19/07/23

JAVA

Program

* Program is a set of instructions to perform task.
* The language which is used to communicate with a machine is known as programming language
* Programming languages are classified into 3 types:
  - High-level programming language
  - Mid-level programming language
  - low-level programming language

High-level programming language:

Programming language which can be readable is known as high level programming language

Eg Java, Javascript, python, c++, etc.,

Mid-level programming language

Programming language which is partially readable is known as mid-level programming language

Eg Bytecode

low-level programming language

Machine understandable language is known as low-level language

Eg Binary language

Binary language

* Machine understable language is known as Binary language
* In binary language, every character is represented in using 0 and 1.

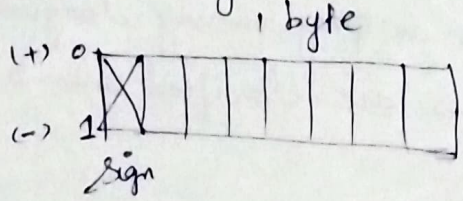
Note

* Generally, the machines will understand only electrical signals that is ON and OFF. which are represented

using 0 and 1.

* The characters we pass to the system will be represented in the binary

* Since we have positive and negative values, the first bit of memory location is used to represent sign.



* To find max value that we can store in a memory location, we have formula

$\boxed{2^n - 1}$ where n = no. of bits

Decimal to binary conversion

1) 10

2	10	
2	5	-0
2	2	-1
	1	-0

$10 \Rightarrow (1010)_2$

2) 59

2	59	
2	29	-1
2	14	-1
2	7	-0
2	3	-1
	1	-1

$59 \Rightarrow (111011)_2$

Binary to Decimal Conversion

1) 1010

1 0 1 0

* * * *

$2^3\ 2^2\ 2^1\ 2^0$

$8 + 0 + 2 + 0 = 10$

$1010 \Rightarrow (10)_{10}$

2) 11001101

1 1 0 0 1 1 0 1

* * * * * * * *

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

$128 + 64 + 0 + 0 + 8 + 4 + 0 + 1 \Rightarrow 205$

$11001101 \Rightarrow (205)_{10}$

20/07/2021

Java

* Java is a high-level class based, object-oriented Programming language.
* Java is developed by Sun Micro systems (part of Oracle corporation) in the year 1995.
* Father of Java is James Gosling.
* Java consists of 3 editions. They are
  - Standard Edition => To develop standalone applications
  - Enterprise Edition => To develop enterprise applications
  - Micro Edition => To develop mobile applications.

Features of Java

* Simple to learn
* Robust & Secure language
* Object oriented language
* Multi threaded language
* platform independent

Why Java is platform independent?

* Java is platform independent programming language because of JVM (Java Virtual machine)
* Java compiler converts source file into class file which is converted into binary and executed by the JVM.

Steps to create and execute Java program

Step 1: Create source file.

* We can create source file by using editors like Notepad / Editplus
* Source file must be saved with .java extension.

Step 2: Generate class file

* We can generate class file by giving source file as input to the compiler using command:

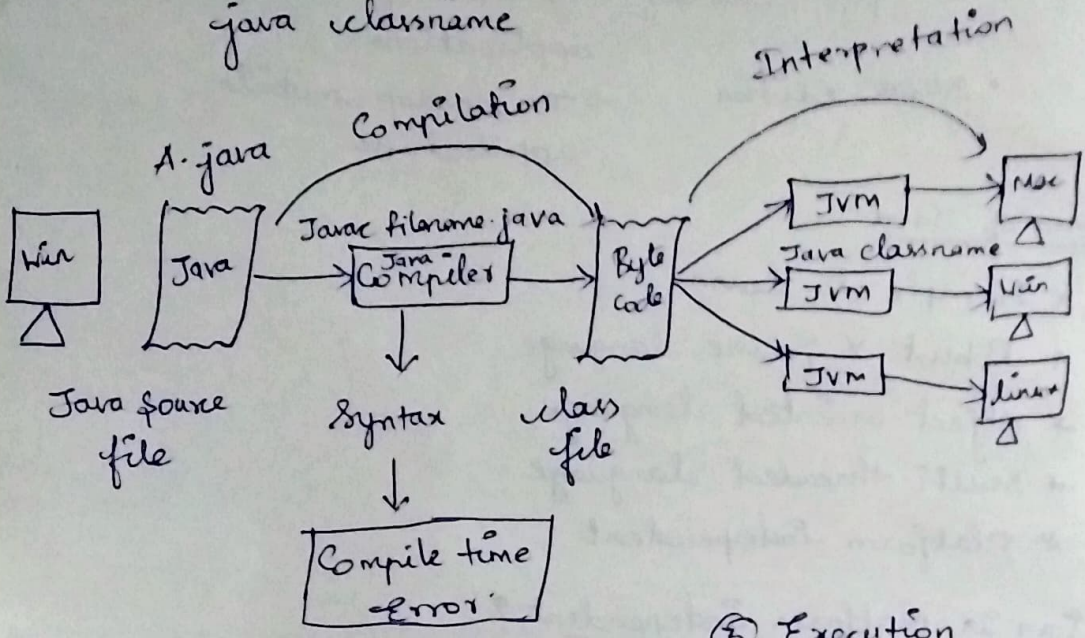
javac filename.java

* If all syntax are proper, Compiler will generate class file else we get an error which is known as compile time error.

Step 3: Execution

* We can execute program by giving class file as input to the JVM by using command:

java classname



21/07/23

Command prompt commands

① change drive

drivename:

② Select folder

cd foldername

③ Remove and Exit folder

cd..

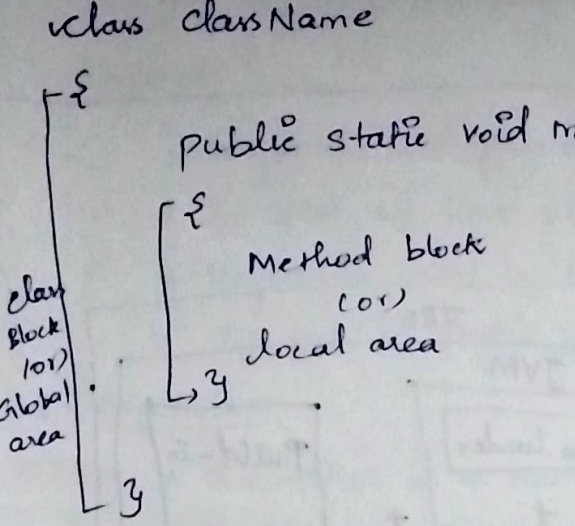
④ clear screen

cls

Note:

To compile a program in the command prompt, we must provide absolute path that is path of an file from the source folder

Structure of Java Program



* In Java, we have two basic blocks They are
  - class block (or) global area
  - Method block (or) local area

Note:

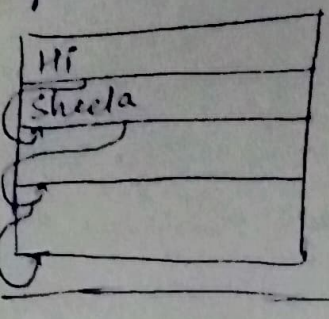
* To compile a program, main method is not mandatory
* To execute a program, main method is mandatory

Print statements

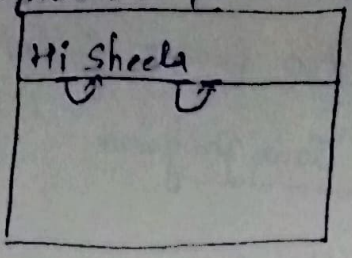
* In Java, we have 2 basic print statements

System.out.println(data)	System.out.print(data)
* Println() statement is used to print the data and move the cursor to next line. * Println statement can be used without passing any data.	* Print() Statement is used to print the data and move the cursor to next position in the same line * Print statement cannot be used without passing any data.

System.out.println("Hi");
System.out.println("Aruna");
System.out.println();

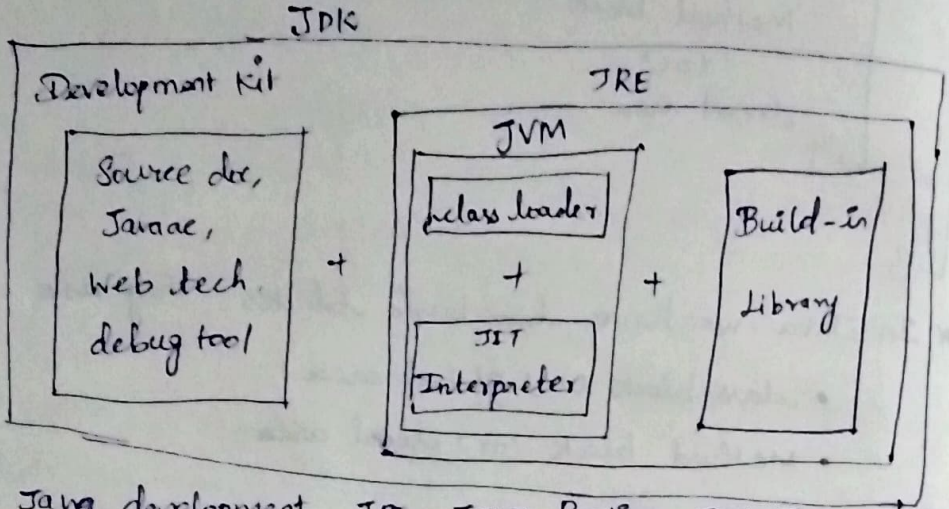


System.out.print("Hi");
System.out.print("Aruna");
System.out.print(); ⇒ Compile time error



24/07/23

1) Difference b/w JDK, JRE & JVM.



JDK - Java development Kit
JRE - Java Runtime Environment
JVM - Java Virtual machine
JIT - Just In Time

JDK	JRE	JVM
* JDK stands java Development Kit	* JRE stands for java runtime Environment	* JVM stands for java virtual machine
* JDK is used to develop and run an application	* JRE is used to execute a java program	* JVM is used to convert bytecode into binary
* JDK consists of development tools and JRE	* JRE consists of JVM and build-in library.	* JVM consists of class loader and JIT interpreter

Tokens

* Tokens are basic elements of a programming language
* In Java, we have 4 tokens They are
    1. Keywords
    2. Identifiers
    3. Literals/ Input / Data
    4. Symbols

1 Keywords

* Keywords are pre-defined words
* In Java, we have 53 keywords
* Keywords must be written in lowercase
* Eg public, static, void, etc.

2 Identifiers

* Identifier is a name given to the java member like variable, method and class.
* Identifier rules:
    i) Identifier should not start with number
    ii) Special characteristics not allowed other than '_' and '$'
    iii) Space is not allowed
    iv) Keywords cannot be used as identifier

Note

Don't use built-in class or package name

Naming convention

* Naming convention is an industry standard followed by programmers for naming elements
* The naming convention for class name is

Pascal Case

* Eg: FileNotFoundException
* Eg: class 123Demo //CTE
  class Dinga@Dingi //CTE

class Dinga Dingi //CTE
class public //CTE
class Voidclass //CTS

3. Literals

* Literals are user inputs.
* In Java, we have 4 literals They are

i) Number
ii) Character
iii) String (group of characters)
iv) Boolean

i) Number

* In java we can pass number directly

Eg System.out.println(20);

ii) character

* In java, we can pass character using single quotes
* Whenever we pass character, ASCII values (American standard Code for Information Interchange) will be considered.

'0' ~ '9'	'A' ~ 'Z'	'a' ~ 'z'	Special characters
48 ~ 57	65 ~ 90	97 - 122	

Eg
* System.out.println(A); //CTE
* System.out.println('A'); //A

iii) String

* String is a group of characters
* In Java, we can pass group of characters or String with using double quotes

Eg System.out.println(sheela); //CTE
System.out.println('sheela'); //CTE

System.out.println("sheela");

iv) Boolean

* In Java, we have 2 Boolean literals They are true and false
* True and False are keywords we can pass directly

-Eg: System.out.println(True);
System.out.println(False);

-Eg: program

```
class Literals
{
  public static void main(String[] args)
  {
    //Number
    System.out.println(143); //143 - Number
    //character
    System.out.println('A'); //A
    System.out.println('*'); //*   } - character
    System.out.println('5'); //5
    //string
    System.out.println("sheela"); //sheela - string
    //Boolean
    System.out.println(true); //true   } - Boolean
    System.out.println(false); //false
  }
}
```

4) Symbols

* Symbols are used to separate one statement from the another statement

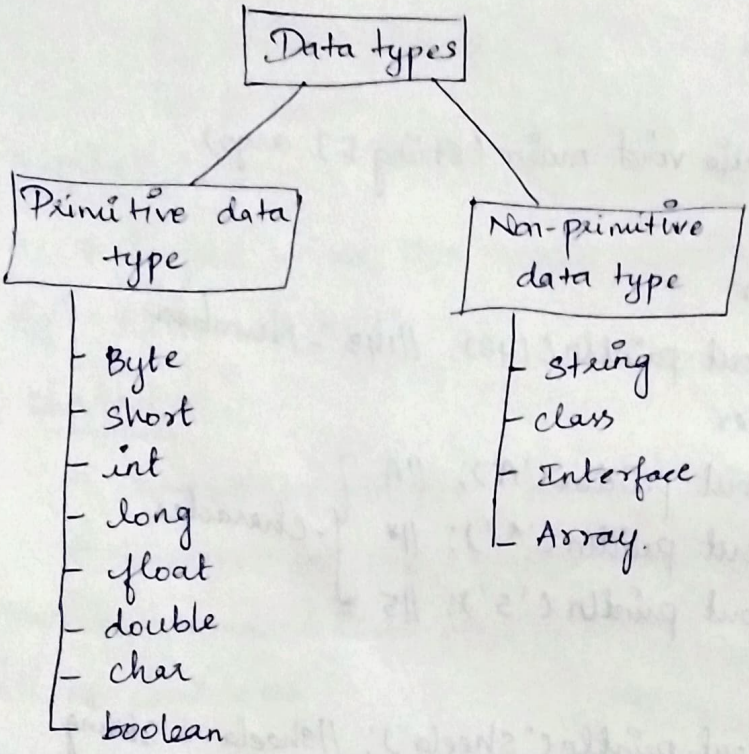
Eg: {, }, (), ;

Note: In Java, every statement must have either a block or semicolon.

10/1/23

Data types

* Data types are used to specify type and size of memory location
* Data types are classified into 2 types. They are
  i) Primitive data type
  ii) Non-primitive data type



Literals		Data types	Default size	Default value
① Number	Whole number	byte	1 byte	0
		short	2 byte	0
		int	4 byte	0
		long	8 byte	0L
	Decimal	float	4 byte	0.0F
		double	8 byte	0.0
② character		char	2 byte	space
③ Boolean		boolean	1 bit	False
④ String		String	NA	null.

Variable

* Variable is the name given to the memory location
* In Java, we can declare and initialize variable in 2 ways.

i) Declaring and Initializing separately.
ii) Declaring and Initializing together

Declaration statements

* Declaration statement is used to declare a variable.

Syntax:

```
datatype identifier;
```

Ex: byte a; → a [0] byte, 1 byte

Ex: String name; → name [null] String, NA

Ex: double bal; → bal [0.0] double, 8 byte

Initialization statement

* Initialization statement is used to assign a value to the variable

Syntax:

```
Identifier = value;
```

Ex:

a = 20; → a [20]

bal = 2500.0; → bal [2500.0]

name = "sheela"; → name [sheela]

- Declare and Initialization statement:

Syntax: `Datatype identifier = value;`

-Ex:

String name = "Allen"; → name | Allen (String)

int age = 25; → age | 25 (int)

Program

```
class Datatypes
{
  public static void main (String[] args)
  {
    // Declaration stmt
    byte a;
    String name;
    double salary;
    // Initialization stmt
    id = 101;
    name = "sheela";
    salary = "2500.0;
  // Declare and Initialization stmt
    int age = 25;
  System.out.println(id); //101
  System.out.println(name); // sheela
  System.out.println(salary); //2500.0
  System.out.println(age);  //25
  }
}
```

Note:

The naming Convention for a variable is Camelcase

Ex:

additionOfTwoNumber

StudentName

StudentIdNumber

1) Write a Java program to store Student details and print it

2) Write a Java program to find addition of two whole numbers

3) Write a Java program to find jump required to cover given distance

i) Distance is 5m

ii) One jump Covers 1.2 m

26/07/23

Typecasting

* The process of converting one data type into another data type is known as typecasting.

* In Java, Typecasting is classified into two types

i) Primitive typecasting

ii) Non-primitive typecasting

Primitive typecasting

* Data type conversion between primitive data types is known as primitive typecasting.

* Primitive typecasting is categorized into 2 types They are

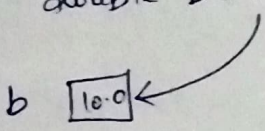
i) Widening

ii) Narrowing

Widening

* Process of converting smaller data type into larger datatype is known as widening
* Widening is done by the compiler implicitly

-Ex: double b = 10;



Narrowing

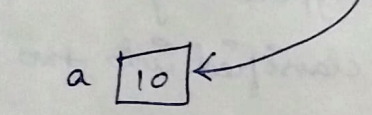
* Process of converting larger datatype into smaller datatype is known as narrowing
* Narrowing is not done by the compiler programmer has to do explicitly using typecast operator

Typecast operator

* Typecast operator is used to convert one datatype into another datatype
* Syntax:

(data type) value;

Ex: int a = 10.5; int(10.5);



byte < short < int < long < float < double

char

Program:

```
class TypeCasting
{
public static void main(String[] args){
{
//Widening => Converting smaller datatype into
//larger datatype
```

```
double b = 10;
System.out.println("Var b:: "+b); //10.0
//Narrowing :=> Converting larger datatype into smaller datatype
int a = (int)50.5;
System.out.println("Var a: "+a); //50
// Type cast operator
System.out.println((char)65); //A
System.out.println((double)65); //65.0
}
}
```

Operator - Input
task
Return type

27/07/23.

Operator

* Operator is an predefined symbol which is used to perform specific tasks.

* Based on number of inputs, operator accepts they are classified into 3 types:

i) Unary operator - accepts only one input.

ii) Binary operator - accepts only two inputs

iii) Ternary operator - accepts only three inputs.

Ex:

i) Arithmetic operators [+, -, /, *, %.]

ii) Relational operators [>, <, >=, <=, ==, !=]

iii) Conditional operator

iv) Increment and Decrement operator [++, --]

v) Logical operator [&& (AND), || (OR), ! (NOT)]

vi) Compound Assignment operator [+=, -=, /=, *=, %=]

Arithmetic operators. [+, -, *, /, %.].

* Arithmetic operators are binary operators.

* The return type of arithmetic operator follows the formula

MAX(int, IP1, IP2)

* the associativity of arithmetic operators is left to right.

Note:

* In Java, '+' operator has two behaviours that is

i) Addition

ii) Concatination (joining).

* Whenever we have string literal present, the '+' operator will act as concatination operator.

Program

```
class P1
{
  public static void main(String[] args)
  {
    // Addition
    System.out.println(10+20);    //30
    System.out.println(10+20.0);  //30.0
    System.out.println(5/2);      //2
    System.out.println(5/2.0);    //2.5
    System.out.println(10+'A');   //75
    System.out.println('A'+'B');  //131
    // Concatenation
    System.out.println(10+"Hi");  //"10Hi"
    System.out.println(10+20+"A"); //"30A"
    System.out.println("A"+10+20); //"A1020"
  }
}
```

2

28/07/23

Relational operators [>, <, >=, <=, ==, !=]

* Relational operators are binary operators
* Relational operators are used to compare two literals
* Relational operator's return type is Boolean

Note:

* String and Boolean literals can be used only with "==" or "!=" operators

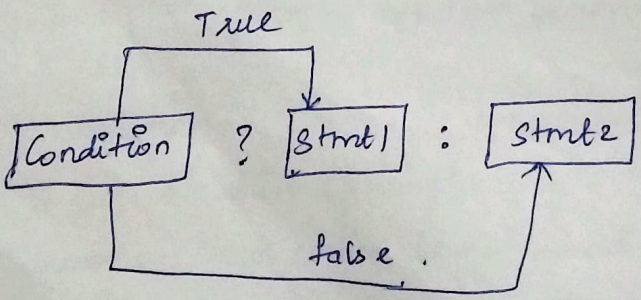
Eg:

Program:

```
class Relational operators
{
 public static void main(String [] args){
    System.out.println(50>40); //True
    System.out.println(50<40); //false
    System.out.println(50==50); //true
    System.out.println(50.5<=50.2); //false
    System.out.println('B'<'A'); //false
    System.out.println("Hello"=="Hi"); //false
    System.out.println("Mala"!="sheela"); //true
 }
}
```

Conditional operator

Syntax:



* Conditional operator is a Ternary operator
* Conditional operator is used to take a decision based on condition

Question 1:

Write a java program to find largest of two numbers using conditional operator.

```
class LargestOfTwoNumbers
{
 public static void main(String[] args)
  {
   int a=80;
   int b=120;
   int res = a>b ? a : b;
   System.out.println("Largest of "+a+" & "+b+
          "is"+res);
  }
}
```

Question 2

1) Write a java program to check given number is even number or odd number
3) Write A JP to check given number is positive number or negative number
4) WAJPT check given name is valid name or not. The valid name is "Dinga".

2)
```
class OddOrEvenNumber
 {
  public static void main(String[] args)
   {
    int number=30;
    int remainder=number%2;
    String result=remainder==0 ? "Even number":
                  "odd number";
    System.out.println(number+" is "+result);
  }}
```

3) class PositiveNegativeNumber

```
{
  public static void main (String[] args)
  {
    int number = -52;
    String result = number < 0 ? "Negative Number":
                                 "Positive Number";
    System.out.println( number + " is " + result);
  }
}
```

3) class ValidName

```
{
  public static void main (String[] args)
  {
    String name = "Aruna";
    String result = name == "Dinga" ? "Valid name":
                                      "Invalid Name";
    System.out.println(name + " is " + result);
  }
}
```

02/08/23

Increment operator [++]:

* Increment operator is used to increase the value of a variable by 1.
* Increment operator must be used along with variable only.
* Increment operator is classified into 2 types

Pre-increment operator

Syntax:

```
++variable;
```

* In pre-increment operator, the variable value is increased first and then increased value will be used.

*Example:

```
int a=10;
S.o.pln(a); //10
S.o.pln(++a); //11
S.o.pln(a); //11
```

int a 11

Post-increment operator

Syntax:

```
variable++;
```

* In post-increment operator, the variable value will be used first and then increased.

*Example

```
int a=10;
S.o.pln(a); //10
S.o.pln(a++); //10
S.o.pln(a); //11
```

int a 11

Decrement operator [--]:

* Decrement operator is used to decrease the value of a variable by 1.
* Decrement operator must be used along with variable only.
* Decrement operator is classified into two types:

Pre-Decrement

Syntax

```
--variable;
```

In pre-decrement operator, the value of the variable is decreased first and then the decreased value is used

Example

```
int a=10;
S.o.pln(a); //10
S.o.pln(--a); //9
S.o.pln(a); //9.
```

int a: 10 → 9

Post-Decrement

Syntax

```
variable--;
```

In post-decrement operator, the value of the variable is used and then decreased.

Example

```
int a=10;
S.o.pln(a); //10
S.o.pln(a--); //10
S.o.pln(a); 9
```

int a: 10 → 9

```
class IncrementOperator
{
  public static void main(String[] args)
  {
    int a=10;
    int b= ++a +15;
      // b = 11+15 = 26
    int c = a++ + ++b-20;
    // c = 11+27-20 = 38-20
    // c = 18
    int d = a-- - --b + ++c;
    // d = 12 - 26 + 19 = 34
    S.o.pln(++a); // 12
    S.o.pln(++b); // 27
    S.o.pln(--c); // 18
    S.o.pln(d--); // 34
  }
}
```

a: 10 → 11 → 12

b: 26 → 27

c: 18 → 19

d: 34

a: 12

b: 27 ⇒ 11+15 = 26

c: 18 ⇒ 11+27-20 = 18

d: 34 ⇒ 12-26+19 = 5

03/08/2023

Logical Operator [&&, ||, !]

* Logical operators are used to merge multiple conditions

* In Java, we have 3 logical operators
  i) Logical AND (&&)
  ii) Logical OR (||)
  iii) Logical NOT (!)

* The return type of logical operator is **Boolean**

① AND (&&)

I/p1	I/p2	Res
F	F	F
F	T	F
T	F	F
T	T	T

② OR (||)

I/p1	I/p2	Res
F	F	F
F	T	T
T	F	T
T	T	T

③ NOT (!)

I/p	Res
T	F
F	T

Note

* In AND (&&) operator, if the first condition is false second condition will not be check.

* In OR (||) operator, if first condition is true, second condition will not be check.

1) Write a Java program to find the student is pass or fail The marks student got are

maths = 95
eng = 75

```
class PassOrFail
{
  public static void main (String[] args)
  {
    String name = "Mala";
    int maths = 25;
    int english = 75;
```

```
String res = maths>=35 && eng>=35 ? "Pass" : "fail";
S.o.pln("Hi"+name+" your result:"+res);
}
}
```

o/p:

Hi Mala your result is fail

2) WAJPT select t shirt The required colours are Blue or red.

```
class Tshirt
{
  public static void main (String[] args)
  {
    String colour1="Red";
    String colour2="Bule";
    String res = colour1=="Red" || colour2=="Blue" ?
                 "Selected" : "Not Selected";
    S.o.pln(res);
  }
}
```

o/p Selected

3) WAJPT validate user

i) The valid user name is Dinga

ii) The password is Dinga143@Dingi.

```
class ValidateUser
{
  public static void main (String[] args)
  {
    String name="Dinga";
    String password="Dinga143@Dingi";
    String res = name=="Dinga" && password=="Dinga143@
                 Dingi" ?
                 "Valid":"Invalid";
```

```
S.o.pln("Hi"+name+" ,Password and name are: "+
                    else);
    }
}
```

O/p

Hi Dinga, name and password are: valid

Compound Assignment Operator [+=, -=, *=, /=, %=]

* Compound assignment operators are combination of arithmetic operator and assignment operator
* Compound assignment operators are used to update the variable

Eg:

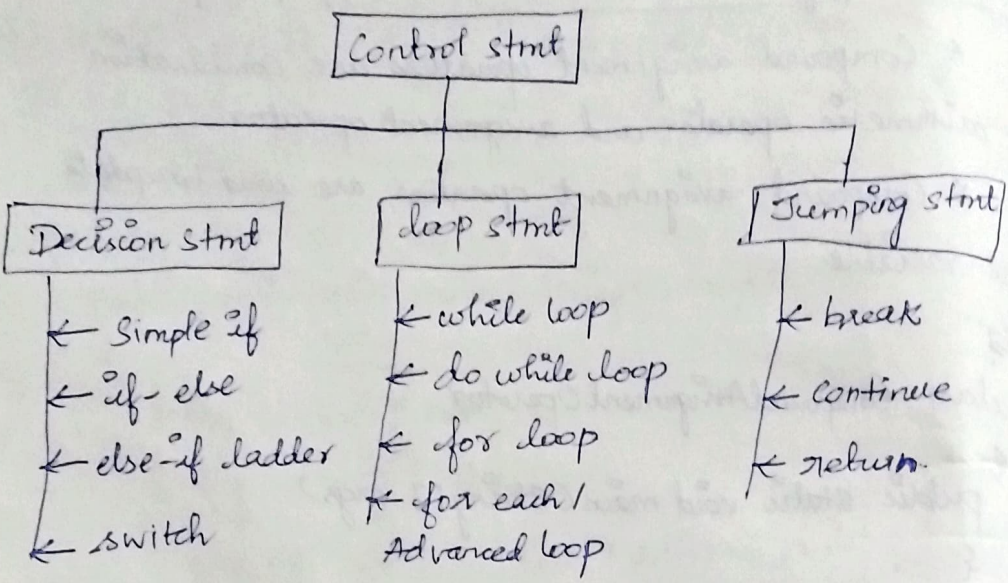
```
class CompoundAssignmentOperator
{
  public static void main(String[] args)
  {
    int bal = 1000;
    S.o.pln("balance:: "+bal); //1000
    bal += 500;
    S.o.pln("balance: "+bal); //1500
    bal -= 300;
    S.o.pln("balance: "+bal); //1200
    bal /= 2;
    S.o.pln("balance: "+bal); //600
    bal *= 2;
    S.o.pln("balance: "+bal); //1200
  }
}
```

04/08/23.

Control Statements

* Control statements are used to control the flow of execution.

* In Java, we have following control statements.



Decision statements

* Decision statements are used to take a decision based on condition

* In Java, we have 4 decision statements. They are

i) Simple if
ii) If else
iii) Else if ladder
iv) Switch.

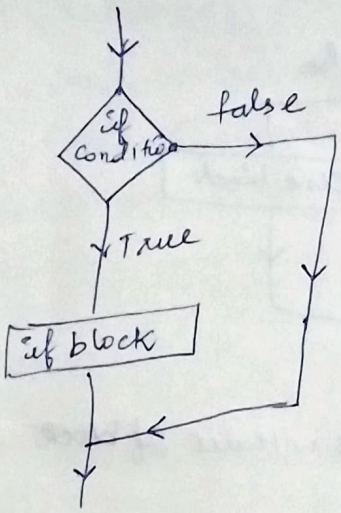
Simple if

* Whenever we have single statement which has to be executed or not executed, we go for simple if statement.

Syntax:

```
if(condition)
{
// stmt
}
```

Flowchart



Eg:

```
class PositiveFormat
{
  public static void main (String[] args)
  {
    int num = 10;
    if (num < 0)
    {
      num = num * (-1);
      System.out.println (" Number is converted ");
    }
    S.o.pln (" Num:: " + num);
  }
}
```

↑

1) WAJPT print given number in positive format

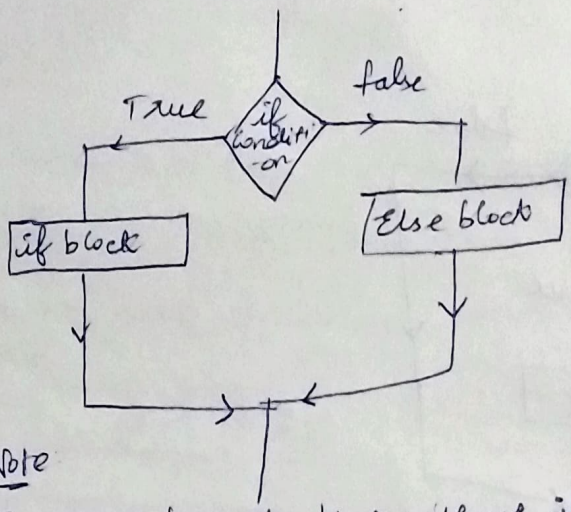
If-else:

* Whenever we have two tasks in which any one has to be performed, we go for if-else statement.

Syntax

```
if(condition)
{
   //stmt1
}
else
{
   //stmt2
}
```

Flow chart



Note

We cannot have else block without if block

2) WAJPT check given number is even number (or) odd number

```
class EvenOrOdd
{
  public static void main(String[] args)
  {
    int num=5;
    int rem = num%2;
    if(rem==0)
    {
      S.o.pln(num+" is Even number");
    }
  else
  {
    S.o.pln(num+" is odd number");
  }
 }
}
```

3) WAJPT to check given number is divisible by 3 and 5 or not.

4) WAJP to check given number is positive number or negative number.

5) WAJP to check user is valid or not. If valid Print "Welcome" else "Invalid user" print.

Valid Username = "Durga"

Valid Password = "Durga143@Durgi"

6) WAJP to check given character is alphabet or not

7) WAJP to check given character is digit or not

8) WAJP to check given year is leap year or not

9) WAJP to check given month is valid or not

bit.ly/CECM29CK

3) class DivisibleBy3And5

```
{
  public static void main(String[] args)
  {
    int num=15;
    if(num%.3==0 && num%.5==0)
    {
      System.out.println(num+" is divisible by 3 and 5");
    }
    else
    {
      System.out.println(num+" is not divisible by 3 and 5");
    }
  }
}
```

4) class PositiveOrNegativeNumber

```
{
  public static void main(String[] args)
  {
    float num=50.5f;
    if(num>0)
    {
      S.o.pln("num+" is a positive number");
    }
```

```
    else
    {
      S.o.pln(num+" is Negative number");
    }
  }
}
```

5) class ValidUser

```
{
  public static void main(String[] args)
  {
    String name="Dingi";
    String password="Dinga143@Dingi";
    if(name=="Dinga"&& password=="Dinga143@Ding
    {
      S.o.pln("Welcome "+name+"!!!!");
    }
    else
    {
      S.o.pln("Invalid User");
    }
  }
}
```

6) class AlphabetOrNot

```
{
  public static void main(String[] args)
  {
    char ch='a';
    if((ch>='A'&& ch<='Z')||(ch>='a'&& ch<='z'))
    {
      S.o.pln(ch+" is a alphabet");
    }
    else
    {
      S.o.pln(ch+" is not a alphabet");
    }
  }
}
```

7) class DigitOrNot

```
{
  public static void main (String[] args)
  {
    char digit = '0';
    if ( digit >= '0' && digit <= '9')
    {
      S.o.pln (digit + " is a digit");
    }
    else
    {
      S.o.pln (digit + " is not a digit");
    }
  }
}
```

8) class LeapYear

```
{
  public static void main (String[] args)
  {
    int year = 2018;
    if ( year % 4 == 0 && (year % 100 != 0 || year % 400 == 0))
    {
      S.o.pln (year + " is a leap year");
    }
    else
    {
      S.o.pln ( year + " is not a leap year");
    }
  }
}
```

9) class ValidMonth

```
{
  public static void main (String[] args)
  {
    String month = "July";
    if ( month == "January" || month == "February" ||
         month == "March" || month == "April" ||
```

```
month == "May" || month == "June" || month == "July" ||
month == "August" || month == "September" || month ==
"October" || month == "November" || month == "December")
{
  S.o.pln("Hurray!!!" + month + " is a Valid month");
}
else
{
  S.o.pln("ohh no!!" + month + " is not a valid month");
}
}
}
```

9) class IsValidMonth

```
{
public static void main(String[] args)
{
  int mm = 4;
  if (mm >= 1 && mm <= 12)
  {
    S.o.pln(mm + " is a valid month");
  }
  else
  {
    S.o.pln(mm + " is invalid month");
  }
}
}
```

09/03/2022

-Else-if ladder

* Whenever we have more than two tasks in which any one has to be performed, we go for else-if ladder
* In-else if ladder, else block is optional

Syntax

```
if (condition1)
{
  Stmt 1
}
else if (condition 2)
{
  stmt 2}
}
else if (condition 3)
{
  stmt 3
}
.
.
.
else
{
  Stmt n
}
```

1) WAJPT find largest of three numbers

```
class largestOfThreeNumbers
{
  public static void main (String[] args)
  {
    int a= 60;
    int b = 80;
    int c = 40;
    if (a>b && a>c)
    {
      System.out.println("Largest of"+a+","+b+"and"+c+"is"
                         +a);
    }
```

```
else if (b>c)
{
  S.o.pln("Largest of "+a+","+b+"&"+c+" is "+b);
}
else
{
  S.o.pln("Largest of "+a+","+b+"&"+c+" is "+c);
}
}
}
```

2) WAJPT check given character is uppercase, lowercase digit or special character.

```
class Checkcharacter
{
  Public static void main(String[] args)
  {
    char ch='5';
    if(ch>='A' && ch<='Z')
    {
      S.o.pln("The given character "+ch+" is UpperCase");
    }
    else if(ch>='a' && ch<='z')
    {
      S.o.pln("The given character"+ch+" is lowercase");
    }
    else if(ch>='0' && ch<='9')
    {
      S.o.pln("The given character "+ch+" is Digit");
    }
    else
    {
      S.o.pln("The given character"+ch+" is Special
                                    character");
    }
  }
}
```

3) WAJP to print Dinga if the number is divisible by 3, print "Dingi" if the number is divisible by 5, Print "Dinga Weds Dingi" if the number is divisible by 5 and 3 else print "Breakup"

```
class DingaDingiLove
{
  public static void main (String[] args)
  {
    int number = 15;
    if (number %5 == 0 && number %3 == 0)
    {
      S.o.pln ("Dinga Weds Dingi");
    }
    else if (number %3 == 0)
    {
      S.o.pln ("Dinga");
    }
    else if (number %5 == 0)
    {
      S.o.pln ("Dingi");
    }
    else
    {
      S.o.pln ("Breakup");
    }
  }
}
```

4) WAJP to validate given date

```
class ValidDate
{
  public static void main (String[] args)
  {
    int dd = 06;
    int mm = 07;
    int yy = 2002;
                    → s.o.pln( dd + "/" + mm + "/" + yy);
    if (dd<1 || dd>31 || mm<1 || mm>12 || yy<1)
    {
      s.o.pln("Invalid date");
    }
    else if (mm==4 || mm==6 || mm==9 || mm==11) && dd>30)
    {
      s.o.pln("Invalid date");
    }
    else if (mm==2 && !(yy%4==0 && yy%100!=0 ||
    {                    yy%400==0) && dd>28)
      s.o.pln("Invalid date");
    }
    else if (mm==2 && (yy%4==0 && yy%100!=0 ||
                       yy%400==0) && dd>29
    {
      s.o.pln("Invalid date");
    }
    else
    {
      s.o.pln("Valid date");
    }
  }
}
```

08/08/2023

Switch

* Switch is a decision statement which works based on equality operator

Syntax

```
Switch (value)
{
  case (value): { // stmt 1 }
  case (value): { // stmt 2 }
  case (value): { // stmt 3 }
  case (value): { // stmt 4 }
  ____
  ____
  ____
  ____
  default : { stmt nth }
}
```

Note

i) In switch, we cannot use long, float, double and Boolean datatypes

ii) In switch, we cannot have duplicate keys labels

iii) In switch, default block is optional

iv) Switch compares switch value with the case values, if any case value is matching that case block will be executed along with the case blocks ~~below~~ it, after it including default block.

Program

```
class Switch
{
  public static void main (String[] args)
  {
    int choice = 3;
    switch (choice)
```

```
{
    case 1: S.o pln(1);
    case 2: S.o pln(2);
    case 3: S.o pln(3);
    case 4: S.o pln(4);
    case 5: S.o pln(5);
    default: S.o pln(6);
}
}
}
```

O/P

```
3
4
5
6
```

Break

* Break is a keyword.
* Break is used to stop the continuous flow of execution (loop).
* Break can be used only inside switch statement or loop statement

Program

```
class S2
{
    public static void main(String[] args)
    {
        int choice = 2;
        Switch (choice)
        {
            case 1: S.o.pln("You have selected English"); break;
            case 2: S.o.pln("Neenga Tamil Select Panirukinga"); break;
            case 3: S.o.pln("Neevu Kannada select madiddira"); break;
            default: S.o.pln("Invalid input");
        }
    }
}
```

1) WAJPT print month names for given month number

2) WAJPT check given number is even or odd using switch

1)
```
class MonthNames
{
public static void main(String[] args)
{
int choice = 3;
switch(ch)
{
case 1:
    System.o.pln("January");
    break;
case 2
    s.o.pln("february");
    break;
case 3
    s.o.pln("March");
    break;
case 4
    s.o.pln("April");
    break;
case 5
    s.o.pln("May");
    break;
case 6.
    s.o.pln("June");
    break;
case 7:
    s.o.pln("August");
    break;
case 8:
    s.o.pln("September");
    break;
case 9:
    s.o.pln("October");
    break;
Case 10:
```

```
        S.o.pln("October");
        break;
    case 11:
        S.o.pln("November");
        break;
    case 12:
        S.o.pln("December");
        break;
    default: S.o.pln("Invalid month"); break;
    }
}
```

2) class EvenOrOddUsingSwitch

```
{
    public static void main(String[] args)
    {
        int no=52;
        int rem=no%2;
        switch(rem)
        {
            case 0: S.o.pln(no+" is even no");
            case 1: S.o.pln(no+" is odd no");
        default: S.o.pln("Error");
        }
}
```

10/08/23

Loop statements

* Whenever we want to perform task repeatedly, we go for loop statements
* In java, we have 4 loop statements. They are
  i) While loop
  ii) Do while loop
  iii) For loop
  iv) For-each loop / Advanced for loop
* Every loop statement must have three segments. They are
  (i) initialization
  (ii) Condition
  (iii) Updation

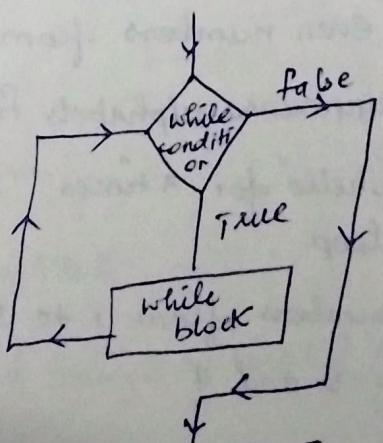
While loop

* While loop is an entry controlled loop in which the condition will be checked first, if the condition is true, the block will be executed until the condition becomes false

Syntax

```
while (condition)
{
   // stmt
}
```

Flow chart



1) WAJPT print numbers from 1 to 5.

```
class Simpleloop
{ public static void main (String[] args)
  {
    int num=1;
```

```
while (num<=5)
{
    sop(num);
    num++;
}
}
}
```

o/p

1 2 3 4 5

2) WAJPT print "Hi" for six times

```
class PrintHi
{
  public static void main(String[] args)
  {
    String
    int n=1;
    while (n<=6)
    {
      sop("Hi");
      n++;
    }
  }
}
```

o/p

Hi
Hi
Hi
Hi
Hi
Hi

3) WAJPT print numbers from 20 to 1.

4) WAJPT print even numbers from 1 to 10

5) WAJPT print uppercase alphabets from A to Z.

6) WAJPT print "hello" for 3 times "bye" for 3 times using single loop.

7) WAJPT print numbers from 1 to 500 which are divisible by both 5 and 7.

3) class PrintNumberFrom20To1

```
{
  public static void main (String[] args)
  {
    int number = 20;
    while (number >=1)
    {
      S.o.pln(number);
      number--;
    }
  }
}
```

O/P

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

4)

4) class EvenNumberFrom1To10

```
{
  public static void main (String[] args)
  {
    int num = 1;
    while (num<=10)
    {
      if (num%2==0)
      {
        S.o.pln(num);
      }
      num++;
    }
  }
}
```

O/P

2
4
6
8
10

5) class UpperCaseFromAToZ

```
{
  public static void main (String[] args)
  {
    char ch = 'A';
    while
    if (ch<='Z')
    {
      S.o.pln(ch);
    } ch++;
}}
```

b) class Hello3Times Bye3Times

```
{
  public static void main (String [] args)
  {
    int no=1;
    while (no<=6)
    {
      if (no<=3)
      {
        S.o.pln("Hello");
      }
      else
      {
        S.o.pln("Bye");
      }
      no++;
    }
  }
}
```

7) class DivisibleBy5And7

```
{
  public static void main (String [] args)
  {
    int num=1;
    while (num<=500)
    {
      if (num%.5==0 && num%.7==0)
      {
        S.o.pln(num);
      }
      num++;
    }
  }
}
```

11/08/23

1) WAJPT print even numbers from 1 to 10

```
class EvenNumbers1To10
{
  public static void main (String[] args)
  {
    int num=1;
    int while (num<=10)
      {
        if (num%2==0)
        {
          S.o.pln (num);
        }
      num++;
    }
  }
}
```

o/p

2
4
6
8
9
10

2) WAJPT find the sum of even numbers from 1 to 10

```
class SumOfEvenNumbers
{
  public static void main (String[] args)
  {
    int num=1;
    int sum=0;
    while (num<=10)
    {
      if (num%2==0)
      {
        sum=sum+num;
      }
    num++;
    }
  S.o.pln (sum);
}
}
```

o/p

30.

3) WAJPT find the product of Even numbers from 1 to 10

```
class ProductofEvenNumbers
{
  public static void main (String[] args)
  {
    int num=1;
    int prod=1;
    while (num<=10)
    {
      if (num%2==0)
      {
        prod=prod*num
      }
    num++;
  }
  S.o.pln("The Product of even numbers is :"+prod);
  }
}
```

4) WAJPT find the count of even numbers from 1 to 10

```
class CountEvenNumbers
{
  public static void main (String[] args
  {
    int num=1;
    int count=0,
    while (num<=10)
    {
      if (num%2==0)
      {
        count++;
      }
      num++;
    }
    S.o.pln("The Count is:"+count);
  }
}
```

O/P

5.

5) WAJPT find Product of numbers from 1 to 5.

```
class Product
{
  public static void main (String[] args)
  {
    int num = 1; int prod = 1;
    while (num <= 5)
    {
      prod = prod * num; } (or) prod *= num++;
      num++;
    }
    System.out.println ("The product is :: " + prod);
  }
}
```

o/p
120.

6) WAJPT to count the numbers which are divisible by 5 and 7 between 1 to 100

7) WAJPT find x^n value.

8) WAJPT count number of digits in the given number

9) WAJPT count the factors of given number.

```
6) class DivisibleCount
{
  public static void main (String[] args)
  {
    int num = 1;
    int count = 0;
    while (num <= 100)
    {
      if (num % 5 == 0 && num % 7 == 0)
      {
        count++;
      }
      num++;
    }
    System.out.println (count);
  }
}
```

6) class XPowerNValue

```
{
  public static void main(String[] args)
  {
    int x = 45;
    int n = 38;
    int prod = 1;
    int temp = n;
    while (n > 0)
    {
      Prod* = x;
      n--;
    }
  S.o.pln((x + "^" + temp + " is " + prod);
}
}
```

7) class CountDigits

```
{
  public static void main (String[] args)
  {
    int num = 4576;
    int count = 0;
    int temp = num;
    while (num > 0)
    {
      num = num/10;
      num++;
    }
  S.o.pln("Digits in the" + num + "is" + Count);
}
}
```

8). class FactorsOfNumber

```
{
  public static void main (String[] args)
  {
    int num=9;
    int i=1;
    int count=0;
    while (i<=num)
    {
      if (num%i==0)
      {
        count++;
      }
      i++;
    }
    System.out.println ("factors of "+num+" is "+count);
  }
}
```

12/8/23

1) Prime number

* Prime number is a number that is greater than 1 and divided by 1 or itself only.

* In other hand, prime numbers can't be divided by other num than itself or 1.

Eg: 2, 3, 5, 7, 11, ...

Note 0 and 1 are not prime number The 2 is the only even prime number

2) Factorial

* Factorial of n is the product of all +ve descending integers Factorial of n is denoted by n!

Eg $4! = 4 \times 3 \times 2 \times 1 = 24$

$2! = 2 \times 1 = 2$.

3) Fibonacci series

* Fibonacci series next number is the sum of previous 2 num.

* The first 2 num of fibonacci series 0 & 1.

4) Palindrome

* Palindrome num is also known as numeral num. Palindrome num is num that remains the same when its digits are reversed.

Eg 123 = 321, mom = mom.

5) Spy number

If the sum of all digits are equal to the product of all digits is known as spy number.

Eg: $1+4+2=8$, $1*4*2=8$ //

6) Armstrong number

* Armstrong num is a num that is equal to the sum of its cube of its digit

Eg 0, 1, 153, 370 $153 = 1^3+5^3+3^3 \Rightarrow 1+125+27 \Rightarrow 153$

7) Happy num

* Happy number is a num when reaches 1 when the digits replaced by the square of each digits

Eg $7 = 7^2 \Rightarrow 49 \Rightarrow 4^2+9^2 \rightarrow 16+81 \Rightarrow 97 \rightarrow 81+49$

$\Rightarrow 130 \rightarrow 1^2+9^2 \rightarrow 1+81 \Rightarrow 82 \Rightarrow 8^2+4$

$\Rightarrow 68 \Rightarrow 36+64 \Rightarrow 100 \rightarrow 1.$

8) Perfect number

* In number theory, a perfect number is the +ve integer that is equal to the sum of +ve divisors excluding the num itself

9) Neon number

A +ve integer whose sum of its digits of square is equal to the num itself is called neon number

Ex: $9 \Rightarrow 9^2 \Rightarrow 81 \rightarrow 8+1 = 9$

10) Strong number

A strong number is a number whose sum of factorial of digits is equal to the original number.

Eg $145 = 1! + 4! + 5! = 1 + 24 + 120$

$= 145$

1) WAJPT check given number is Prime number or not

```
class PrimeNumber
{
  public static void main (String[] args)
  {
    int num=23;
    int i=1;
    int count=0;
    while (i<=num)
    {
      if (num%i==0)
      {
        count++;
      }
      i++;
    }
  if (count==2)
  {
    So.pln (num+" is a prime number");
  }
  else
  {
    So.pln (num+" is not a prime number");
  }
 }
}
```

O/P

23 is a prime number

2) WAJPT find factorial of given number

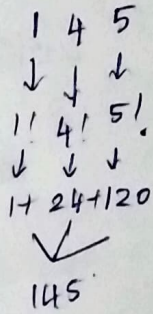
```
class Factorial
{
  public static void main (String[] args)
  {
    int num=5;
    int fact=1;
    int temp=num;
```

```
while (num>0)
{
    fact *= num--;
}
S.o.pln("The factorial of "+temp+" is "+fact);
}
}
```

op

The factorial of 5 is 120

3] WAJPT to check whether the number is strong number or not




```
class StrongNumber
{
  public static void main(String[] args)
  {
    int num=145;
    int sum=0;
    int temp=num;
    while(num>0)
    {
      int last=num/10;  → Take last digit
      int fact=1;
      while(last>0)  → find factorial of last digit
      {
        fact *= last--;
      }
      sum=sum+fact;  → Add factorial of last digit to sum
      num=num/10;  → Remove the last digit
    }
    if(temp==sum)
      S.o.pln(temp+" is strong number");
    else
      S.o.pln(temp+" is not a strong number");
  }
}
```

op

145 is strong number

4) WAJP1 to check the number is spy number or not

```
class SpyNumber
{
  public static void main (String[] args)
  {
    int num = 123;
    int sum = 0;
    int prod = 1; int temp = num;
    while (num > 0)
    {
      int last = num % 10,   - Take last digit
      sum = last;            - Add to sum
      prod *= last;          - Add to product
      num = num / 10,        - Remove last digit
    }
  if (sum == prod)
    S.o.pln (temp + " is a spy number");
  else
    S.o.pln (temp + " is not a spy number");
  }
}
```

o/p

123 is a spy number.

14/8/25

1) WAJPT to check whether the number is palindrome or not

```
class Palindrome
{
  public static void main (String[] args)
  {
    int num=121, rev=0, temp=num;
    while (num>0)
    {
      // Take last digit
      int last = num%10;
      // Add last digit into rev
      rev=(rev*10)+last;
      // Remove last digit
      num=num/10;
    }
  if (temp==rev)
    System.out.println(temp+" is a palindrome");
  else
    System.out.println(temp+" is not a palindrome");
  }
}
```

o/p
123 is not a palindrome.

2) WAJPT to check whether the number is Armstrong number or not.

```
class ArmstrongNumber
{
  public static void main (String[] args)
  {
    int num=153, temp=num, temp1=num, sum=0;
      count=0;
```

```
while (num > 0) // count the number of digits
{
    num = num/10;
    Count++;
} // finding Armstrong number
while (temp > 0)
{ // Take last digit
  int last = temp % 10;
  int x = last, n = count, prod = 1;
  while (n > 0)
  {
      prod = prod * x;
      n--;
  }
  sum = sum + prod; // Add to sum
  temp = temp/10; // Remove the last digit
}
if (temp1 == sum)
    S.o.println(temp1 + " is a Armstrong number");
else
    S.o.println(temp + " is not a Armstrong number");
}
}
```

- // find x^n (last count) value

3) WAJPT to check a number is neon number or not.

$9^2 = 81 = 8+1 = 9$

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

```
int num = 9, sum = 0;
int sq = num * num; // Square the number
while (sq > 0)
{
    int last = sq % 10; // Take last digit
    sum = sum + last; // Add to sum
    sq = sq / 10;     // Remove last digit
}
if (num == sum)
    System.out.println(num + " is a Neon number");
else
    System.out.println(num + " is not a Neon number");
}
}
```

16/8/23

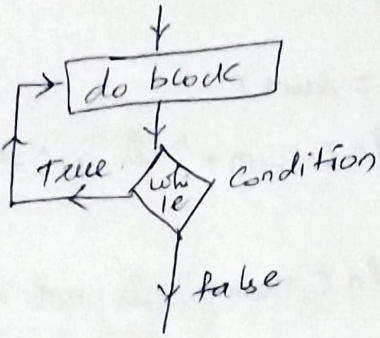
Do-while loop

* Do-while loop is an exit-controlled loop in which condition will be checked after do block execution

Syntax

```
do
{
   // stmt
}
while (condition);
```

Flowchart



Note

* In do-while loop, even false condition will execute minimum one time.

1] WAJPT print the numbers from 1 to 5

```
class P1
{
   public static void main (String[] args)
   {
      int num=1;
      do
      {
         S.o.pln (num);
         num++;
      }
      while (num<=5)
   }
}
```

o/p

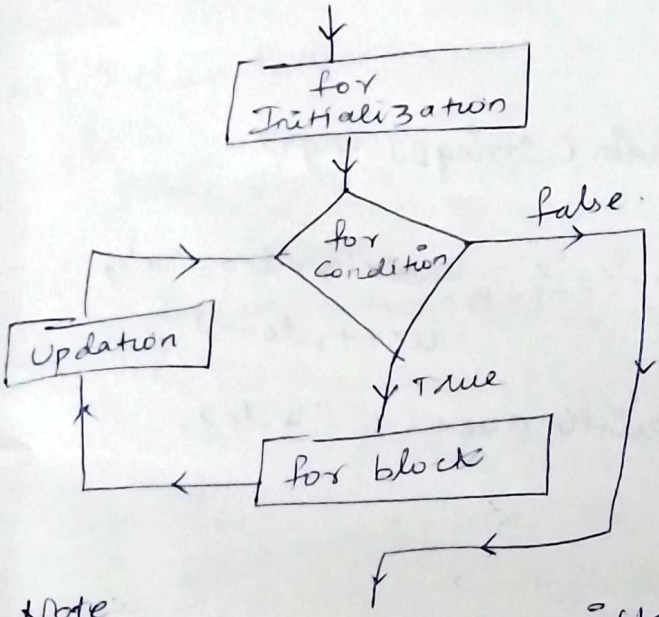
1
2
3
4
5

for-loop

Syntax

```
for (initialization , condition , updation)
{
    // stmt
}
```

flow chart



Note

* If we declare a variable in for loop initialization, its visibility within for ~~loop~~ block
* The default condition of for loop is true

Example

1) WAJPT print numbers from 1 to 5

```
class Forloop
{
    public static void main(String[] args)
    {
        for(int num=1; num<=5; num++)
        {
            S.o.pln(num);
        }
    }
}
```

O/P

```
1
2
3
4
5
```

2) WAJPT print upper case alphabets from A to Z

3) WAJPT to print numbers from 1 to 500 which are divisible by 5 and 7

4) WAJPT print following patterns

① A : Z
B : Y
Z : a

② 

③ 

④ 

4) i) class Pattern1
{
public static void main (String[] args)
{
for (char uc='A', lc='z'; uc<='z' && lc>='a';
uc++, lc--)
{
System.out.println(uc+" : "+lc);
}
}
}

2) class UpperCase
{
public static void main (String[] args)
{
for (char ch='A'; ch<='z'; ch++)
{
S.o.println(ch);
}
}
}

3) class DivisibleBy5And7
{
public static void main (String[] args)
{
for (int i=1; i<=500; i++)

```
{
    if (i%5==0 && i%7==0)
    {
        s.o.pln(i);
    }
}
}
```

4) iv) class Pattern4

```
{
  public static void main(String[] args)
  {    int n=4;
    for(int i=0; i<n; i++)
    {
      for(int j=0; j<n; j++)
      {
        s.o.print("*  ");
      }
      s.o.pln();
    }
  }
```

4. ii) class Pattern2

```
{
  public static void main(String[] args)
  {
    for(int i=0; i<4; i++)
    {
      s.o.pln("*");
    }
  }
}
```

4.iii) class Pattern3

```
{
  public static void main (String[] args)
  {
    for (int i=0; i<4; i++)
    {
      System.out.print (" * ");
    }
  }
}
```

17/8/23

Pattern:

Tips for pattern printing:

Step 1: Observe the shape of the pattern

Step 2: Identify the characters to be printed.

Step 3: Observe the sequence of pattern

i) If each row starting with same character, declare the variable inside outer loop.

ii) If each row starting with different character declare the variable outside outer loop.

Step 4: Observe the updation.

Program

1)
```
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
```

```
class P1
{
  public static void main (String[] args)
  {
    int n=4;
    for (int i=0; i<n; i++)
    {
```

```
        int K=1;
        for(int j=0;j<n;j++)
        {
          S.o.print(K+" ");
        }
      S.o.println();
    }
  }
}
```

2)

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

- Outside outer loop

```
class P2
{
  public static void main(String[] args)
  {
    int n=4, K=1;
    for(int i=0;i<n;i++)
    {
      S.o.print
      for(int j=0;j<n;j++)
      {
        S.o.print(K<10? K++ + "  " : K++ + " ");
      }
      S.o.println();
    }
  }
}
```

3)

1	2	3	4
1	2	3	4
1	2	3	4
1	2	3	4

- Inside outer loop

```
class P3
{
  public static void main (String[] args)
  {
    int n=4;
    for (int i=0; i<n; i++)
    {
      int K=1;
      for (int j=0; j<n; j++)
      {
        S.o.print (K++ + " ");
      }
      S.o.println();
    }
  }
}
```

4)
```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4.
```

```
class P4
{
  public static void main (String[] args)
  {
    int n=4, K=1;
    for (int i=0; i<n; i++)
    {
      for (int j=0; j<n; j++)
      {
        S.o.print (K + " ");
      }
    S.o.println();
    K++;
    }
}
```

5) A A A A
B B B B
C C C C
D D D D

```
class P5
{
  public static void main (String [] args)
  {
    int n=4;
    char ch='A';
    for(int i=0; i<n; i++)
    {
      for(int j=0; j<n; j++)
      {
        System.out.print(ch+" ");
      }
      System.out.println();
      ch++;
    }
  }
}
```

6) A B C D
A B C D
A B C D
A B C D

7) 4 3 2 1
4 3 2 1
4 3 2 1
4 3 2 1

8) 1 A 1 A
2 B 2 B
3 C 3 C
4 D 4 D

9) 1 2 3 4
A B C D
1 2 3 4
A B C D

10) 1 2 3 4
A A A A
1 2 3 4
B B B B

11) D C B A
D C B A
D C B A
D C B A

6) class P6

```
{
  public static void main(String[] args)
  {
    int n=4;
    for(int i=0; i<n; i++)
    {
      char ch='A';
      for(int j=0; j<n; j++)
      {
        S.o.println(ch+" ");
        ch++;
      }
      S.o.println();
    }
  }
}
```

7) class P7

```
{
  public static void main(String[] args)
  {
    int n=4;
    for(int i=0; i<n; i++)
    {
      int k=n;
      for(int j=0; j<n; j++)
      {
        S.o.print(k-- +" ");
      }
      S.o.println();
    }
  }
}
```

8) class P8

```
{
  public static void main(String[] args)
  {
    int n=4, k=1;
    char ch='A';
```

```
for(int i=0; i<n; i++)
{
  for(int j=0; j<n; j++)
  {
    S.o.print(j%2==0 ? k++ + " " : ch++ + " ");
  }
  S.o.println();
  k++;
  ch++;
}
}
}
```

9)
```
class P9
{
  public static void main(String[] args)
  {
    int n=4;
    for(int i=0; i<n; i++)
    {
      int k=1;
      char ch='A';
      for(int j=0; j<n; j++)
      {
        S.o.print(i%2==0 ? k++ + " " : ch++ + " ");
      }
      S.o.println();
    }
  }
}
```

10)
```
class P10
{
  public static void main(String[] args)
  {
    int n=4;
    for(int i=0; i<n; i++)
    {
      int k=1;
      char ch='A';
      for(int j=0; j<n; j++)
      {
```

```
        S.o.print(i%2==0 ? k++ + " " : ch + " ");
    }
    S.o.println();
    if(i%2!=0)
    {
        ch++;
    }
  }
}
}
```

11) class P11

```
{
    Public static void main(String[] args)
    {
        int n=4;
        for(int i=0; i<n; i++)
        {
            char ch=(char)(n+64);
            for(int j=0; j<n; j++)
            {
                S.o.print(ch-- + " ");
            }
            S.o.println();
        }
    }
}
```

22/8/22

shape of the pattern

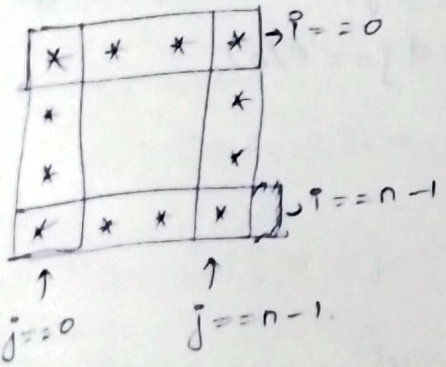
- i (or) j common
- Compare i with j
- i+j with n-1

Coordinates

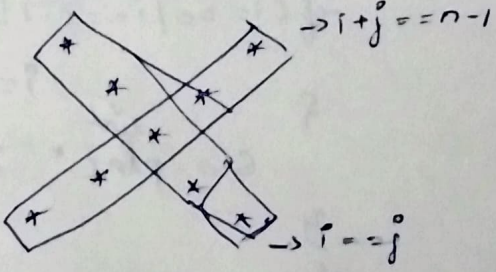
n = 4

(i,j)	j=0	j=1	j=2	j=n-1
i=0	0,0	0,1	0,2	0,3
i=1	1,0	1,1	1,2	1,3
i=2	2,0	2,1	2,2	2,3
i=n-1	3,0	3,1	3,2	3,3

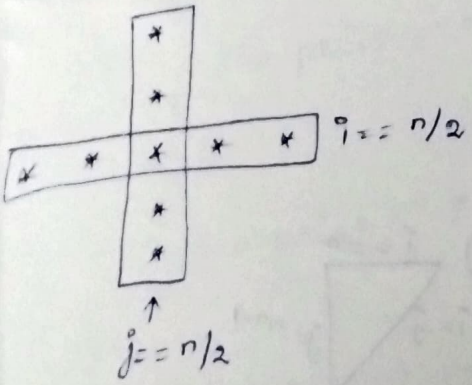
shape-01



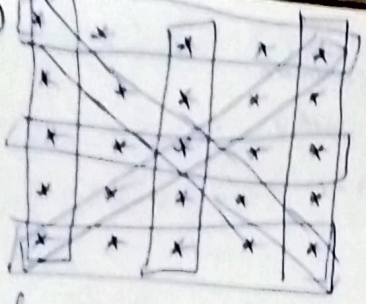
shape-02 - Diagnals



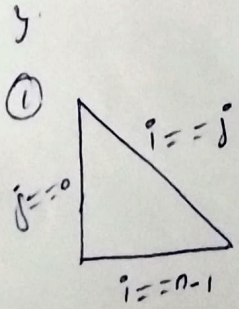
shape-03

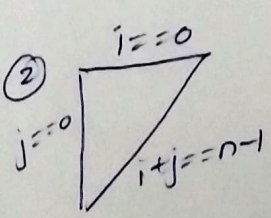


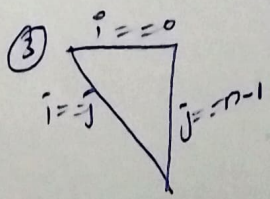
1)

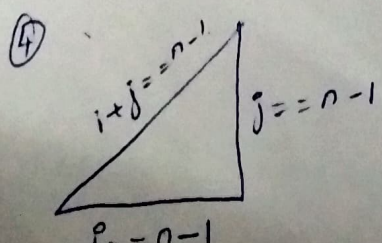


```
class S1
{
  public static void main(String[] args)
  {
    int n=5;
    for(int i=0;i<n;i++)
    {
      for(int j=0;j<n;j++)
      {
        if(i==0||i==n-1||j==0||j==n-1||i==j||i+j==n-1||
                                    i==n/2||j==n/2)
        {
          System.out.println("*");
        }
        else
        {
          System.out.print(" ");
        }
      }
      System.out.println();
    }
  }
}
```

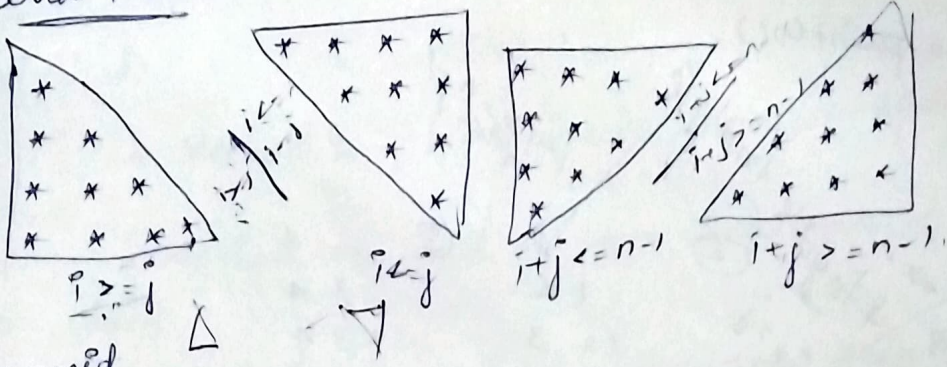






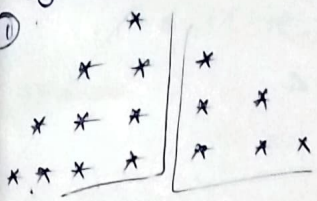


Solid Patterns



Pyramid

(1)



```
class ss1
{
 public static void main (String[] args)
  {
   int n=8;
   for( int i=0; i<n; i++)
   {
    for( int j=0 ; j<n; j++
    {
     if(i+j>=n-1)
      {
        s.o.print("* ");
      }
     else
     {
       s.o.print(" ");
     }
   }
for(int j=0; j<n; j++)
{ if(i>j)
  {  s.o.print(" * ");
  }
else
 {
 } s.o.print(" ");
```

}

S.o.println();

}

}

Floyd's triangle

① 1
2 3
4 5 6
7 8 9 10

② 1
1 2
1 2 3
1 2 3 4

③ 4
3 4
2 3 4
1 2 3 4

④ 1 2 3 4 4 3 2 1
1 2 3 3 2 1
1 2 2 1
1 1

⑤ A
B B
C
D D
E E

⑥ 1
1 2 1
1 2 3 2 1
1 2 3 4 3 2 1

⑦ 1 2 3 4 3 2 1
2 3 4 3 2
3 4 3
4

⑧ A
B
A B C D E
D
E

⑨ 1
1 2
1 2 3
1 2 3 4
1 2 3
1 2
1

⑩ 1
A B
1 2 3
A B C D
1 2 3
A B
1

⑪ C J D A 1

int

for

i<=j
i=j
i>=j

23/08/23

Methods

* Method is a block of code to perform specific task.
* Methods are used to store behaviour of an object
* Methods are used to achieve code reusability
* Methods are executed only when they are called
* We can call the methods using method signature

Note

* Method can be declared anywhere in the class block but not in local area

Syntax

Method declaration

```
[modifier] Returntype methodname ( [args] )
                                 method signature
{
    // task
}
```

method body (or) method implementation

Example

```
class M1
{
    public static void demo ( String[] args)
    {
        s.o.pln("demo begin");
        s.o.pln("demo end");
    }
    public static void main (String[] args)
    {
        s.o.pln("Main begin");
        demo();
        s.o.pln("Main end");
    }
}
```

o/p:

Main begin
Demo begin
Demo end
Main end.

Note

* Based on arguments, methods are classified into 2 types
They are
  i) No argument method
  ii) Parameterized method
* Whenever we don't want to pass input to the method, we go for no argument method
* Whenever we want to pass input to the method, we go for parameterized method
* Whenever we are calling parameterized method,
  i) length of actual and formal arguments must be same
  ii) Datatypes must be same
  iii) Order of declaration must be same.

```
class M2
{
  public static void main (String[] args)
  {
    print();
    add (10, 20); //30
    add (50, 30); //80
  }
  public static void print()  //No Argument method
  {
    s.o.pln ("Hello Sheela");
  }
  public static void add (int a, int b) // Parameterized method
  {
    s.o.pln (a+b);
  }
}
```

O/P:
Hello Sheela
30
80

1) Design a method to check given number is even or odd no

2) WAJPT print prime numbers between given range using methods

3) WAJPT check given number is prime or not using methods

4)

```
1) class EvenOrOdd
{
  public static void evenOrOdd (int num)
  {
    if (num %. 2 ==0)
    {
      s.o.pln (num+" is an even num");
    }
    else
    {
      s.o.pln (num+" is an odd num");
    }
}
public static void main (String[] args)
{
  evenOrOdd(8);
}
}.
3) class IsPrime
{
  public static void main(String[] args)
  {
    isprime(23);
  }
  public static void isPrime (int num){
    int count=0, a=1;
    while (a<=num)
    {
      if (num % a==0)
      {
        count++;
```

```
    }
    a++;
  }
  if(count==2)
  {
    System.out.println(num+" is prime number");
  }
  else
  {
    System.out.println(num+" is not prime number");
  }
 }
}
```

2)
```
class A2
{
  public static void main(String[] args)
  {
    primeGivnR(50);
  }
  public static void primeGivnR(int num)
  {
    for(int i=1;i<=num;i++)
    {
      int count=0;
      for(int j=1;j<=i;j++)
      {
        if(i%j==0)
        {
          count++;
        }
      }
      if(count==2)
      {
        System.out.println(i+" ");
      }
    }
  }
}
```

24/8/23

JRE Memory Areas

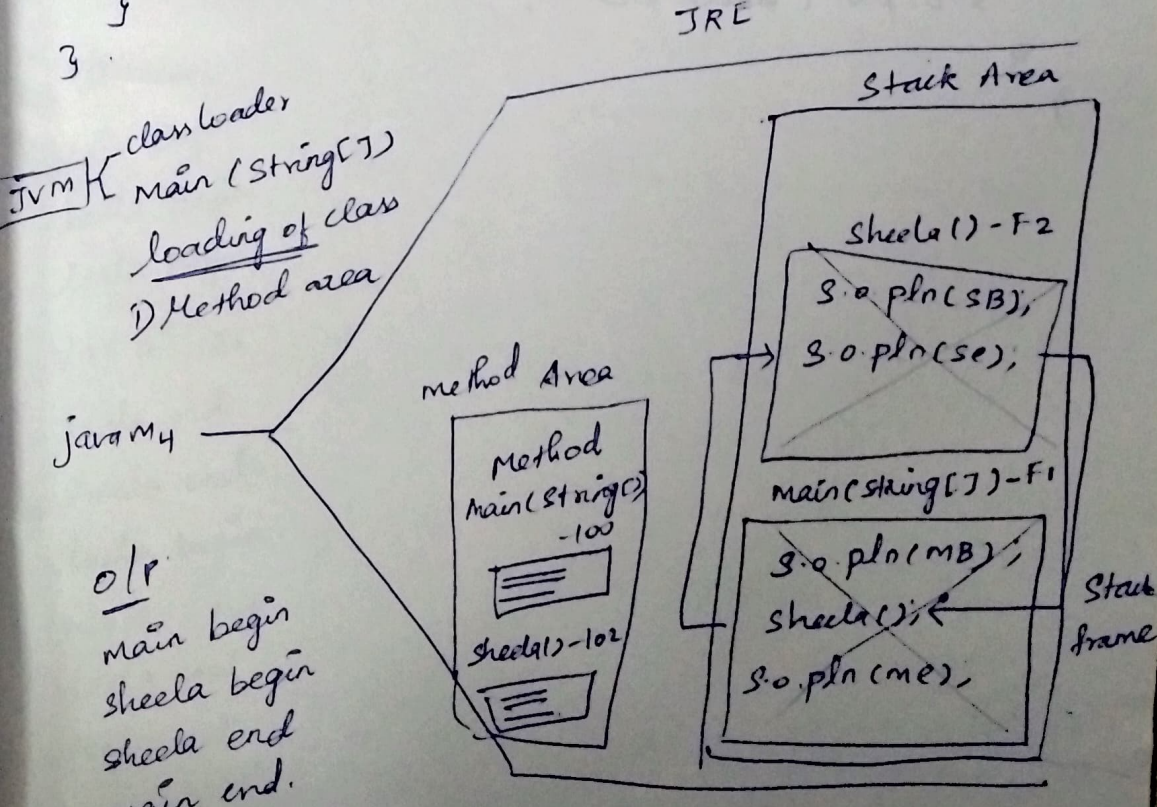
i) Method Area - To store all the methods of the class

ii) class static Area - To store static members

iii) Heap Area - To store non-static members

iv) Stack Area - To execute programs

```
class M4
{
    public static void main(String[] args)
    {
        S.o.pln("Main begin");
        sheela();
        S.o.pln("Main end");
    }
    public static void sheela()
    {
        S.o.pln("sheela begin");
        S.o.pln("sheela end");
    }
}
```

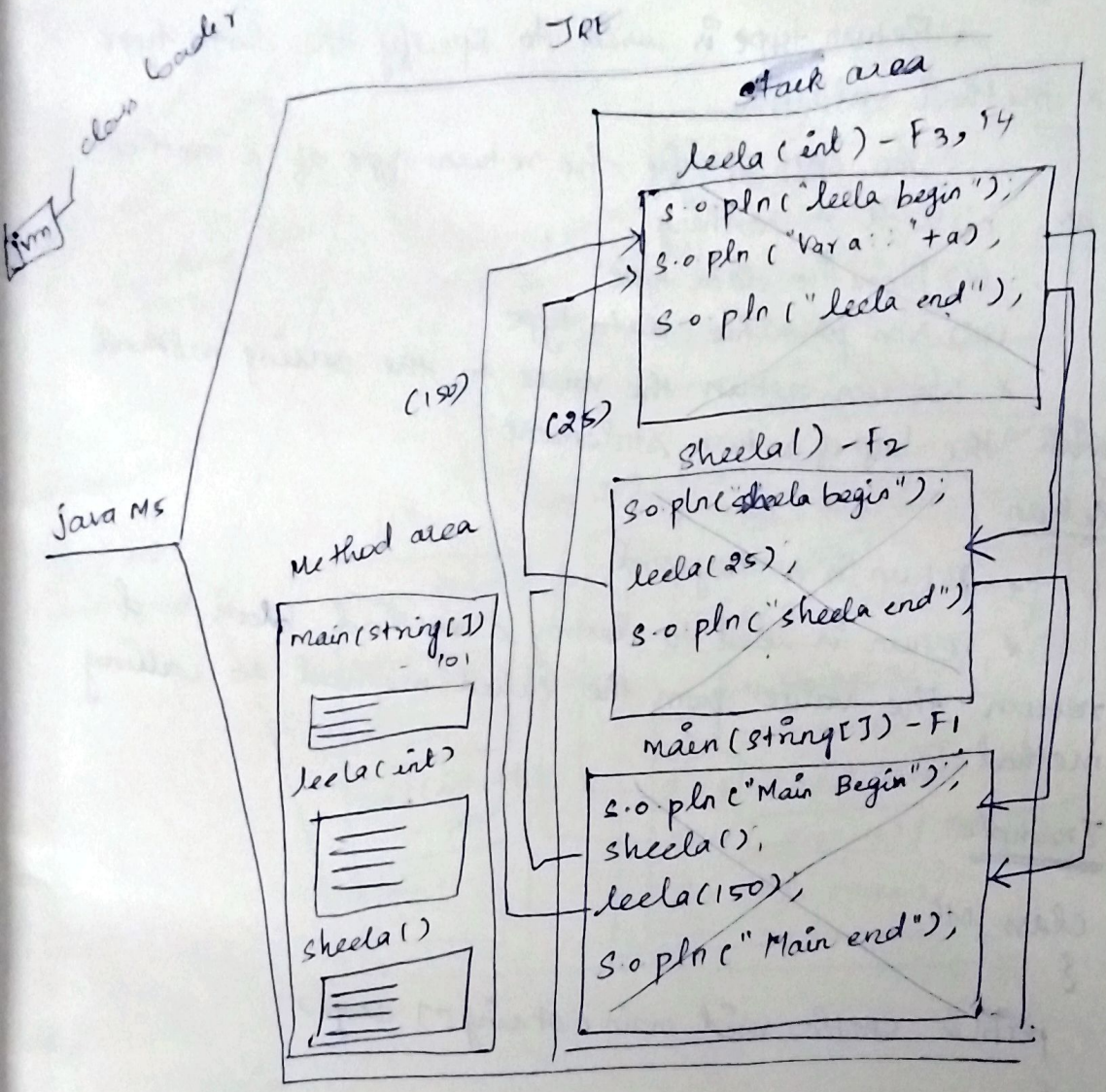


o/p

main begin
sheela begin
sheela end
main end.

2) class M5

```
{
public static void leela(int a)
{
  S.o.pln("leela begin");
  S.o pln("Var a:: "+a);
  S.o pln("leela end");
}
Public static void main(String[] args)
{
  S.o. pln("Main Begin");
  Sheela();
  leela(150);
  S.o pln("Main end");
}
Public static void sheela()
{
  S.o. pln("sheela Begin");
  leela(25);
  S.o pln("sheela end");
}
}
```



O/P

Main Begin
sheela Begin
leela Begin
Var a:: 25
leela end
sheela end
leela begin
var a:: 150
leela end
Main end

Return type

* Return type is used to specify the data type a method returns.
* You can specify the return type of a method as (i) void - Nothing
  (ii) Primitive data type
  (iii) Non-primitive data type
* We can return the value to the calling method with the help of return statement

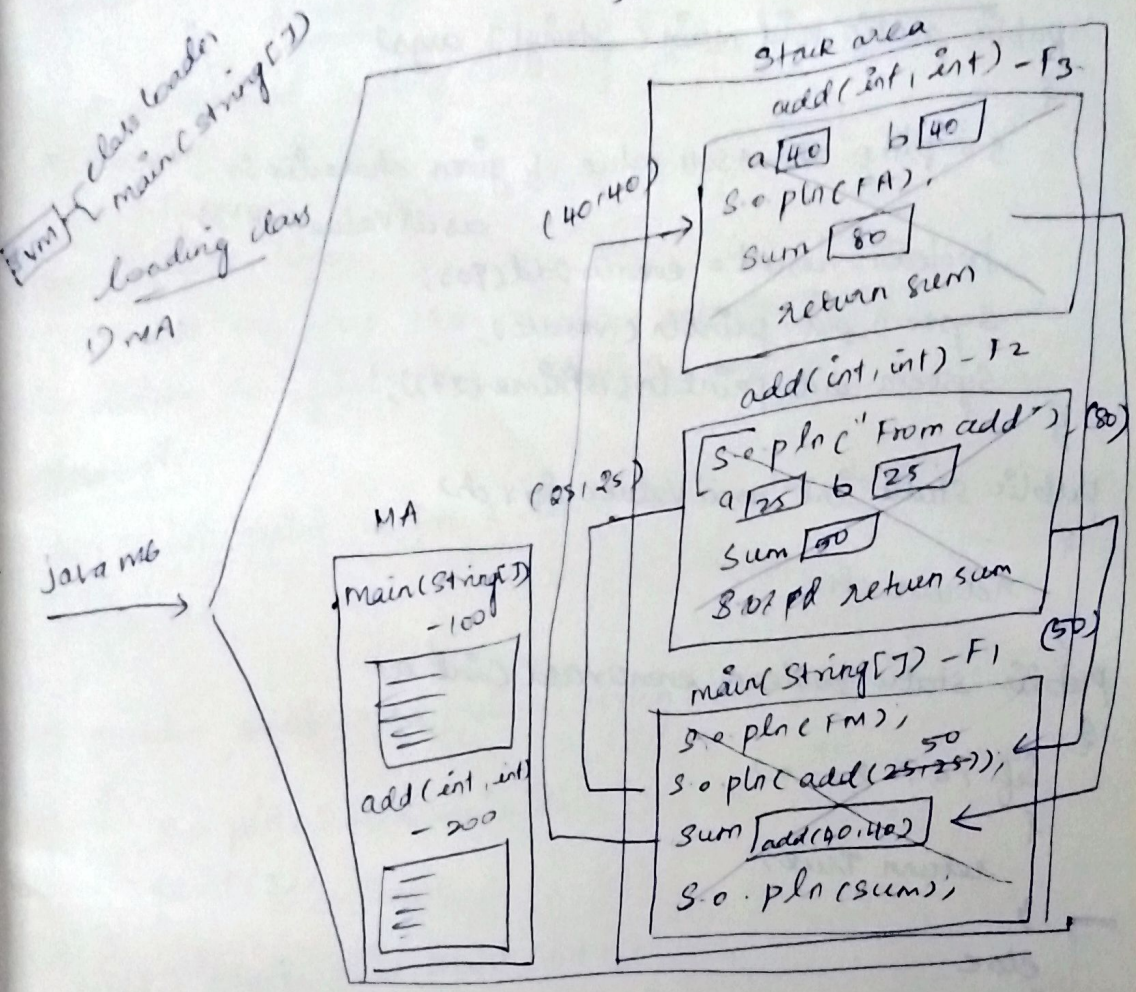
Return

* return is a keyword
* return is used to destroy a method block and return the value from the called method to calling method

Example

```
class M6
{
  public static void main(String[] args)
  {
    S.o.pln("From main");
    S.o.pln(add(25,25));
    int sum = add(40,40);
    S.o.pln(sum);
  }
  public Static int add(int a, int b)
  {
      S.o.pln("from add");
      int sum=a+b);
      return sum;
  }
}
```

JRE



o/p

From main

From add

50

From add

80.

1) WAJP to design a method to return ASCII value of given character

2) WAJP to design a method which returns true if the given no is even number else false.

3) WAJP to return true if the number is prime else false.

```
class ReturnType
{
  public static void main( String[] args)
  {
    s.o.pln(" The ASCII value of given character is : "+
                                    asciiValue('H'));
    boolean result = evenOrOdd(90);
    System.out.println(result);
    System.out.println(isPrime(27));
  }
public static int asciiValue(char ch)
  {
    return ch;
  }
Public static boolean evenOrOdd(int a)
{
   if (a%2==0)
   {
     return True;
   }
   else
   {
     return false;
   }
}
public static boolean isPrime(int num)
{
  int count=0;
  for(int i=0; i<=num; i++)
  {
    if(num%i==0)
    {
      count++;
    }
  }
  if(count==2)
     return true;
  else
     return false;
  }
}
```

25/8/23

Recursion

* A method calling himself is known as recursion
* Whenever recursion occurs, program will run for infinite times.
* We can stop the recursion by designing base condition as first instruction in method

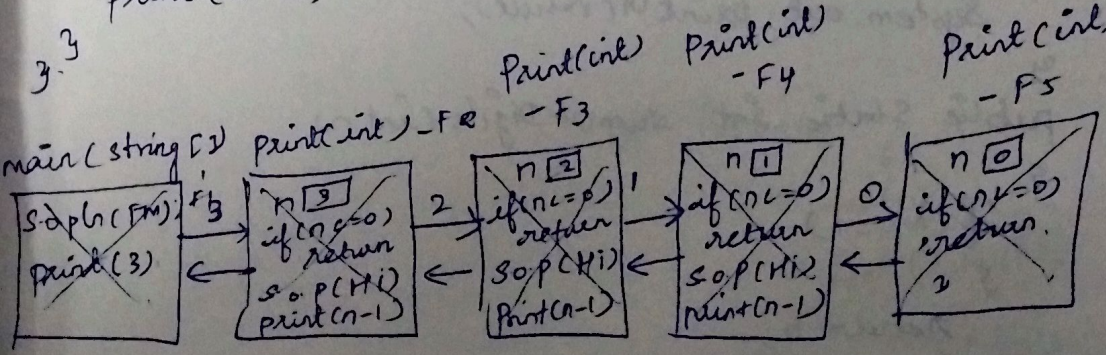
Example

WAJP to print hi for 3 times using recursion

```
class M8
{
public static void main (String [] args)
{
  S.o.pln("from main");
  print(3);
}
public static void print (int n)
{
  // Base Condition          3<=0
  if (n<=0)   (3<=0)
  {
    return;
  }
// task
  S.o.pln("hi");
// recursive call stmt
  print(n-1);
}
}
```

o/p

From main
Hi
Hi
Hi



1) WAJPT print numbers from 1 to 5.

```
class M9
{
 public static void main (String[] args)
 {
   S.o.pln("From main");
   print(1);
 }
 public static void print(int n)
 {
   if(n>5)
   {
     return;
   }
   S.o.pln(n);
   print(n+1);
 }
}
```

2) WAJP to find sum of given number and its below digits using recursion.

3) WAJP to find factorial of given number using recursion.

4) WAJP to find n^{th} fibonacci no using recursion.

2) class SumOfBelowDigitsUsingRecursion

```
{
 public static void main (String[] args)
 {
   int result = sumofDigits(7);
   System.out.println(result);
 }
 public static int sumofDigits(int n)
 {
  if(n==1)
  {
    return n;
  }
  int sum = n + sumofDigits(n-1);
  return sum;
 }
}
```

3) class FactorialUsingRecursion

```
{
  public static void main(String[] args)
  {
    int result = factorial(5);
    S.o.pln(result);
  }
  public static int factorial(int n)
  {
    if(n==1)
      return n;
    int fact = n * factorial(n-1);
    return fact;
  }
}
```

5 * fact(4)
4 * fact(3)

4) class Fibonacci

```
{
  public static void main(String[] args)
  {
    int result = fib(8);
    S.o.pln(result);
  }
  public static int fib(int n)
  {
    if(n==0)
      return 0;
    else if(n==1||n==2)
      return 1;
    else
      return (fib(n-1)+fib(n-2));
  }
}
```

0
1<=1

```
if(n<=1)
{
  return n;
}
return fib(n-1)+fib(n-2);
```

5 * fact(4)
24
4 * fact(3)
6
3 * fact(2)
2
2 * fact(1)
1

28/08/23

1) WAJPT print numbers from 1 to 500 which are Palindrome and sum of digits of number must be less than or equal to 5

```
class PalindromeSumOfDigits
{
  public static void main (String[] args)
  {
    for(int i=0; i<=500; i++)
    {
      if ( isPalindrome(i) && sumOfDigits(i))
      {
        S.o.pln(i);
      }
    }
  }
  public static boolean isPalindrome (int num)
  {
    int rev=0, temp = num;
    while (num>0)
    {
      rev= (rev*10) + num%10;
      num/=10;
    }
    return temp==num;
  }
  public static boolean sumOfDigits (int num)
  {
    int sum=0;
    while (num>0)
    {
      sum = sum+ num%10;
      num/=10;
    }
    return sum<=5;
  }
}
```

Method Overloading

* Designing multiple methods with same name but different arguments is known as method overloading.
* The advantage of method overloading is we can perform a task in multiple ways
* In method overloading, Compiler will decide the implementation of the method to be executed based on arguments

Note

* In method overloading, the compiler will decide method implementation in following order
  i) Similar datatypes
  ii) Widening

Example

```
class MethodOverloading
{
  public static void main(String[] args)
  {
    print(10); //int
    print((double)10); //double
    print((char)10); //char
    print((long)10); //long
  }
  public static void print(double a)
    S.o.pln("print(double)");
  Public static void print(long a)
    S.o.pln("print(long)");
  public static void print(char a)
    S.o.pln("print(char)");
  public static void print(int a);
    S.o.pln("print(int)");
```