~explicit type casting~~ explicit

2) explicit type casting :- way of arraugment changes.

1 → 2 → 4 → 8 → 4 → 8
byte → short → int → long → float → double
char → 2

2) Explicit type casting :- Based on our requirment we can perform the convertion of 1 type to another forcebly is called explicit type casting or narrowing or down casting

```
class Type Cast
{
 public static void main (string [] args)
 {
  byte a = (byte) 188;
  short b = a;
  int c = 45678;
  short d = (short) c;
  long e = 866 - 6662111;
  float f = e;
  System.out.println("A = " + a);
  System.out.println("B = " + b);
```

```
System.out.println(" C= "+c);
System.out.println(" D= "+d);
System.out.println("E= "+e);
System.out.println("F= "+f);
}
}
```

o/p:- A = -68
B = -68
C = 45678
D = -19858
E = 0866666122
F = 0.8666661

(2) Array operation (Array operator) :-

Single dimensional array

Declaration:- 1) int a[];
2) int []a,b,c;
(a - array)
3) int[] a,b,c
(a - array)

Memory allocation :-
1) int a[];
2) a = new int[4];

if we use System.out.println()
after ① -> null
② -> [I @ ___]
[class name @ Hash code Hexadecimal]

Initialization:-
a[0] = 45, a[1] = 60.

Allocation & initialization in single step.
a = new int[]{10,20,30};

Declaration, memory allocation initialization in single step
int a[]={10,20,30};

```
class ArrayDemo
{
 Static int a[],b;
 Static int[] b,c;
 Public Static void main (String[] args)
 {
  System.out.println("A= "+a);
  System.out.println("B= "+b);
  System.out.println("C= "+c);
  System.out.println("D= "+d);
  a=new int[4];
  b= new int[]{35,44,53,62,71};
  System.out.println("A= "+a);
  System.out.println("B="+b);
  System.out.println("Values before =");
  a[0]=44;
```

o/p:- A = null
B = null
C = null
D = 0
A = [I@ ___
B = [I@ ___
a[0

```
a[2]=44;
System.out.println("\nAfter assign some values;");
for(int i=0; i<a.length; i++)
System.out.println(a[i]+"\t");
ArrayDemo ad=new ArrayDemo(); //Proof for showing
System.out.println("\n"+ad); //2) Gets code as output
                              //for empty classes.
}
}
```

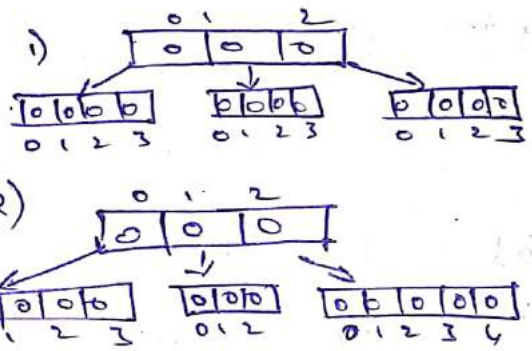
## Two Dimensional Arrays :-

**Declaration:** — 1) `int a[][];`

2) `int []a[] ;` 3) `int [][]a ;`

4) `int[] a[] ;` 5) `int[] []a;`

6) `int [][] a ;`




**Memory allocation :-**

1) `a = new int[3][4);`

2). `a = new int[3][];`
`a[0] = new int[4];`
`a[1] = new int[3];`
`a[2] = new int[5];`

```
Class ArrayDemo1
{
public static void main(String[] args)
{
int a[]
 a=new int[3][];
 a[0]=new int[3];
a[1]=new int[2];
 a[2]=new int[5];
 System.out.println("List of arrays :");
for(int i=0; i<a.length; i++)
{
 for(int j=0; j<a[i].length; j++)
  {
    System.out.println(a[i][j]+"\t");
  }
    System.out.println()
}
System.out.println("A:"+a);
```

o/p:-
List of arrays
0 0 0
0 0 0
0 0 0
A=[[I@ ——
A[0]=[I@ ——

System.out.println ("A[0] : " +a[0]);

}

}

Class operators

{

public static void main (string[] args)

{

int i=8; // i=8+9+9+10+12+13+13+14+16

i = ++i + i++ + i++ + ++i + ++i + i++ + i++ + ++i;

System.out.println ("I = "+i);

}

}

o/p:- I = 104.

Control statments :-

In Java there are 3 types of control statments :

i) Conditional control statments ii) Loop statments 3) Jump st

i) Conditional Control statments :-

In this there are 2 types.

i) if statment.

```
if (condition)
{
statments;
}
```

```
else if (condition)
{
Statments;
}
```

```
else
{
Statments;
}
```

write a program weather the given number is even or odd by.

```
class if
{
public static void main (string [] args)
{
int i = 4
if (i == 4)
System.out.println ("I= " + even number);
else
system.out.println ("I= " + odd number);
}
```

o/p:- I = even number

ii) switch statment.

```
switch (exp or variable)
{ case (value 1): statments;
                  break;
  case (value 2): statments;
                  break;
   :
  case (value n): statments;
                  break;
  default : statments;
}
```

Allowed values in switch.

| upto 1.4 version | from 1.5 V |
|---|---|
| byte | byte |
| short | short |
| int | int |
| long | long |
| char | char |
| | + strings. |

Auto Boxing / Auto Unboxing :- Automatic conversion of primitive data into wraper object is called auto boxing. Automatic Conversion of wraper object into primitive type is called auto unboxing. These 2 concepts are introduced in 1.5 V. Here onwards we can access wraper object insted of primitive value & vice-versa.

write a program to perform all the arithematic operations by using switch case and read the values from cmd prompt as string (operation : add, sub, mod, mul, div) ; (double) value1 , (double) value2.

```
class switch demo
{
public static void main(String[] args)
{
if (args.length == 3)
```

String concadination.

+ , Concat(String s)

Comparision

1) equals.

boolean variable = equals(String value)

2) in compare to (String s)

to lower case :-

tolower case(~~String~~)

opt = opt.to lower case

to upper case(~~String~~)

trim().

```
{
String opt= args[0];
double a= Double.parseDouble (args[1]);
double b= Double.parseDouble (args[2]);
opt = opt.toLowerCase();
switch (opt)
{
  Case "add" : System.out.println("Addition = "+(a+b));
               break;
  Case "sub" : system.out.println ("Subtraction= "+(a-b));
               break;
  Case "mul" : System.out.println("Mul = "+(a*b));
               break;
  Case "div" : system.out.println("Div = "+(a/b));
               break;
  Case "mod" : System.out.println("Mod = "+(a%b)
               break;
  default.   : System.out.println("provide correct input"

}
else
system.out.println("Provide 3 arguments");
}
}
```

Loops:- In java we have 4 different types of loops

1) while 2) do-while 3) for 4) for-each or enhanced for loop

While loop:- If we donot know the number of itteration to perform then we should prefer while loop.

Syntax:-

```
initialization
while (condition)
{statments;
 incre/decrement;
}
```

Note:- If we want to excute single statment as the body of loop then there is no need of block.

while (1) ~ while (true). ~

a) while (true)
System.out.println ("Hai");
System.out.println ("Hello");

o/p:- C.E:- Unreachable statements :-

b) while (false)
System.out.println ("Hai");
System.out.println ("Hello");

11. Here we have true / false which will be identified by the compailer & and we can know how many times it executes if we fail to ~~exec~~ compaile a single statment then it shows C.E

c) int a=10;
while (a==10)
System.out.println ("Hai");
System.out.println ("Hello");

o/p:- hai
hai

d) int a=10;
while (a!=10)
System.out.println ("Hai");
System.out.println ("Hello");

o/p:- hello.

// Here a's value is not know untill execution begins hence there will be no compilation error.

e) final int a=10;
while (a==10)
System.out.println ("Hai");
System.out.println ("Hello");

o/p:- C.E:- Unreachable statements.
As the a is declared as constant (final keyword) here compiler

f) final int a=10;
while (a!=10)
System.out.println ("Hai");
System.out.println ("Hello");

```
{
  Statements;
}
```

```
for(exp1 ; condition; exp2)
{ statements;
}
```

```
for(i=1; i<=10; i++);
{ System.out.println("I=" +i); }
```

o/p:- IF 11

```
for(;;)
{
  system.out.println("I=" +i);
  i++;
}
```

o/p:- I = 1
I = 2
I = 3
⋮
I = ∞ but all must be

In exp-1 we are allowed to declare n no of variables of same type

```
for (i=1; j=45; i<=10; i++, j--)
System.out.println("I= " +i+ "\tJ=" +j);
```

o/p:- I=1 J=45
I=2 J=44
⋮
I=10 J=36

Any valid expression is allowed in place of exp-1 & exp-2

```
int i=1, j=45
for( System.out.println("Hai"); i<=10) ; system.out.println("Hello"));
{ system.out.println("I=" +i+ "\tJ=" +j);
  i++;
  j--;
}
```

o/p:- Hai
I = 1 J = 45
Hello
I = 2 J = 44
⋮
Hello
I = 10 J = 36

// we can place incre/decrement operators in exp-3 along with printing statement

**for-each loop** :- If you want to read 1 by 1 element from collection of elements without loosing any element

It was introduced in 1.5V.

```
class foreachdemo
{
  public static void main (String [] args)
  {
    int a= {55, 44, 33, 22, 11, 99, 88, 77, 66, 76, 45, 43, 32, 63};
    int sum=0
    float avg;
    for( int x : a)
    {system.out.println(" x + "\t");
     Sum + = x;
    }
    System.out.println("\n Sum=" +sum);
```

```
for(i=0; i<a.length; i++)
{system.out.println(a[i]+
 Sum+=a[i];
}
```

```
avg = (float) sum / 6;
System.out.println("Avg= "+avg);
}
}
```

o/p:- 55 44 33 ... 32. 67
sum = 921
avg = 57.62

Jump
1) goto ✗
2) break ✓
3) continue ✓

**Break** :- 1) It must be used inside the loop or switch only.
2) Break is used to terminate the loop or switch follow through.
3) In java we can use break with lable.
4) Break with lable must be inside the labled loop or labled block.

```
class BreakDemo
{
public static void main(String[] args)
{
 fl:
 for(int i=1; i<=3; i++)
 {
  jl:
  for(int j=1; j<=3; j++)
  {
   kl:
   for(int k=1; k<=3; k++)
   {
    if(i==j)
     break jl;
     System.out.println("I= "+i+"\tJ=
     System.out.printf("I= %d \tJ= %d \tK= %d\n",i,j,k);
   }
  }
 }
}
```

o/p:-
i=2 j=1 k=1
i=2 j=1 k=2
i=2 j=1 k=3
i=3 j=1 k=1
i=3 j=1 k=2
i=3 j=1 k=3
i=3 j=2 k=1
i=3 j=2 k=2
i=3 j=2 k=3

```
class BreakDemo1
{
 public static void main(String[] args)
 {
  System.out.println("Main started");
  fir:
  {
   System.out.println("First started");
```

o/p:- Main started
first started
second started
third started
main ended

```
Sec:
{
  System.out.println("second started");
  third:
  {
    system.out.println("third started");
      if(true)
        break fir;
    system.out.println("Hai");
    System.out.println("third ended");
  }
  system.out.println("second ended");
}
system.out.println("first ended");
}
system.out.println("main ended");
}
}
```

Continue :-

1) Continue is used to skip the particular iteration based on some condition.
2) Continue must be inside the loops
3) We can use lable with continue in java
4) Continue with lable must be inside the labled loop only.

~~System~~. _____ _____ :-

You are allowed to write the return statment inside main method as following way.

```
void main()
{
  ========
  return;
}
```

The main method return statment will term the program execution.

System.exit();

integer → 0 normal termination
integer → 1 abnormal termination.

```
Class DoContinue
{
public static void main(String[] args)
```

```
{ int i=0                                    1        3
  do
  {
   ++i;                                  o/p:- 3
   if(i++==5)                                  10
    continue;                                  14
   System.out.println(++i);                    18
  } while(i++<=20);                            22
 }
}
```

OOPS :- (Object Oriented Programming Concepts)

If any programming language follows the following 6 concepts then the programming language is object oriented programming language. 1) Class 2) Object 3) Data Encapsulation. 4) IS-A Relation (Inheritance) 5) Polymorphism 6) Data Abstraction

Class :- It is nothing but blueprint of object which provides specifications / properties and implementation of further features

```
class ClassName
{
 variables ;
 methods ;  } members ──→ Instance members / object members
 blocks ;                ──→ Static members / class level members
}
```

Note :- For class physical existence cannot be possible. For that we should go for object.

Object :- Object is nothing but entity. which is existed physically in real time environment.

Object is also called as instance of a class.

Every object contains some properties and having some actions.

Properties are represented by variables, features by methods.

Constructor with the class name is nothing but

```
class ref.variable = new Classname();  -> Constructor.
```

3) Data Encaptualation:- Wrapping up of group of data elements into a single entity is called data encaptualation.

Data Encaptualation mainly concentrate on hiding the data behind the methods.

We can hide the data with the help of private modifier.

We are not allowed to access private members of a class from outside the ~~package~~ class.

We can access them with the help of getter and setter methods only.

Getter method is used to read the data from variable and setter method is used to update or add the value

```
Class Student
{
 Private int StdNum;
 Private string StdName;
  Public int get StdNum()
  {return StdNum;
  }
  Public void set StdNum()
  { ~~retu~~ StdNum = num;
  }
  Public string get StdName()
  { return std name()
  }
  Public void set StdName()
  {
    // this operator is used to represent current running object.
     this.StdName = StdName;
  }
}
Class Encapsulation
{
public static void main (string[] args)
{
  Student s= new student();
System.out.println ("Student Number is :" + StuNum);
```

```
System.out.println("Student Name is = "+Std Name);
S.setStdNum(234);
s.setStdName("BDPs");
System.out.println("Student Number is = "+S.getStdNum());
System.out.println("Student Name is = "+S.getStdName());
}
}
```

o/p:- Student Num = 0
Student Name = null;
Student Num = 234
Student Name = BDPs

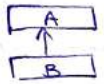
Note:- To provide the reusability in OOPS concept or classes we are maintaining the relationships b/w the classes. This relation ships can be classified into 2 types

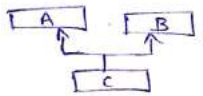
1) Is-A-Relationship (inheritance)
2) Has-A-Relation ship. (Composition / Aggregation).

Is-A-Relationship / inheritance :-

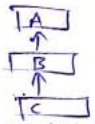
The mechanism of deriving or extending the features of existing class to newly created class is called "Is-A-Relationship" or inheritance. Existing class is also called as "Parent class" or "Base Class" or "Super Class". The newly created class is also called as "child class" or "derived class" or "Subclass".

Based on OOPS classes inheritance can be classified into 5 types. They are 1) single or simple inheritance. 2) Multiple inheritance 3) Multi level inheritance 4) Hirical inheritance 5) Hybrede inheritance

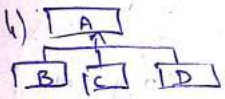
1) 


2) 

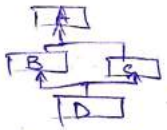
multiple inheritance

3) 

multi level inheritance

4) 

Hirarical inheritance

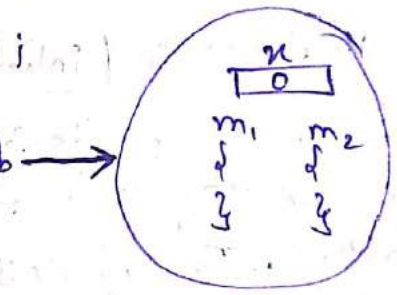
5) 

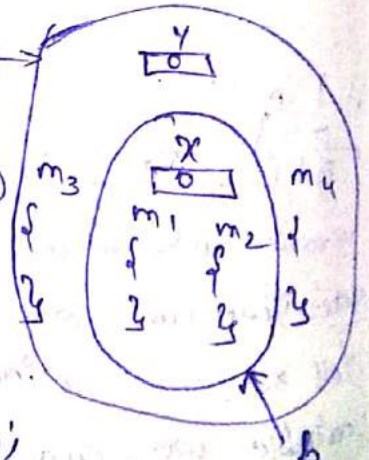
Hybrid inheritance

Note:-1) we can store the child class object into parent class reference variable. But with the help of this reference variable we can able to access parent class specified methods only.

2) But we are not allowed to store the parent class object into child class reference variable.

```
Class Base
{
int x;
Void m1()
{
System.out.println("Base class method1");
    x=333;
}
Void m2()
{
  System.out.println("Base class method2");
  System.out.println("x="+x);
  }
}
Class Derived extends Base
{
int y;
void m3()
{
 System.out.println("Derived class method1");
  Y=123;
}
 Void m2()
{
 System.out.println("Derived class method2");
  System.out.println("x"+"+"+"y="+(x+y));
}
}
Class IASDemo
{
public static void main(String[] args)
{
   Base b=new Base();
    b.m1();
    b.m2();
    Derived d= new Derived();
    d.m1();
```



```
d.m2();
d.m3();
d.m4();
Base b1 = new Derived();
b1.m1();
b2.m2();
// b3.m3(); [Refer note]
}
}
```

Multiple Inheritance :- Java cannot support multiple inheritance through classes. But we can achive this with the help of interfaces.

Reasons for removing multiple inheritance through classes

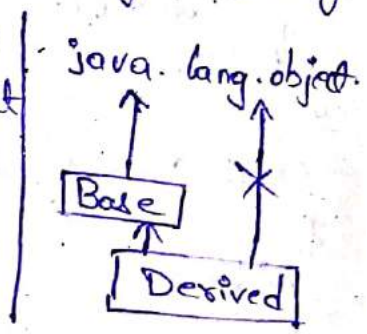
1) If a class extends the features of more than 1 class at a time in the child class there may be a chance of occurance of method naming and variable naming ambiguity.
2) Some times we may create dublicate copies.

To avoid these problems a class can extend only 1 class at a time in java.

Note :- i) For every object comonly required features are availd in object class which is located in java.lang package. i.e java.lang.Object.

2) To get all this features object class is a pre default parent class for every class in java either directly or indirectly.

If a class cannot extend any other classes then only our class is the direct child class of object.




```
class A
{
void m1()
{
System.out.println("Class A method M1");
}
}
class B extends A
{
void m2()
{
System.out.println("Class B method M2");
}}
```

```
class C extends A
{
void m3()
{
System.out.println("class C method M3");
}
}
class HeirDemo
{
public static void main (String[] args)
{
A a = new A();
a.m1();
B b = new B();
b.m1();
b.m2();
C c = new C();
c.m1();
c.m3();
// Here sub childs object can also be stored in parents ref. variable
D d = new D();
d.m1();
d.m2();
d.m4();
}
}
class D extends B
{
void m4()
{
System.out.println("class D method M4");
}
}
```

→ `A a1 = new D(); a1.m1();`

o/p:- class A method M1
class A method M1
class B method M2
class A method M1
class C method M3
class A method M1
class B method M2
class D method M4
class A method M1

## Has-A-Relationship :-

Either Composition or Aggregation is called has a relationship

**Composition :-** There is no chance of existence of contained objects without existance of container object.

Ex:- University contains several departments there is no chance

Polymorphism name itself says that same name having many forms. In this different forms which form has to be executed is decided by compailer and JVM. Based on this polymorphism can be classified into 2 types.

1) Compile time polymorphism 2) Run time polymorphism.

Compile time polymorphism :- If compailer decides which form has to be executed based on reference time is called compile ~~mor~~polymorphism or static polymorphism or early binding.

We can achive this feature with the help of following concepts

1) Method overloading 2) Method Hiding 3) Constructor Overloading.

Run time polymorphism :- If JVM decides which form has to be executed based on run time object is called Run time polymorphism or dynamic polymorphism or late binding.

we can achive this concept with the help of method over riding.

Method signature :- Method name followed by arguments list is called method signature.

1) Method overloading :- Two methods are said to be overloaded methods if and only if both the methods having same name with different signature.

* In this situation method invocation (calling) can be take care by java compiler based on reference type. Hence this concept comes under compile time topolymorphism.

Ex:- Sum (int, int), sum (int, float), sum (float, int)
Sum (int, int, int), sum (float, float)

```
class overloading
{
void sum (int x, int y, int z)
{System.out.println("sum of 3 integers= "+(x+y+z));}
Void sum (int x, int y)
{System.out.println("sum of 2 integers ="+(x+y));}
void sum (int x, float y)
{System.out.println("sum of int and float="+(x+y));}
Void sum (float x, float y)
{System.out.println("sum of 2 floats ="+(x+y));}
Void sum (float x, int y)
{System.out.println("sum of float and int="+(x+y));}
Public static void main (String[] args)
{
overloading ol = new overloading();
ol.sum(44,33,66); //ol.sum(int, int, int)
ol.sum(67, 33.55f);
ol.sum (34, 11);
ol.sum (23.44f, 56);
ol.sum (46.55f, 33.44f);
byte a=25;
ol.sum(a, 67); //ol.sum (byte, int)->ol.sum(short,int)-> ol.sum(int,int
long b = 567L;
ol.sum(45,6);
ol.sum (b, 34.55f); //ol.sum(long, float) -> ol.sum (float, float)
ol.sum (b,a); //ol.sum(long, byte)-> ol.sum(float, short)-> ol.sum(float, int)
}
}
```

Var-Argument-Method- Based on overloading concept when ever no of arguments vary we have to write the new method with specified signature.

Hence for every requirment we have to write new method implementation. But this process will increase the length of the code and reduces readibility.

To resolve this problem Sun people introduced var-Argument method which can hold. 0 to n arguments of same data type.

It will reduce the length of code.

The var-Argument store n number of arguments as single dimension array internally we can replace array argument with var-argum
But vice-versa is not possible.

```
/*
int a[], int[] a, int[]a;
int... a, int a...
*/
class VarArgDemo
{
 static void sum (int ... x)
 {
  System.out.println("Var-Arg Method");
  int sum=0;
  for (int i : x
  sum+=i;
  System.out.println("sum="+sum);
 }
public static void main(String... Args)
{
  Sum(44,55);
  sum(55,33,56);
  Sum(67,78,83,34,57);
  Sum(23,45,69,78,89,96);
  sum(46);
  sum();
  int x[]={55,33,77,66,55,12,
           46,67,84};
  Sum(x);
 }
}
```

Method over-riding :- 1) Two methods are said to be over-ridden methods if and only if both the methods have same signature

2) what ever the features available in parent class pre-defaultly available to the child class. If child class doesn't satisfy with parent class implementation we are allowed to rewrite it.
This process is called Method Over Riding.

Rules for method over riding :-

1) Both parent and child class method signature must be same
2) child class access previlazation must be greater than or equal to parent

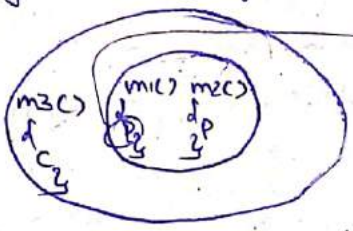
Acess previlazation

| Acess Modifier/ Acess Area. | Same class | child class in same package | other class in same package | Child other class in others package |
|---|---|---|---|---|
| Private | Yes | No | No | No |
| Default | Yes | Yes | Yes | No |
| Protected | Yes | Yes | Yes | Yes |
| Public | Yes | Yes | Yes | Yes |

other class in other package
private - No, default - No, Protected - No, Public - Yes

Access previlage order:- private < default < protected < public

In this concept method invocation can be take care by JVM based on run time object Hence this concept comes under run time polymorphism.




initially it will point to P but after method overriding it turns to P.

```
Class Parent
{
Void m1()
{ System.out.println("Parent class Method M1"); }
Void m2()
{System.out.println("Parent class M2 Method "); }
}
Class child
{
 Void m3()
 {System.out.println("child class Method M3"); }
 Void m4()
 {System.out.println("child class Method M1"); }
}
class OverRiding
{
public static void main (String[] args)
{
 Parent p = new Parent();
 p.m1();
 p.m2();
 Child c = new Child();
 c.m1();
 c.m2();
 c.m3();
 parent p1 = new child();
 p1.m1();
 p1.m2();
}
}
```

dp:-

Parent class Method M1
Parent class M2 Method
child class Method M1
Parent class M2 Method
child class Method M3
child class Method M1
Parent class M2 Method

ult < protected < public

n be take care by JVM
s concept comes under run
→ initially it will be
p but after method
overriding it turns
to P.

ethod M1"); }

M2 Method "); }

ethod M3"); }

ethod M1"); }

s)

t class Method M1
ut class M2 Method.
d class Method M1
nt class M2 Method
d class Method M3
ild class Method M1
ent class M2 Method

we can write parent class private method in the child class. But it is not a method overriding. It comes under method hiding because parent class m1 private method is not visible to child class.

we are not allowed to override the parent class static method as non static in the child class. and vice-versa is also not possible.

But we are allowed to write both parent & child class method as static.

It seems like method overriding but it internally works as method hiding.

Method hiding:-

In this concept method invocation can be taken care by java compailer based on reference time.

Note:- If we declare both parent & child class m1 method as static in the above program then the output is

parent class Method M1
Parent class M2 Method
child class Method M1
Parent class Method M3
Child class Method M1
Parent class Method M1
Parent class Method M2

| Parent/child | OOPS Concept | type of polymorphism |
|---|---|---|
| both static | Method hiding | compile time polymorphism |
| not static. | Method overriding | run time polymorphism |

Constructor:- Constructor is nothing but a method which is having the same name of the class without any return type. Constructors are invoced only once for every object creation. Constructors are used to provide memory allocation and initialization for instance members.

In Java Constructors are classified into 2 types. They are
1) Default constructor 2) Parameteristic constructor.

Default constructor:- Constructor without arguments

Parameteristic constructor:- Constructor with arguments

If a class doesn't contain any Constructor in it then only java Compiler provides default constructor as follows

Super c); → It is the method that calls parent class and allocate memory to its variables

before Compilation
class Test
{
}

after Compilation
class Test
{
Test()
{
Super c);
}
}

```
class Parent
{
  int x;
  Parent()
  {
   System.out.println("Parent class Demo");
    x=444;
  }
 void display()
 {System.out.println("x="+x);}
}
 class ConsDemo
 {
  public static void main(String[] args)
  {
   Parent p=new Parent();
   p.display();
  }
}
```

```
class Parent
{
 int x;
 parent()
 { System.out.println("Parent class Default constructor");
   x:=444;
 }  void display()
 class child extend
    {System.out.println("x="+x);}
}
class Child extends Parent

{ int y;
  child();
  { // super() is used to call parent class constructor.
    super();
    System.out.println("child class Default Constructor");
    y=333;
  }
   void display()
  {
   System.out.println("y="+y);}
   //super keyword represents parent class object.
```

```
    super.display();
  }
}
class ConstDemo.
{public static void main (String[] args).
 {parent p= new Parent();
  p.display();
  Child c = new Child();
  c.display();
 }
}
```

Op:-
Parent class Default Cons
x = 444;
Child class Default Cons
y = 333;

Parameterized Constructor:-

Note:- ~~Ba~~ In) We are allowed to write more than 1 constructor in a class but with different signature. This process is called Constructor overloading.

2) super() in child can be initilize either from default constructor or from parameterized constructor

```
class Parent
{
 int x;
 Parent()
 {
  System.out.println("Parent class Default constructor");
  x=111;
 }
 Parent(int x)
 {System.out.println("Parent class Parametrized constructor");
  this.x=x;
 }
 void display().
 { System.out.println(" x = "+x);}
}
class child extends parent.
{int y;
 child()
 {super();
  System.out.println("Child class Default Constructor");
  y=333;
 }
```

```
Child (int x, int y)
{
  super(x);
  System.out.println("Child class parameterized Constructor");
  this.y = y
}
void display()
{
  super.display();
  System.out.println("y = "+y);
  System.out.println("x = "+x+" y = "+y);
}
}
class ConstCons
{
public static void main (String[] args)
{
  Parent p = new Parent(345);
  p.display();
  Child c = new Child(); Parent p1 = new Parent();
  c.child();                p1.display();
  Child c = new child();
  c.child();
  Child c1 = new child (345, 355)
  c1.child();
}
}
```

```
O/p:-
Parent class Parameterized Co
x = 345;
Parent class Default Const
x = 111
Child class Default Cons
y = 333
Child class Parameterized Co
x = 345  y = 355
```

## Copy Constructor :-

A constructor having the object of the same class as argument is class copy constructor. It is used to create dublicate objects.

```
class test
{
int a,b,c;
Test()
{ System.out.println("Default Constructor");
  a=b=c=22;
}
Test (int a, int b, int c)
{
System.out.println("Parameterized Constructor");
this.a=a;
this.b=b;
```

```
this. c = c;
}test (int a, int b)
{this();
 System. out. println ("Parameterized Constructor ");
  this. a = a;
  this. b = b;
}void display()
{System. out. println ("A = "+a+"\t B = "+b+ "\t C = "+c);}

Test (Test t)
 {
  System. out. println ("Copy Constructor");
   a = t. a;
   b = t. b;
   this. c = t. c;
  }
}
Class Copy Const.
{public static void main (String [] args)
 {
  Test t = new Test();
   t. display();
   Test t1 = new Test (44, 55, 66);
   t1. display();
   Test t2 = new Test (35, 65);
   t2. display
   Test t3 = new Test (Test t)
 }}
```

Note:-

1) Allowed modifiers for constructors are private, default, protected, public.
2) We are not allowed to access private constructors directly from outside class.
3) But we can access them with the help of factory method. Factory method is a method which is declared as a public static and return the object of that class.

```
Class Employee
{
 private Employee()
 {System. out. println ("Private Constructor");}
 Public static Employee getEmployee()
 {
  System. out. println ("Factory method ");
  Employee e = new Employee();
```

```
        return e;
    }
}
class SingleTurn
{
public static void main (String[] args)
{
  Employee e1 = Employee.getEmployee();
  Employee e2 = Employee.getEmployee();
  Employee e3 = Employee.getEmployee();
  System.out.println (e1);
  System.out.println (e2.toString());
  System.out.println (e3);
}
}
```

o/p:- Private Constructor
Factory method
Private Constructor
Factory method
Private Constructor
Factory method.
Hexadecimal Hash

## SingleTurn classes :-

we ~~can~~ are allowed to create only one object for a class. we can use that object for every instance. such type of classes are ~~t~~called single turn classes. The main advantage of this concept is it reduces the memory Wastage. we can also generate our own single turn classes by using private constructor, factory method, static reference method.

```
class Employee
{ static Employee e;
  private Employee()
  {System.out.println("Private Constructor");}
  public static void main (String[] args) Employee getEmployee()
  {
    System.out.println("Factory method");
    if(e==null)
      e = new Employee();
      return e;
  }
}
class SingleTurn
{ public static void main (String[] args)
{ Employee e1 = Employee.getEmployee();
  Employee e2 = Employee.getEmployee();
  Employee e3 = Employee.getEmployee();
  System.out.println (e1);
  System.out.println (e2.toString());
  System.out.println (e3);
}
}
```

Modifiers:- 1) Private 2) Default 3) [illegible] 4) static 5) [illegible] 6) Strictfp 7) final 8) synchronized 9) abstract 10) native 11) transient 12) volatile.

Class level Modifiers (5) :- 1) default 2) Public 3) Strictfp 4) final 5) abstract

Inner classes (8) :- 1) default 2) public 3) Strictfy 4) final 5) abstract 6) private 7) protected 8) Static

Member level modifiers

a) Method level modifiers (10) :- 1) Private 2) default 3) protected 4) public 5) static 6) Strictfp 7) final 8) synchronized 9) abstract 10) native

b) Variable level modifiers (8) :- 1) private 2) default 3) protected 4) public 5) static 6) final 7) transient 8) volatile.

The classes which are declared as default which can be accesible within the current package only.

static :- Static modifier is applicable for methods & classes only. we can directly access the static members without creating the object.

strictfp (floating point notation) :- It is applicable only for methods & class if we declare a method with strictfp modifier then all floating point calculation of that method has to follow IEEE standard 754.

If we declare a class with strictfp modify the every implimented method of that class has to follow IEEE standard for their floating point calculation.

Synchronized :- It is applicable only for methods & blocks, if we declare a method with synchronize modifier then the implimentation of that method can be allowed to access only 1 thread at a time

Internal process of object creation:-

1) Identify all the instance members & provide memory allocation from top to bottom

2) Initilize instance variables & execute instance blocks from top to bottom.

3) Execute the constructor.

For final variable :- for final variable we have to provide the initialization manually. otherwise we will get compile time error

For final local variables we have to provide initialization either at time of declaration itself or before accessing it

For instance final variables we have to provide initialization before the completion of object creation. that means we can provide the initilization either at time of declaration or with in instance block

For static final variables we have to provide the initialization or within the class loading. that means either within declara itself or in static block.

Final method :- The method which is declared with final modifer we are allowed to rewrite it. that means it will restrict the polymorphi Concept. Hence never recomended to go for final method if there is no specif requirment

Final class :- If we declare a class with final modifier we are no allowed to create any child class.

Every method inside the final class is final. but there is no need of every variable as final.

Final classes also disturb the inheritance concept. Hence never recomen to go for it, if there is no requirment

```
class FinalDemo
{
 final int a;
 { System.out.println("Instance block");
   a=888;
 }
 FinalDemo
 {
   System.out.println("Constructor");
 // a=555;
 }
  public static void main(String[] args)
  {
      FinalDemo fd = new FinalDemo();
      //fd.a=333; Here we can modify final variable
      System.out.println(fd.a);
  }
}
```

o/p:- Instance block
Constructor
888

Data Abstraction :- Data abstraction means hiding the internal implem of data is called data abstraction.

We can achive the data abstraction in java with abstract modifier.

Abstract modifier is applicable for method & classes only.

Abstract class :- If the implementation of the class is not completed or we are not allowed to create the object directly then we

should declare

For abstract classes we are not allowed to create objects. To access the elements of abstract class.

For every abstact class we must create at least 1 child class. with that child class we can access abstract ~~class~~ members

```
abstact class Parent
{
void m1()
{System.out.println("Class Parent method M1");}
void m2()
{System.out.println("class Parent method M2");}
}
class child extends parent
{
void m3()
{System.out.println("Class ~~Parent~~ Child method M3");}
}
class AbstractDemo
{
Public static void main (String[] args)
{
  Child c = new Child();
  c.m1();
  c.m2();
  c.m3();
  Parent p = new Child();
  p.m1();
  p.m2();
}
}
```

o/p:- class Parent method M1
class Parent method M2
class child method M3
class Parent method M1
class Parent method M2

These comes after final modifiers

Native:- It is applicable only for methods. If we declare a method with native modifier then implimentation of the method is already completed in other languages like C & C++.

Transient:- These are allowed only for variables. If we declare the variable with transient modifiers it can hide the value of the variable & provide the default values at the time of transfering the java

object into either file format or network support format

Volatile:- It is applicable only for variables. If the value of t Variable changes frequently then we should declare those varia with volatile modifier.

For volatile variable every thread creates our seperate copy & th modifications are saved into the original copy.

Final :- It is applicable only for variables, methods & classes.

a

1) If we know the method specification but not method implem then we should go for abstract methods.
2) For abstract methods there is no body.
3) If a class contains any abstract method then we have to declare those classes with abstract modifier.
4) For every abstract class we must create atleast 1 chil class. The child class is responsible to create implementatio for parent class all the abstract methods
Otherwise we have to declare the child class as abstract.

The main advantage of abstract method is to provide guidlines to the child classes.

Note:- We are allowed to write constructor in abstract class. Here we can access the constructor class is called automa when child class is called

```
abstract class Parent
{
  Parent()
  {System.out.println("Abstract class constructor");}
  void m1()
  {System.out.println("Parent class method m1");}
  void m2();
}
class child extends parent
{
  void m3();
  {System.out.println("Child class method m3");}
```

void m2()
{ System.out.println ("Parent class method m2"); }

class Abstractdemo1
{
public static void main (String[] args)
{
Parent p = new Class();
p.m1();
p.m2();
Child c = new classc();
c.m1();
c.m2();
c.m3();
}
}

o/p:- Abstract class Constructor
Parent class method m1
Child class method m2
Parent class method m1
Child class method m2
Child class method m3

Illegal combination of modifiers at method level are.

1) private - abstract 2) static - abstract 3) strictfp - abstract
4) Syncronized - abstract 5) native - abstract 6) final - abstract.

Illegal combination of modifiers at class level:-

1) final - abstract

Interfaces:-

1) Introduction:-

Def:- 1) Any service requirments specification is called an interface

Ex:- 1) Sun people defined servelet API to develop web applications web server is responsible to provide implementation.

2) From the user/point of view and interface define the set of various services what he is expecting. From the service provide point of view an interface define the set of services he is offered.

Hence an interface considered as a contract blw client & service provider

3) Inside interface every method is abstract whether we are declaring abstract or not. Hence interface is considered as 100% pure abstract class).

Summary definition:- Any service requirment specification (or any contract blw client and service provide or 100% pure abstract class is considered as an interface.

Interface declaration

```
interface interfacename
{
public static final variable;
Public abstract methods
default methods; } → from 1.8 v
static methods;
private methods; → from java 9v
}
```

Note:- 1) For all the variables of interface we have to provide initialization at the time of declaration itself.

2) We can inherit the features of interface to a class by using implements keyword

3) The implemented class has to provide the implementation for each and every abstract method available in interface. Otherwise the class has to be declared as abstract.

```
interface One
{
int
```

note:- 1 class can implement n no of interfaces simultaniously. In this kind of situation there may be a chance of occurance of method naming ambiguality and variable naming ambiguality. It can be resolved as following ways.

i) Method naming ambiguity:-

i) More than 1 interface containg method with same signature then in the implemented class provide implementation only one because those methods cannot have body in interface.

ii) If more than 1 interface containg method with different signature then we have to provide the implementation for both of them method overloading

contract b/w client and

abstract whether we
nterface is considered

ment specification (SRS)
rovide or 100% pure
erface.

we have to provide
tion itself.
ce to a class

mplementation for
le in interface.
as abstract.

ces simultaniously.
of occurance of
ambiguativity.

same signature
mentation only one
face.
ent signature then
of them

~~Variable name~~ ~~variable~~
~~If more than 1 interface~~ containg

1) If more than 1 interface containg variable with same name then we will get Variable naming ambiguarity
2) we can resolve this problem with the help of interface name

~~we can~~ Note:- To provide the reusability we can also apply both IS-A and HAS-A Relationships b/w Interfaces.

3) we can maintain the Is-A relationship b/w interfaces with the help of extends Keyword only.
3) One interface can extends the features of more than interface
4) One class can extends and implement interfaces both simultaneously.

```
interface One
{int x=333;
 void m1();
 void m2();
}
interface Two
{int x=555;
 void m1();
 void m3();
}
interface Three extends One,Two
{
void m6();
}
abstract class Parent implements One,Two
{public void m1()
{
System.out.println("Parent class M1 Method");
System.out.println(" One x = "+One.x);
System.out.println("Two x ="+Two.x);
}
 void m3()
{ System.out.println("Parent class M3 Method");
}
Public void m5()
{System.out.println(" Parent class M5 Method");
```

```
}
child class child extends parent implements three.
{
void mu()
{System.out.println("child class mu method");}
public void m2()
{System.out.println("child class m2 method");}
Public void m6()
{System.out.println("child class m6 method");}
}
class InterfaceDemo
{
public static void main(String[] args)
{
parent p= new child();
p.m1();
p.m2();
p.m3();
p.m5();
child c= new child();
c.m1();
c.m2();
c.m3();
c.mu();
c.m5();
c.m6();
One o=new child();
o.m1();
o.m2();
Three t =new child();
t.m1();
t.m2();
t.m5();
t.m6();
}
}
```

o/p:-

s three.

ethod");}

2 method");}

method");}

Var-Arg ... method!-

It was introduced in Java-1.5V

According to method overloading if the no of arguments are varied of same data type. For every variation we have to write a new method implementation. This process will increase the lines of code and reduces the readibality. Hence to resolve this problem Sun people introduced var-Argument.

Single argument variable can hold 0 to n number of arguments of same data type.

Ex:-
```
void m1(int x, int y)
{
}
void m2(int x, int y, int z)
{
}
void m3(int ... x)
{
}
```

Syntax

Return type method (data type ... Varx)
name

Marker interfaces:- Interface doesn't contain any methods in it. But by implementing this interface our class will get some ability to perform. Those interfaces are called Marker interfaces or ability interfaces.

Ex:- Serializable, cloneable.

Marker interfaces behaviour is defined at the time of implementation of JVM level coding

Serializable :- It provides ability to convert java formated object into either file formated or network formated objects.

cloneable :- It provides the ability to create dublicate objects

Adapter classes:- Adapter class is a class which implements a interface and provides the empty implementation for every method in it.

If we implements an interface directly for each and every method compulsory we should provide implementation for every method in the interface wether it is required or not. This approch increases the length of code decreases readibility.

we can resolve this problem with the help of Adapter class.
Instead of implementing an interface if we can extend Adapter class we have to provide implementation only for required methods but not for all the methods of that interface. thus application decreases length of code and improves readibility.

Imp:-

Differences b/w Interface & abstract class:-

Interface:-

1) If we just know the features but not the implementation then we should go for interfaces
2) Every method present in interface must be abstract and public
3) There is restriction on interface modifiers
4) Every interface variable must be public static final
5) As every interface variable is always public static final, we cannot declare with private, protected, transient, volatile.
6) It is required to perform initialization at the time of declaration
7) Inside abstract class we cannot take both static & instance
8) Inside abstract class we cannot take constructors

Abstract class:-

1) If we are talking about implementation but not complete then we should go for abstract classes
2) Every method present inside abstract class need not be public and abstract.
3) There is no restriction on abstract class method modifiers.
4) Every abstract class variable need not be public static final.
5) There are no restrictions on abstract class variable modifiers
6) It is not required to perform initialization at the time of declaration
7) Inside abstract classes we can take both static and instance
8) Inside abstract class we can take constructors

1.8V Java features in interface:-

From 1.8V onwards we can write default & static methods inside the interface

1) Default methods:- These methods have the body.

Default is a normal keyword. It is not a access modifiers.

If you want to provide a new implementation feature implementation for already compleated software or product then we should go for default methods.

If you want to provide minimum guidlines for implemented classes then also we can go for default methods

```
interface Left
{
 default void m1()
 { System.out.println("Default method M1 in Left "); }
  void m2();
}
interface Right
 {
  default void m1()
  {System.out.println("Default method M1 in Right");}
 }
Class DefaultInterface implements Left, Right
 {
  public void m1()
  { System.out.println("Default Interface M2 method");
     Left.super.m1();
     Right.super.m1();
  }

  public static void main (String[] args)
  {     public void m2()
 Default{System.out.println("M2 Method");}
  public Static void main (String[] args)
  {
   DefaultInterface di = new DefaultInterface();
    di.m1();
    di.m2();
  }
}
```

Static methods :- It was introduced in 1.8V if you want to provide any feature to acess without (implementing

of creating any object
Static method inside the interface this type of methods are invoced with the help of interface name only:

```
interface Test
{
  void m1(); static void main()
  {
    System.out.println("Static method of interface");
  }
}
class NewoEnterface
{
  public static void main(String[] args)
  {
    Test.m1();
  }
}
```

Private methods:- It was introduced in java 9 version. Private methods are used to provide commnly accessed code for more than 1 default method.

```
interface Test
{
  default void m2()
  {
    System.out.println("M2 method started");
    System.out.println("M2 method ended");
    mu();
  }
  default void m3()
  {
    System.out.println("M3 method started");
    mu();
    System.out.println("M3 method ended");
  } → Private void mu() {System.out.println("private method in inte
  static void m4();
  {System.out.println("Static method of interface");}
}
class Testing implements Test
{
}
class Interface New
{
```

```
public static void main (String [] args)
{
  Test.m1();
  Testing t1 = new Testing();
  t1.m2();
  t1.m3();
  // t1.m4();  coe can't access private methods from an interface from main class.
  }
}
```

~~Two one~~

**Package:-**

It is nothing but a directory which can contain set of classes and interfaces which are used to perform similar type of operations.

We can create our own packages with the help of package statement.

We are allowed to write at most one package statement in a single program.

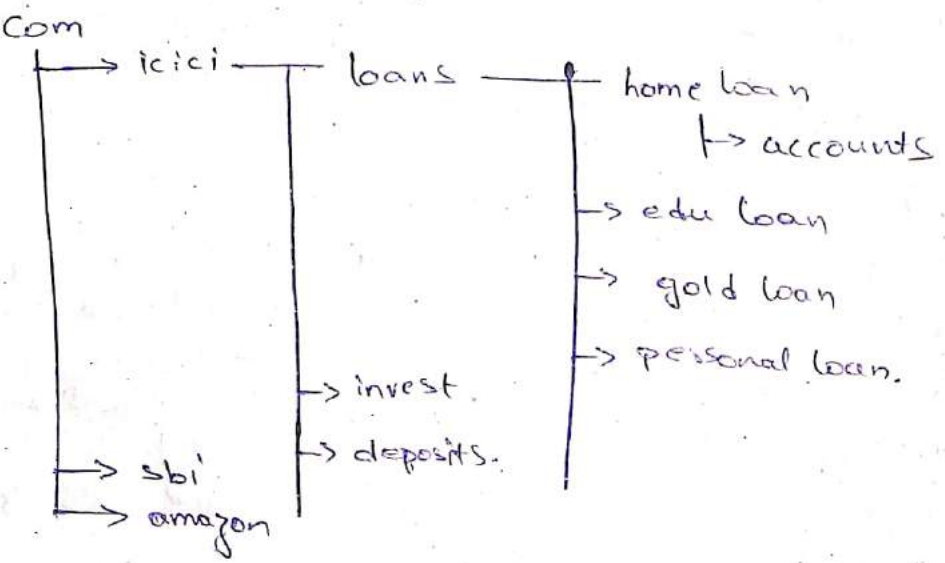
Syntax:- package packagename;

(It is) key word

Package name must be in the format of universal naming convention i.e reverse order of domain name

1) google.com → dns server.
   ↓
   Server name

2) URL => www.icic.com\loans\homeloan\Accounts
   domain name.




3) package com.icic.loams.homeloans
   class Accounts
   {
   }

```
package in.bdps.java
class parent
{
 int x ;
 void m1();
 {
  System.out.println ("Parent class M1 method");
    x=333
 }
 Void m2();
  {
  system.out.println ("Parent class M2 method");
  }
}
```

compilation — dot indicates current directory

```
javac -d . parent.java
javac -d . child.java  Packagedemo.java
java  fully qualified name of class
java in.bdps.java.packagedemo
```

2)
```
package in.bdps.java
class child extends parent.
{
 Void m3()
 {
 System.out.println("child class M3 method");
 System.out.println ("x = "+x);
 }
 Void m4()
 {
 System.out.println ("child class M4 method");
 }
}
```

3)
```
import java.bdps.in
class Packagedemo
{
 public static void main (String[] args)
 {
  Parent p = new Parent();
  p.m1();
  p.m2();
  Child c = new child();
  c.m1();
  c.m2();
  c.m3();
  c.m4();
  parent p1 = new child();
  p1.m1();
  p2.m2();
 }
}
```

o/p:-
parent class M1 method
parent class M2 method
~~child class M~~
Parent class M1 method
parent class M2 method
child class M3 method
x = 333
child class M4 method
Parent class M1 method
Parent class M2 method

To compile the class files in the specified directory in the specified package use -d option in javac command

Syntax:- javc -d directoryname programname.java.

Ex:- javac -d . parent.java

We can also compile more than 1 source code at a time as follows

Syntax: javac -d directoryname program1.java program2.java

Ex:- javac -d . child.java

We can execute the classes which are available in packages from location of package directory as follows

Syntax:- java fully qualified name of class

Ex:- java in.bdps.java.packageDemo

We can execute the classes of a package from any location of the system by using -cp option in java command.

Syntax:- java -cp classpath fullyqualified name.

Ex:- java -cp f:\jun6to7\ in.bdps.java.PackageDemo

Accessing the classes from outside the package:-

```
Package in.bdps.java1;
Public class Parent1
{
protected int x;
protected void m1() // it can be accessed by other classes in same package
{
System.out.println("Parent class m1 method");
x=333;
}
Public void m2()
{System.out.println("Parent class m2 method");
}
}
```

2)
```
Package in.bdps.java2;
public class Child1 extends in.bdps.java.parent1
{
public void m3()
{System.out.println("child class m3 method");
System.out.println("x="+x);
}
protected void m4() // it cannt be accessed by other classes in other packages
{
System.out.println("child class m4 method");
}}
```

```
package in.bdps.java1;
class package Demo1
{
public static void main (String [] args)
{
parent1 p= new parent1();
p.m1();
p.m2();
in.bdps.java2.child1 c = new in.bdps.java2.child1();
c.m1();
c.m2();
c.m3();
// c.m4();
parent p1= new in.bdps.java2.child();
p1.m1();
p1.m2();
}
}
```

o/p:- parent class M1 meth
parent class M2 meth
Parent class M1 meth
Parent class M2 meth
child class M1 meth
x = 33 }
Parent class M1 meth
Parent class M2 meth

Note :- 1) If you want to access classes which are located in other packages for every usage you have to provide fully qualified name of the class.

2) But this process will increase the length of code and reduces readabil

3) To resolve this problem Sun people introduced import statment.

Import statment :-

Import statment provides the ability to acess classes which are available in other classes like current package class.

That means if we import a class which is available in other package by using import statment then we can acess that class any number of times with short names.

Import statments can be classified into 2 types.

1) Explicit import 2) Implicit import.

import packagename.classname import package name.*;

1) Explicit import :- In this we have to name the specific class in specified package to access.

import package name.class name;

Ex:- import java.util.Date;
import java.util.Arraylist
import java.util.Scanner
import java.io.File;

2) Implicit import :- In this we name the specific package to get access of all the classes in it.

```
import packagename.*;
```

Ex:- import java.util.*;
import java.lang.*;
import java.io.*;
import java.sql.*;

Note :- 1) Explicit import is recomended to use if our priority is used to maintain readibility.

2) If we are using implicit import there may be chance of occurance of class naming ambiguavity.

3) To resolve this problem java compiler follows ~~same~~ some priority

1) Explicit import class 2) Current package class 3) Implicit import class

Explicit > current package > Implicit.

import ~~when we call the method with~~

```
class Date
{
public String toString()
{
return "User Defined Date Object";
}
}
```

Static import :

If you want to access ~~classes~~ Static members which are available in other classes we have to ~~to~~ call them with the help of class name for every usage.

But this process will increas the length of code & decreases readibility

To increase the readibility Sun people introduces Static import in java 1.5v

It provides the ability to access static members of other class as current class members.

It is also classified into 2 types they
1) Explicit static import 2) Implicit static import.

1) Explicit static import.

```
import static java.lang.Math.random;
import static java.lang.Math.sqrt;
class StaticImportDemo
{
public static void main (String [] args)
{
System.out.println ("Random value = "+(int)(random()*100));
System.out.println ("Square root of 125 = "+sqrt(125));
System.out.println ("Cube root of 125 = "+Math.cbrt(125));
System.out.println ("5 power 4 is = "+Math.pow(5,4));
System.out.println ("Max value of 46&77 = "+Math.max(46,77));
}
}
```

o/p :- Random value is : 63
Square root of 125 : 11.180255 988→498949
Cube root of 125 : 5.0
5 power 4 is :- 625.0
MaxValue of 46 & 77 = 77.

Note :- If you are using implicit static import there may be a chance of occurance of ambiguity. To resolve this java compiler followes the following priority.

1) Current class static member
2) Explicitly imported static member
3) Implicitly imported static member.

According to the developers point of view static import reduce the readibility because occurance of ambiguity is high.

Except

## Exception Handeling :-

**Introduction :-** If a program terminates unwantedly or unexpectedly is called abnormal termination. The errors which are caused to such situations are generally called exceptions.

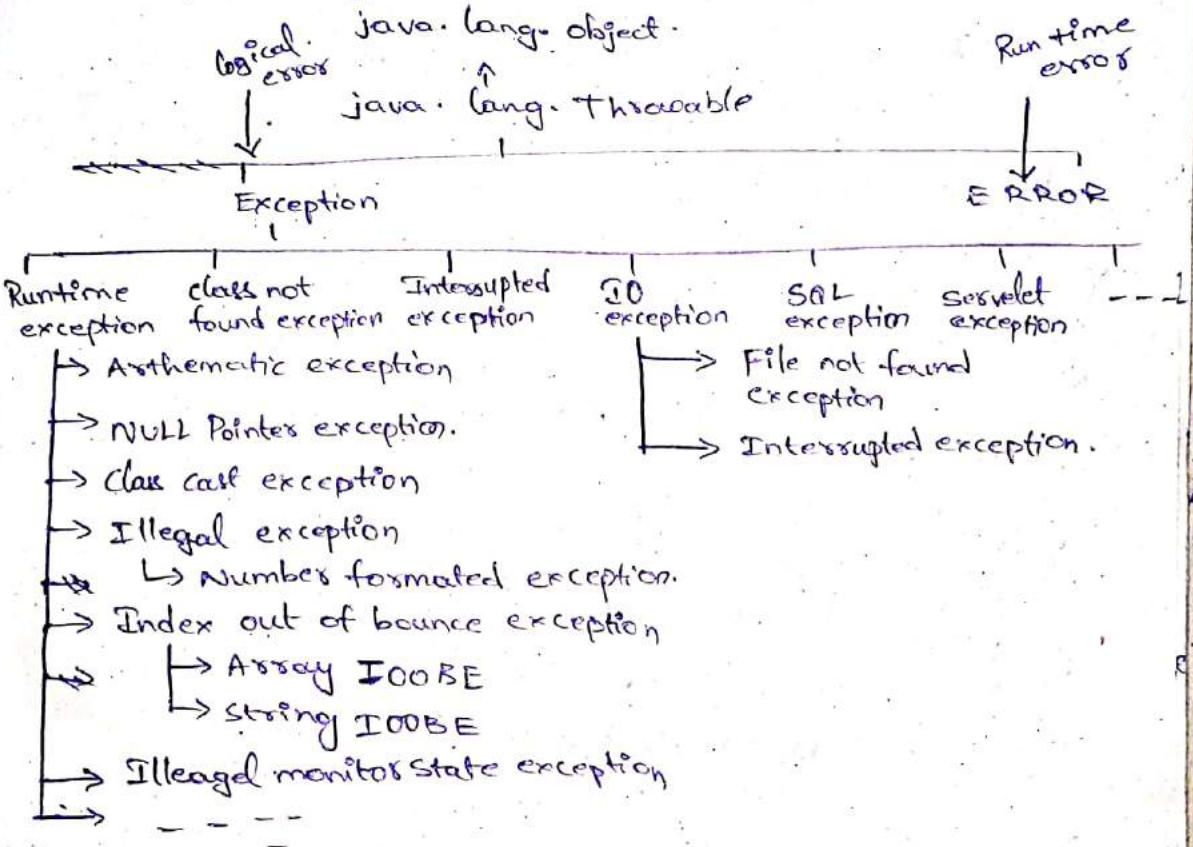
At the time of execution we may get a different types of errors
1) Logical error 2) Run time error

**Logical error :-** 1) The errors which are occured because of the mistakes at the time of logic implementation.
2) These type of errors can be resolved ~~through~~ programatically.

**Run time error :-** 1) The errors which are occared because of lack of resources.
2) This errors cannt be resolved through programatically

**Note :-** The errors which are resolved through programatically are categorised as exception in java. otherwise it is considered as error.

**Exception hirearsity :-**




Exceptions are classified into 2 types based on area of identification. They are 1) Checked exceptions 2) Unchecked exceptions

**Checked exception :-** The exceptions which are identified by the compiler at the time of compilation are called

checked exception.

Ex:- class not found, interrupted, Io exception

Unchecked exceptions:- Exceptions which are not identified by the Compiler is called unchecked exceptions

Ex:- All the exceptions under runtime exception.

A checked exception is said to be "fully checked exception" if and only if all of its child exceptions are checked.

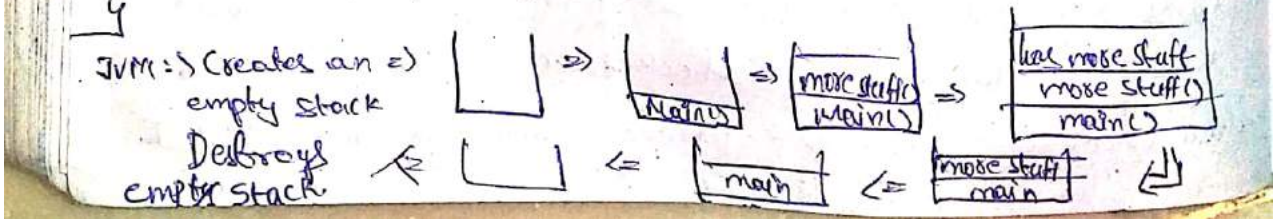
Ex:- I/O exception

A checked exception is said to be "partially checked exception" if and only if some of its child exceptions are checked and rest un checked. Ex:- Exception

Exception handling:- Providing the alternative solution for raised exception is called exception handling. In java we have 2 types of handling mechanism.

1) Default exception handling 2) Customized exception handling

```
class StackTraceDemo
{
 public static void main (String[] args)
 {
    System.out.println ("main started");
    hasmorestuff();
    System.out.println ("main ended");
 }
  static void hasmorestuff()
  {
   System.out.println ("More stuff started");
   hasmorestuff();
    System.out.println ("more stuff ended.");
  }
  static void hasmorestuff()
  {
   System.out.println ("has more stuff started");
   System.out.println (3/0);
    System.out.println ("Has more stuff ended");
  }
}
```



Note:- i) The default exception handling cannot avoid abnormal termination. To resolve this problem and perform smooth termination. we should go for customer handeling.

ii) Customized exception handeling :-

We can perform the customize exception handling by the following keywords i) try 2) catch 3) throw 4) throws 5) finally.

i) Try :- Try is a block. In this block we have to place all the risky statments. The statments which may cause exception at run time are called risky statments. Try block is always associated with either catch or finally blocks

ii) Catch :- Catch block is used to provide the handeling code for a specified exception. Catch block is always associated with try block.

iii) Throw :- Throw keyword is used to hand over the exception object to JVM.

iv) Throws :- Throws key word is used to transfer the exception handeling responsibility to the calling method.

v) Finally :- finally is a block. In this block we have to keep all the clean up code. The statments which are available in finally block will be executed ~~ideathes~~ whether exception raised or not, handled or not. It is always ~~tried~~ associated with try block.

```
class ExceptionDemo
{
 static String name;
  public void static void main (String[] args)
  {
    try
    {
      name = "hello";
       System.out.println ("-try block");
       System.out.println (34/4);
       System.out.println ("Hai");
       System.out.println (name.tostring());
    }
    Catch (Asthematic Exception ae)
    {
    system.out.println ("Catch 1");
    }
    catch (Nullpointer Exception ne)
    {
```

```
System.out.println("catch 2");
}
finally
{
System.out.println("Finally block");
}
System.out.println("Hello world");
}
}
```

dp:- Try block
8
Hai
Finally block
Hello world.

In throwable class we have following to display the about the exception

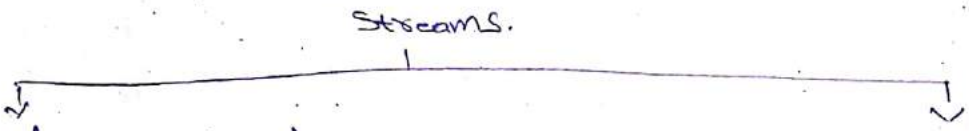
Throwable():-

i) Void printStackTrace():-
Exception name : Description of Exception
Stack trace

ii) String toString()
Exception Name : Description of Exception.

iii) String getMessage();-
display the description of Exception

Streams (For Reading)




Input stream (Read)
File Input Stream.
Readers:-
Input Stream Reader
File input stream Reader } These all will read line by line
File reader.
✓ Buffer Reader. → This will read an entire line at a time

output Stream (writ
File output stream
writers:-
output stream write
File writer.
Buffer writter
✓ Print writer.

System.in => keyboard

i) Attach the keyboard to Input Stream Reader.
InputStreamReader isr = new InputStreamReader(System.in);
2) Connect the ISR to BufferReader
BufferReader br = new BufferedReader(isr);
3) By using BufferReader Methods we can read the data.
a) int read() throws IOException => to read single char

b) String readLine() throws IOException ==> to read entire line as string.

Here read and readline methods throws IO exception which is a checked exception. Hence when ever we are calling these methods we have to handel it manually.

```
import java.io.BufferReader;
import java.io.InputStreamReader;
import java.io.IOException;
class ReadingDemo
{
int a,b,c;
InputStreamReader isr;
BufferedReader br;
ReadingDemo
{
  isr = new InputStreamReader();
   br = new BuffesedReader(isr);
}
void readA() throws IOException
{
   String s = br.readLine();
    a = Integer.parseInt(s.trim()); //trim() is used to remove
    a = Integer.                    // prefix & postfix spaces in string
}
Void readB() throws IOException
{
 b = Integer.parseInt(br.readLine().trim());
}
Void division()
{
 try
 {c = a/b;
  System.out.println("c = "+c);
 }
 catch (Arithematic Exception ae)
 {system.out.println("Provide value of B other than zero(0)");
  try()
  {
   read B();
  }
  Catch (IOException ie)
  {System.out.println(ie);
  }
```

```
        divison();
    }
}
public Static Void main (String[] args) throws IOException.
{
 ReadingDemo rd= new ReadingDemo();
  System.out.print ("Enter a value. =");
 rd.readA();
   System.out.print ("Enter b value =");
    rd.readB();
    rd.divison();
  }
}
```

o/p:- Enter a value = 45
Enter b value = 0
Provide value other than 0
25
c = 1

To resolve all these problems sun people introduced scanner class which is available in util package.

Scanner :-

1) By using scanner class we can directly communicate with input stream. Ex:- Scanner sc = new Scanner (System.in);

2) In scanner class we have specified methods to read primitive data
byte = nextByte(); int:-nextInt(); short = nextShort(); long=next
String next(); or nextLine(); float:-nextfloat(); double=next
for a single word for a single line boolean:-nextBoolean();

3) All these methods cannot throw any exceptions hence there is no need to perform exception handeling manually.

```
import java.util.*;
class Younger Exception extends Runtime Exception
{
 Younger Exception (String desc)
 {
  Super (desc);
 }
}
```

```
class OlderException extends RuntimeException
{
 olderException (String desc)
 {super (desc);}

}
class CustException
{
 public static void name (String[] args)
 {
   Scanner sc = new Scanner (System.in);
   System.out.println ("Enter your name = ");
   String name = sc.nextLine();
   System.out.println ("Enter your age = ");
    try
    {
     int age = nextInt();
     if (age < 18)
     {
      YoungerException ye = new YoungerException(
       throw ye;                    name + "... You are younge");
     }else if (age > 30)
     {throw new Older exception (name + "... You are older");
     }
      else
     {System.out.println("You are eligible "+name);
     } // from 1.7V onwards multiple exceptions can be handeled through 1 catch block.
     catch (YoungerException | OlderException ye)
     {
       throw ye System.out.println (ye);
     }
    }

}
```

final vs finally vs finalize :-

final :- It is a modifier which is used to define the constant members

finally :- It is a block which is always associated with try block.
In this block we are providing cleanup code for exception

handling.

Finalize is a method which is used to provide clean up operations and it was invoked before destroying the object by the garbage collector.

## Multi Threading:-

Introduction:- Performing more than 1 task simultaneously is called multi tasking. Multi-tasking can be classified into 2 types

i) Process Based multi tasking ii) Thread based multitasking.

Process:- The step by step operations which are performed from the starting of the application to the closing of the application is called process.

Executing more the one process is called process based multitasking

Thread:- Thread is nothing but a light weight process or sub-individual part of the program.

Executing more than 1 thread simultaneously is called Thread based multitasking or multithreading.

The main advantage of multithreading is It increases the performance of the system and decreases the waiting time.

Note:- 1) The threads execution can be controlled by the thread schedular which is working under the JVM and the behaviours of thread schedular completely depends on OS or vendor

2) Java internally supports threads concept for every program execution JVM internally creates a thread with the name of main

3) We can also create our own threads in a different ways

i) By extending a thread class ii) By implementing runnable interface

java.lang.Runnable(I) ==> public abstract void run();

↓ implements

java.lang.Thread(C)

1) Public void run(){}

2) Public void start() => To register our class as Thread at Thread Sch

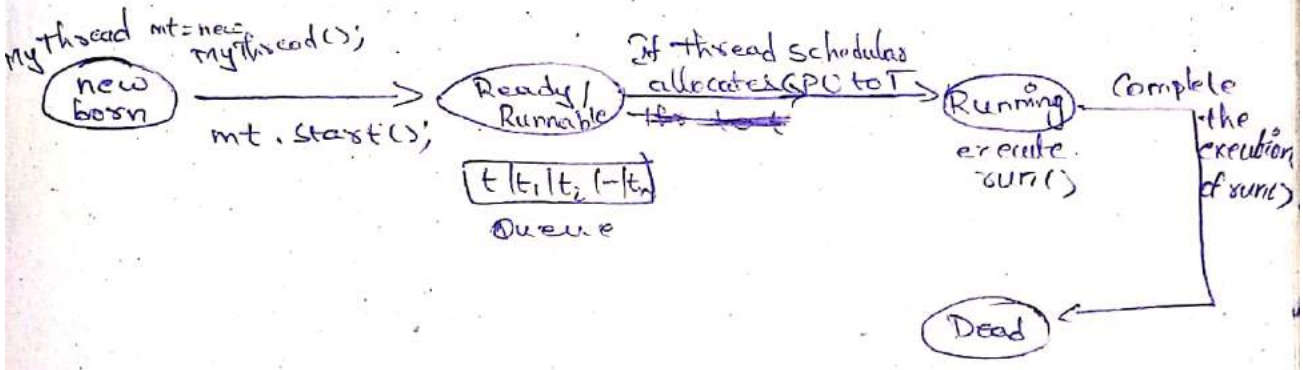
3) Public static Thread currentThread() => Calling:- Thread.currentTh

4) String getName(); => To get the name of current thread

5) Public void setName (String name) => To change thread name
6) Public int getPriority() => 1 to 10 (range) => To get priority of thread
7) void setPriority (int priority) => To change the priority of thread

```
class MyThread extends Thread
{
public void run()
{
for (int i=1 ; i<=10; i++)
System.out.println (" User thread = "+i);
}
}
class ThreadDemo
{
public static void main (String[] args)
{
MyThread mt = new Mythread();
mt.start
for(int i=1 ; i<=10; i++)
System.out.println ("Main thread ="+i);
}
}
```

Basic thread life cycle :-



## Thread Prevension

we can prevent the execution of a thread by using following 3 methods. They are

1) Yeild() 2) join() 3) Sleep()

1) Yeild():- Yeild method is used to pass the execution of a current thread by giving the chance to other waiting threads which is having the same priority.
If these is no thread in the waiting list or all the waiting threads having less priority then current thread will continue the execution.

Method signature:- public static void yeild()

```
class MyThread demo extends Thread
{
  public void main()
  {
    from (int i=1; ic=10; i++)
    {
      Thread.yield();
      System.out.println("Uses thread : "+i);
    }
  }
  class Yeild Demo
  {
    public static void main (String[] args)
    {
      MyThread t= new MyThread();
      t.start();
      for(int i=1; i<=10; i++)
        System.out.println("Main thread : "+i);
    }
  }
```

2) join():- Join method is used to If a thread wants to wait untill the compleation of some other thread then we should go for join method.

Let us assume if 2 threads are executing with the name of t1 and t2. If t1 calling t2.join method then thread t1 enter into the waiting state untill the compleation of t2

Note:- Here join method always throws interrupted exception which is a checked exception. Hence for every usage we have to handel it manually.

Different join methods:-

public void join() throws InterruptedException

public void join(long ms) throws InterruptedException

public void join(long ms, int ns) throws InterruptedException.

ms → milliseconds

ns → nanoseconds

Sleep():- If a thread wants to pause the execution for a particular period of time then we should go for sleep method.

Sleep method also throws interrupted Exception.

Different sleep method

public static void sleep(long ms) throws InterruptedException.

public static void sleep(long ms, int ns) throws InterruptedException.

```
class MyThread extends Thread.
{
public void run()
 {
 for (int i=0; i<=n; i++)
 {
  try{
  {
   Thread.sleep(2000);
  }
  catch (InterruptedException ie)
  {System.out.println(ie);}
  System.out.println("Sitha Thread :");
  }
 }
}
 class JoinDemo:
 {
 public static void main (String[] args) throws InterruptedExcep
 {
   MyThread t = new MyThread();
   t.start();
   t.interrupt();
   t.join(10000);
```

```
for (i=1; i<=10; i++)
{
System.out.println ("Rama Thread = ");
}
}
}
```

Synchronization :-

If more than 1 thread trying to operate on the same object simultaneously then there may be a chance of occurance of data inconsistency. This type of problem is called as race condition.

We can resolve this problem with the help of syncronization concept. Syncronization concept is activated by using syncronized keyword. Syncronized keyword is applicable only for methods and blocks.

Syncronized methods can be allowed to access only 1 thread at a time on the given object.

The main advantage of syncronization is to provide the thread safe. But the main disadvantage it increases the waiting time blw the threads and decreases the performance of the system.

Internal working of synchronization.

1) Every object in java has unique lock. when ever we are trying to syncronized area it will be activated.
2) If a thread wants to execute a synchronized method first it has to get the lock of given object.
3) Once a thread got the lock. it is allowed to execute any synchronized method on that object.
4) while a thread executing synchronizing area all the remaining threads have to wait to access any synchronized method on that given object.
5) Once a thread completes execution of synchronized (method / block) area it will releases the lock automatically.

```
class Display
{
synchronized public void wish (String name)
{
for (int i=1; i<=0; i++)
{
System.out.println ("Good Morning: ");
System.out.println ("
```

```
        }
        Thread.sleep(2000);
        }
        Catch (InterruptedException ie)
        {
        }
        System.out.println(name);
        }
    }
}
Class MyThread implements Runnable
{
    Display d;
    String name;
    MyThread (Display d, String name)
    {
    this.d=d;
    this.name=name;
    Thread t=new Thread();
    t.start();
    }
    public void run()
    {
    d.wish (name);
    }
}
    Class SynchronizedDemo
    {
    Public static void main(String[] args)
    {
    Display d=new Display();
    MyThread t1= new MyThread (d, "Prameeda");
    MyThread t2 =new MyThread (d, "Divya");
    MyThread t3 = new MyThread (d, "Ravi");
    }
}
```

In this situation we have to go

class level Lock:-

Every class in java has unique lock. If a thread trying to access static synchronized method first it has to get the class level lock On a thread get the class level lock it allowed to execute any static synchronized method on the class.

Once a thread executing static synchronized method- All the remaining threads are not allowed to access static synchronized method on the class.

But the remaining threads are allowed to execute instance methods, static methods, non synchronized method simultaneously.

Note:- Declare the wish method as static synchronized and change the code of the main method above as follows.

In main method

```
class synchronizedDemo
{
public static void main(String[] args)
{
Display d1 = new Display();
Display d2 = new Display();
Display d3 = new Display();
MyThread t1 = new MyThread(d1, "Parameela");
MyThread t2 = new MyThread(d2, "Divya");
MyThread t3 = new MyThread(d3, "Ravi");
}
}
```

Along with it add static before wish() in Display class.

Synchronized blocks:-

If you want to provide the thread safty very few lines of code. then never recomended to declare entire method as a

synchronized.
In this kind of situation place those few lines of code inside the synchronized block.
The main advantage of synchronized blocks over methods is it reduces waiting time blw the threads and increases the performance of system.
To get the lock of the current running object then we should declare synchronized block as follows

```
synchronized(this)
{
==========
}
```

To get the lock of particular object b then we should declare the synchronized block as follows.

```
synchronized (b)
{
==========
}
```

```
synchronized (classname.class)
{
==========
}
```

To get the lock of display class then we should declare the synchronized block as follows

Inter thread communication:-

We can provide the communication blw 2 threads by using wait(), notify(), notifyAll():
All this methods are available in object class hence we can call this 3 methods on any object
A thread requires the updation has to call wait method. A thread which is providing the updations has to call notify or notify all methods.
All this 3 methods must be inside the synchronized area otherwise we will get runtime exception says that illegal monitor state exception. IllegalMonitorStateException.
If a thread calls the wait method it immediatly releases the lock and enter into waiting state for notification.
If a thread calls notify() or nitifyall() it releases the lock may or may not immediatly.
Except these 3 methods there is no concept of release the lock.