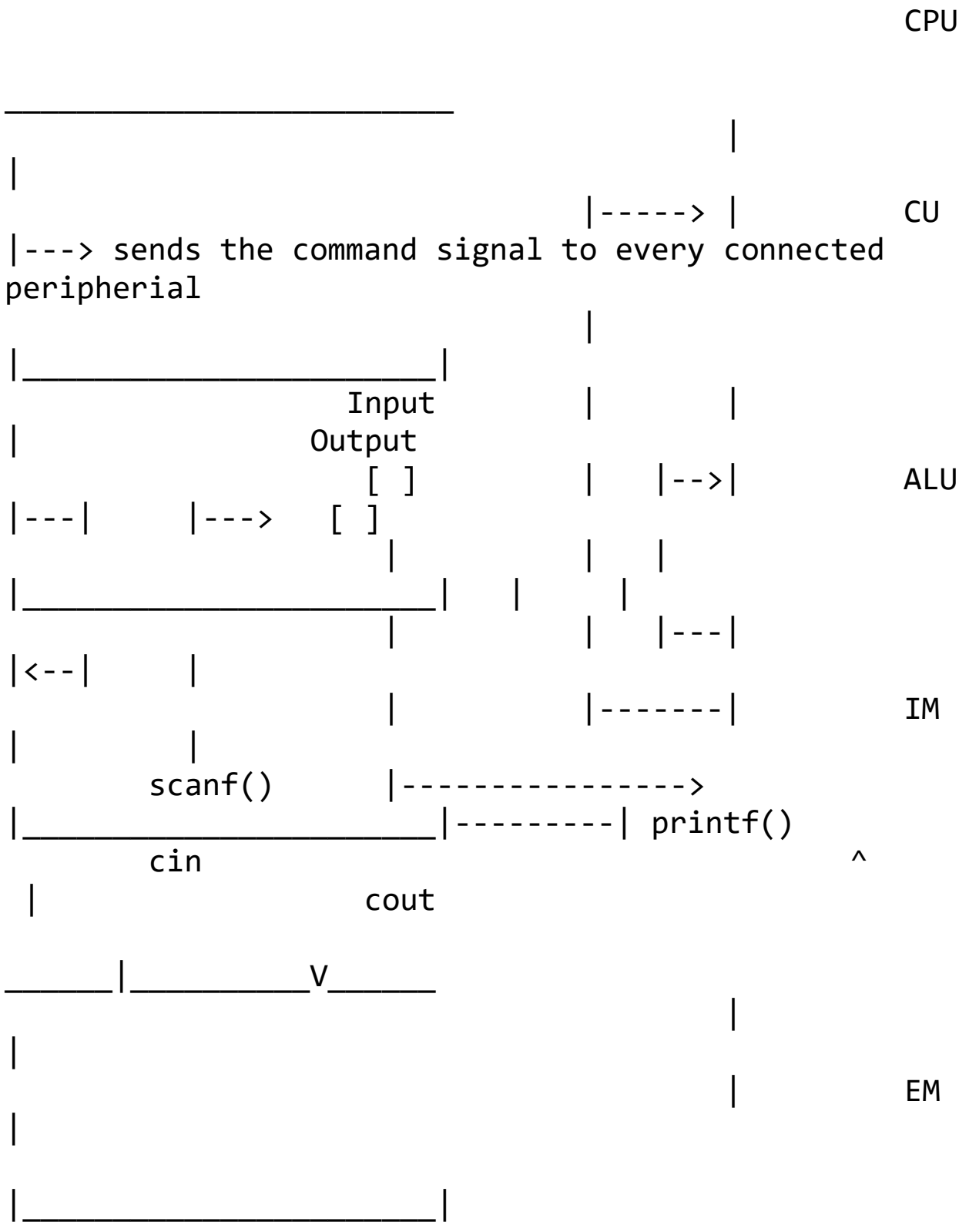
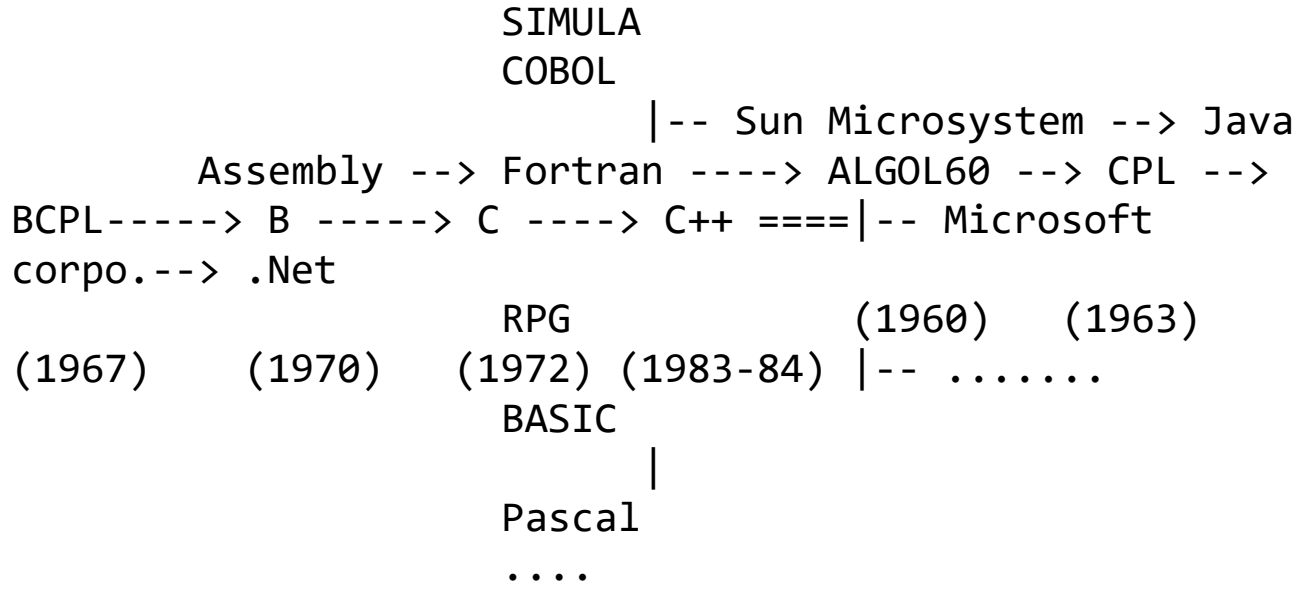
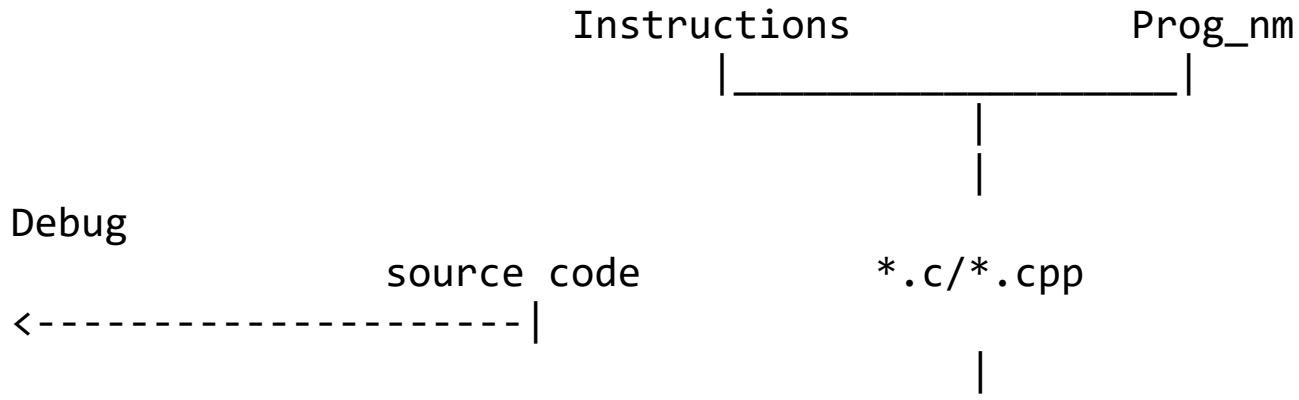
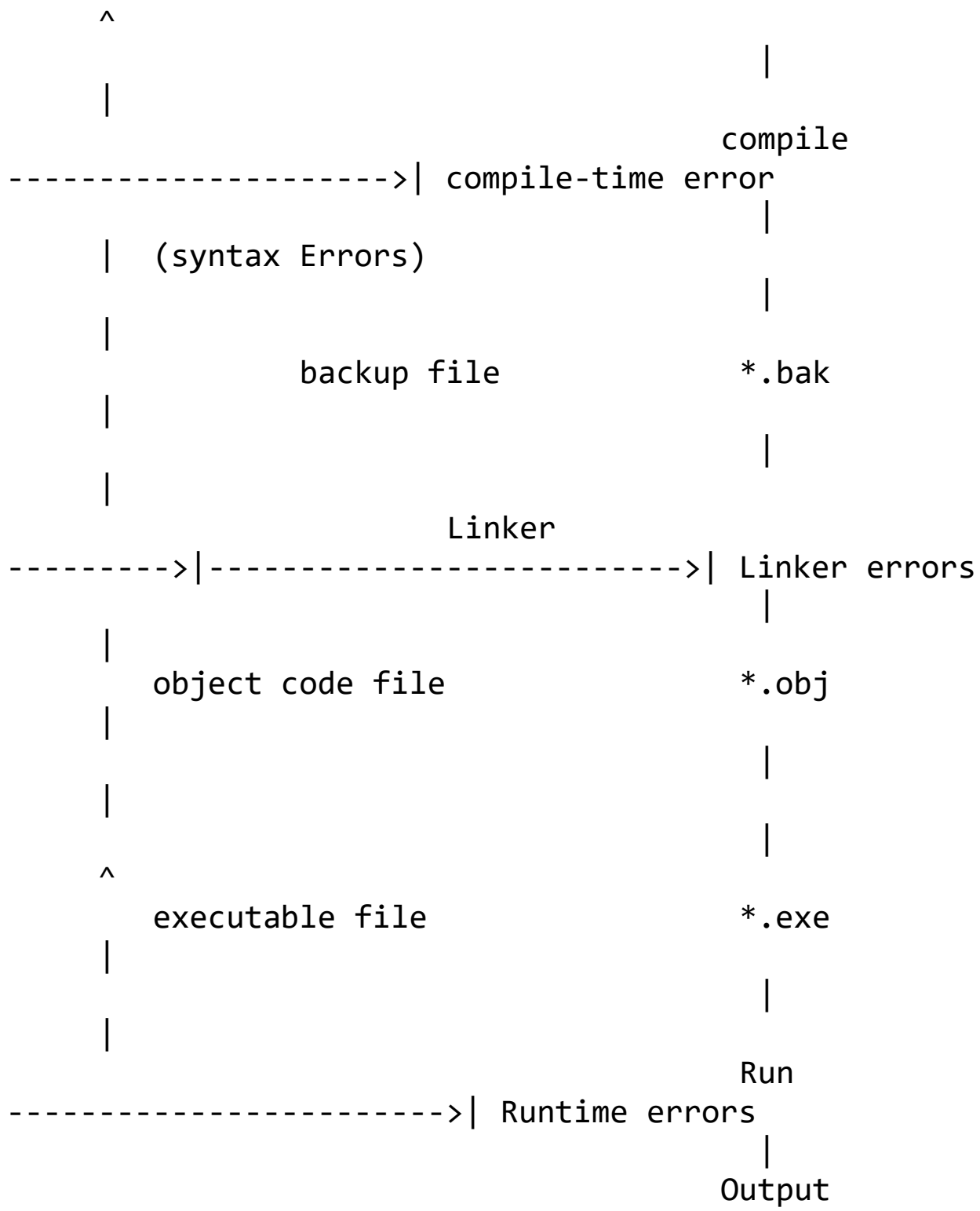

CPU
|
|-----> |
CU
|---> sends the command signal to every connected peripherial
Input
Output
[ ]
[ ]
|---|
|--->
|<--|
ALU
IM
scanf()
cin
printf()
cout
EM

---------- Java Programming ------------

Computer is two state, Multipurpose, programmable, electronic device, Which takes input from user, store it,
process on it and gives the output in desired format.





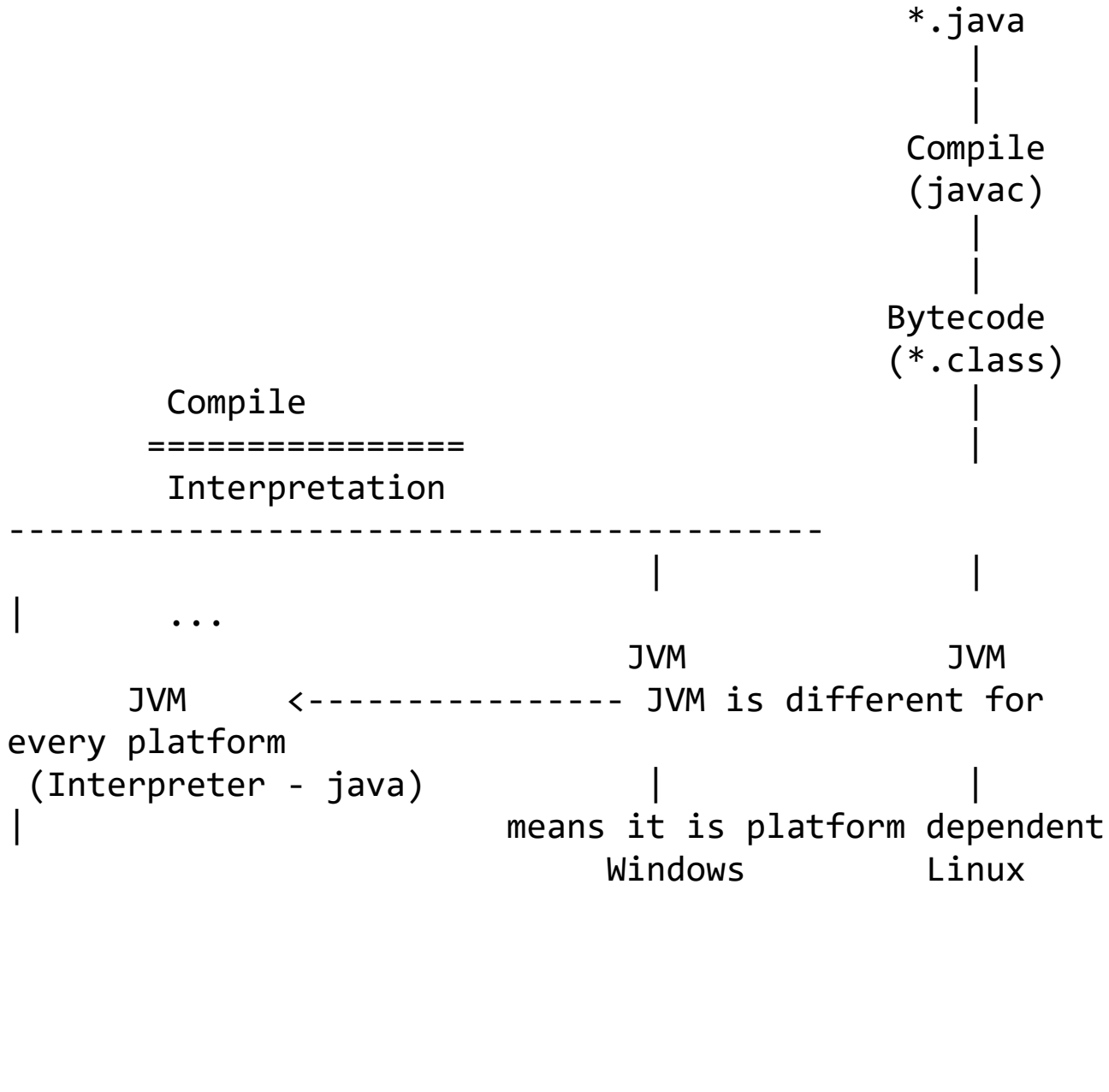

compile
compile-time error
(syntax Errors)
backup file
*.bak
Linker
Linker errors
object code file
*.obj
executable file
*.exe
Run
Runtime errors
Output

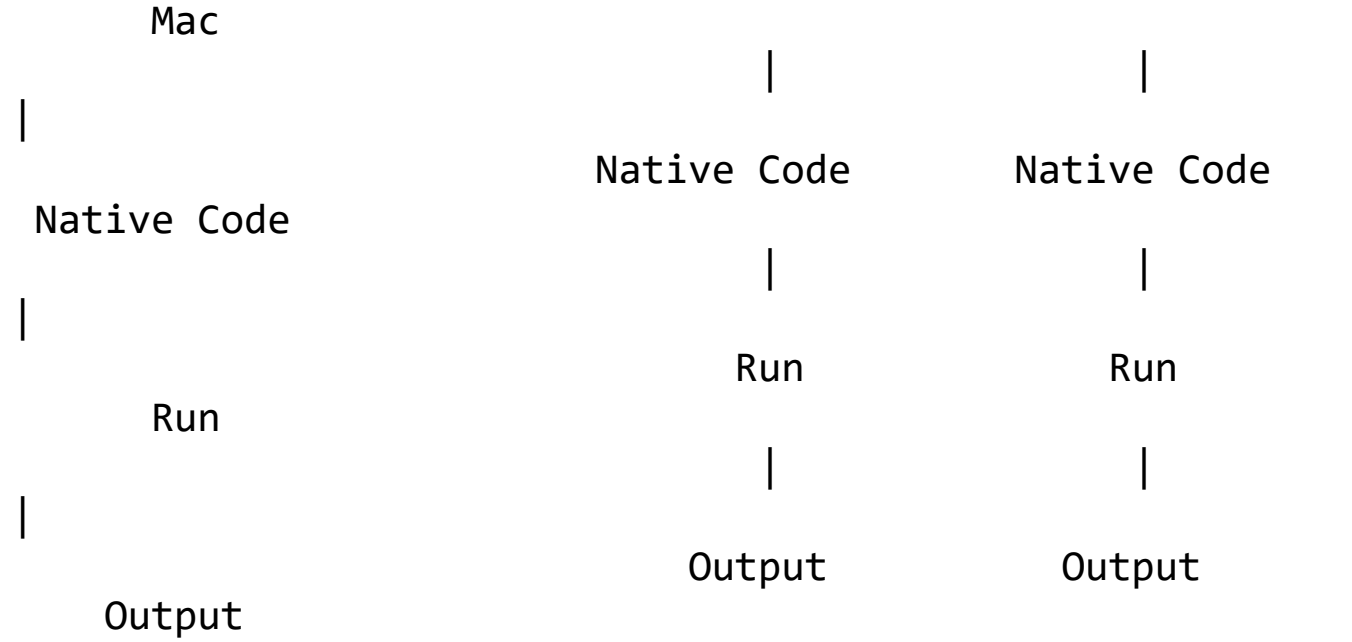
Any hardware or software environment in which a program runs, is known as a platform.

Platform = Processor architecture + OS

In C and CPP, the machine code is generated on the machine on which code is compiled, due to that the application will run only on machines having same config.(platform)which is known as platform dependent. This is the limitation of C and C++.

Now we want the platform independent code.





Java Editors and IDE's

==========

Editor: Notepad, Editplus, Notepad++ ...

IDE: Best Java IDEs

Eclipse. Platform – Linux/macOS/Solaris/Windows. ...

NetBeans. Platform – Linux/macOS/Solaris/Windows. ...

IntelliJ IDEA. Platform – Linux/macOS/Windows. ...

BlueJ. Platform – Linux/macOS/Windows. ...

(Oracle) JDeveloper. Platform – Linux/macOS/Windows.

Now we have see, how to write a code where:

1. Use any editor/ide

2. Install jdk/jre (https://www.oracle.com/in/java/technologies/javase/javase8-archive-downloads.html)

3. Use notepad as a editor and write a code as

```
class <cls_nm>
{
        public static void main(String []args)
        {

------------------;

------------------;
                program_body ;

------------------;

------------------;
        }
}

class Demo
{
        public static void main(String []args)
        {

System.out.print("Welcome to Java Programming");
        }
}
```

4. Save the code in C:\Program Files\Java\jdk-17.0.1\bin As <class_nm>.java

5. win+r --> cmd --> enter (attend the folder where the source file, compiler and interpreter is present) as

```
C:\Users\hp>cd\
C:\>cd "Program Files\Java\jdk-17.0.1\bin"
C:\Program Files\Java\jdk-17.0.1\bin>javac Demo.java
(compilation where you get the bytecode (*.class) )
C:\Program Files\Java\jdk-17.0.1\bin>java Demo      (Byte code interpretation)
                                Welcome to Java Programming
C:\Program Files\Java\jdk-17.0.1\bin>
```

```
//-------------------------------------------------------------------------------------
```

How to run, same code when source file (*.java) is in different folder

```
C:\Users\hp>e:
E:\>cd myjavafiles
E:\myjavafiles>javac First.java
```

'javac' is not recognized as an internal or external command, operable program or batch file.

```
E:\myjavafiles>set path=C:\Program Files\Java\jdk-17.0.1\bin
E:\myjavafiles>javac First.java
E:\myjavafiles>java First
        Welcome to Java Programming-First
E:\myjavafiles>
```

Note that the path is applicable till the current session of the command prompt.
to set the path in the permanent manner set the path in Environmeent variable

Setting the environment variable: this pc --> rh+ click ---> properties --> adv. system settings --> Advanced tab --> environment variable --> user variable path

-if already path is there -> edit --> new-> paste path (C:\Program Files\Java\jdk-17.0.1\bin)

otherwise user variable path--> new and write

variable name --> path
variable value -->
C:\Program Files\Java\jdk-17.0.1\bin) --> ok....
//---

Youtube Link:
https://youtu.be/RBxum7M3B94?si=jepmNZAtetZfJKFp
//---

Details of welcome program:
==========

```
class WelcomeProg
{
        public static void main(String []args)
        {
                System.out.print("Welcome to
Java");
        }
}
```

Line 1: class WelcomeProg:

class: It is keyword which
allows you to create your own type.
WelcomeProg: this is name of
UDT, it must be valide identifier. Internally in java
lib, the have
choosen, First
letter of class name in uppercase and all other in
lowercase if it is
made from from

one word, if multiple words then first character of each word in ucase
and all other in lcase.

e.g. First, Demo, FirstProgram, ExampleDemoWelcome
It is recommended, not compalsory

Line 3: public static void main(String []args)

public: it is used to define the visiblity of method main(), coz the javac and java are not members of class
as a outsiders they must have access to class members therefore visibility is public.

static: The static members gains the memory space when class is loaded into memory, no need of object
creation. therefore the method main() decl. as static

void: It is returning type of method main(), it is void coz java program does not return any value to OS

main(): It is method name, and as it is main(), it is considered as a starting point of of your program

String []args: String is

Built-in class from java.lang package, it is language support package, which is
imported bydefault. []args it is array of arguments, which is passed automatically at the
time of execution from commandline, in absence null is collected.
simply it is array of objects.

Line 5: System.out.print("Welcome to Java");

"Welcome to Java" : It is data, to be displayed
print() is a method from PrintStream class used to display the data on screen.
out is predefined object of PrintStream class, decleared as a static in System class
System is a class from java.lang package.

```
//-------------------------------------------------------------------------
class FirstProgram
{
	public static void main(String []args)
	{
		System.out.println("Welcome to Java Programming");

		System.out.println(args[0]);
```

```
                System.out.println(args[1]);
                System.out.println(args[2]);

                System.out.println("Bye Bye..!!");

        }
}
```

Note that, the IO in java is in form of string only.

Now we need to proceed using the path fillowed in the C and C++

i.e.

charcter set -----> constant / keyword ------> variable ----> Instructions ----> program ----> module ---> software

Java Character set:

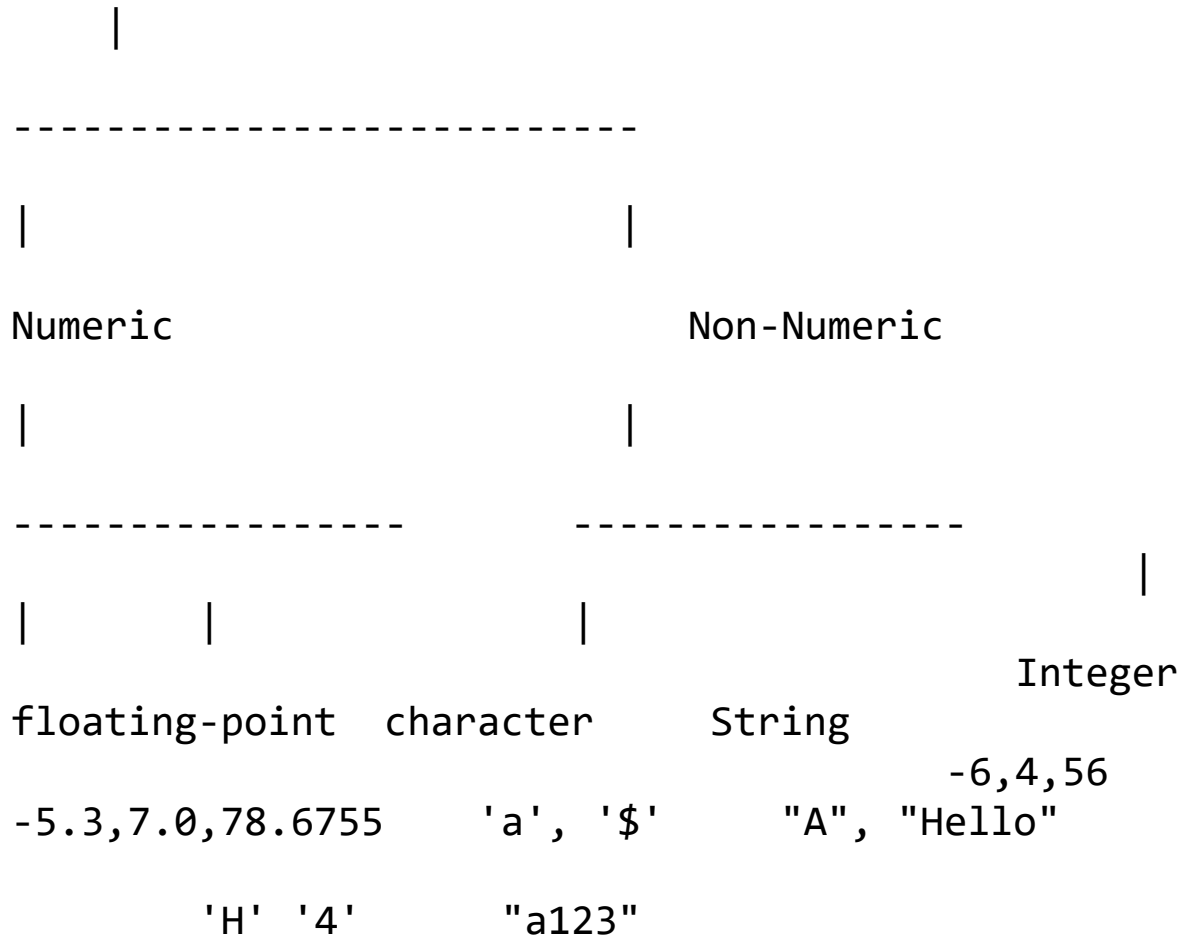
- ASCII(American Standard Code for Information Interchange): Provides the binary string to all symbols present in the US English, which are used in different electronic devices.

- ASCII used to code in english, but java supports different human understable languages for coding. means java

having rich character set as cmp to c/c++. The Standard Code system names unicode system used in the
java which provides the 16 bit binary string to each symbol for different languages.

// Constants: These are the elements in the program having fix value.

Java Constants



// Keyword: These are reserved words, whose meaning is already known to compiler.

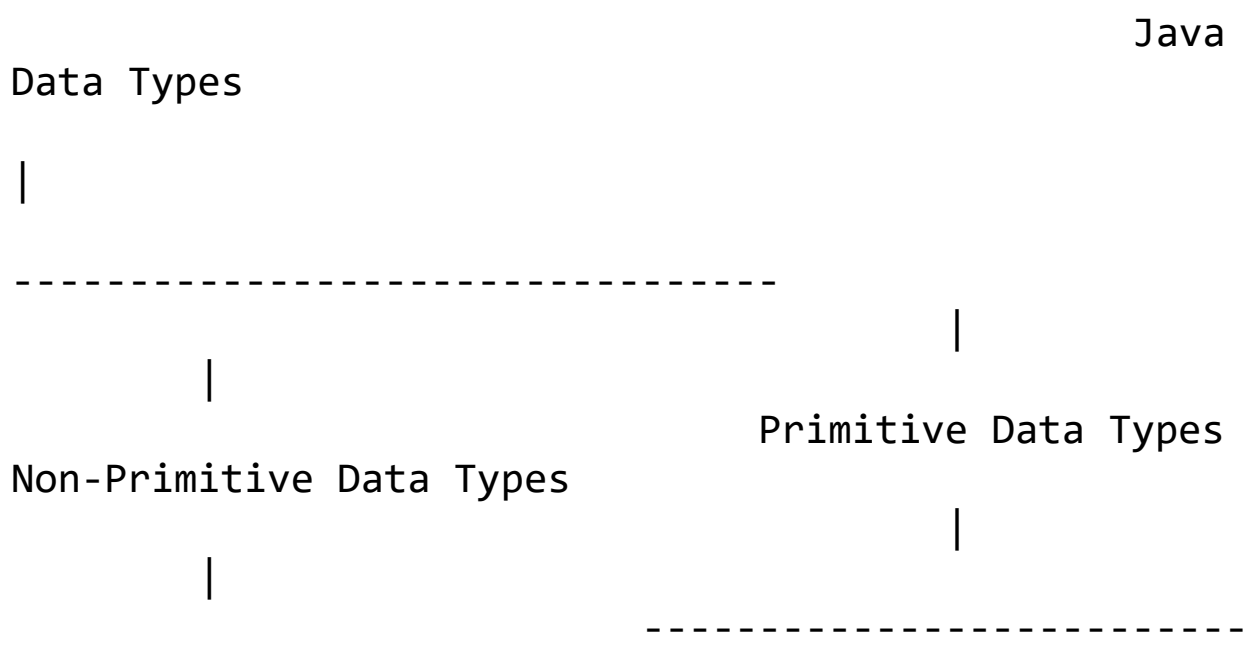
	abstract	continue	for
new	switch		
	assert***	default	goto*
package	synchronized		
	boolean	do	if
private	this		
	break	double	
implements	protected	throw	
	byte	else	import
public	throws		
	case	enum****	
instanceof	return	transient	
	catch	extends	int
short	try		
	char	final	
interface	static	void	
	class	finally	long
strictfp**	volatile		
	const*	float	native
super	while		

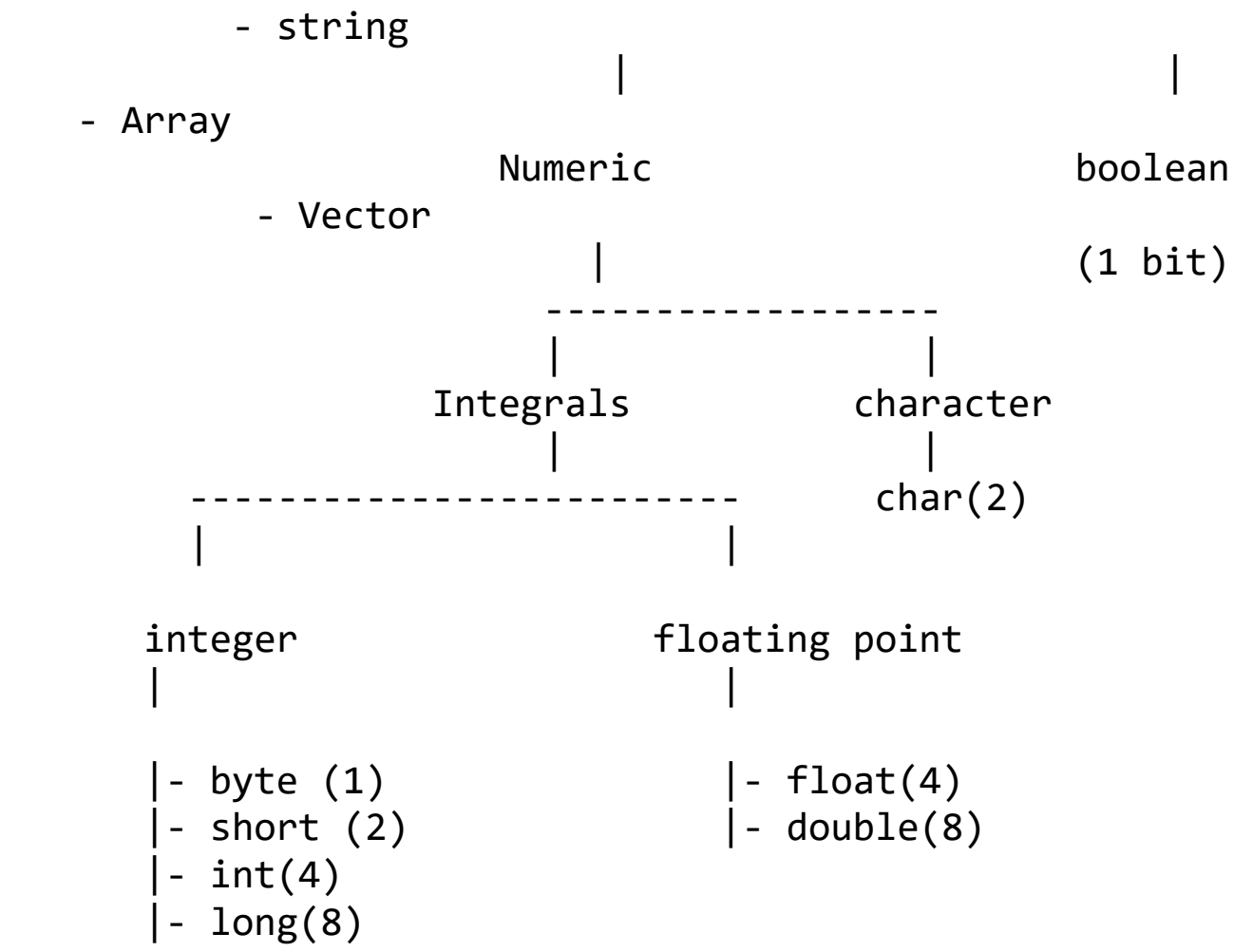
*	not used
**	added in 1.2
***	added in 1.4
****	added in 5.0

(https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html)

Java Data Types: Tool used for the memory allocation.

- Primitive data types: These are provided by the language itself. e.g. int, char, byte..
- Non-Primitive data types: The are defined by the programmers according to the need. e.g. class, interface





- The defalt integer value is considered as a int and default fractional value considered as double.
- When you decl the variable, java demands for the init. of variables, otherwise it will generate the error message,

"variable xxx might not have been initialized"

- When the value of variable having higher type is assigned to variable of lower type, then it will generate the error message

e.g.

a=c; gives the following error, when a is byte variable and c is int variable

"possible lossy conversion from int to byte"

In such case, where you want to convert the value from higther type to lower type, go for the type casting;

i.e. a=(byte)c;

Note carefully that, lower type to higher type promoted automatically.

- The long costant is represented using 'l' or 'L' as a prefix and for the float 'f' or 'F' is used.

- Java allows you to decl. the variables anywhere in the program, just decl before using it.

```
class IntData
{
        public static void main(String []args)
        {
                byte b=10; // If the literal it will be considered as a byte value
                System.out.println("value of b: "+b);

                int x=100;
                // b=x; // if downcast then it will generate an error
```

```
                b=(byte)x;
                System.out.println("value of b: "+b);

                long t;
                t=x;
                System.out.println("value of t: "+t);
        }
}

//-------------------------------------------------------
-------------------------------------------------------
---------------

        /// Different ways of input in Java

        // 1. init and display

class InitID
{
        public static void main(String []args)
        {
                byte b=10;
                short s=23;
                int i=50;
                long no=456L;
                System.out.println("\n b="+b+"\t
s="+s+"\t i="+i+"\t no="+no);

                float ft=4.5F;
                double db=56.233;
                System.out.println("\n ft="+ft+"\t
db="+db);

                boolean ans=true;
```

```
                System.out.println("\n ans is: "+ans);
        }
}

//-------------------------------------------------------
-----------------------------------------

        // Data input using commandline

class CmdlnID
{
        public static void main(String []args)
        {
                int a,b;
                a=args[0];
                b=args[1];
                System.out.println("\n a="+a+"\t b="+b);
        }
}

output on compile
E:\jprodyp>javac CmdlnID.java
CmdlnID.java:6: error: incompatible types: String cannot
be converted to int
                a=args[0];
                      ^
CmdlnID.java:7: error: incompatible types: String cannot
be converted to int
                b=args[1];
                      ^
2 errors

//-------------------------------------------------------
-----------------------
```

```
class CmdlnID
{
        public static void main(String []args)
        {
                int a=0,b=0;
                a=Integer.parseInt(args[0]);
                b=Integer.parseInt(args[1]);
                System.out.println("\n a="+a+"\t b="+b);
                System.out.println("\n Sum: "+(a+b));
        }
}

E:\jprodyp>javac CmdlnID.java

E:\jprodyp>java CmdlnID
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of
bounds for length 0
        at CmdlnID.main(CmdlnID.java:6)


E:\jprodyp>java CmdlnID 11 22

 a=11    b=22

 Sum: 33
//-------------------------------------------------
----------------------------------------
 // Dealing with the exception

class CmdlnID
{
        public static void main(String []args)
        {
```

```
                int a=0,b=0;

                try
                {
                        a=Integer.parseInt(args[0]);
                        b=Integer.parseInt(args[1]);
                }
                catch(Exception e)
                {
                        System.out.println("\n Please
Pass the arguments thw commandline");
                }
                System.out.println("\n a="+a+"\t b="+b);
                System.out.println("\n Sum: "+(a+b));
        }
}

output:
E:\jprodyp>javac CmdlnID.java

E:\jprodyp>java CmdlnID

 Please Pass the arguments thw commandline

 a=0     b=0

 Sum: 0

//=================================================
================================================

        // Using BufferedReader and InputStreamReader

import java.io.InputStreamReader;
```

```
import java.io.BufferedReader;
class BrIsrID
{
        public static void main(String []args)
        {
                InputStreamReader isr=new
InputStreamReader(System.in);
                BufferedReader br=new
BufferedReader(isr);

                int x=0;
                double y=0.0;

                System.out.println("Enter the int value:
");
                x=br.readLine();
                System.out.println("Enter the double
value: ");
                y=br.readLine();

                System.out.println("\n x="+x+"\t y="+y);

        }
}
output on compile


E:\jprodyp>javac BrIsrID.java
BrIsrID.java:15: error: incompatible types: String
cannot be converted to int
                x=br.readLine();
                             ^
BrIsrID.java:17: error: incompatible types: String
cannot be converted to double
```

```
                y=br.readLine();
                              ^
2 errors

//-----------------------------


import java.io.InputStreamReader;
import java.io.BufferedReader;
class BrIsrID
{
        public static void main(String []args)
        {
                InputStreamReader isr=new
InputStreamReader(System.in);
                BufferedReader br=new
BufferedReader(isr);

                int x=0;
                double y=0.0;

                System.out.println("Enter the int value:
");
                x=Integer.parseInt(br.readLine());
                System.out.println("Enter the double
value: ");
                y=Double.parseDouble(br.readLine());

                System.out.println("\n x="+x+"\t y="+y);

        }
}
output on Compile
```

```
E:\jprodyp>javac BrIsrID.java
BrIsrID.java:15: error: unreported exception
IOException; must be caught or declared to be thrown
                x=Integer.parseInt(br.readLine());
                                                 ^
BrIsrID.java:17: error: unreported exception
IOException; must be caught or declared to be thrown
                y=Double.parseDouble(br.readLine());
                                                   ^
2 errors

//-------------------------------------------------------
-----------------------------

        There are two different ways to eliminate these
Exceptions

        a) use try catch
        b) use throws clause


import java.io.InputStreamReader;
import java.io.BufferedReader;
class BrIsrID
{
        public static void main(String []args) throws
Exception
        {
                InputStreamReader isr=new
InputStreamReader(System.in);
                BufferedReader br=new
BufferedReader(isr);

                int x=0;
```

```
                double y=0.0;

                System.out.println("Enter the int value:
");
                x=Integer.parseInt(br.readLine());
                System.out.println("Enter the double
value: ");
                y=Double.parseDouble(br.readLine());

                System.out.println("\n x="+x+"\t y="+y);

        }
}
```

output

```
E:\jprodyp>javac BrIsrID.java

E:\jprodyp>java BrIsrID
Enter the int value:
12
Enter the double value:
67.34

 x=12    y=67.34
```

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
class BrIsrID
{
        public static void main(String []args)
        {
                InputStreamReader isr=new
```

```
InputStreamReader(System.in);
                BufferedReader br=new
BufferedReader(isr);

                int x=0;
                double y=0.0;

                try
                {
                        System.out.println("Enter the
int value: ");

x=Integer.parseInt(br.readLine());
                        System.out.println("Enter the
double value: ");

y=Double.parseDouble(br.readLine());
                }
                catch(Exception e){}
                System.out.println("\n x="+x+"\t y="+y);

        }
}
```

output:

```
E:\jprodyp>javac BrIsrID.java

E:\jprodyp>java BrIsrID
Enter the int value:
23
Enter the double value:
45.78
```

```
 x=23    y=45.78

//=================================================
=======================================

        4. Input using Scanner class

import java.util.Scanner;
class ScannerID
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);

                int x=0;
                double y=0.0;
                System.out.println("Enter the int value:
");
                x=sc.nextInt();
                System.out.println("Enter the double
value: ");
                y=sc.nextDouble();

                System.out.println("\n x="+x+"\t y="+y);
        }
}

output:


E:\jprodyp>javac ScannerID.java

E:\jprodyp>java ScannerID
Enter the int value:
```

```
11
Enter the double value:
45.67

 x=11    y=45.67

//=================================================
=======================================

        5. Input using
javax.swing.JOptionPane.showInputDialog()

import javax.swing.JOptionPane;
class InDialogID
{
        public static void main(String []args)
        {
                int x=0;
                double y=0.0;


x=Integer.parseInt(JOptionPane.showInputDialog("Enter
the int value: "));


y=Double.parseDouble(JOptionPane.showInputDialog("Enter
the double value: "));

                System.out.println("\n x="+x+"\t y="+y);
        }
}

//-------------------------------------------------
---------------------------------
```

// Operators in Java: Operators are used to process the data. There are following operators
present in the java.

- Assignment Operators (= and short-hand operators)
- Unary Operators (- ++ -- (type))
- Arithmetic Operators (+ - * / %)
- Relational Operators (< <= > >= == !=)
- Logical Operators (&& || !)
- conditional Operator (? :)
- Bitwise Operators (& | ^ >> << >>>)
- special Operators (. and instanceof)

- Assignment Operators (= and short-hand operators): will assigns constant value at its rh+, value of variable at its rh+ or
answer of exper at its rh+ to variable at left.

e.g.
int x=10; int y=x; int z=x+y;

shorthand expr: suppose, x=x+10 can be written as x+=10;

```
x=x/10  --> x/=10

....

class DemoAssignment
{
        public static void main(String []args)
        {
                int x=10;
                int y=x;
                int z=x+y;

                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

                x+=100;
                y*=2;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

        }
}


- Unary Operators ( - ++ -- (type) )

        - will gives oppisite value
        ++ incr by 1
        -- decr by 1

                inc/dec
                  |
        -------------------------
        |                       |
```

```
   pre                        post
(++x, --x)                (x++, x--)


      ++x  <-----> x=x+1      <------> x++
      --x  <-----> x=x-1      <------> x--

      when these operators are used in the expression,


            pre --> expr --> post

      suppose x=5, and y=9

            z = ++x + y-- ;

      - find the basic expr
      - operate all pre operators
      - calc. the basic expr. with current values
      - operator all post operators

      - x becomes 6
      - assigned 15 to z
      -y becomes 8

import java.util.Scanner;
class DemoUnary
{
      public static void main(String []args)
      {
            int x=0;
            int y=0;
            int z=0;
```

```
                Scanner sc= new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextInt();

                z=-x;
                x++;
                --y;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

                z=++x+y--;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);
        }
}

//-------------------------------------------------------
-----------------

        (type): It refers to type casting, means
changing the data type of variable obly at the of calc.


import java.util.Scanner;
class DemoCasting
{
        public static void main(String []args)
        {
                int x=0,y=0;
                double z=0;

                Scanner sc= new Scanner(System.in);
```

```
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextInt();  // 13, 5

                z=x/y;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

                z=(double)x/y;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

                z=x/(double)y;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

                z=(double)x/(double)y;
                System.out.println("\n x="+x+"\t
y="+y+"\t z="+z);

        }
}
```

output:

```
E:\jprodyp>javac DemoCasting.java

E:\jprodyp>java DemoCasting

 Enter the values of x and y:
13
5

 x=13    y=5     z=2.0
```

```
 x=13    y=5     z=2.6

 x=13    y=5     z=2.6

 x=13    y=5     z=2.6

//-------------------------------------------------------
-----------------------------------

- Arithmetic Operators ( + - * / % )

        // program to calc the simple interst.

import java.io.InputStreamReader;
import java.io.BufferedReader;
class DemoArith
{
        public static void main(String []args) throws
Exception
        {
                BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
                int p=0,n=0;
                double r=0.0,si=0.0;

                System.out.println("Enter the value of
p: ");
                p=Integer.parseInt(br.readLine());

                System.out.println("Enter the value of
r: ");
                r=Double.parseDouble(br.readLine());
```

```
                System.out.println("Enter the value of
n: ");
                n=Integer.parseInt(br.readLine());

                si=(p*r*n)/100;

                System.out.println("Simple Interst is:
"+si);
        }
}
```

Using % and / operator

lets see simple example, we have to calculate 13/5

```
                2 <------------- (13/5)
              _________
         5 )  13
            - 10
              ---------
                 3 <------------ (13%5)
```

lets see some examples, ovserve the result and write the conclusion

13/5=2	13%5=3
27/7=3	27%7=6
67/9=7	67%9=4
123/10=12	123%10=3
459/10=45	459%10=9
3857/10=385	3857%10=7
7/10=0	7%10=7

```
        - Div by 10 eliminates the last digit from
number.
          and mod by 10 gives the last digit.
        - In N/D, when N<D then div is 0 and rem is N

   // Enter any 3 digit number from keyboard and find
addition of its all digits.
  // no=285 then ans = 5+8+2 => 15

import java.util.Scanner;
class DemoDivMod
{
        public static void main(String []args)
        {
                int no=0,rem=0,tot=0;
                Scanner sc=new Scanner(System.in);

                System.out.println("\n Enter any 3 digit
number: ");
                no=sc.nextInt(); //285

                rem=no%10; //5
                tot=tot+rem; // 0+5=5
                no=no/10; //28

                rem=no%10; //8
                tot=tot+rem; // 5+8=13
                no=no/10; //2

                rem=no%10; //2
                tot=tot+rem; // 13+2=15
                no=no/10; //0
```

```
                System.out.println("\n Total is "+tot);
        }
}
```

Unlike C/C++, Here in java you can operate the % operator on fractional and -ve values,
when you operate the % operator on -ve values the sign of ans is taken as the sign of N form N/D.

```
//-------------------------------------------------------------------------------
```

```
        // Relational Operators: (<, <=, >, >= ==, !=): These operators are used to find the
                relation between two operands. It will forms the condition which is useful in the
                conditional conditional control statements.

                thw ans of condition is boolean value true when it is true and false when false.


        suppose x=23    y=5;

                x>y     ----> true      means if we write z=x>y then true assigned to z.

                x!=y    ----> true

                y<1     ----> false

                x%10==0 ----> false
```

```
                100%y==0 ---> true

class DemoRel
{
        public static void main(String []args)
        {
                int x=23,y=5;
                boolean b;
                System.out.println("\n x is: "+x+"\t y
is: "+y);

                b=x>y;
                System.out.println("\n (x>y) is: "+b);

                b=x!=y;
                System.out.println("\n (x!=y) is: "+b);

                b=y<1;
                System.out.println("\n (y<1) is: "+b);

                b=x%10==0;
                System.out.println("\n (x%10==0) is:
"+b);

                b=100%y==0;
                System.out.println("\n (100%y==0) is:
"+b);

        }
}
output

E:\jprodyp>javac DemoRel.java
```

```
E:\jprodyp>java DemoRel

 (x is: 23y is: 5

 (x>y) is: true

 (x!=y) is: true

 (y<1) is: false

 (x%10==0) is: false

 (100%y==0) is: true




//-------------------------------------------------------
----------------------------------------

        /// Logical Operator ( && || !): These operators
are used to join two or more conditions

                when the conditions are joind by

                - && --> gives true only when both true
otherwise false
                - || --> gives false only when both
false otherwise true
                - !  --> gives
                                !(true) ---> false
                                !(false) ---> true
```

```
        suppose x=23    y=5;

                (x>y)&&(y<100)  ----> true
                (x>y)&&(y>100)  ----> false

                (x>y)||(y>100)  ----> true
                (x<y)||(y>100)  ----> false

                !(x!=y) ----> false
                !(y<1)  ----> true


class DemoLogical
{
        public static void main(String []args)
        {
                int x=23,y=5;
                boolean b;
                System.out.println("\n x is: "+x+"\t y
is: "+y);

                b=(x>y)&&(y<100);
                System.out.println("\n ((x>y)&&(y<100))
is: "+b);

                b=(x>y)&&(y>100);
                System.out.println("\n ((x>y)&&(y>100))
is: "+b);

                b=(x>y)||(y>100);
                System.out.println("\n ((x>y)||(y>100))
is: "+b);

                b=(x<y)||(y>100);
```

```
                System.out.println("\n ((x<y)||(y>100)) 
is: "+b);

                b=!(x!=y);
                System.out.println("\n (!(x!=y)) is: 
"+b);

                b=!(y<1);
                System.out.println("\n (!(y<1)) is: 
"+b);

        }
}
```

output:

```
E:\jprodyp>javac DemoLogical.java

E:\jprodyp>java DemoLogical

 x is: 23        y is: 5

 ((x>y)&&(y<100)) is: true

 ((x>y)&&(y>100)) is: false

 ((x>y)||(y>100)) is: true

 ((x<y)||(y>100)) is: false

 (!(x!=y)) is: false

 (!(y<1)) is: true

///-----------------------------------------------------
```

// Conditional operator or ternary operator or if-then-else operator(?:):

This is the only operator which has decision abiity.

syntax:

<condition> ? <options>;

<condition> ? <true_part> : <false_part> ;

// WAP to find the max from 2 nos

```
import java.util.Scanner;
class DemoConditionalOperator
{
	public static void main(String []args)
	{
		Scanner sc=new Scanner(System.in);
		int x=0,y=0;

		System.out.println("\n Enter any two nos: ");
		x=sc.nextInt();
		y=sc.nextInt();

		int z = (x>y) ? x : y ;
		System.out.println("\n Max no: "+z);
	}
}
```

```
output:
E:\jprodyp>javac DemoConditionalOperator.java

E:\jprodyp>java DemoConditionalOperator

 Enter any two nos:
45
78

 Max no: 78

E:\jprodyp>java DemoConditionalOperator

 Enter any two nos:
90
23

 Max no: 90

///------- Nesting of conditional operators


        // WAP to find the max from 3 nos

import java.util.Scanner;
class DemoConditionalOperator1
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);
                int x=0,y=0,z=0;

                System.out.println("\n Enter any three
```

```
nos: ");
                x=sc.nextInt();
                y=sc.nextInt();
                z=sc.nextInt();

                int max = (x>y) ? (x>z?x:z) : (y>z?y:z)
;
                System.out.println("\n Max no: "+max);
        }
}
```

output:

```
E:\jprodyp>javac DemoConditionalOperator1.java

E:\jprodyp>java DemoConditionalOperator1

 Enter any three  nos:
11
22
33

 Max no: 33

E:\jprodyp>java DemoConditionalOperator1

 Enter any three nos:
111
22
33

 Max no: 111

E:\jprodyp>java DemoConditionalOperator1
```

```
 Enter any three nos:
11
222
33

 Max no: 222


//-----------------------------------------------------
-----------------------------------------------------
---------

        /// Bitwise Operator: [ & | ^ >> << >>> ]

    These operators are used in the bit level
operations.

        & ==> 1 & 1 -> 1 otherwise 0
        | ==> 0 | 0 -> 0 otherwise 1

              0^0
        ^ -->     ==> 0 otherwise 1
              1^1

    suppose x=10       y=12

        (0000 1010)     (0000 1100)

    (x&y)       (x|y)       (x^y)

     1010        1010        1010
    &1100       |1100       ^1100
  ========    ========    =======
     1000        1110        0110
```

```
    (8)          (14)        (6)


    x=10 (0000 1010)     y=12  (0000 1100)

     z=x<<2               z=y>>2
       (0010 1000)        (0000 0011)
       ==>40             ==> 3


class DemoBitwise
{
        public static void main(String []args)
        {
                int x=10,y=12;
                System.out.println("\n (x&y) is"+ (x&y)
);

                System.out.println("\n (x|y) is"+ (x|y)
);

                System.out.println("\n (x^y) is"+ (x^y)
);

                System.out.println("\n (x<<2) is"+
(x<<2) );

                System.out.println("\n (y>>2) is"+
(y>>2) );

        }
}

//-------------------------------------------------
```

```
-------------------------------------------------------------------
//-----------------------------------------------------------------
-------------------------------------------------------------------

// Control Statements in Java:


                                    Control Statements
                                            |

-------------------------------------
                        |
   |
                  Conditional
Un-Conditional
                        |
|
        ------------------------------
---------------------------------
        |               |           |           |
|               |       |
      Decision        Loop        case         break
continue     lbl.break  return
        |               |           |
    and
        |- if()         |- for()   switch
    lbl. continue
        |- if() else    |- while()
        |- nesting      |- do..while()
        |- ladder
```

// Decision Conditional Control statment:

        // Using if(): used to decide, execute the block of code or not. That block is mentioned in the program as

                        syntax:
                                if(<condi>)
                                {
                                        -------------;
                                        -------------;
                                        block of code;
                                        -------------;
                                        -------------;
                                }

        - Block of code will be executed only when the condition is true otherwise it will be skipped

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
class DemoIf
{
        public static void main(String []args)
        {
                InputStreamReader isr=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(isr);

                int a=0;
                try
                {
```

```
                        System.out.println("Enter the
value of a: ");

a=Integer.parseInt(br.readLine());
                }
                catch(Exception e){}

                if(a%7==0)
                {
                        System.out.println("Entered no
is div by 7 ");
                }
                if(a%7!=0)
                {
                        System.out.println("Entered no
is not div by 7 ");
                }
        }
}

//-----------------------------------------------------
----------------------------


        // Using if() else: It is used when you want to
execute any one code block from two different blocks
        //                  according to condition.


                syntax:
                        if(<condi>)
                        {
                                ----------------;
                                ---------------;
```

```
                        ----------------;
                }
                else
                {
                        ------------------;
                        ---------------;
                        ------------------;
                }
```

when <condi> is TRUE --> will execute the if() block only

FALSE -> will execute the else block only

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
class DemoIfElse
{
        public static void main(String []args)
        {
                InputStreamReader isr=new InputStreamReader(System.in);
                BufferedReader br=new BufferedReader(isr);

                int a=0;
                try
                {
                        System.out.println("Enter the value of a: ");
```

```
a=Integer.parseInt(br.readLine());
                }
                catch(Exception e){}

                if(a%7==0)
                {
                        System.out.println("Entered no
is div by 7 ");
                }
                else
                {
                        System.out.println("Entered no
is not div by 7 ");
                }
        }
}


        //      Using Nesting of if() else:

                        Nesting refers to using one
control statment in to same or another control statement


        some combinations:


                if()                                    if()
                if()                    if()
                {                                       {
                {                       {
                        if()
if()                    }
```

```
if()
                        {
{                       else
{
                        }
}                       {
}
                }
else                            if()            }

{                               {               else

}                               }               {
                                                }
                }                               if()

                                                {

                                                }

                                        }



                if()
if()
                {
{
                        if()
if()
                        {
{
                        }
}
```

```
                }
else
                else
{
                {
}
                        if()
}
                        {
else
                        }
{
                        else
if()
                        {
{
                        }
}
                }
else

{

}

}


class DemoIfElseNesting
{
        public static void main(String []args)
        {
                int a=Integer.parseInt(args[0]);
                int b=Integer.parseInt(args[1]);
```

```
                int c=Integer.parseInt(args[2]);

                if(a>b)
                {
                        if(a>c)
                        {
                                System.out.println("\n a
is max");
                        }
                        else
                        {
                                System.out.println("\n c
is max");
                        }
                }
                else
                {
                        if(b>c)
                        {
                                System.out.println("\n b
is max");
                        }
                        else
                        {
                                System.out.println("\n c
is max");
                        }
                }
        }
}

//-------------------------------------------------------
-------------------------------------------------------
-------
```

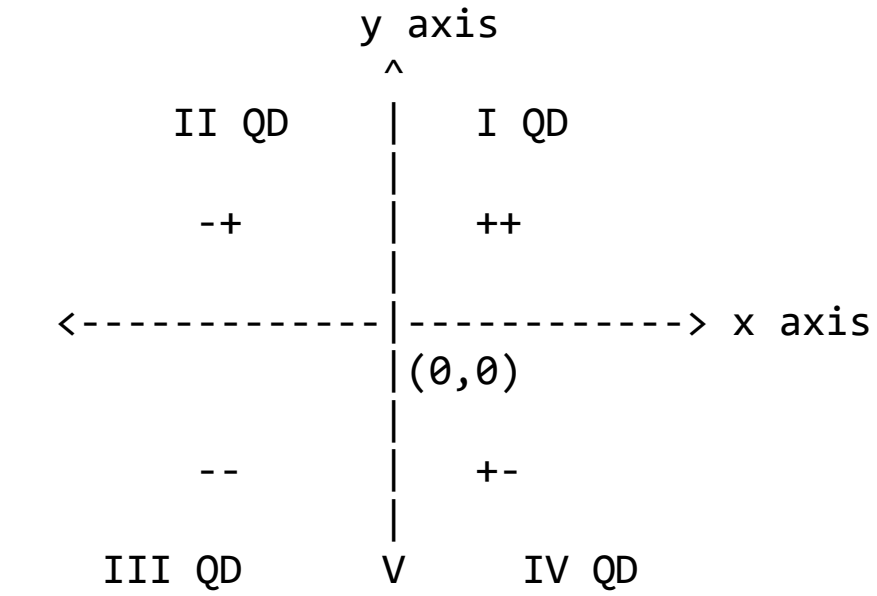
// Using if() else Ladder

syntax:

```
        if(<>)
        {
                ---------;
                ---------;
        }
        else if(<>)
        {
                ---------;
                ---------;
        }
        else if(<>)
        {
                ---------;
                ---------;
        }
        else if(<>)
        {
                ---------;
                ---------;
        }
        [<else>]
        {
                ---------;
                ---------;
        }
```

// Enter the co-ordinates of point in 2D system,

and display the exact location of that point.



There are 7 different possibilities.

```
import java.util.Scanner;
class DemoIfElseLadder
{
        public static void main(String []args)
        {
                int x=0,y=0;
                Scanner sc=new Scanner(System.in);

                System.out.println("\n Enter the x cord:
");
                x=sc.nextInt();
                System.out.println("\n Enter the y cord:
");
                y=sc.nextInt();
```

```
                if(x>0&&y>0)
                {
                        System.out.println("\n Point
present in I st qd");
                }
                else if(x<0&&y>0)
                {
                        System.out.println("\n point
present in II nd qd");
                }
                else if(x<0&&y<0)
                {
                        System.out.println("\n Point is
present in 3 rd qd");
                }
                else if(x>0&&y<0)
                {
                        System.out.println("\n Point is
present in 4 th qd");
                }
                else if(x!=0&&y==0)
                {
                        System.out.println("\n Point is
present on x axis");
                }
                else if(x==0&&y!=0)
                {
                        System.out.println("\n Point is
present on y axis");
                }
                else
                {
                        System.out.println("\n Point
```

```
present at org");
                }
        }
}
```

```
                // Using the Loops in Java: Loops are used to avoid the continue repitition of code in the program.
                                        There are three different loops in C.

                                1. For() loop
2. While() loop         3. do..While() loop
```

1. For() loop:

```
                syntax:

                        for( [<init>] ; <condi> ; [<inc/dec/stat/expr>] )
                        {

                                --------------;

                                --------------;

                                --------------;
```

```
                                        ------------;

                                        ------------;

                                }


import javax.swing.JOptionPane;
class DemoFor
{
        public static void main(String []args)
        {
                int
no=Integer.parseInt(JOptionPane.showInputDialog("Enteran
y number:"));

                int t=0,tot=0;
                for(t=no;no!=0;no=no/10)
                {
                        tot=tot+(no%10);
                }
                System.out.println("\n Addition of all
digits from "+t+" is "+tot");
        }
}


///-----------------------------------------------------
-----------------------------------------------------
------------

        //  Using while loop:
```

Again the aim is same i.e. used to avoid the code rep.

syntax:

```
while(<cond>)
{
        ----------------;
        ----------------;
        ----------------;
        ----------------;
        ****************;
}
```

- It will execute the body of loop, till the condition is true.

- <init> block is absent in while() but you have init. the iterator before starting of loop.

- <inc/dec> block is absent, but you have to add atleast one statement which will make the <cond> false
  after some iterations.
otherwise it will attend the infinite looping

```
/// WAP to display 1 to 15 nos using while loop

class DemoWhile
{
        public static void main(String []args)
        {
```

```
                int i;

                i=1;
                while(i<=15)
                {
                        System.out.println(" "+i);
                        i++;
                }
        }
}

//-------------------------------------------------------------------------------
        /// WAP to display list of odd nos from 1 to 50 using while loop.

class DemoWhile
{
        public static void main(String []args)
        {
                int i;

                i=1;
                while(i<=50)
                {
                        if(i%2!=0)
                        {
                                System.out.println(" "+i);
                        }
                        i++;
                }
        }
}
```

```
//-------------------------------------------------------------------

        // WAP to find the entered number is prime

import java.util.*;
class DemoWhile
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("\n Enter any no: ");
                int no=sc.nextInt();
                int d=2;
                int flg=0;
                while(d<=(no/2))
                {
                        if(no%d==0)
                        {
                                flg=1;
                                break;
                        }
                        d++;
                }
                if(flg==0)
                {
                        System.out.println("\n Entered
no is prime ");
                }
                else
                {
```

```
                        System.out.println("\n Entered
no is not prime ");
                }
        }
}

//-------------------------------------------------------
-----------------------------------

        /// using  do while()

        syntax:

        do
        {
                ----------;
                ----------;
                ----------;
                ----------;
                ----------;
        }while(<cond>);

// Display list of prime nos from given range

import java.util.*;
class DemoWhile
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("\n Enter the range
starts from: ");
                int n1=sc.nextInt();
```

```
                System.out.println("\n Enter the range
ends to: ");
                int n2=sc.nextInt();

                int d=2,flg=0;
                System.out.println("\n List of prime
nos: ");
                for(no=n1;no<=n2;no++)
                {
                        d=2;
                        flg=0;
                        while(d<=(no/2))
                        {
                                if(no%d==0)
                                {
                                        flg=1;
                                        break;
                                }
                                d++;
                        }
                        if(flg==0)
                                System.out.println("
"+no);
                }
        }

//-----------------------------------------------------
-----------------------------------------------------
---------------

        // WAP to display *

class StarPattern
{
```

```
        public static void main(String []args)
        {
                System.out.print("*")
        }
}


class StarPattern
{
        public static void main(String []args)
        {
                System.out.print("*");
        }
}

//-------------------------------------------------

        WAP to print
        *****

class StarPattern
{
        public static void main(String []args)
        {
                int j=0;
                for(j=0;j<5;j++)
                {
                        System.out.print("*");
                }
        }
}
```

```
//-------------------------------------------------

        WAP to print
        *****
        *****
        *****
        *****
        *****

class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;
                for(i=0;i<5;i++)
                {
                        for(j=0;j<5;j++)
                        {
                                System.out.print("*");
                        }
                        System.out.print("\n");
                }
        }
}

//-------------------------------------------------

        WAP to print
           j
        01234
        *****  i=0
        *   *  i=1
        *   *  i=2
        *   *  i=3
```

```
        *****  i=4

class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;
                for(i=0;i<5;i++) // No of lines
                {
                        for(j=0;j<5;j++) // no of cols
                        {
                                if(i==0 || i==4)
                                {

System.out.print("*");
                                }
                                else if(j==0 || j==4)
                                {

System.out.print("*");
                                }
                                else
                                {

System.out.print(" ");
                                }
                        }
                        System.out.print("\n");
                }
        }
}

//-------------------------------------------------------
------------------------------
```

WAP to print

```
*
**
***
****
*****
```

```
class StarPattern
{
	public static void main(String []args)
	{
		int i, j=0;

		for(i=0;i<5;i++)
		{
			for(j=0;j<5;j++)
			{
				if(j<=i)
				{

System.out.print("*");
				}
			}
			System.out.print("\n");
		}
	}
}
```

```
//-------------------------------------------------
```

WAP to print

```
*****
****
***
**
*
```

```
class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;

                for(i=0;i<5;i++)
                {
                        for(j=0;j<5;j++)
                        {
                                if(j<(5-i))
                                {

System.out.print("*");
                                }
                        }
                        System.out.print("\n");
                }
        }
}

//--------------------------------------------------
---------------
```

```
                *
               ***
              *****
```

```
             *******
            *********
           ***********

class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;
                for(i=0;i<6;i++)
                {
                        for(j=0;j<(6+i);j++)
                        {
                                if(j<(5-i))
                                {

System.out.print(" ");
                                }
                                else
                                {

System.out.print("*");
                                }
                        }
                        System.out.print("\n");
                }
        }
}

//-------------------------------------------------------
-------------------

     1
    222
```

```
   33333
  4444444
 555555555
66666666666

class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;
                for(i=0;i<6;i++)
                {
                        for(j=0;j<(6+i);j++)
                        {
                                if(j<(5-i))
                                {

System.out.print(" ");
                                }
                                else
                                {

System.out.print(i+1);
                                }
                        }
                        System.out.print("\n");
                }
        }
}

//-------------------------------------------------------
---------------------------
```

```
     A
    ABC
   ABCDE
  ABCDEFG
 ABCDEFGHI
ABCDEFGHIJK

class StarPattern
{
        public static void main(String []args)
        {
                int i, j=0;
                char ch;
                for(i=0;i<6;i++)
                {
                        ch='A';
                        for(j=0;j<(6+i);j++)
                        {
                                if(j<(5-i))
                                {

System.out.print(" ");
                                }
                                else
                                {

System.out.print(ch);
                                        ch++;
                                }
                        }
                        System.out.print("\n");
                }
        }
}
```

```
//-------------------------------------------------------------------------------
```

switch(): It is used in the menu driven programming.

syntax:

```
switch(<opt>)
{
        case <CC>:
                ---------;
                ---------;
                break;
        case <CC>:
                ---------;
                ---------;
                break;
        case <CC>:
                ---------;
                ---------;
                break;
        case <CC>:
                ---------;
                ---------;
                break;
        [<default>]:
                -----------;
                ----------;
}
------------------;
```

The switch case is used when there are more

possibilities, and from which we have to
choose any one according to users choice.

```
import java.util.Scanner;
class DemoSwitch
{
	public static void main(String []args)
	{
		Scanner sc=new Scanner(System.in);
		int opt=0;
		double a=0.0, b=0.0, ans=0.0;

		System.out.println("\n*** Menu ****");
		System.out.println("1.add \n 2.sub \n
3.multi \n 4.div ");
		System.out.println("select your option:
");
		opt=sc.nextInt();

		System.out.println("Enter any two nos:
");
		a=sc.nextDouble();
		b=sc.nextDouble();

		switch(opt)
		{
			case 1:
				ans=a+b;
				break;
			case 2:
				ans=a-b;
```

```
                                break;
                        case 3:
                                ans=a*b;
                                break;
                        case 4:
                                ans=a/b;
                                break;
                        default:
                                System.out.println("\n
Incorrect Option");
                }
                System.out.println("\n Ans is: "+ans);

        }
}

//-----------------------------------------------------
--------------------------------

        // Using the character as a option (both lcase
and ucase for a single case)

import java.util.Scanner;
class DemoSwitch
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);
                char opt=0;
                double a=0.0, b=0.0, ans=0.0;

                System.out.println("\n*** Menu ****");
                System.out.println(" a.add \n b.sub \n
c.multi \n d.div ");
```

```
                System.out.println("select your option:
");
                opt=sc.nextLine().charAt(0);

                System.out.println("Enter any two nos:
");
                a=sc.nextDouble();
                b=sc.nextDouble();

                switch(opt)
                {
                        case 'A':
                        case 'a':
                                ans=a+b;
                                break;
                        case 'B':
                        case 'b':
                                ans=a-b;
                                break;
                        case 'C':
                        case 'c':
                                ans=a*b;
                                break;
                        case 'D':
                        case 'd':
                                ans=a/b;
                                break;
                        default:
                                System.out.println("\n
Incorrect Option");
                }
                System.out.println("\n Ans is: "+ans);

        }
```

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

import java.util.Scanner;
class DemoSwitch
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);
                int i=0, opt=0;
                double a=0.0, b=0.0, ans=0.0;

                while(i<3)
                {
                        i++;
                        System.out.println("\n*** Menu
****");
                        System.out.println("1.add \n
2.sub \n 3.multi \n 4.div ");
                        System.out.println("select your
option: ");
                        opt=sc.nextInt();

                        System.out.println("Enter any
two nos: ");
                        a=sc.nextDouble();
                        b=sc.nextDouble();

                        switch(opt)
                        {
                                case 1:
                                        ans=a+b;
                                        break;
```

```
                                        case 2:
                                                ans=a-b;
                                                break;
                                        case 3:
                                                ans=a*b;
                                                break;
                                        case 4:
                                                ans=a/b;
                                                break;
                                        default:

System.out.println("\n Incorrect Option");

                                }
                                System.out.println("\n Ans is:
"+ans);
                        }
                }
}
//-------------------------------------------------------
-------------------------------------

        // Using the switch within infinite loop
(Termination using break)

import java.util.Scanner;
class DemoSwitch
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);
                int opt=0;
                double a=0.0, b=0.0, ans=0.0;
```

```
                while(true)
                {
                        System.out.println("\n*** Menu
****");
                        System.out.println("1.add \n
2.sub \n 3.multi \n 4.div \n 5.stop");
                        System.out.println("select your
option: ");
                        opt=sc.nextInt();

                        if(opt==5)
                                break;

                        System.out.println("Enter any
two nos: ");
                        a=sc.nextDouble();
                        b=sc.nextDouble();

                        switch(opt)
                        {
                                case 1:
                                        ans=a+b;
                                        break;
                                case 2:
                                        ans=a-b;
                                        break;
                                case 3:
                                        ans=a*b;
                                        break;
                                case 4:
                                        ans=a/b;
                                        break;
                                default:
```

```
System.out.println("\n Incorrect Option");

                        }
                        System.out.println("\n Ans is:
"+ans);
                }
        }
}



//-------------------------------------------------------
-------------------------------------

        // Using the switch within infinite loop
(Termination using System.exit(0) )

import java.util.Scanner;
class DemoSwitch
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);
                int opt=0;
                double a=0.0, b=0.0, ans=0.0;

                while(true)
                {
                        System.out.println("\n*** Menu
****");
                        System.out.println("1.add \n
2.sub \n 3.multi \n 4.div \n 5.stop");
                        System.out.println("select your
option: ");
```

```
                        opt=sc.nextInt();
                        System.out.println("Enter any
two nos: ");
                        a=sc.nextDouble();
                        b=sc.nextDouble();

                        switch(opt)
                        {
                                case 1:
                                        ans=a+b;
                                        break;
                                case 2:
                                        ans=a-b;
                                        break;
                                case 3:
                                        ans=a*b;
                                        break;
                                case 4:
                                        ans=a/b;
                                        break;
                                case 5:
                                        System.exit(0);
                                default:

System.out.println("\n Incorrect Option");

                        }
                        System.out.println("\n Ans is:
"+ans);
                }
        }
}

//-------------------------------------------------
```

Unconditional control statements: The control statement does not need any condition.

NOTE: In Java goto is not present.

- using the continue: It will keep the enclosing loop in the running condition without considering the remaining body of loop.

```
class DemoContinue
{
	public static void main(String []args)
	{
		int i,j;

		for(i=0;i<10;i++)
		{
			System.out.print("-");
			for(j=0;j<10;j++)
			{
				if(j>i)
				{
					continue;
				}
				System.out.print("*");
			}
			System.out.println();
		}
	}
}
```

```
//-------------------------------------------------------
----------------

//-------------------------------------------------------
----------------


                // Using Labelled Continue: It will keep
loop in running condition, not only
                                                enclosing,
but outer loops using label(tag)

class DemoLabelledContinue
{
        public static void main(String []args)
        {
                int i,j;

                outer:for(i=0;i<10;i++)
                {
                        System.out.print("-");
                        for(j=0;j<10;j++)
                        {
                                if(i>5)
                                {
                                        continue outer;
                                }
                                if(j>i)
                                {
                                        continue;
                                }
                                System.out.print("*");
                        }
                        System.out.println();
```

```
                }
        }
}


//-------------------------------------------------------
-----------------

// Using the break ans labelled break


class DemoBreak
{
        public static void main(String []args)
        {
                int i,j;

                for(i=0;i<10;i++)
                {
                        System.out.print("-");
                        for(j=0;j<10;j++)
                        {
                                if(j>i)
                                {
                                        break;
                                }
                                System.out.print(""+j);
                        }
                        System.out.println();
                }
        }
}

//-------------------------------------------------------
```

```
----------------------------------------

class DemoLabelledBreak
{
        public static void main(String []args)
        {
                int i,j;

                outer:for(i=0;i<10;i++)
                {
                        System.out.print("-");
                        for(j=0;j<10;j++)
                        {
                                if(i>5)
                                {
                                        break outer;
                                }
                                if(j>i)
                                {
                                        break;
                                }
                                System.out.print(""+j);
                        }
                        System.out.println();
                }
        }
}


//-------------------------------------------------------
---------------------------------

        // Using return: pass back the value from called
method to calling function method.
```

/// Array in Java Programming

// Def: It is collection of elements having same data type which are conti.
arranged in the memory.

// Decl. of array in java:

we have seen the array decl. in c/c++ as,

<data_type> <ar_nm>[<index>];

int x[5]; --> // in C/C++ it will allocate the memory

x
[] [] [] [] []

but in java, if we decl array as,

int x[] or int []x;

then it will create the reference variable only.

x
[ ]

this location same as pointer variable, it able to refer towards the
location where data is or will be stored.

Note carefully that, array gains the memory space dynamically.
and for that we have to use the new keyword as shown below.

syntax:
<data_type> []<ar_nm> = new <data_type>[<index>]; OR
<data_type> <ar_nm>[] = new <data_type>[<index>];

e.g.
int []x=new int[6];
x
[] -------> [][][][][][]

```
class DemoArrayInit
{
	public static void main(String []args)
	{
		int []x={11,67,89,45,2};

		System.out.println("\n Array elements are: ");
		for(i=0;i<5;i++)
		{
			System.out.print("  "+x[i]);
		}
	}
}
```

// Note that there is one property named length in array, which

```
        // will gives the number of elements in the
array. so it can be used as

class DemoArrayInit
{
        public static void main(String []args)
        {
                int []x={11,67,89,45,2};

                System.out.println("\n Array elements
are: ");
                for(int i=0;i<x.length;i++)
                {
                        System.out.print("  "+x[i]);
                }
        }
}

//-----------------------------------------------------
-------------------------------------
        /// Array input and display - using commandline

        x
         [ ] ------> [][][][][][]

class DemoArrayInCmdln
{
        public static void main(String []args)
        {
                // decl.
                int []x=new int[6];

                System.out.println("\n Enter any 5 nos:
");
```

```
                for(int i=0;i<x.length;i++)
                {
                        x[i]=Integer.parseInt(args[i]);
                }
                // display
                System.out.println("\n Array elements
are: ");
                for(int i=0;i<x.length;i++)
                {
                        System.out.print("  "+x[i]);
                }
        }
}

//-----------------------------------------------------
-------------------------------------
        /// Array input and display - BufferedReader and
InputStreamReader
import java.io.BufferedReader;
import java.io.InputStreamReader;
class DemoArrayInBr
{
        public static void main(String []args) throws
Exception
        {
                // decl.
                int []x=new int[6];
                BufferedReader br=new BufferedReader
(new InputStreamReader(System.in));

                System.out.println("\n Enter nos: ");
                for(int i=0;i<x.length;i++)
                {
```

```
x[i]=Integer.parseInt(br.readLine());
                }
                // display
                System.out.println("\n Array elements
are: ");
                for(int i=0;i<x.length;i++)
                {
                        System.out.print("  "+x[i]);
                }
        }
}


//-----------------------------------------------------
-------------------------------------
        /// Array input and display - Scanner
import java.util.Scanner;
class DemoArrayInsc
{
        public static void main(String []args)
        {
                // decl.
                int []x=new int[6];
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter nos: ");
                for(int i=0;i<x.length;i++)
                {
                        x[i]=sc.nextInt();
                }
                // display
                System.out.println("\n Array elements
are: ");
                for(int i=0;i<x.length;i++)
                {
```

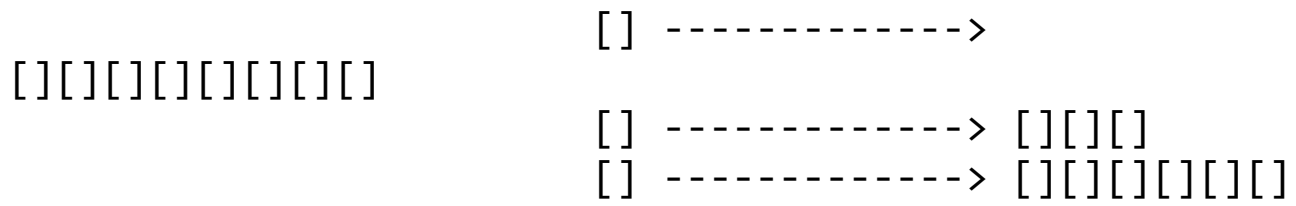
```
                        System.out.print("  "+x[i]);
                }
        }
}

//-------------------------------------------------------
-------------------------------------
        /// input Array and display all prime nos from
it

import java.util.Scanner;
class DemoArrayInsc
{
        public static void main(String []args)
        {
                // decl.
                int d=0,flg=0;
                int []x=new int[6];
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter nos: ");
                for(int i=0;i<x.length;i++)
                {
                        x[i]=sc.nextInt();
                }
                // process - display all primes
                System.out.println("\n Prime elements
are: ");
                for(int i=0;i<x.length;i++)
                {
                        d=2;
                        flg=0;
                        while(d<=(x[i]/2) )
                        {
                                if(x[i]%d==0)
```

```
                                {
                                        flg=1;
                                        break
                                }
                                d++;
                        }
                        if(flg==0)
                                System.out.print("
"+x[i]);
                }
                // display
                System.out.println("\n Array elements
are: ");
                for(int i=0;i<x.length;i++)
                {
                        System.out.print("  "+x[i]);
                }
        }
}

//-------------------------------------------------------
----------------------------------------

        // Using the 2D array:

        Decl syntax:

                <data_type> <ar_nm>[][];    // It will
create the reference variable only

        Inti. of array:
                <data_type> <ar_nm>[][]= { {},{},{}};

                        [][][]
```

```
                        [][][]
                        [][][]

class Demo2DArray
{
        public static void main(String []args)
        {
                int
a[][]={{11,22,33},{44,55,66},{77,88,99}};
                int i,j;

                for(i=0;i<3;i++)
                {
                        for(j=0;j<3;j++)
                        {
                                System.out.print("
"+a[i][j]);
                        }
                        System.out.println();
                }
        }
}


or


class Demo2DArray
{
        public static void main(String []args)
        {
                int
a[][]={{11,22,33},{44,55,66},{77,88,99}};
                int i,j;
```

```
                for(i=0;i<a.length;i++)
                {
                        for(j=0;j<a[i].length;j++)
                        {
                                System.out.print("
"+a[i][j]);
                        }
                        System.out.println();
                }
        }
}

//-------------------------------------------------------
-----------------

        decl. of array:

                <data_type> <ar_nm>[][]=new
<data_type>[<rows>][<cols>];

import java.util.Scanner;
class Demo2DArrayIO
{
        public static void main(String []args)
        {
                Scanner sc = new Scanner(System.in);
                int a[][]=new int [3][4];
                int i,j;

                System.out.println(" Enter the array of
3x4: ");
                for(i=0;i<3;i++)
                {
```

```
                        for(j=0;j<4;j++)
                        {
                                a[i][j]=sc.nextInt();
                        }
                }
                System.out.println("\n the array of 3x4:
");
                for(i=0;i<a.length;i++)
                {
                        for(j=0;j<a[i].length;j++)
                        {
                                System.out.print("
"+a[i][j]);
                        }
                        System.out.println();
                }
        }
}

//-----------------------------------------------------
----------------------------------


// Variable size array (Jagged Array)

        This the array with number of rows mentioned in
the decl. but the number of cols
        are decided at runtime.

                The data strucure of array look like
as..
     x
     [] ---------------> [] -------------> [][][][][]
                         [] -------------> [][][][]
```



such array can be decl. as...

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

x[0]=new int[5];
x[1]=new int[4];
x[2]=new int[8];
x[3]=new int[3];
x[4]=new int[6];
```

// WAP using array to input and display the marks of N students in m subjects.

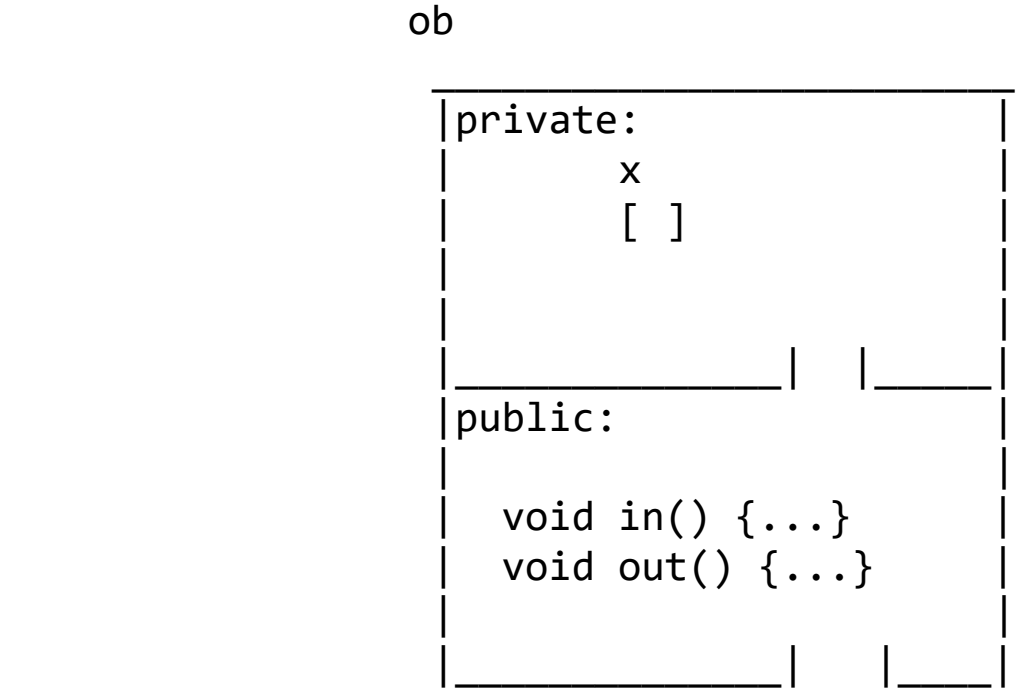
```
import java.util.*;
class JaggedDemo
{
        public static void main(String []args)
        {
                Scanner sc=new Scanner(System.in);

                System.out.println("\n Enter the student count: ");
                int nos=sc.nextInt();


                int x[][]=new int[nos][];
```

```
                int j=0,scnt=0;
                System.out.println("\n Enter the Marks
of "+nos+" students: ");
                for(int i=0;i<nos;i++)
                {
                        System.out.println("\n Enter the
subject count: ");
                        scnt=sc.nextInt();
                        x[i]=new int[scnt];
                        System.out.println("\n Enter the
marks in "+scnt+" subjects:  ");
                        for(j=0;j<scnt;j++)
                        {
                                x[i][j]=sc.nextInt();
                        }
                }

                System.out.println("\n Marks of "+nos+"
students ");
                for(int i=0;i<x.length;i++)
                {
                        System.out.print("\n Student
"+(i+1)+": ");
                        for(j=0;j<x[i].length;j++)
                        {
                                System.out.print("
"+x[i][j]);
                        }
                }
        }
}

//-------------------------------------------------------
-------------------------------------------------------
```

```
-----------
//-------------------------------------------------------
-----------------------------------------------------------
----------
        OOPS in java:
        =============

        In the oop we must know the followong things.

                - class
                - Object
                - data hiding
                - encapsulation
                - abstraction
                - Instance variables and methods
                - constructors and destructors
                - static variable and static method
                - method overloading and method
overidding
                - operator overloading
                - Interitance
                - polymorphism
                - virtual and abstract

        All these things are mentioned with ref to C++
background.


        // Lets consider a simple program in C++

void abc(int a1)
{
        ........;
}
```

```
int pqr(int a1, int a2)
{
        ..........;
        return --;
}
int main()
{
        int x,y,z;
        .......
        abc(x);
        .......
        .......
        pqr(y,z);
        .......
        return 0;
}
```

        // You can provide the security to data using class as shown below.



```
class Number
{
    private:
        int x;
    public:
        void in()
        {
        }
        void out()
        {
        }
};
int main()
{
        Number ob;

        ob.x=10;  xxxxxx

        ob.in();

        return 0;
}
```

Using class in java: It allows you to create your own data type. In java standred library(JSL) all the methods are provided as a member methods of class, and in many cases there are different classes which are used to some exsisting types in its object form.

syntax:

```
class <cls_nm>
{
        <instance_variables>;
        <Member_methods>;
        .
        .
}
```

* Note: There are 4 different visibility modifiers for java classes.

- private: Not accessible directly using . operator
- public: can be accessed using the . operator directly
- protected: ....
- default: can be accessed using the . operator directly

* Bydefault the class members are default in nature.
* Here in java you have to mention the visibility of every class member using public, private, or protected. No keyword for default.

* There is one unwritten rule. CHOOSE THE FIRST CHARACTER OF CLASS NAME IN UPPERCASE AND ALL OTHER IN LOWER IF IT IS MADE FROM SINGLE WORD. IF IT IS MADE FROM MORE WORDS THEN FIRST CHARACTER OF EACH WORD IN UCASE AND ALL OTHERS IN LCASE, WITH NO SPACE IN BETWEEN.

//---

- Creating the Object in Java: In simple words the object is variable of class type. or
It is instance of class.

```
class Demo
{
        ...
}
```

If i used the statement like

```
Demo d;

d
[ ]  (reference variable)
```

It is called as reference variable (It is same like pointer variable), means it able to hold the reference(address) where the data is stored.

Means we have to take some additional efforts to allocate the memory for object and for that we need to use the new keyword as ...

```
<class_nm> <Obj_nm>=new <cls_nm>();

i.e.
```

```
                Demo d=new Demo();

                d
                 [ ]-----------> [     ]
                                 [     ]  SPACE
ALLOCATED TO
                                 [     ]  STORE THE DATA
                                 [     ]
```

Note that the java object is dynamic in nature.

```
class Number
{
        private int x;
        private double y;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the val of x:
");
                x=sc.nextInt();
                System.out.println("Enter the val of y:
");
                y=sc.nextDouble();
        }
        public void out()
        {
                System.out.println("val of x: "+x);
                System.out.println("val of y: "+y);
        }
}
                                // ob1
```

```
class MainNumber                //  [] -----> [      ]
{
        public static void main(String []args)
        {
                Number ob1;     // Only reference
variable is created in java
                ob1=new Number(); // as java object
gains the memory space dynamically, new is used

                ob1.in();
                ob1.out();

                new Number().in(); // Anonymous Object
        }
}

//-------------------------------------------------------
--------------------------------------

        // Defining main() method in a same class

import java.util.Scanner;
class Number
{
        private int x;
        private double y;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the val of x:
");
                x=sc.nextInt();
                System.out.println("Enter the val of y:
");
```

```
                y=sc.nextDouble();
        }
        public void out()
        {
                System.out.println("val of x: "+x);
                System.out.println("val of y: "+y);
        }

        public static void main(String []args)
        {
                Number ob=new Number();
                ob.in();
                ob.out();
        }
}

//-------------------------------------------------------
-----------------------------------------------

        /// A class containing methods with arguments
class Student
{
        private String name;
        private int id;
        private double per;
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
```

```
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
        }
}
class DemoStudent
{
        public static void main(String []args)
        {
                String nm="Amol";
                int i=45;
                double p=90.56;

                Student s1=new Student();
                s1.setData(nm, i, p);
                s1.showData();
        }
}

//-----------------------------------------------------
-------------------

        // creating another object

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double per;
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
```

```
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
        }
}
class DemoStudent
{
        public static void main(String []args)
        {
                String nm="Amol";
                int i=45;
                double p=90.56;

                Student s1=new Student();
                s1.setData(nm, i, p);
                s1.showData();

//-----------------------------------------
                String b1;
                int b2;
                double b3;

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                b1=sc.nextLine();
                System.out.println("Enter the student
id: ");
                b2=sc.nextInt();
                System.out.println("Enter the
Percentage: ");
```

```
                b3=sc.nextDouble();

                Student s2=new Student();
                s2.setData(b1,b2,b3);
                s2.showData();
        }
}

//-----------------------------------------------------
----------------------------------------
        /// Method overloading

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double per;
        public void setData()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
```

```
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
        }
}
class DemoStudent
{
        public static void main(String []args)
        {
                String nm="Amol";
                int i=45;
                double p=90.56;

                Student s1=new Student();
                s1.setData(nm, i, p);
                s1.showData();

//-----------------------------------------
                String b1;
                int b2;
                double b3;

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                b1=sc.nextLine();
                System.out.println("Enter the student
id: ");
                b2=sc.nextInt();
```

```
                System.out.println("Enter the
Percentage: ");
                b3=sc.nextDouble();

                Student s2=new Student();
                s2.setData(b1,b2,b3);
                s2.showData();


//-------------------------------------------------------
-----------

                Student s3=new Student();
                s3.setData();
                s3.showData();

        }
}

//-------------------------------------------------------
-----------------------------------------

        /// Defining the Constructors in class

        - default constructor

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double per;
        public Student() // Default Constructor
        {
```

```
                name="NA";
                id=0;
                per=0.0;
        }
        public void setData()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
        }
}
class DemoStudent
{
        public static void main(String []args)
        {
```

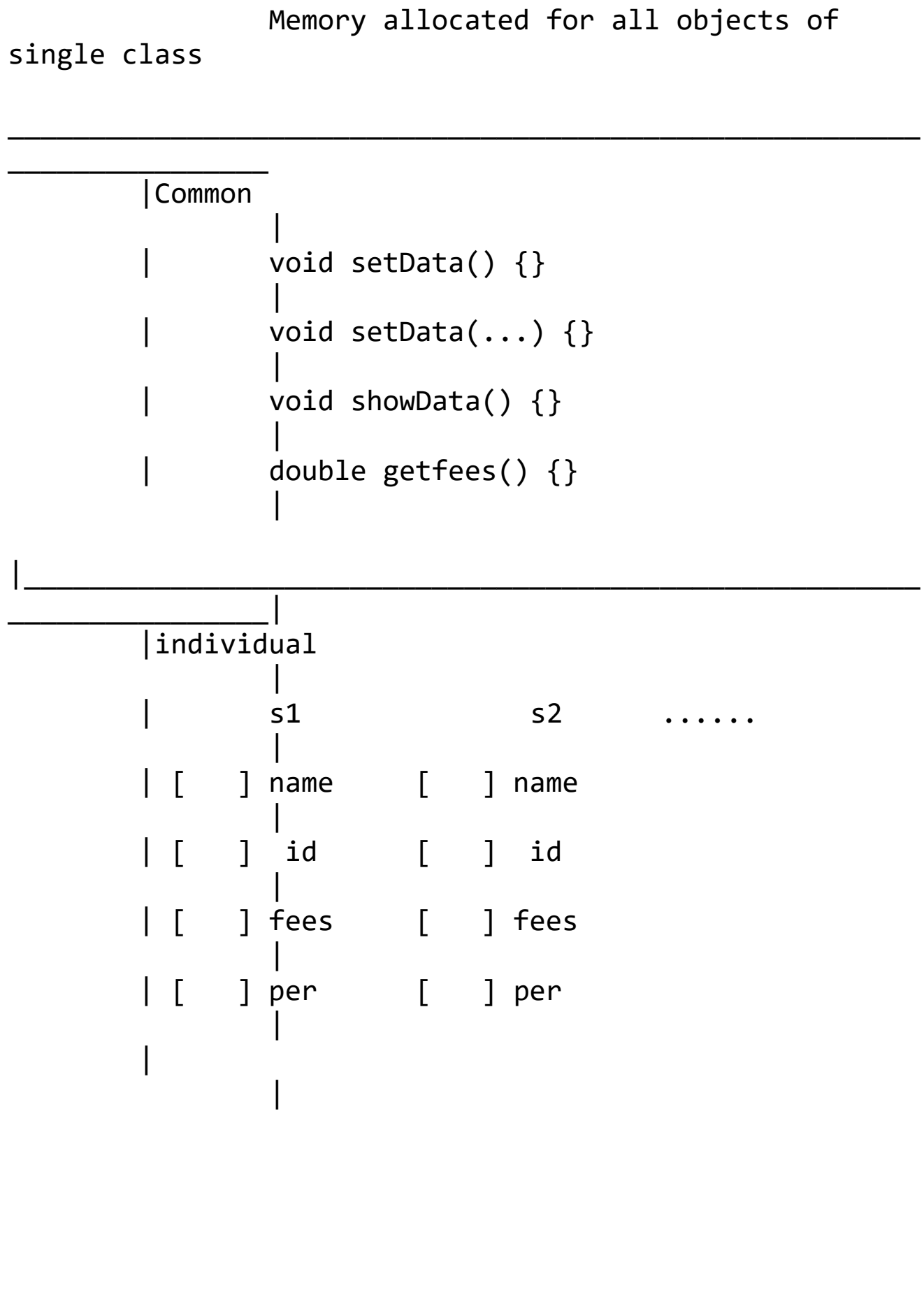
```
                Student s1=new Student();
                s1.showData();

        }
}

//-------------------------------------------------------
-----------------------------------------------

        // Parameterized and Copy Constructor

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double per;
        public Student() // Default Constructor
        {
                name="NA";
                id=0;
                per=0.0;
        }
        public Student(String a1, int a2, double a3) //
Parame. Constructor
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public Student(Student a) // copy Constructor
        {
                name=a.name;
                id=a.id;
```

```
                per=a.per;
        }
        public void setData()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
        }
}
class DemoStudentConstructor
{
        public static void main(String []args)
        {
                Student s1=new Student();
                s1.showData();
```

```
                Student s2=new
Student("Kiran",12,98.34);
                s2.showData();

                Student s3=new Student(s2);
                s3.showData();

        }
}

//-----------------------------------------------------
-----------------------------------------------------

        /// Array Of Objects

                s                   0     1    2
                [] -------------> [  ] [  ] [  ]

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double per;
        public Student() // Default Constructor
        {
                name="NA";
                id=0;
                per=0.0;
        }
        public Student(String a1, int a2, double a3) //
Parame. Constructor
        {
```

```
                name=a1;
                id=a2;
                per=a3;
        }
        public Student(Student a) // copy Constructor
        {
                name=a.name;
                id=a.id;
                per=a.per;
        }
        public void setData()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Percentage: "+per);
```

```
        }
}
class DemoStudentArray
s
{
[] -----> [ ][ ][ ]
        public static void main(String []args)
           |  |  |
        {
           V  V  V
                Student []s=new Student[3];
           {} {} {}


                System.out.println("Enter the
information of 3 student: ");
                for(int i=0;i<3;i++)
                {
                        s[i]=new Student();
                        s[i].setData();
                }
                System.out.println(" information of 3
student: ");
                for(int i=0;i<3;i++)
                {
                        s[i].showData();
                }
        }
}

//-------------------------------------------------------
---------------------------------------

        // static member in a class
```

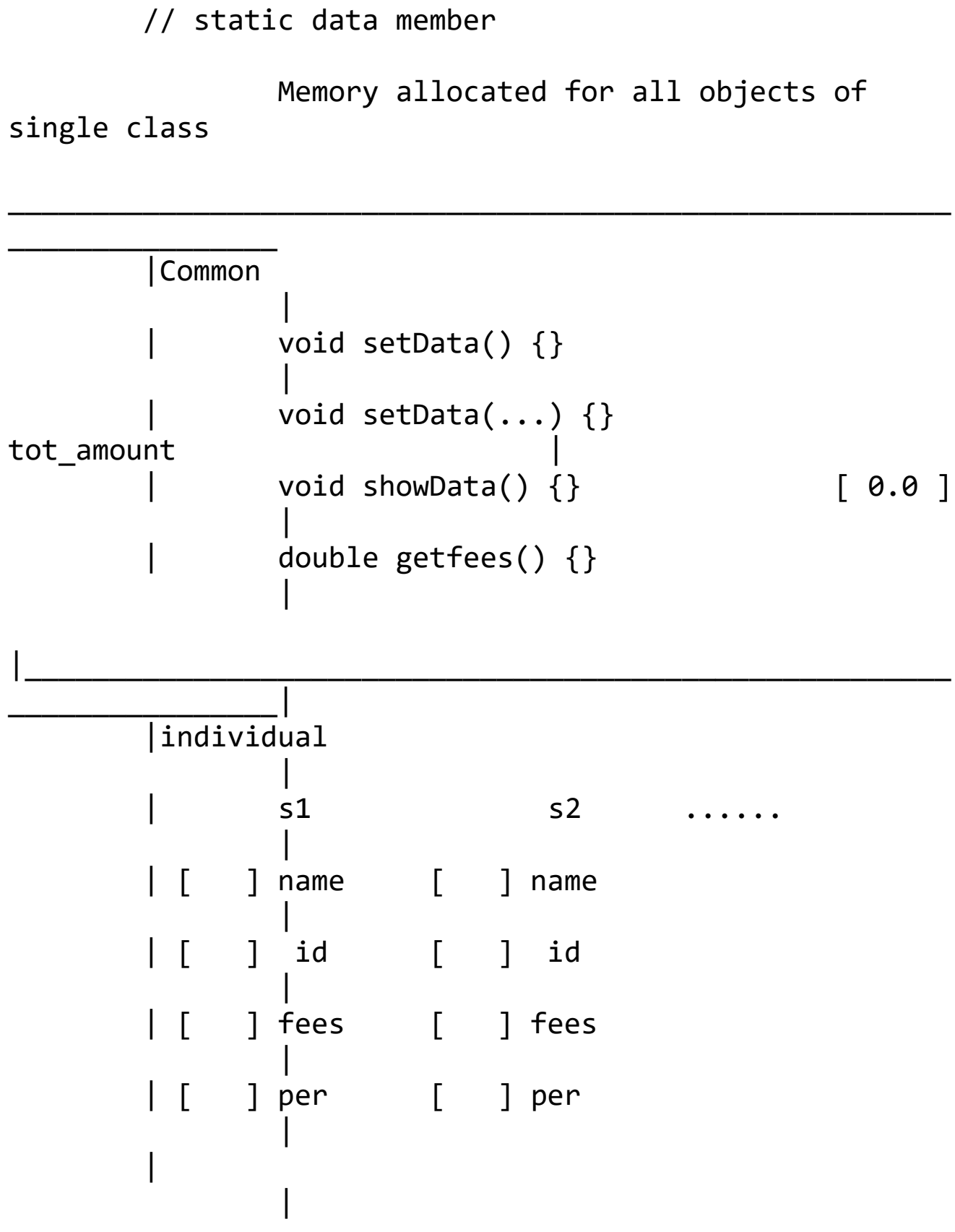

Memory allocated for all objects of single class
Common
void setData() {}
void setData(...) {}
void showData() {}
double getfees() {}
individual
s1
s2
.....
name
id
fees
per
name
id
fees
per

|_____
_____|

```
import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double fees;
        private double per;
        public Student() // Default Constructor
        {
                name="NA";
                id=0;
                per=0.0;
        }
        public Student(String a1, int a2, double a3) //
Parame. Constructor
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public Student(Student a) // copy Constructor
        {
                name=a.name;
                id=a.id;
                per=a.per;
        }
        public void setData()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
```

```
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the fees paid:
");
                fees=sc.nextDouble();
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public double getFees(){ return fees; }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t
Percentage: "+per);
        }
}
class DemoStudentStatic1

{

        public static void main(String []args)

        {
```

```
                Student []s=new Student[3];



                System.out.println("Enter the
information of 3 student: ");
                for(int i=0;i<3;i++)
                {
                        s[i]=new Student();
                        s[i].setData();
                }
                System.out.println(" information of 3
student: ");
                for(int i=0;i<3;i++)
                {
                        s[i].showData();
                }

                double Total_fees_Collected=0.0;
                for(int i=0;i<3;i++)
                {

Total_fees_Collected+=s[i].getFees();
                }

                System.out.println("\n Amount Received:
"+Total_fees_Collected);
        }
}

//-------------------------------------------------------
-------------------------------------------------------
```


// static data member
Memory allocated for all objects of single class
Common
void setData() {}
void setData(...) {}
tot_amount
void showData() {}
[0.0]
double getfees() {}
individual
s1
s2
.....
name
id
fees
per

```
|_____
_____|


import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double fees;
        private double per;
        static private double tot_amount;
        public Student() // Default Constructor
        {
                name="NA";
                id=0;
                per=0.0;
        }
        public Student(String a1, int a2, double a3) //
Parame. Constructor
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public Student(Student a) // copy Constructor
        {
                name=a.name;
                id=a.id;
                per=a.per;
        }
        public void setData()
        {
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the fees paid:
");
                fees=sc.nextDouble();
                tot_amount=tot_amount+fees;
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t
Percentage: "+per);
        }
        public void showTotalAmount()
        {
                System.out.println("Total Fees
Collected: "+tot_amount);
        }
}
```

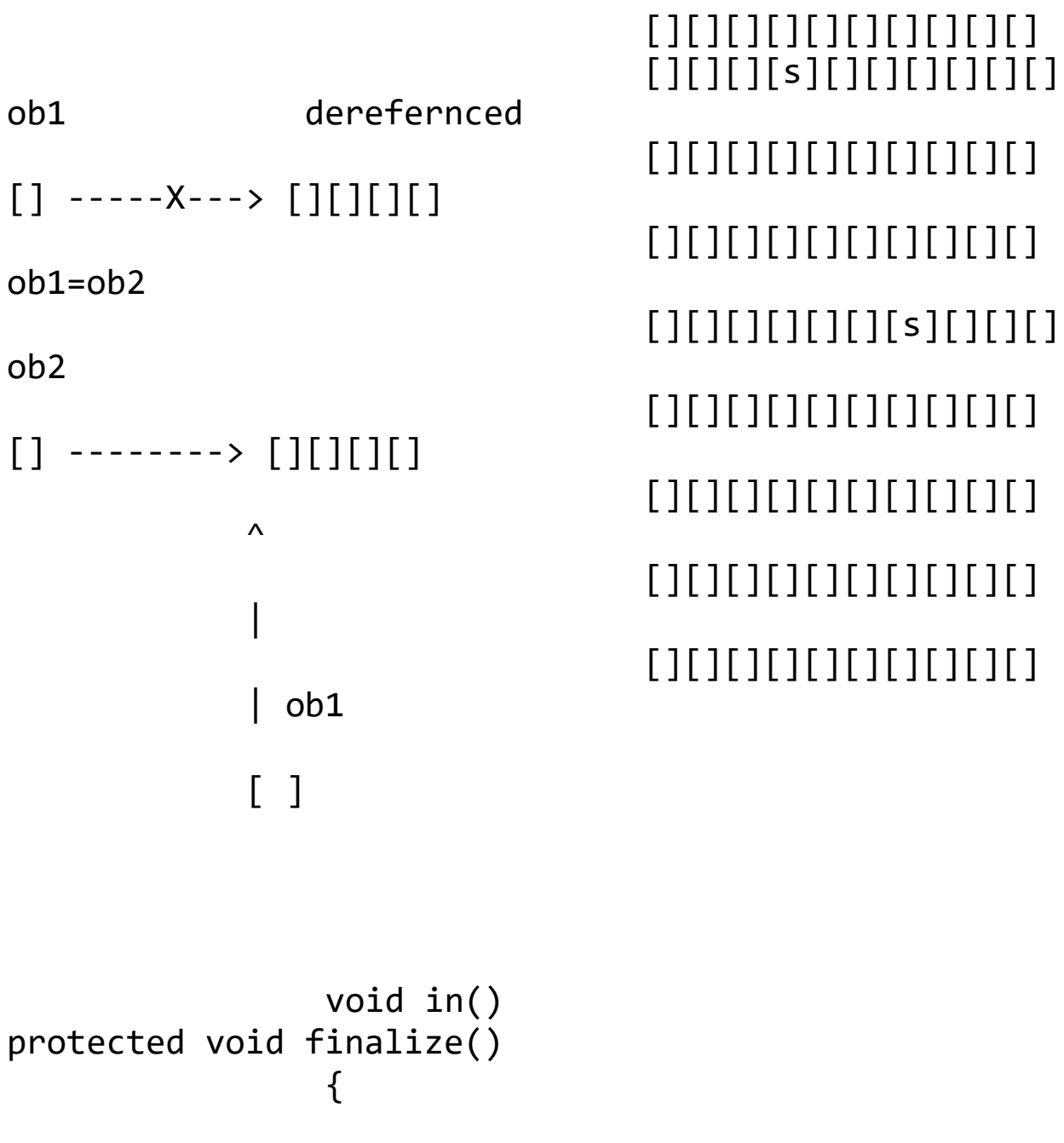
```
class DemoStudentStatic1

{

        public static void main(String []args)

        {

                Student []s=new Student[3];



                System.out.println("Enter the
information of 3 student: ");
                for(int i=0;i<3;i++)
                {
                        s[i]=new Student();
                        s[i].setData();
                }
                System.out.println(" information of 3
student: ");
                for(int i=0;i<3;i++)
                {
                        s[i].showData();
                }

                s[0].showTotalAmount();
        }
}


//-------------------------------------------------------
-------------------------------------------------
```

```
// Static method and static block

import java.util.Scanner;
class Student
{
        private String name;
        private int id;
        private double fees;
        private double per;
        static private double tot_amount;
        static
        {
                tot_amount=1000;
        }
        public Student() // Default Constructor
        {
                name="NA";
                id=0;
                per=0.0;
        }
        public Student(String a1, int a2, double a3) //
Parame. Constructor
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public Student(Student a) // copy Constructor
        {
                name=a.name;
                id=a.id;
                per=a.per;
        }
        public void setData()
```

```
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the student
name: ");
                name=sc.nextLine();
                System.out.println("Enter the student
id: ");
                id=sc.nextInt();
                System.out.println("Enter the fees paid:
");
                fees=sc.nextDouble();
                tot_amount=tot_amount+fees;
                System.out.println("Enter the
Percentage: ");
                per=sc.nextDouble();
        }
        public void setData(String a1, int a2, double
a3)
        {
                name=a1;
                id=a2;
                per=a3;
        }
        public void showData()
        {
                System.out.println("Student Information:
Name:"+name+"\t ID: "+id+"\t Fees Paid: "+fees+"\t
Percentage: "+per);
        }
        public static void showTotalAmount()
        {
                System.out.println("Total Fees
Collected: "+tot_amount);
        }
```

```
}
class DemoStudentStatic3

{

        public static void main(String []args)

        {
                Student.showTotalAmount();

                Student []s=new Student[3];



                System.out.println("Enter the
information of 3 student: ");
                for(int i=0;i<3;i++)
                {
                        s[i]=new Student();
                        s[i].setData();
                }
                System.out.println(" information of 3
student: ");
                for(int i=0;i<3;i++)
                {
                        s[i].showData();
                }
                Student.showTotalAmount();
        }
}

//-------------------------------------------------------
---------------------------------------
```

// Garbage Collector: Java dont have destructors, then the derefernced memory locations are relesed automatically by a Garbage Collector.



void in()
protected void finalize()
{

```
{
                        int a;
                        ......
}
                }

                System.gc()



//-------------------------------------------------------
-------------------------------------------------------


                                    forming the relationship

                    Inheritance
    Association

                        IS-A
      HAS-A

                        person
        Car
                          |
         |
                ---------------------
---------------------
                |       |          |
|                   |
            Trainer   Student    Driver
   Engine               Music Player
```

Composion Agrigation

CPP Code just for example

```
class A
{
};
Class B
{
};
class Z:public A, public B
{
};
A ob1;

B ob2;

};
```

// Inheritance in java: It is art of defining the new class using the per-defined classes.

consider the example, in normal case,

class employee class student
class Teacher class Driver

```
    {                          {
{                          {
        char name[50];             char name[50];
  char name[50];             char name[50];
        int age;                   int age;
   int age;                   int age;
        -----------;               -----------;
   -----------;               -----------;
        -----------;               -----------;
   -----------;               -----------;
        -----------;               -----------;
   -----------;               -----------;
        -----------;               -----------;
   -----------;               -----------;
    };                             };
};                         };
```

```
        /// syntax of inheritance

    class <base/super/parent_class>
    {
        ........;
    };
    class <derived/sub/child_class>  extends
<base/super/parent_class>
    {
        --------;
        --------;
    };
```

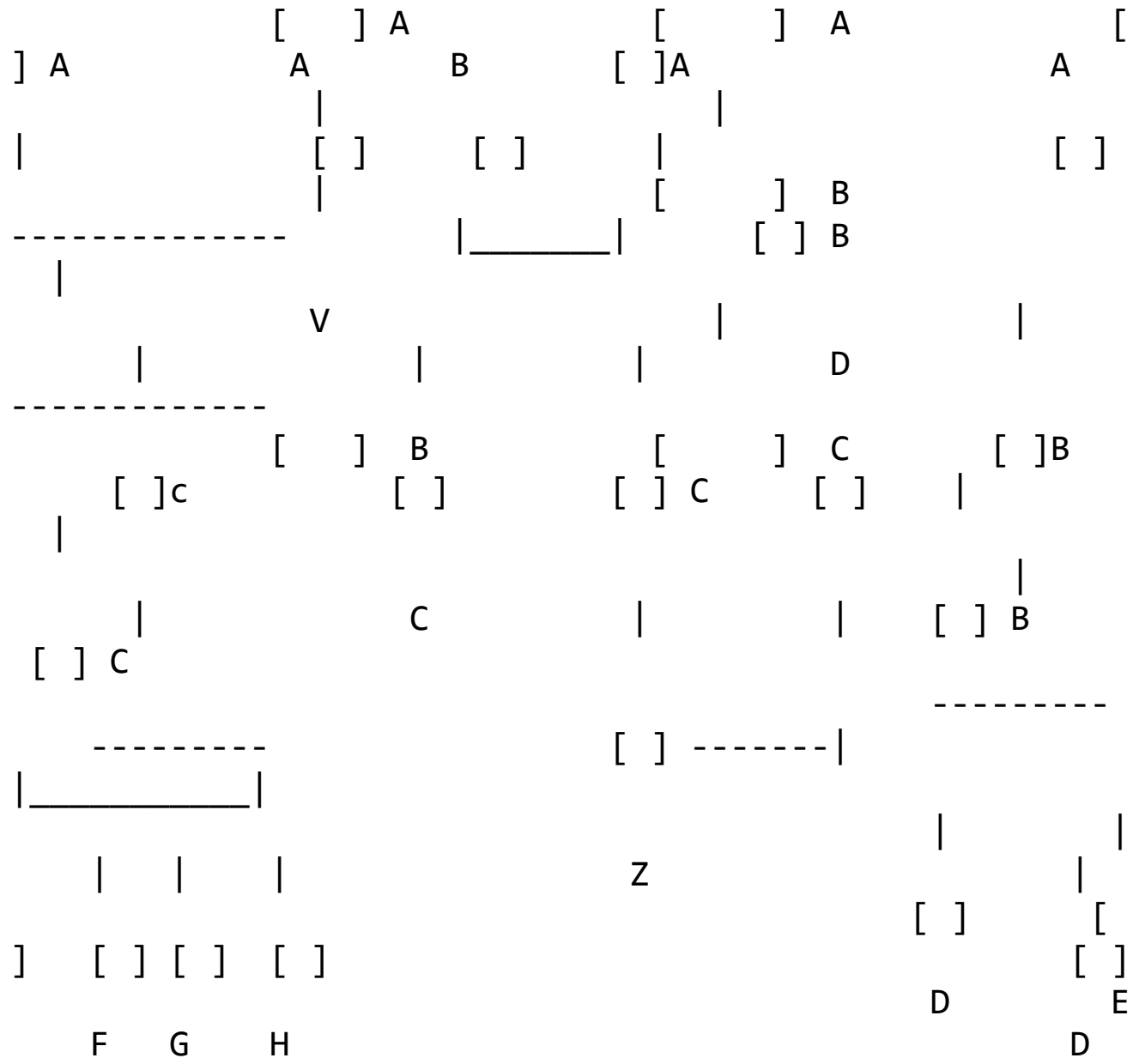
lets see with the example

class Person

```
{
   char name[50];
   int age;
   -----------;
}


class employee extends Person   class student extends
Person    class Teacher extends Person       class
Driver extends Person
{                               {
       {                                {
    -----------;                     -----------;
            -----------;
-----------;
    -----------;                     -----------;
            -----------;
-----------;
    -----------;                     -----------;
            -----------;
-----------;
};                              };
       };                               };
```

/// Types of Inheritance in C++

```
            single          Multi-level
hierarchical        Multiple         Hybrid
          Inheritance       Inheritance
Inheritance       Inheritance     Inheritance
```



- single Inheritance: One to one relationship, two layers
- Multi-level Inheritance: one to one relationship, more than two layers
- hierarchical Inheritance: One to many relationship
- Multiple Inheritance: many to one relationship
- Hybrid Inheritance: combinations of any two or

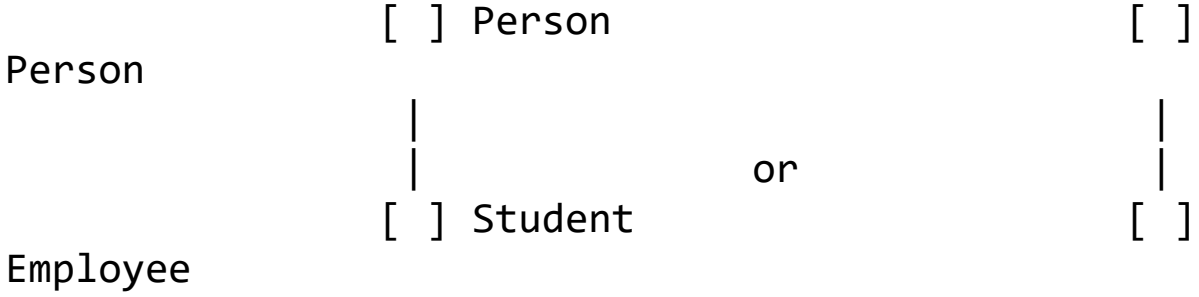
more inheritance

To avoid the ambiguity error java eliminated the Multiple and Hybrid inheritance. to achive the plus points of these inheritances java introduced the "interface."

So Java having only 3 inheritances

- single Inheritance: One to one relationship, two layers
- Multi-level Inheritance: one to one relationship, more than two layers
- hierarchical Inheritance: One to many relationship

- single Inheritance:



- if we include both in single program, it become hierarchical Inheritance.

```
import java.util.Scanner;
class Person
{
```

```
        private String nm="";
        private int age;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the name of
person: ");
                nm=sc.nextLine();

                System.out.println("\n Enter the age of
person: ");
                age=sc.nextInt();
        }
        public void out()
        {
                System.out.println("\n Name: "+nm+"\t
age: "+age);
        }
}


class Student extends Person
{
        private int rno;
        private double per;
        public void input()
        {
                in();
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the rno of
Student: ");
                rno=sc.nextInt();

                System.out.println("\n Enter the
```

```
percentage of Student: ");
                per=sc.nextDouble();
        }
        public void output()
        {
                out();
                System.out.println("\n RNO: "+rno+"\t
Percentage: "+per);
        }
}


class MainStudent
{
        public static void main(String []args)
        {
                Student s=new Student();
                s.input();
                s.output();
        }
}


//-------------------------------------------------
-------------------------

        Or can be called as



import java.util.Scanner;
class Person
{
        private String nm="";
```

```
        private int age;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the name of
person: ");
                nm=sc.nextLine();

                System.out.println("\n Enter the age of
person: ");
                age=sc.nextInt();
        }
        public void out()
        {
                System.out.println("\n Name: "+nm+"\t
age: "+age);
        }
}

class Student extends Person
{
        private int rno;
        private double per;
        public void input()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the rno of
Student: ");
                rno=sc.nextInt();

                System.out.println("\n Enter the
percentage of Student: ");
                per=sc.nextDouble();
        }
```

```
        public void output()
        {
                System.out.println("\n RNO: "+rno+"\t
Percentage: "+per);
        }
}

class MainStudent
{
        public static void main(String []args)
        {
                Student s=new Student();

                s.in();
                s.input();

                s.out();
                s.output();
        }
}
```

As the in() and out() from Person class decl. as public, visibility remains as it is
in derived class. so both can be called from main() or from methods of child class.

```
//-------------------------------------------------
-----------------------------------------
```

- Multilevel Inheritance:

```
                [ ] Person              in() and out()
                 |
```

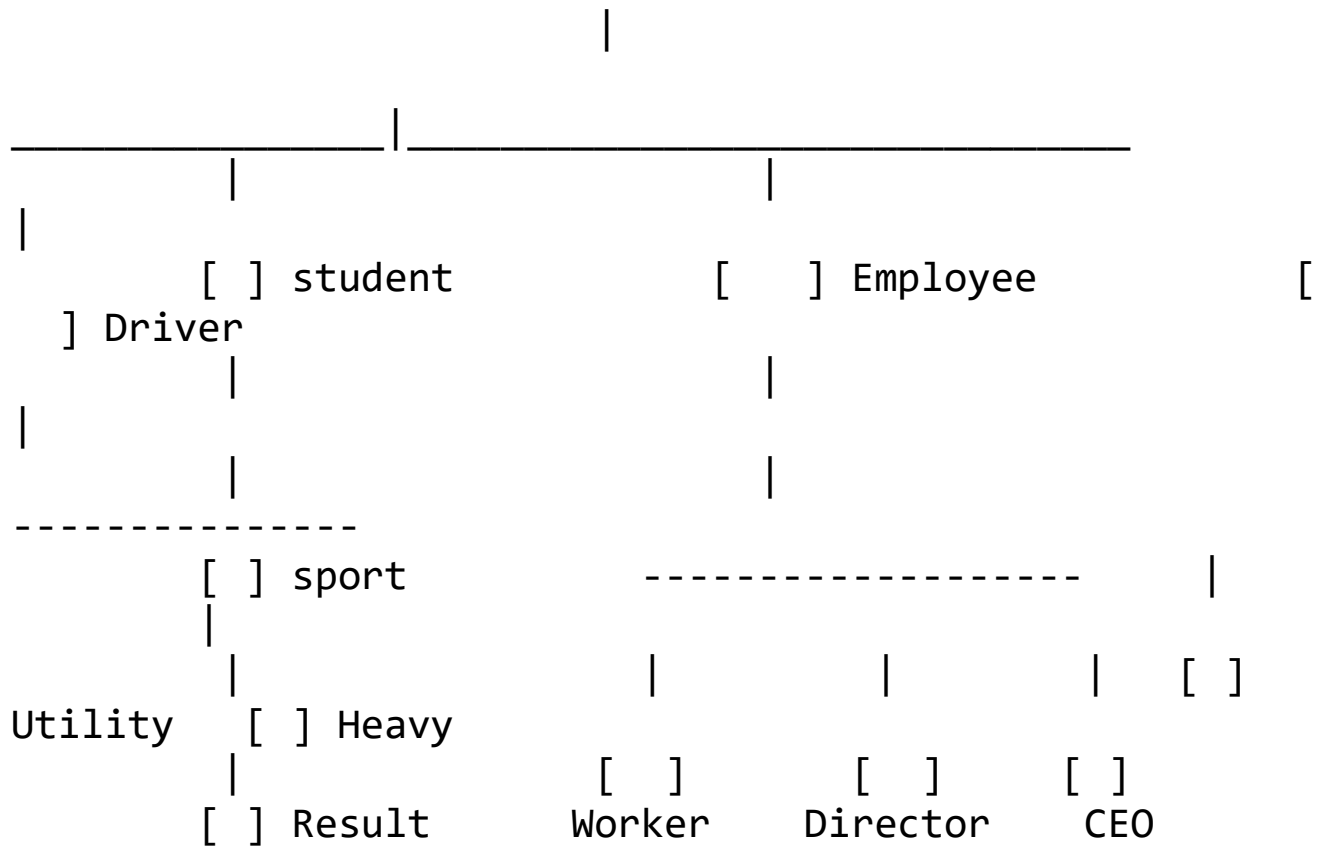
```
                 |
                [ ] Student             input() and
output()
                 |
                 |
                [ ] Sport               set() and show()

import java.util.Scanner;
class Person
{
        private String nm="";
        private int age;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the name of
person: ");
                nm=sc.nextLine();

                System.out.println("\n Enter the age of
person: ");
                age=sc.nextInt();
        }
        public void out()
        {
                System.out.println("\n Name: "+nm+"\t
age: "+age);
        }
}

class Student extends Person
{
        private int rno;
        private double per;
```

```
        public void input()
        {
                in();
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the rno of
Student: ");
                rno=sc.nextInt();

                System.out.println("\n Enter the
percentage of Student: ");
                per=sc.nextDouble();
        }
        public void output()
        {
                out();
                System.out.println("\n RNO: "+rno+"\t
Percentage: "+per);
        }
}
class Sport extends Student
{
        private int points;
        public void set()
        {
                input();
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the grade
points of Student: ");
                points=sc.nextInt();
        }
        public void show()
        {
                output();
                System.out.println("\n Points:
```

```
"+points);
        }
}
class MainStudentMultilevel
{
        public static void main(String []args)
        {
                Sport s=new Sport();
                s.set();
                s.show();
        }
}

//-------------------------------------------------------
-------------------------

        /// Using the constructors in inheritance

class Base
{
        private int a;
        public Base()
        {
                System.out.println("\n In Base
default");
                a=10;
        }
        public Base(int x)
        {
                System.out.println("\n In Base para");
                a=x;
        }
        public void showBase()
        {
```

```
                System.out.println("\n a="+a);
        }
}
class ImdBase extends Base
{
        private int b;
        public ImdBase()
        {
                System.out.println("\n In ImdBase
default");
                b=20;
        }
        public ImdBase(int x, int y)
        {
                super(x);
                System.out.println("\n In ImdBase
para");
                b=y;
        }
        public void showImdBase()
        {
                showBase();
                System.out.println("\n b="+b);
        }
}
class Derived extends ImdBase
{
        private int c;
        public Derived()
        {
                System.out.println("\n In Derived
default");
                c=30;
        }
```

```
        public Derived(int x, int y, int z)
        {
                super(x,y);
                System.out.println("\n In Derived
para");
                c=z;
        }
        public void showDerived()
        {
                showImdBase();
                System.out.println("\n c="+c);
        }
}

class MainPassParamInh
{
        public static void main(String []args)
        {
                Derived ob1=new Derived();
                ob1.showDerived();

                Derived ob2=new Derived(100,200,300);
                ob2.showDerived();
        }
}


///================================================
================================================
============

        // Hierarchical Inheritnce

                        [ ] Person
```



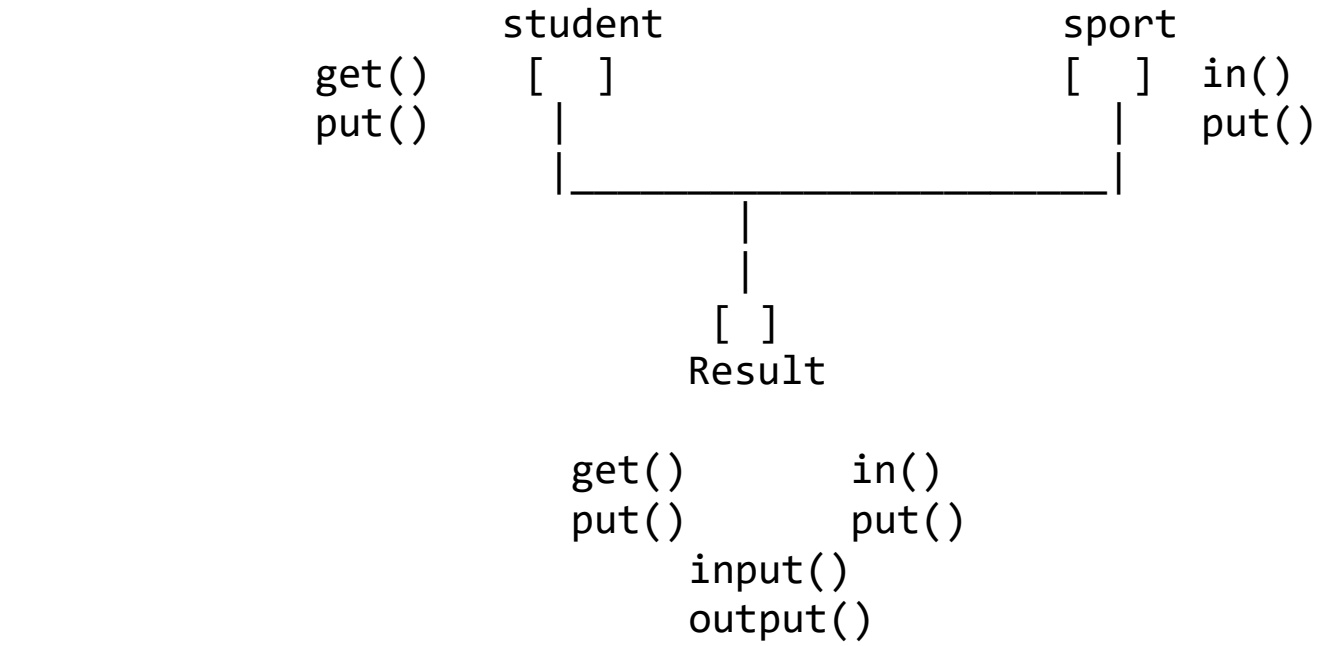
///==========

We have two more inheritances multiple and hierarchical, these inheritances are present in C++
but as both create the Ambiguous condition both are eliminated from Java.

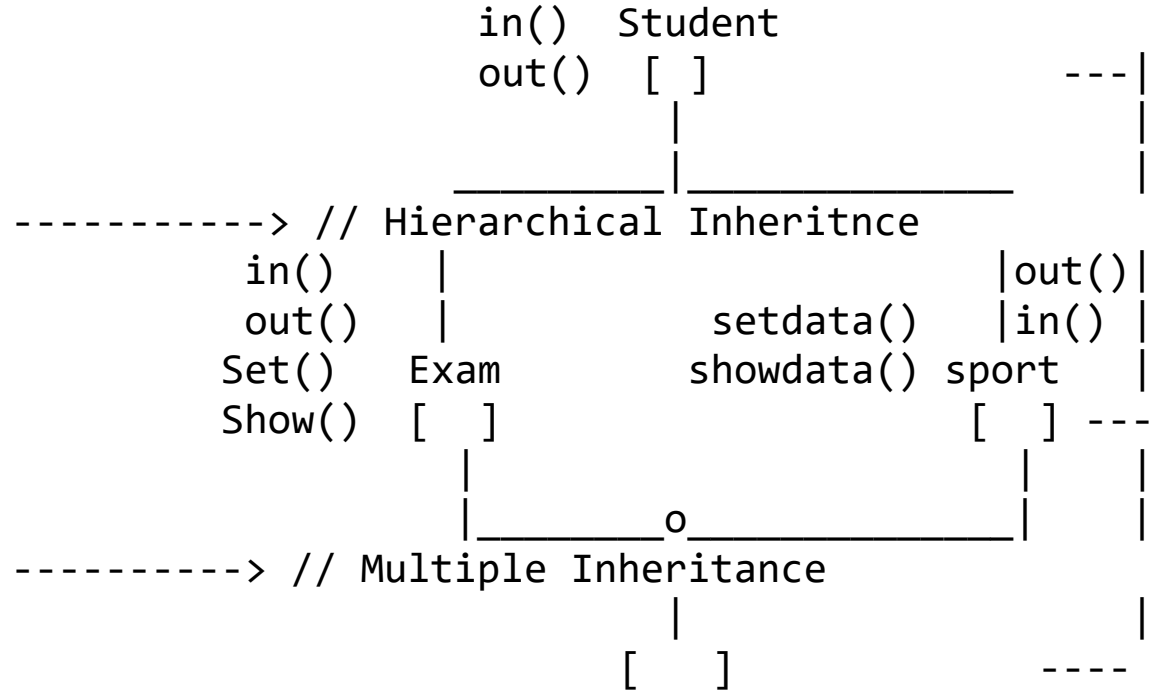
NOTE TAHT TO GAIN GOOD PART OF THESE INHERITANCES JAVA INTRODUCED THE INTERFACE.

Lets see, how the ambiguous condition is cerated by the both

// Multiple Inheritance:



// Hybrid Inheritance:



```
           in()   out()  Result  in()       out()
          Set()   Show()         setdata()  showdata()


///=============================================================
=============================================================
===========

//=============================================================
=============================================================
==========

         /// Method Overriding

    When base class function is redefined in child
class, and we called the same function
    using the object of child class. then the native
copy of child class will be executed
    and the inherited copy from base class is overruled
by the native copt of function. which
    is called as function overriding.


        class A [ ] show(){..}
                   |
                   |
        class B [ ] show(){...} --> Own copy of B
                    show(){..} --> Inhe. from A


  suppose we create an object of class B,

        i.e.    B ob=new B();
                ob.show(); --> the owned copy of B will
```

be executed.
which overrides the inh. copy of show()

```
/*
import java.util.Scanner;
class Person
{
        private String nm="";
        private int age;
        public void in()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the name of person: ");
                nm=sc.nextLine();

                System.out.println("\n Enter the age of person: ");
                age=sc.nextInt();
        }
        public void out()
        {
                System.out.println("\n Name: "+nm+"\t age: "+age);
        }
}

class Student extends Person
{
        private int rno;
        private double per;
        public void input()
```

```
        {
                in();
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the rno of
Student: ");
                rno=sc.nextInt();

                System.out.println("\n Enter the
percentage of Student: ");
                per=sc.nextDouble();
        }
        public void out()
        {
                // out(); // resursive call
                super.out();
                System.out.println("\n RNO: "+rno+"\t
Percentage: "+per);
        }
}

class MainStudentOverriding
{
        public static void main(String []args)
        {
                Student s=new Student();
                s.input();
                s.out();
        }
}

//-------------------------------------------------------
------------

        // Accessing the inherited instance variable
```

using super keyword

```
class Base
{
        public int a;
        Base()
        {
                a=10;
        }
}
class Derived extends Base
{
        public int a;
        Derived()
        {
                a=100;
        }
        public void out()
        {
                System.out.println("\n In Derived Inh
a="+super.a+" and a="+a);
        }
}
class DemoInitBlockAndSuper_InstanceVariable
{
        public static void main(String []args)
        {
                Derived ob=new Derived();
                ob.out();
        }
}

//-----------------------------------------------------
------------------------
```

```
//-------------------------------------------------------
------------------------

        /// Using final keyword

1. Final Variables:
        When a variable is declared as final, its value
cannot be changed after initialization.
        This is often used for constants Decl.

/*
public class FinalVariableExample {
    public static void main(String[] args) {
        final int MAX_VALUE = 100;
        // MAX_VALUE = 200; // Compilation error, as
MAX_VALUE is final
        System.out.println("Maximum value: " +
MAX_VALUE);
    }
}

class DemoFinal
{
        final public int x;
        public DemoFinal()
        {
                x=10;
        }
        public void show()
        {
                System.out.println("\n x="+x);
        }
}
class MainFinal
```

```
{
        public static void main(String []args)
        {
                DemoFinal ob=new DemoFinal();
                // ob.x=20;
                ob.show();
        }
}
```

- Note that we have to assign the value to final variable either using
  = operator directly or using initilizer block or by using default
  constructor.

- when we not assign any value, in that case it will generate an error message as
"MainFinal.java:16: error: variable x might not have been initialized"

- If we try to assign any other value we will face error as
"MainFinal.java:27: error: cannot assign a value to final variable x"

```
//-----------------
```

2. Final Methods: When a method is declared as final, it cannot be
overridden by subclasses.

```
class Parent {
```

```
    final void display() {
        System.out.println("This is a final method.");
    }
}

class Child extends Parent {
    // Compilation error: Cannot override the final
method from Parent
     void display() { }
}
class MainFinalMethodExample
{
        public static void main(String []args)
        {
                Child ob=new Child();
                ob.display();
        }
}

Output: (Compile time error)
MainFinalMethodExample.java:56: error: display() in
Child cannot override display() in Parent
     void display() { }
          ^
  overridden method is final
1 error
Press any key to continue . . .

//--------------------

3.Final Classes:
 - When a class is declared as final, it cannot be
extended
   by other classes.
```

```
final class Person
{
}
class Student extends Person
{
}
class mainDemoFinalClass
{
        public static void main(String []args)
        {
                Student ob=new Student();
        }
}
output: (Comile time error)
MainFinalMethodExample.java:83: error: cannot inherit
from final Person
class Student extends Person
                      ^
1 error
Press any key to continue . . .

//---------------------------------------


4. Final Parameters:
When a parameter is declared as final in a method, it
means that
the value of the parameter cannot be changed within the
method.

*/
class Demo
{
```

```
        private int x;
        public void in(final int t)
        {
                t=78;
                x=t;
        }
        public void show()
        {
                System.out.println("\n x="+x);
        }
}
class DemoFinalParameter
{
        public static void main(String []args)
        {
                Demo ob=new Demo();
                ob.in(100);
                ob.show();
        }
}
```

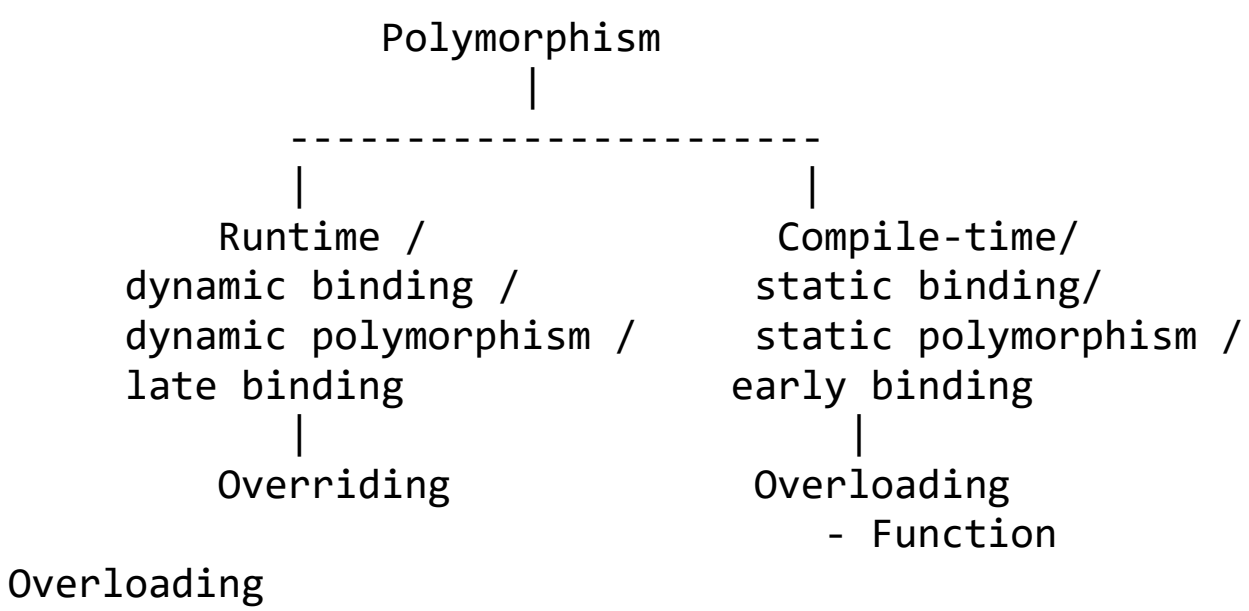
Output (Error on Compie)

```
DemoFinalParameter.java:114: error: final parameter t
may not be assigned
                t=78;
                ^
1 error
Press any key to continue . . .


//-------------------------------------------------------
-----------------------
```

// Polymorphism in Java: It refers one name many forms. there are two
different types as discussed below.

the word polymorphism forms as shown

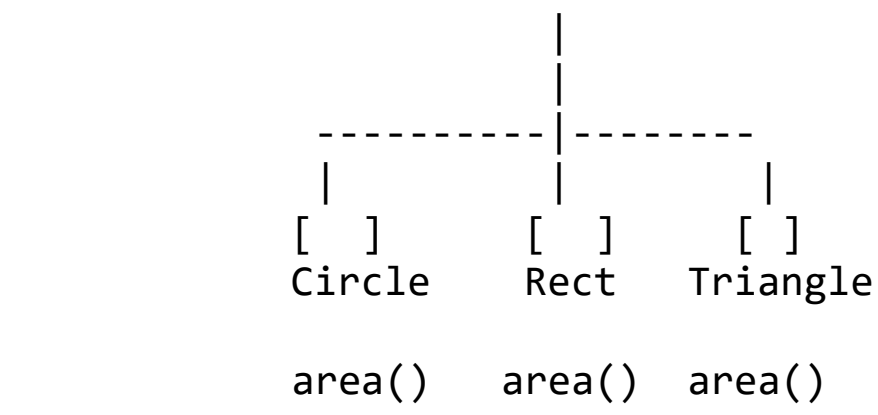
- Poly: many or more
- morphism: forms or copies



We have seen, the method overloading and Constructor overloading these
are the examples of static/compile time polymorphism.

lets see Runtime polymorphism now,

shape
[ ]




```
class Shape
{
}
class Circle extends Shape
{
        public void area()
        {
                System.out.println("In Circle area()
method");
        }
}

class Rect extends Shape
{
        public void area()
        {
                System.out.println("In Rect area()
method");
        }
}

class Triangle extends Shape
{
        public void area()
```

```
        {
                System.out.println("In Triangle area()
method");
        }
}

class DemoRuntimePoly
{
        public static void main(String []args)
        {
                Shape s=null;

                s=new Rect();
                s.area();
        }
}


output (on Compile)
DemoRuntimePoly.java:35: error: cannot find symbol
                s.area();
                 ^
  symbol:   method area()
  location: variable s of type Shape
1 error
Press any key to continue . . .

//-------------------------------------------------

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

```
        public void area()
        {
                System.out.println("In Circle area()
method");
        }
}

class Rect extends Shape
{
        public void area()
        {
                System.out.println("In Rect area()
method");
        }
}

class Triangle extends Shape
{
        public void area()
        {
                System.out.println("In Triangle area()
method");
        }
}

class DemoRuntimePoly
{
        public static void main(String []args)
        {
                Shape s=null;

                s=new Rect();
                s.area();
```

```
                s=new Circle();
                s.area();

                s=new Triangle();
                s.area();
        }
}
```

output: (On Compile)

```
DemoRuntimePoly.java:35: error: cannot find symbol
                s.area();
                 ^
  symbol:   method area()
  location: variable s of type Shape
DemoRuntimePoly.java:38: error: cannot find symbol
                s.area();
                 ^
  symbol:   method area()
  location: variable s of type Shape
DemoRuntimePoly.java:41: error: cannot find symbol
                s.area();
                 ^
  symbol:   method area()
  location: variable s of type Shape
3 errors
Press any key to continue . . .
```

Note that it is searching the method area() in the base class Shape.
as it is not present over there, it is generating the error message.
lets try after defining the area() in base class shape.

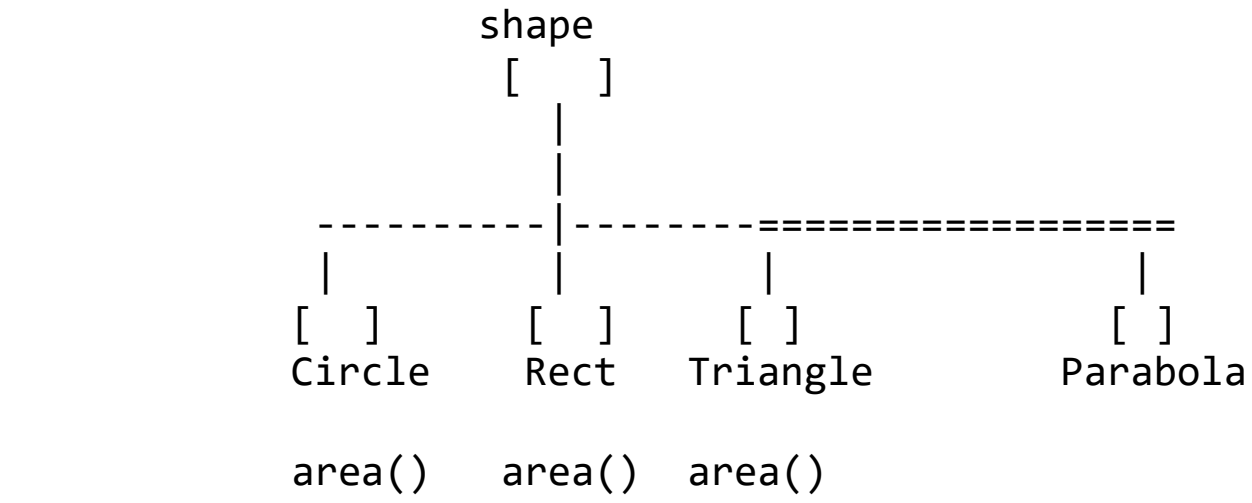
```
class Shape
{
        public void area()
        {
                System.out.println("In Shape area()
method");
        }
}
class Circle extends Shape
{
        public void area()
        {
                System.out.println("In Circle area()
method");
        }
}

class Rect extends Shape
{
        public void area()
        {
                System.out.println("In Rect area()
method");
        }
}

class Triangle extends Shape
{
        public void area()
        {
                System.out.println("In Triangle area()
method");
        }
```

```
}

class DemoRuntimePoly
{
        public static void main(String []args)
        {
                Shape s=null;

                s=new Rect();
                s.area();

                s=new Circle();
                s.area();

                s=new Triangle();
                s.area();
        }
}
output:

In Rect area() method
In Circle area() method
In Triangle area() method
Press any key to continue . . .
```

Just analyse the output, without using virtual keyword the methods of
class object where the reference of shape is pointing is executed
this inteligence due to fact which is known as "Dynamic method Dispatch"
and therefore virtual keyword not present is Java.

```
//-----------------------------------------------
```

Note that it is not compulsory to all childs of Shape class to
override the area() method. So to make compulsory in C++ pure
virtual method is there in base but as virtual keyword is absent
in java, it is done with the help of abstract keyword.




```
abstract class Shape
{
        abstract public void area();
}
class Circle extends Shape
{
        public void area()
        {
                System.out.println("In Circle area()
method");
```

```
        }
}

class Rect extends Shape
{
        public void area()
        {
                System.out.println("In Rect area()
method");
        }
}

class Triangle extends Shape
{
}

class DemoRuntimePoly
{
        public static void main(String []args)
        {
                Shape s=null;

                s=new Rect();
                s.area();

                s=new Circle();
                s.area();

                s=new Triangle();
                s.area();
        }
}
```

- Output on compile when only area() method in Shape decl. as abstract and tringle does not override area():

DemoRuntimePoly.java:139: error: Shape is not abstract and does not override abstract method area() in Shape
class Shape
^
1 error
Press any key to continue . . .

- Output on compile when Shape class and area() method from it decl. as abstract and tringle does not override area():

DemoRuntimePoly.java:159: error: Triangle is not abstract and does not override abstract method area() in Shape
class Triangle extends Shape
^
1 error
Press any key to continue . . .

//--

// After overriding area() in class Triangle and class shape and its area() as a abstract:

```
abstract class Shape
{
	abstract public void area();
}
class Circle extends Shape
```

```
{
        public void area()
        {
                System.out.println("In Circle area()
method");
        }
}

class Rect extends Shape
{
        public void area()
        {
                System.out.println("In Rect area()
method");
        }
}

class Triangle extends Shape
{
        public void area()
        {
                System.out.println("In Triangle area()
method");
        }
}

class DemoRuntimePoly
{
        public static void main(String []args)
        {
                Shape s=null;

                s=new Rect();
                s.area();
```

```
                s=new Circle();
                s.area();

                s=new Triangle();
                s.area();
        }
}
```

Output:

```
In Rect area() method
In Circle area() method
In Triangle area() method
Press any key to continue . . .
```

Abstract Class and Abstract Methods:

Abstract Class:

In Java, an abstract class is a class that cannot be instantiated on its own and is meant to be subclassed by other classes. It serves as a blueprint for other classes, providing common functionality and structure. Abstract classes can have both abstract and concrete methods.

Here are key characteristics of abstract classes in Java:

- It is decared as an abstract using abstract keyword.

```
        abstract class Demo
```

```
{
        .......
}
```

- Abstract class have atleast one abstract method, and child class must override the all abstract methods of base otherwise we need to declare the child class as abstract, and remember that we are not allowed to create the object of an abstract class.

```
abstract class Demo
{
        abstract void show();
        .......
}
```

- Abstract classes can also have regular (concrete) methods with a complete implementation

```
abstract class Demo
{
        abstract void show();
        public void display()
        {
        }
        .......
}
```

- Objects cannot be created from abstract classes using the new keyword. They are meant to be extended by subclasses.

- Abstract classes can have constructors, and they are called when a subclass object is created.

- Abstract classes can be used as a base class for other classes. Subclasses extend the abstract class and provide concrete implementations for the abstract methods.

- Abstract classes can have instance variables just like regular classes.

```
abstract class Demo
{
        private int x=10;
        public Demo()
        {
                System.out.println("\n In the Demo Class Constructor");
        }
        public void out()
        {
                System.out.println("\n In out() method - x is "+x);
        }
        abstract void show();
}
class Test extends Demo
{
        int y;
        public Test()
        {
                y=100;
                System.out.println("\n In child Test
```

```
class Constructor");
        }
        public void show()
        {
                System.out.println("\n Hi from show()
Test and y="+y);
        }
}
class DemoAbstractExample
{
        public static void main(String[] args)
        {
                Test ob=new Test();
                ob.show();
                ob.out();
        }
}
```

- The inner class can be declared abstract by declaring it local.
- A static method and constructor can be part of an abstract class.
- When the final keyword is used, the abstract keyword cannot be used.
- Abstract methods cannot be declared private.
- Abstract methods cannot be declared static.
- An abstract keyword cannot be used with variables or constructors.

```
//-----------------------------------------
```

Abstract method:

In Java, an abstract method is a method that is declared without an implementation in an abstract class. An

abstract class is a class that cannot be instantiated on its own and may contain abstract methods, which are meant to be implemented by concrete (non-abstract) subclasses. Abstract methods serve as a blueprint for concrete classes to provide their own implementation.

Here are the key points about abstract methods in Java:

- The abstract method decleared as abstract using abstract keyword and it dont have body.

```
i.e. abstract void dispay();
```

- When any method decleared as abstract, the containing class must be decleared as abstract.

i.e.

```
abstract class Test
{
        abstract void display();
        ....
}
```

- Subclasses that extend an abstract class must provide implementations for all the abstract methods declared in the superclass, otherwise you have to declear that class as abstract.

```
abstract class Test
{
        abstract void display();
        ....
}

class Demo extends Test
{
```

```
                        .....
                        public void dispay()
                        {
                                .....
                        }
                        ....
                }
```

- Abstract methods provide a way to define a common interface for a group of related classes, ensuring that each concrete subclass provides its own implementation. This promotes code reusability and helps in creating a consistent interface for different classes.

```
//-------------------------------------------------------
-----------------------------------------
//-------------------------------------------------------
-----------------------------------------
//-------------------------------------------------------
-----------------------------------------
```

Interface in Java:

It id actually introduced to achive the advantages of multiple inheritance in java.

In interfaces, all methods are implicitly abstract and public. They don't use the abstract keyword.

interface variables in Java are constants, and they are implicitly public, static, and final.

Decl. Syntax:

```
        interface  <interface_nm>
        {
                --------------;
```

```
                        ---------------;
                }
          e.g.
                interface area
                {
                        static final float pi=3.14F;
                        void compute(float  x, float
y);
                        void show();
                }
```

A class can extend only one class but it may implements one or more than one interface

```
                In C++
In Java

        class Exam      class Sport             class
Exam    interface Sport
     show() [ ]        show()[ ]             show()[ ]
            [ ]show() -abstract
             |                |                    |
             |
             |_______________|
|_______________|
                        |
show()  |
                show()  [  ] show()
     [  ] show(){}
         class Result:public Exam, Public Sport
class Result extends Exam implements Sport

                Result rob;
Result rob=new Result();
```

```
                rob.show(); // ambiguous error
rob.show();  // No Error

         - if() result having his own copy of show()
           then there will not be a ambiguous error.


/*
abstract class Test
{
        abstract public void in();
}
*/
interface Test
{
        void in();
}
class Demo implements Test
{
        public void in()
        {
        }
}
class DemoMain
{
        public static void main(String []args)
        {
                Demo ob=new Demo();
        }
}
//-------------------------------------------------------
--------------

// class implements interface
```

```
import java.util.Scanner;
interface Sample
{
        void in();
        void out();
}
class Test implements Sample
{
        private int x;
        private double y;
        public Test()
        {
                x=23;
                y=45.21;
        }
        public void in() // visibility must be public
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextDouble();
        }
        public void out() // visibility must be public
        {
                System.out.println("\n x="+x+"\t y="+y);
        }
}
class MainDemoInterfaceExample
{
        public static void main(String []args)
        {
                Test ob=new Test();
```

```
                ob.in();
                ob.out();
        }
}

//-------------------------------------------------------
----------------------

///-----------------------------------------------------
---------------------------------------

        // class extends class and implements interface

import java.util.Scanner;
interface Sample
{
        void in();
        void out();
}
class Demo
{       private int a;
        public Demo()
        {
                a=100;
        }
        public void show()
        {
                System.out.println("\n a="+a);
        }
}
class Test extends Demo implements Sample
{
        private int x;
        private double y;
```

```
        public Test()
        {
                x=23;
                y=45.21;
        }
        public void in() // visibility must be public
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextDouble();
        }
        public void out() // visibility must be public
        {
                show();
                System.out.println("\n x="+x+"\t y="+y);
        }
}
class MainDemoInterfaceExample
{
        public static void main(String []args)
        {
                Test ob=new Test();
                ob.in();
                ob.out();
        }
}

//-----------------------------------------------------
-----------------------

//-----------------------------------------------------
-----------------------------------
```

```
        // A class extends class and implements multiple
interfaces.
import java.util.Scanner;
interface SampleIn
{
        void in();
}
interface SampleOut
{
        void out();
}
class Demo
{       private int a;
        public Demo()
        {
                a=100;
        }
        public void show()
        {
                System.out.println("\n a="+a);
        }
}
class Test extends Demo implements SampleIn, SampleOut
{
        private int x;
        private double y;
        public Test()
        {
                x=23;
                y=45.21;
        }
        public void in() // visibility must be public
        {
```

```
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextDouble();
        }
        public void out() // visibility must be public
        {
                show();
                System.out.println("\n x="+x+"\t y="+y);
        }
}
class MainDemoInterfaceExample
{
        public static void main(String []args)
        {
                Test ob=new Test();
                ob.in();
                ob.out();
        }
}

//-----------------------------------------

        // An inteface extends another interface

import java.util.Scanner;
interface SampleIn
{
        void in();
}
interface SampleOut extends SampleIn
{
        void out();
```

```
}
class Demo
{       private int a;
        public Demo()
        {
                a=100;
        }
        public void show()
        {
                System.out.println("\n a="+a);
        }
}
class Test extends Demo implements SampleOut
{
        private int x;
        private double y;
        public Test()
        {
                x=23;
                y=45.21;
        }
        public void in() // visibility must be public
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextDouble();
        }
        public void out() // visibility must be public
        {
                show();
                System.out.println("\n x="+x+"\t y="+y);
        }
```

```
}
class MainDemoInterfaceExample
{
        public static void main(String []args)
        {
                Test ob=new Test();
                ob.in();
                ob.out();
        }
}


//-----------------------------------------------------
-------------------------

        // refering the base class object using
interface reference

import java.util.Scanner;

interface SampleOut
{
        void out();
}

class Demo
{       private int a;
        public Demo()
        {
                a=100;
        }
        public void show()
        {
                System.out.println("\n a="+a);
```

```
        }
}
class Test extends Demo implements SampleOut
{
        private int x;
        private double y;
        public Test()
        {
                x=23;
                y=45.21;
        }
        public void in() // visibility must be public
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter the values
of x and y: ");
                x=sc.nextInt();
                y=sc.nextDouble();
        }
        public void out() // visibility must be public
        {
                System.out.println("\n x="+x+"\t y="+y);
        }
}
class MainDemoInterfaceExample
{
        public static void main(String []args)
        {
                SampleOut ob=new Test();
                ob.in();
                ob.out();
        }
}
```

```
output on compile:
MainDemoInterfaceExample.java:33: error: cannot find
symbol
                ob.in();
                  ^
  symbol:   method in()
  location: variable ob of type SampleOut
1 error
Press any key to continue . . .
```

Note that an interface reference able to refer an object of its every child but
using it we can access only those methods which are declleared with that interface.

```
//-------------------------------------------------
```

In version 8

- Java interface can have default methods
- Java Interface may contain static method

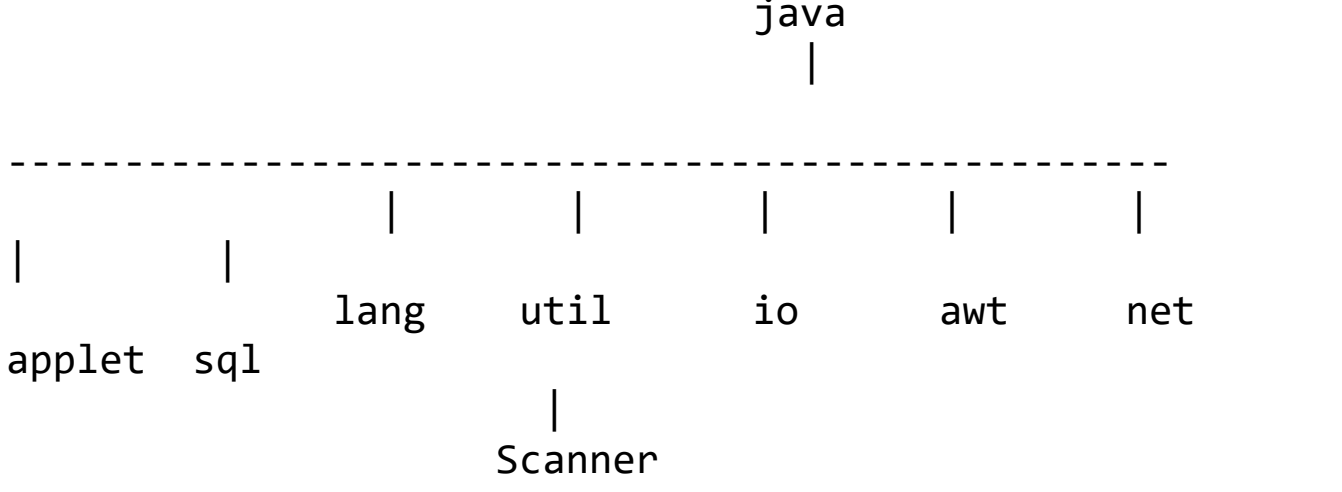
```
interface A
{
        default void show()
        {
        }
        static void xxx()
        {
        }
}
```

```
///-------------------------------------------------
```

Packages in Java: It is Collection of classes and interfaces.

There are two types of packages ----> Built-in packages --> provided with JDK
|-> User defined packages -> defined by programmer

Java provides the standred library known as JSL or API, which is divided into
different packages as shown (It is same as that of fie folder system)



It contains some other subpackages
classes and interfaces together with
Scanner class

So it can be refered as java.util.Scanner

as this is provided class, we import it from the

JSL using import statement as

import java.util.Scanner

You know the concept of path

we used the command >>set path=c:\program files\.....\bin

which will gives the location of java compiler.
in same manner class path command is there, which will gives the location from
where we have to load the class when that class not present at current location or
not present at default path(i.e.in the JSL)

Defining our own package:

Java source file can any (or all) of the following four internal parts:

- A single package statement (optional)
- Any number of import statements (optional)
- A single public class declaration (required)
- Any number of classes private

// Importing predefined class

// main class: E:\javaprogramsdacoe ->

```
MainPackageDemoExample.java
import java.util.Scanner;
public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
");
                int no=sc.nextInt();
        }
}

//-----------------------------------------------------
---------------------------------

        // Using class from same file (means same
package)

        // Using class from file - i.e. same package

        Note - All the member i.e. private, public,
protected and default are accessible within class.
                    and the private member not accessible
directly within same package

//ArithOperations class:
E:\javaprogramsdacoe\MainPackageDemoExample.java ->
ArithOperations.java (same file)
// same package means write above as
E:\javaprogramsdacoe\ -> ArithOperations.java (Same
package)
// (same file and same package are exactly same)
package javaprogramsdacoe;
```

```
import java.util.Scanner;
class ArithOperations
{
        private int ans; // visibility - private
        public int b;
        protected int c;
        int d;
        public ArithOperations() //
        {

                ans=0;
                b=10;
                c=100;
                d=1000;
        }
        //.....
        public void add10(int no) // visibility - public
        {
                ans=10+no;
        }
        int getAns()  // visibility - default
        {
                return ans;
        }
}

//main class: E:\javaprogramsdacoe ->
MainPackageDemoExample.java
public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
```

```
");
                int no=sc.nextInt();

                ArithOperations ob=new
ArithOperations();
                ob.add10(no);
                System.out.println("ans after adding 10:
"+ob.getAns());

                // ob.ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
        ob.b=171;
        ob.c=1781;
        ob.d=511;

        }
}

//-------------------------------------------------------
---------------------------------------

// Using class from same package: (non-subclass)
// same package means write above as
E:\javaprogramsdacoe\ -> ArithOperations.java (Same
package)
// (same file and same package are exactly same)

        // In same package non subclass unable to access
only private all others are accessible.

package javaprogramsdacoe;
import java.util.Scanner;
class ArithOperations
```

```
{
        private int ans; // visibility - private
        public int b;
        protected int c;
        int d;
        public ArithOperations() //
        {

                ans=0;
                b=10;
                c=100;
                d=1000;
        }
        //.....
        public void add10(int no) // visibility - public
        {
                ans=10+no;
        }
        int getAns()  // visibility - default
        {
                return ans;
        }
}

//main class: E:\javaprogramsdacoe ->
MainPackageDemoExample.java
public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
");
                int no=sc.nextInt();
```

```
                ArithOperations ob=new
ArithOperations();
                ob.add10(no);
                System.out.println("ans after adding 10:
"+ob.getAns());

                // ob.ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
        ob.b=171;
        ob.c=1781;
        ob.d=511;

        }
}

        compile and run option
E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

E:\javaprogramsdacoe>cd..

E:\>java javaprogramsdacoe.MainPackageDemoExample
Enter any number:
11
ans after adding 10: 21

//
------------------------------------------------------------
-----------------------------------------------

/// Subclass in same package
```

```
// In same package subclass unable to access only
private all others are accessible.


package javaprogramsdacoe;
import java.util.Scanner;
class MyWork extends ArithOperations
{
        public void setData()
        {
                ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
        b=171;
        c=1781;
        d=511;
        }
        public void showData()
        {
                //System.out.println("\n a="+a);
                System.out.println("\n b="+b+"\t
c="+c+"\t d="+d);
        }
}
//main class: E:\javaprogramsdacoe ->
MainPackageDemoExample.java
public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
");
                int no=sc.nextInt();
```

```
                MyWork ob=new MyWork();
                ob.setData();
                ob.showData();
        }
}
/*
        on compile before commenting a

E:\>cd javaprogramsdacoe

E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java
MainPackageDemoExample.java:9: error: ans has private
access in ArithOperations
                ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
                ^
1 error

*/

/*
        After commenting ans
E:\javaprogramsdacoe>javac ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java

E:\javaprogramsdacoe>cd..

E:\>java javaprogramsdacoe.MainPackageDemoExample
Enter any number:
```

```
12

 b=171   c=1781  d=511
*/

//-----------------------------------------------------
-----------------------------------------------------

/// Sub class and from different package
        // private and default not accessible

// E:\pack1\demo\ArithOperations.java
package demo;
public class ArithOperations
{
        private int ans; // visibility - private
        public int b;
        protected int c;
        int d;
        public ArithOperations() //
        {
//
                ans=0;
                b=10;
                c=100;
                d=1000;
        }
        //.....
        public void add10(int no) // visibility - public
        {
                ans=10+no;
        }
        int getAns()  // visibility - default
        {
```

```
                return ans;
        }
}

//main class: E:\javaprogramsdacoe ->
MainPackageDemoExample.java
package javaprogramsdacoe;
import demo.ArithOperations;
import java.util.Scanner;
class MyWork extends ArithOperations
{
        public void setData()
        {
                ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
        b=171;
        c=1781;
        d=511;
        }
        public void showData()
        {
                //System.out.println("\n a="+a);
                System.out.println("\n b="+b+"\t
c="+c+"\t d="+d);
        }
}

public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
```

```
");
                int no=sc.nextInt();

                MyWork ob=new MyWork();
                ob.setData();
                ob.showData();
        }
}

/*
         Errors while compile and execution

E:\javaprogramsdacoe>javac MainPackageDemoExample.java
MainPackageDemoExample.java:5: error: cannot find symbol
class MyWork extends ArithOperations
                                    ^

//-------------------------------------------------------
------
E:\javaprogramsdacoe>set classpath=.*;e:\pack1;

E:\javaprogramsdacoe>javac
e:\pack1\demo\ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java
MainPackageDemoExample.java:4: error: ArithOperations is
not public in demo; cannot be accessed from outside
package
import demo.ArithOperations;
           ^
MainPackageDemoExample.java:6: error: ArithOperations is
not public in demo; cannot be accessed from outside
package
class MyWork extends ArithOperations
```

```
//-------------------------------------------------------
--------------

E:\javaprogramsdacoe>javac
e:\pack1\demo\ArithOperations.java
e:\pack1\demo\ArithOperations.java:3: error: modifier
protected not allowed here
protected class ArithOperations

//-------------------------------------------------------
-------------------

E:\javaprogramsdacoe>javac
e:\pack1\demo\ArithOperations.java

E:\javaprogramsdacoe>javac MainPackageDemoExample.java
MainPackageDemoExample.java:10: error: ans has private
access in ArithOperations
                ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
                ^
MainPackageDemoExample.java:13: error: d is not public
in ArithOperations; cannot be accessed from outside
package
        d=511;
        ^
MainPackageDemoExample.java:18: error: d is not public
in ArithOperations; cannot be accessed from outside
package
                System.out.println("\n b="+b+"\t
c="+c+"\t d="+d);
```

```
      ^
3 errors

Note that as ans and d are private and default, thay are
not accessible

*/

//-----------------------------------------------------
-----------------------------------------------

        // class from different package non-subclass
                // - having only access to public

// E:\pack1\demo\ArithOperations.java
package demo;
public class ArithOperations
{
        private int ans; // visibility - private
        public int b;
        protected int c;
        int d;
        public ArithOperations() //
        {
//
                ans=0;
                b=10;
                c=100;
                d=1000;
        }
        //.....
        public void add10(int no) // visibility - public
        {
                ans=10+no;
```

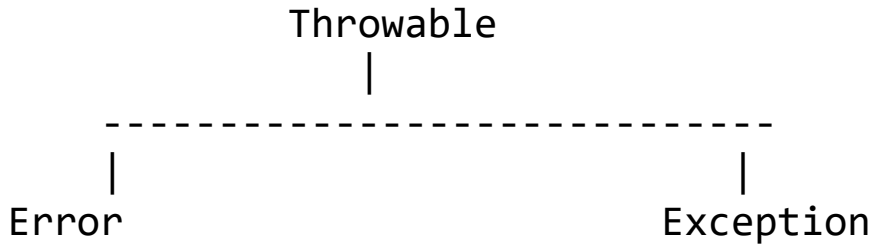
```
        }
        int getAns()  // visibility - default
        {
                return ans;
        }
}

//main class: E:\javaprogramsdacoe ->
MainPackageDemoExample.java
package javaprogramsdacoe;
import demo.ArithOperations;
import java.util.Scanner;

public class MainPackageDemoExample
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter any number:
");
                int no=sc.nextInt();

                ArithOperations ob=new
ArithOperations();
                ob.add10(no);
                System.out.println("ans after adding 10:
"+ob.getAns());

                ob.ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
        ob.b=171;
        ob.c=1781;
        ob.d=511;
```

```
        }
}

/*
         Errors while compile and execution

E:\javaprogramsdacoe>javac MainPackageDemoExample.java
MainPackageDemoExample.java:16: error: getAns() is not
public in ArithOperations; cannot be accessed from
outside package
                System.out.println("ans after adding 10:
"+ob.getAns());

    ^
MainPackageDemoExample.java:18: error: ans has private
access in ArithOperations
                ob.ans=11;
//MainPackageDemoExample.java:44: error: ans has private
access in ArithOperations
                  ^
MainPackageDemoExample.java:20: error: c has protected
access in ArithOperations
        ob.c=1781;
          ^
MainPackageDemoExample.java:21: error: d is not public
in ArithOperations; cannot be accessed from outside
package
        ob.d=511;
          ^
4 errors

note: for class from different package non-subclass
having only access to public
*/
```

```
//-------------------------------------------------------
-------------------------------------------------------
------
                Access Protection Chart

                                                private
default         protected       public

  same class                                    Yes
Yes             Yes             Yes

  same package subclass                         No
Yes             Yes             Yes

  same package non-subclass                     No
Yes             Yes             Yes

  Different package subclass                    No
No              Yes             Yes

  Different package non-subclass                No
No              No              Yes


//=======================================================
=======================================================
=========
//=======================================================
=======================================================
=========
```

/// Exception Handing

Java Exception hierarchy
Here is a simplified diagram of the exception hierarchy in Java.



the Throwable class is the root class in the hierarchy.

Note that the hierarchy splits into two branches: Error and Except

Errors:

An error is the mistake in developing program causes a unexpected output or we are unable to execute the program.

The error may of two types
- Compile time errors:
- Run time errors:

Compile time errors: these are the syntax errors which are displayed by java compiler and therefore these are known as Compile time errors

e.g. use of undeclared variable
bad reference to object

missing semi comma
misspelling of identifier and keyword
missing/mismatching brackets in classes and methods

Run time errors: some time the program may compile but gives error at the time of execution due to some mistakes
at run time such errors are known as Run time errors
e.g. divide an integer by zero
access an element that is out of bounds of an array
attempting to use a negative size for an array

Exceptions
Exceptions can be caught and handled by the program. When an exception occurs within a method,
it creates an object. This object is called the exception object. It contains information about the
exception such as the name and description of the exception and state of the program when the
exception occurred.

now focus on different types of exceptions in Java.

Java Exception Types
The exception hierarchy also has two branches: RuntimeException and IOException.

1. RuntimeException
A runtime exception happens due to a programming error. They are also known as unchecked exceptions.
These exceptions are not checked at compile-time but run-time.

Some of the common runtime exceptions are:

- Improper use of an API - IllegalArgumentException
- Null pointer access (missing the initialization of a variable) - NullPointerException
- Out-of-bounds array access - ArrayIndexOutOfBoundsException
- Dividing a number by 0 - ArithmeticException

You can think about it in this way. “If it is a runtime exception, it is your fault”.
The NullPointerException would not have occurred if you had checked whether the variable was initialized or not before using it.
An ArrayIndexOutOfBoundsException would not have occurred if you tested the array index against the array bounds.

2. IOException
An IOException is also known as a checked exception. They are checked by the compiler at the compile-time and the programmer is prompted to handle these exceptions.

Some of the examples of checked exceptions are:

- Trying to open a file that doesn’t exist results in FileNotFoundException

- Trying to read past the end of a file

Java Exception Handling - Note that handling the exception means not elimination it.
We know that exceptions abnormally terminate the execution of a program, to avoid it, it is important to handle exceptions. And it will allows us to take some corrective efforts.

There are 5 different keywords provided to handle the exceptions. and these are

try catch finally throw throws

```
try                         try
try
{                           {
{
  .........                   ..........
}
}                           }
catch(Exception e)
catch(Exception e)          catch(XException e)
{
{                           {
}
  ---------                 }
finally
}                           catch(YException e)
{
                            {
}
                            }
```

```
                        catch(Exception e)
                        {
                        }
```

```
try
{

}
catch (ExceptionType1 Ob)
{

}
catch (ExceptionType2 Ob)
{

}
finally
{
}
```

```
try
{

}
finally
{
}
```

```
//-------------------------------------------------------------
------------------------------

class DemoErrorException
{
        public static void main()
        {
                double y;

                x=10;
                y=12.45;
```

```
                System.out.println("x="+x+"\t y="+y);
        }
}/*
E:\jprodyp>javac DemoErrorExceprion.java
DemoErrorExceprion.java:8: error: cannot find symbol
                x=10;
                ^
  symbol:   variable x
  location: class DemoErrorExceprion
DemoErrorExceprion.java:11: error: cannot find symbol
                System.out.println("x="+x+"\t y="+y);
                                           ^
  symbol:   variable x
  location: class DemoErrorExceprion
2 errors
*/

//---------------------------------------


class DemoErrorException
{
        public static void main(String []args)
        {
                int x;
                double y;

                x=args[0];
                y=args[1];

                System.out.println("x="+x+"\t y="+y);
        }
}
```

```
E:\jprodyp>javac DemoErrorException.java

E:\jprodyp>javac DemoErrorException.java
DemoErrorException.java:9: error: incompatible types:
String cannot be converted to int
                x=args[0];
                      ^
DemoErrorException.java:10: error: incompatible types:
String cannot be converted to double
                y=args[1];
                      ^
2 errors

//-----------------------------------------------------
------------------------------

class DemoErrorException
{
        public static void main(String []args)
        {
                String s1;
                String s2;

                s1=args[0];
                s2=args[1];

                System.out.println("s1="+s1+"\t
s2="+s2);
        }
}


E:\jprodyp>javac DemoErrorException.java
```

```
E:\jprodyp>java DemoErrorException
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of
bounds for length 0
        at
DemoErrorException.main(DemoErrorException.java:9)


//-------------------------------------------------------
--------------------------

        // Using try catch block

class DemoErrorException
{
        public static void main(String []args)
        {
                String s1="";
                String s2="";

                try
                {
                        s1=args[0];
                        s2=args[1];
                }
                catch(Exception e)
                {

System.out.println("---------------------------------");
                        System.out.println("One:
"+e.getMessage());

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

```
                        System.out.println("Two: "+e);

System.out.println("---------------------------------");
                        System.out.println("Three:");
                        e.printStackTrace();

System.out.println("---------------------------------");
                        System.out.println("Four: Index
Issue");
                }

                System.out.println("s1="+s1+"\t
s2="+s2);
        }
}

/*
        --------------- Run Without argiments
-------------------

E:\jprodyp>javac DemoErrorException.java

E:\jprodyp>java DemoErrorException
---------------------------------
One: Index 0 out of bounds for length 0
---------------------------------
Two: java.lang.ArrayIndexOutOfBoundsException: Index 0
out of bounds for length 0
---------------------------------
Three:
java.lang.ArrayIndexOutOfBoundsException: Index 0 out of
bounds for length 0
        at
DemoErrorException.main(DemoErrorException.java:11)
```

```
--------------------------------
Four: Index Issue
s1=      s2=

        ------------------------- Run with single
argument --------------------------------
E:\jprodyp>java DemoErrorException hello
--------------------------------
One: Index 1 out of bounds for length 1
--------------------------------
Two: java.lang.ArrayIndexOutOfBoundsException: Index 1
out of bounds for length 1
--------------------------------
Three:
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of
bounds for length 1
        at
DemoErrorException.main(DemoErrorException.java:12)
--------------------------------
Four: Index Issue
s1=hello         s2=

        ------------------------------- Run with 2
arguments -----------------

E:\jprodyp>java DemoErrorException hello all
s1=hello         s2=all

*/

//-----------------------------------------------------
---------------------------------------

//-----------------------------------------------------
```

```
------------------------------

        // Using try Multiple catch blocks

try to run program and handle all possible exceptions

import java.util.Scanner;
class ExampleUsingMultipleCatch
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter any count:
");
                int cnt=sc.nextInt();
                int []x=new int[cnt];
                for(int i=0;i<5;i++)
                {
                        x[i]=Integer.parseInt(args[i]);
                }

                System.out.println("\n Enter any
position: ");
                int pos=sc.nextInt();
                System.out.println("\n Value at
position: "+x[pos]);
        }
}
//-----------------------------------------------------
---------------------------------------

import java.util.*;
class ExampleUsingMultipleCatch
{
```

```
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter any count:
");

                try
                {       int cnt=sc.nextInt();
                        int []x=new int[cnt];
                        for(int i=0;i<5;i++)
                        {
                                x[i]=sc.nextInt();
                        }

                        System.out.println("\n Enter any
position: ");
                        int pos=sc.nextInt();
                        System.out.println("\n Value at
position: "+x[pos]);
                }
                catch(InputMismatchException ime)
                {
                        System.out.println("\n Immproper
input");
                }
                catch(ArrayIndexOutOfBoundsException ae)
                {
                        System.out.println("\n Index
Issue");
                }
                catch(Exception e)
                {
                        System.out.println("\n Execution
problem");
```

```
                }
                System.out.println("\n End of program");
        }
}
//-----------------------------------------------------
---------------------------------------
```

Java finally block:
In Java, the finally block is always executed no matter whether there is an exception or not.
The finally block is optional. And, for each try block, there can be only one finally block.
The finally block is typically used to perform cleanup operations or release resources that need to be executed regardless of whether an exception occurred.

```
import java.util.*;
class ExampleUsingMultipleCatch
{
        public static void main(String[] args)
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("\n Enter any count:
");

                try
                {       int cnt=sc.nextInt();
                        int []x=new int[cnt];
                        for(int i=0;i<5;i++)
                        {
                                x[i]=sc.nextInt();
                        }
```

```
                        System.out.println("\n Enter any
position: ");
                        int pos=sc.nextInt();
                        System.out.println("\n Value at
position: "+x[pos]);
                }
                catch(InputMismatchException ime)
                {
                        System.out.println("\n Immproper
input");
                }
                catch(ArrayIndexOutOfBoundsException ae)
                {
                        System.out.println("\n Index
Issue");
                }
                catch(Exception e)
                {
                        System.out.println("\n Execution
problem");
                }
                finally
                {
                        System.out.println("\n in
finally block");
                }
                System.out.println("\n End of program");
        }
}

//-----------------------------------------------------
-----------------------------------
```

// Using throw keyword

In Java, the throw statement is used to explicitly throw an exception. This allows you to create and throw your own exceptions exceptions that are caught in your code. The throw statement is typically used inside the try block of a try-catch statement or within a method that declares to throw an exception using the throws keyword.

```
import java.util.Scanner;
class FormFill
{
        private String name;
        private int age;
        private double per;
        public FormFill()
        {
                name="";
                age=0;
                per=0.0;
        }
        public void input()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the name: ");
                name=sc.nextLine();
                System.out.println("Enter the age: ");
                age=sc.nextInt();
                if(age<=20 || age>30)
                {
                        throw new
ArrayIndexOutOfBoundsException();
                }
                System.out.println("Enter the
percentage: ");
```

```
                per=sc.nextDouble();
        }
        public void output()
        {
                System.out.println("Student Details:
Name: "+name+"\n Age: "+age+"\t Percentage: "+per);
        }
}

class FormFillProcess
{
        public static void main(String []args)
        {
                FormFill ob=new FormFill();
                ob.input();
                ob.output();
        }
}


/*
E:\jprodyp>java FormFillProcess
Enter the name:
amit
Enter the age:
23
Enter the percentage:
90.45
Student Details: Name: amit
 Age: 23         Percentage: 90.45

//---------------
E:\jprodyp>java FormFillProcess
Enter the name:
```

```
sumit
Enter the age:
33
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException
        at FormFill.input(FormFill.java:22)
        at FormFillProcess.main(FormFill.java:38)

*/

//-----------------------------------------------------
-------------------------------------

        // handling exception
import java.util.Scanner;
class FormFill
{
        private String name;
        private int age;
        private double per;
        public FormFill()
        {
                name="";
                age=0;
                per=0.0;
        }
        public void input()
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the name: ");
                name=sc.nextLine();
                System.out.println("Enter the age: ");
                age=sc.nextInt();
                if(age<=20 || age>30)
```

```
                {
                        throw new
ArrayIndexOutOfBoundsException("Problem due to age is
not within age window");
                }
                System.out.println("Enter the
percentage: ");
                per=sc.nextDouble();
        }
        public void output()
        {
                System.out.println("Student Details:
Name: "+name+"\n Age: "+age+"\t Percentage: "+per);
        }
}

class FormFillProcess
{
        public static void main(String []args)
        {
                FormFill ob=new FormFill();
                try
                {
                        ob.input();
                        ob.output();
                }
                catch(Exception e)
                {

System.out.println(e.getMessage());
                }
        }
}
//-----------------------------------------------------
```

/ Using throws keyword

In Java, the throws clause is used in a method signature to declare that the method may throw certain types of exceptions. This provides information to the caller of the method about the potential exceptions that need to be handled. The throws clause is used to delegate the responsibility of exception handling to the calling code.

- You can declare multiple exceptions in the throws clause, separating them with commas.
- Checked exceptions (those that extend Exception but not RuntimeException) must be declared in the throws clause. Unchecked exceptions (those that extend RuntimeException) do not need to be declared.
- If a method declares that it throws an exception, means it is not handing it. It is just to inform to the caller about the his responsibility.
- Methods are not required to declare unchecked (runtime) exceptions in the throws clause.

```
import java.util.Scanner;
import java.io.*;
class FormFill
{
        private String name;
        private int age;
        private double per;
        public FormFill()
        {
```

```
                name="";
                age=0;
                per=0.0;
        }
        public void input() throws IOException
        {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the name: ");
                name=sc.nextLine();
                System.out.println("Enter the age: ");
                age=sc.nextInt();
                if(age<=20 || age>30)
                {
                        throw new IOException("Problem
due to age is not within age window");
                }
                System.out.println("Enter the
percentage: ");
                per=sc.nextDouble();
        }
        public void output()
        {
                System.out.println("Student Details:
Name: "+name+"\n Age: "+age+"\t Percentage: "+per);
        }
}

class FormFillProcess
{
        public static void main(String []args)
        {
                FormFill ob=new FormFill();
                try
                {
```

```
                        ob.input();
                        ob.output();
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        }
}
```