

# Chapter 1: Structure of JAVA

All the instructions are always written inside the class in java is called as source file.

class is a keyword

Extension given to java file is .java

File name should be same as that of the class name

Eg

```
class Book  
{  
  
}
```

Contents to be written or created in the java file

Class

Interface

Enum

Class

- It is a keyword
- Keywords: keywords are predefined words which a java compiler can understand
- Keywords should always be written in lowercase

Used to create a class block

## Syntax

```
class ClassName  
{  
  
}
```

className should always be written in lowercase

class block: Anything written inside the opening and closing parenthesis is called as class block.

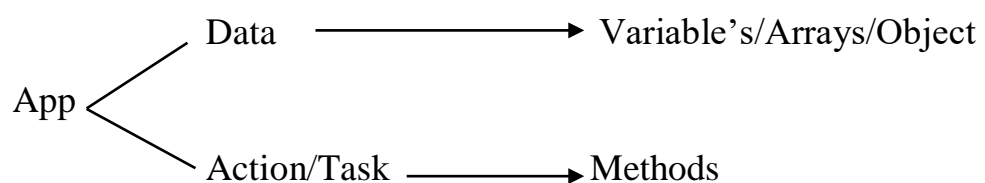
Things we can write in class block

- |                           |                         |
|---------------------------|-------------------------|
|                           | variables               |
| 1) Declaration statements | methods                 |
|                           | class/enums             |
|                           |                         |
|                           | static initializers     |
| 2) Initializer's          | Non static initializers |
|                           | Constructors            |

Create=>Declare

Statements used to declare statements in a class one called Declaration Statements

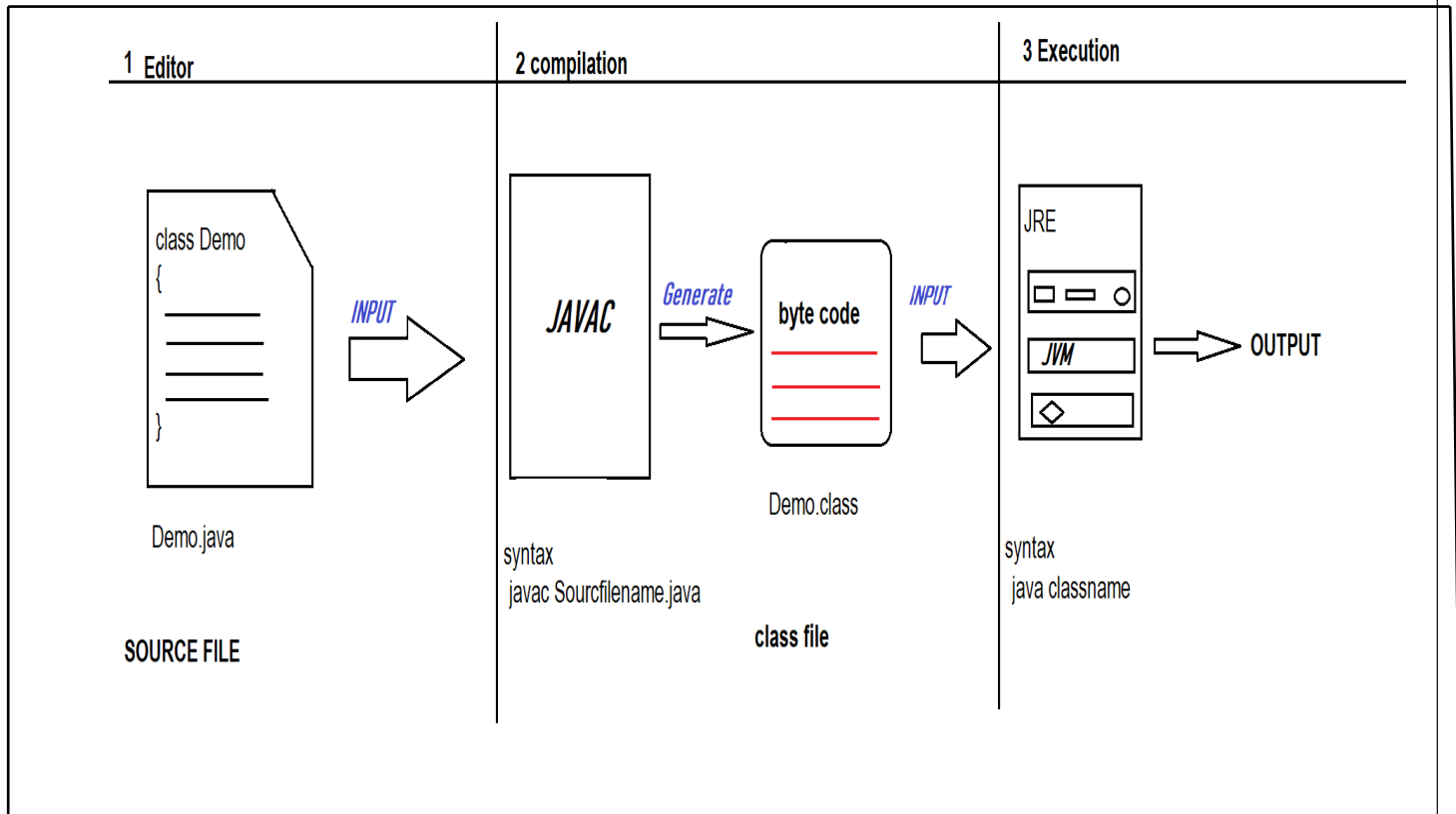
Variable's : Variables is a container or the data holder which is used to store the data.



## Steps to execute a java program

Machines cannot understand the java source file .

Hence they have to be translated.



Byte code is a file generated by compiler which is not understandable by machines nor by humans

Javac is a software

Step 1:	Step 2: Compilation	Step 3: Execution
Editor choice Program1.java  Class Program1 {  }	1.javac complier 2.source file  syntax javac srcfile.java ex: javac Program1.java	1.JRE 2.class file  Syntax java classname eg: Java Program1

A class file cannot be generated for an empty class

Java src file.java

Java classname

Empty class can be compiled

Empty class cannot be used for execution

## Print Statements

*System.out.print(data)*

*System.out.print(10)*

*System.out.print(20)*


*System.out.print()* -----> C.T.E

*System.out.println(data)*

*System.out.println(20)*

*System.out.println(85)*


*System.out.println()* -----> C.T.S

in `System.out.print();`

The cursor prints the data and moves right does not go for next line

in `System.out.println();`

The cursor first prints the data and moves to the next line.

class Program2

{

`System.out.println("java");` // Compile Time Error

}

Compile Time error is shown because inside the class we can declare variable's and methods

We can have only declaration statement's not print statements in java

## Chapter 2: Tokens

Token's: Smallest component of any programming language is called as Tokens

Types of Tokens

- 1) Keywords
- 2) Identifiers
- 3) Literals/values
- 4) Separator's
- 5) Comments

### Keywords

Keywords are predefined words which have some meaning & can be understood by the compiler

Keywords are also called as reserved words

Because they should not be used by programmer for his convenience

Rules

- 1) They are case sensitive should be written in lowercase
1. **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean** : Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.

4. **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.
5. **case**: Java case keyword is used with the switch statements to mark blocks of text.
6. **catch**: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class**: Java class keyword is used to declare a class.
9. **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default**: Java default keyword is used to specify the default block of code in a switch statement.
11. **do**: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double**: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else**: Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final**: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.

19. **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if**: Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements**: Java implements keyword is used to implement an interface.
22. **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.
23. **instanceof**: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int**: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long**: Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native**: Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new**: Java new keyword is used to create new objects.
29. **null**: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package**: Java package keyword is used to declare a Java package that includes the classes.
31. **private**: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. **protected**: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return**: Java return keyword is used to return from a method when its execution is complete.

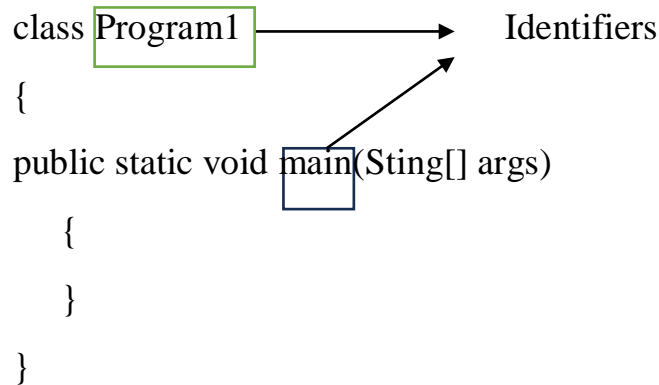
35. **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static** : Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized**: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
41. **this**: Java this keyword can be used to refer the current object in a method or constructor.
42. **throw**: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws**: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient**: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try**: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void**: Java void keyword is used to specify that a method does not have a return value.
47. **volatile**: Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **while**: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.



# Identifiers

Identifier's : The names given to components of java like variables ,methods ,class, interface ,Enum's is called as identifiers.

```
class Program1  
{  
    public static void main(Sting[] args)  
    {  
    }  
}
```



## Rules of Identifiers

- 1) It should not start with a number

Program1



1Program

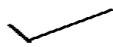


- 2) Identifiers should not have any special characters apart from

(\$) and ( \_ )

Eg

\$Program1



\_Program1



#Program



- 3) we cannot use identifiers are keywords

Eg :

class



if

while

we cannot use keywords as identifier's but if we want to use keywords are identifiers then we can do it by changing their case.

# Literals

The data written by the programmer in the program is called as literal's

Class Program1

```
{  
    Public static void main(String[] args)  
    {  
        System.out.println("hi"); //hi is the data or literals  
    }  
}
```

Types of literals

1)Number literals

i)Integer 1,-4,98,-104.....

ii)decimal: 1.2 , -3.2, 9658.25...

2)Character literals: Enclosed in single quotes

The length of the character literals should always be one

Eg

'a'

We cannot store two values in characters

Eg

'ab' ✗

3)Boolean literals: literals which has only values

true, false

When ever the answer is either yes or no type

Then we use Boolean literals

Yes is represented as **true** in java.

No is represented as **false** in java.

4)String Literals: Group of characters is called as String literals

Eg: hello world

String should always be represented in double quotes

Eg: " Hello World "

## Seperators

They are special characters that separate different parts of code statements or expressions.

1) ( ) => Used to enclose parameters in method declaration and invocation

And also used for surrounding cast types

Eg

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

2) { } => used to define the block of codes for classes ,methods and local scopes and also for automatically initialized arrays.

Eg

Class Program1

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

}

3) [] => used to declare an array

Eg: public static void main(String[] args)

{  
}

4) Semicolon ; => used to terminate the statements

Eg

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

5) Comma , => used to separate identifier's in a variable declaration

Eg:

int a , b , c , d , e ;

6)

Period . => used to separate package names, from subpackages names and  
Classes and also used for calling the classes and methods.

Eg

Import java.util.Scanner;

Classname.Methodname;

## Comments

Comments are used to explain the code and to make it

More readable and understandable

Types of comments

1)single line comments

2)Mult line comments

3)Documentation comments

1) Single line comments :

Used to comment down the single line of code in the java program

Represented by two forward slashes //

2)Multi Line comments :

Used to comment multi line of code like a method, block etc..

Represented by

```
/*
```

```
*write the code here
```

```
*
```

3)Documentation comments:

Used for creating the documentation for project or any software applications

Represented by

```
/**
```

```
*
```

```
* we can write our documentation here
```

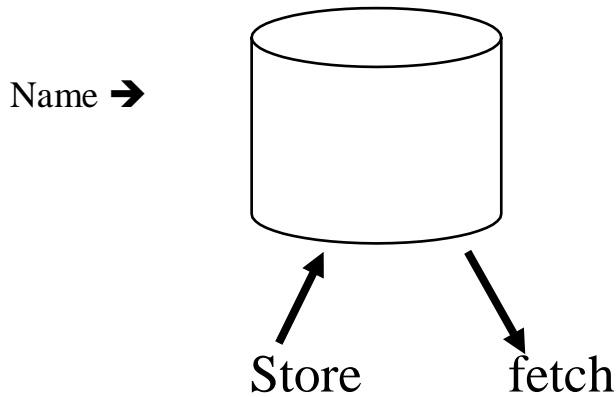
```
*
```

```
*/
```

## Chapter 3: Variables and DataTypes

Variable is a container or the data holder which is used to store the data.

A named block of memory is called as variables .



### Charactersticas of a variables

- 1)Variable can store/fetch data using name
- 2)We can store only on value inside variables
- 3)Data stored in the varaibles are not permanent they are temporary
- 4)Every variable has a lifespan i.e after sometimes it gets expired
- 5) Every variables has a scope.

### **Scope===visibility level**

How to create a variable

Create a container to store a specific type of data

Datatype: Datatype will help us to inform we are crating a container to store a what type of data.

### **Datatype**

Defines what type of data we are storing in the varaible or container

For eg;

Water : bottle

Data => container

book : bag

data=> container

Syntax for creating variables

Datatype identifier; → variable declaration statements

To create multiple variables in single statements

Datatype identifier1 , identifier2 , identifier3 , identifier 4.....;

Multiple variables can be created in a single statement

To store integer data we have predefined datatype called as int

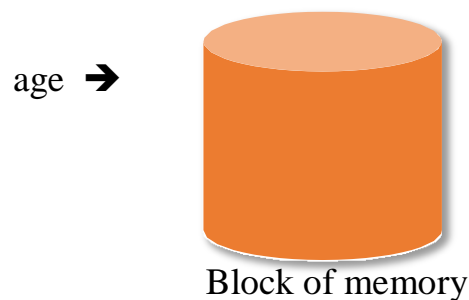
Int → integer no

Step 1:

Create variables

int age;

during execution inside the jre a memory block is created



Step 2: Store data in variable

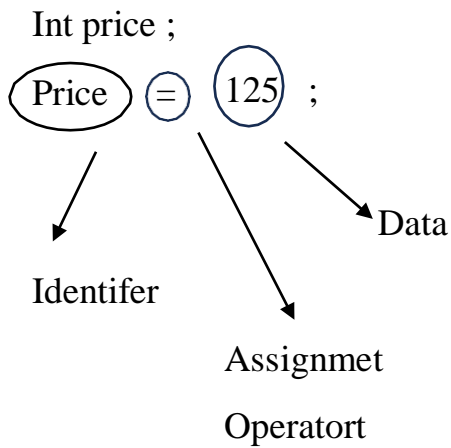
To store data we need help of assignment operator

Represented by = equals to

Assignment operator is used to store data inside variables



Book has a property called price



Eg

```
class program1
```

```
{
```

```
    Public static void main(String[] args)
```

```
    {
```

```
        Int price;
```

```
        Price=125;
```

```
    }
```

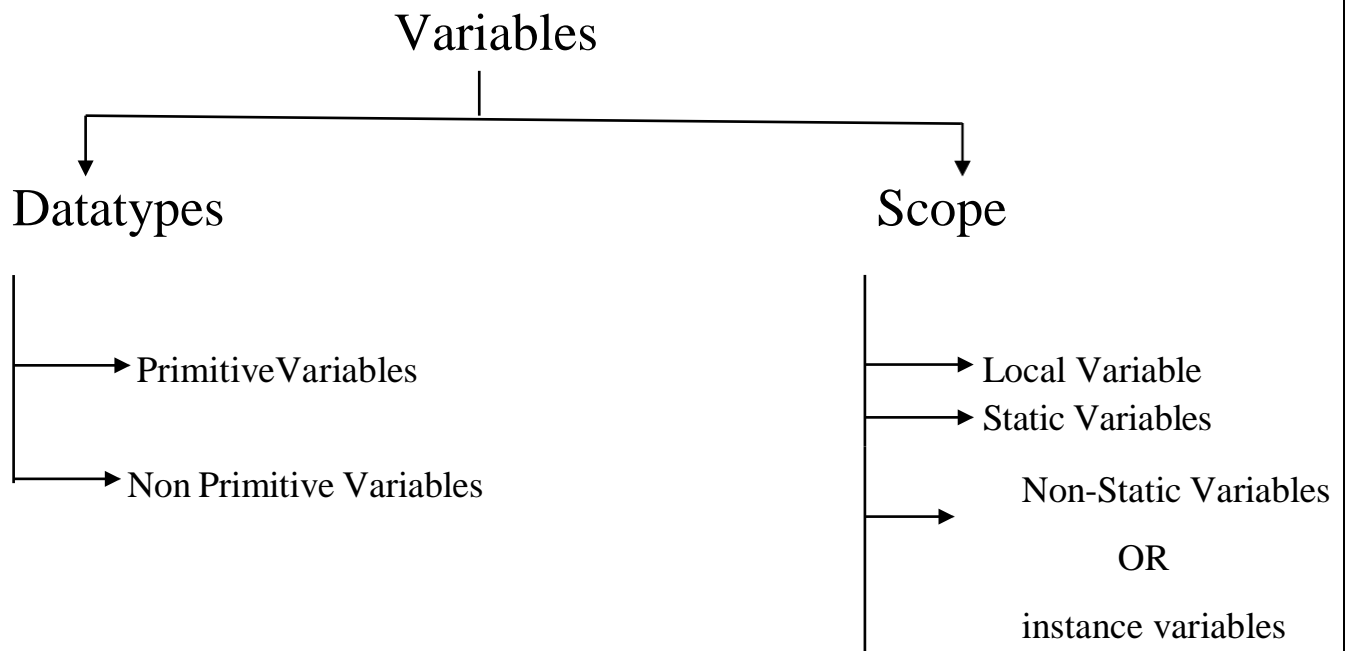
```
}
```

### ***Advantages of variables***

- 1) Easy to fetch
- 2) We can modify/re-assign data
- 3) We can update variables

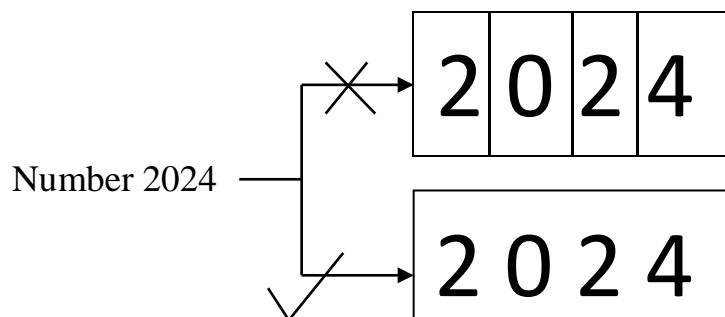


## Types of variables



### **Primitive Variables**

Primitive data : The data which can be store in a single block of memory is calles as Primitive data.

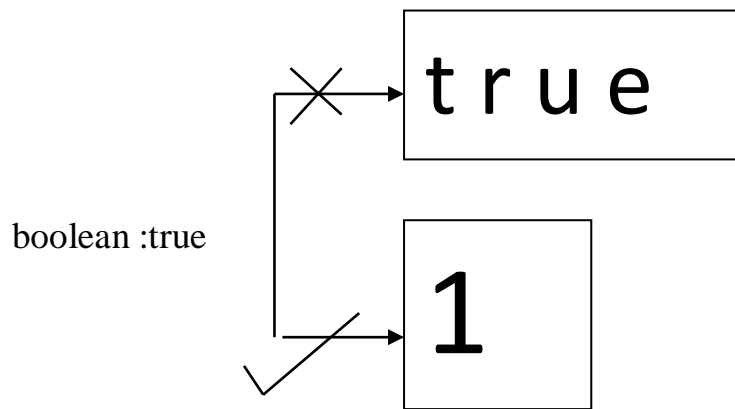


Character: 'a' →

a

ASCII value of a is stored

97



boolean true is not stored but representation 1 is stored

if false then 0 is stored

**Primitive data** :The data which can be stored in a single block of memory is called as primitive data

The variables created to store primitive data is called as primitive variables.

Eg

Number literals

Character literals

Boolean literals

Values are stored inside primitive variables

int a;

a=20;

a        20

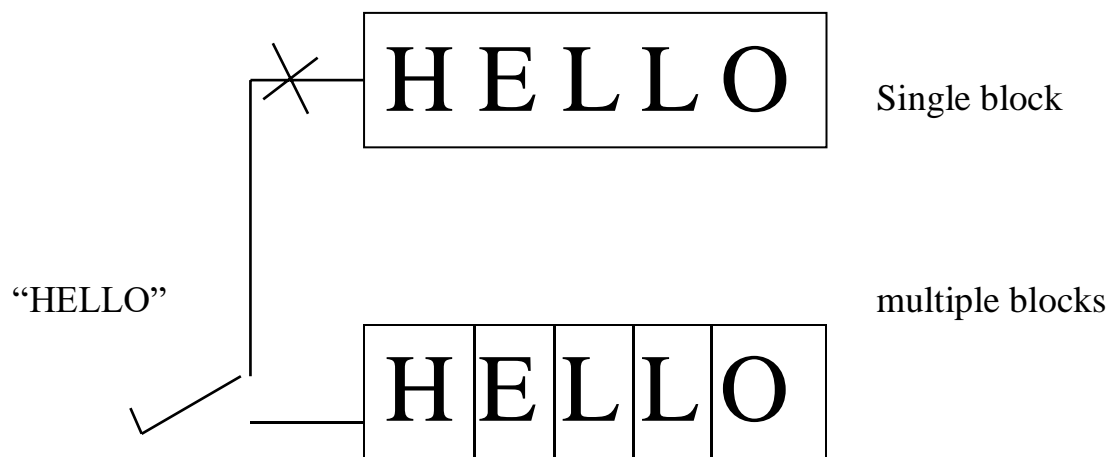
here when you create a variable a block of memory is created and value 20 is stored in that.

Non primitive variables

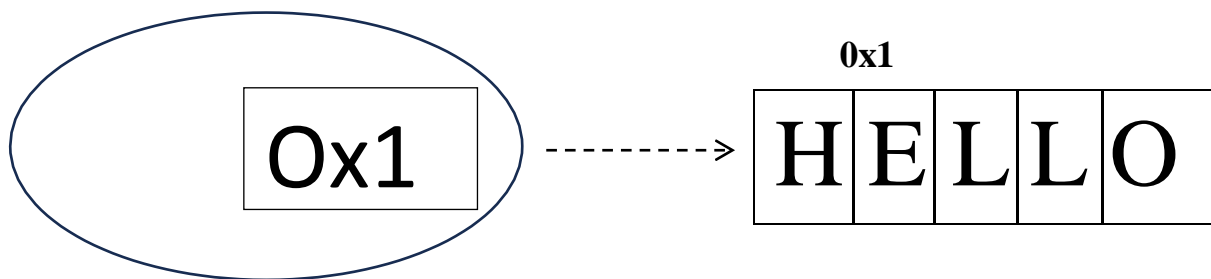
Non primitive data:

The data which cannot be stored in the single block of memory but requires multiple blocks to store data is called as non primitive data

Eg Arrays, object ,Strings



### Non Primitive variables:



Non primitive variables is used to store the reference of memory.

Hence it called as non pimitive variables or reference variables.

Or

The varibale created for non primitive datatype which can be used to store the reference of an object or array is called as non primitive variables or reference variables

## Local varaibles

A variable declared in a method block or instialiazer block or constructor block or any other block is called as local variables.

Eg

Class Name

{

Int b;// not a local variables

Public static void main(String[] args)

```
{  
    int a ; ➔ a is local variables because declared inside method block other  
    then class block  
}  
}
```

Rules for local variables

- 1) We cannot use local variables without assigning data  
[because local variables does not have default data.]
- 2) We can only use the variables inside the block where it is declared  
[we cannot use outside the block]
- 3) We cannot declare two local variables with same name inside same scope

## Primitive datatype

Primitive data

1.Number

2.character

3.boolean

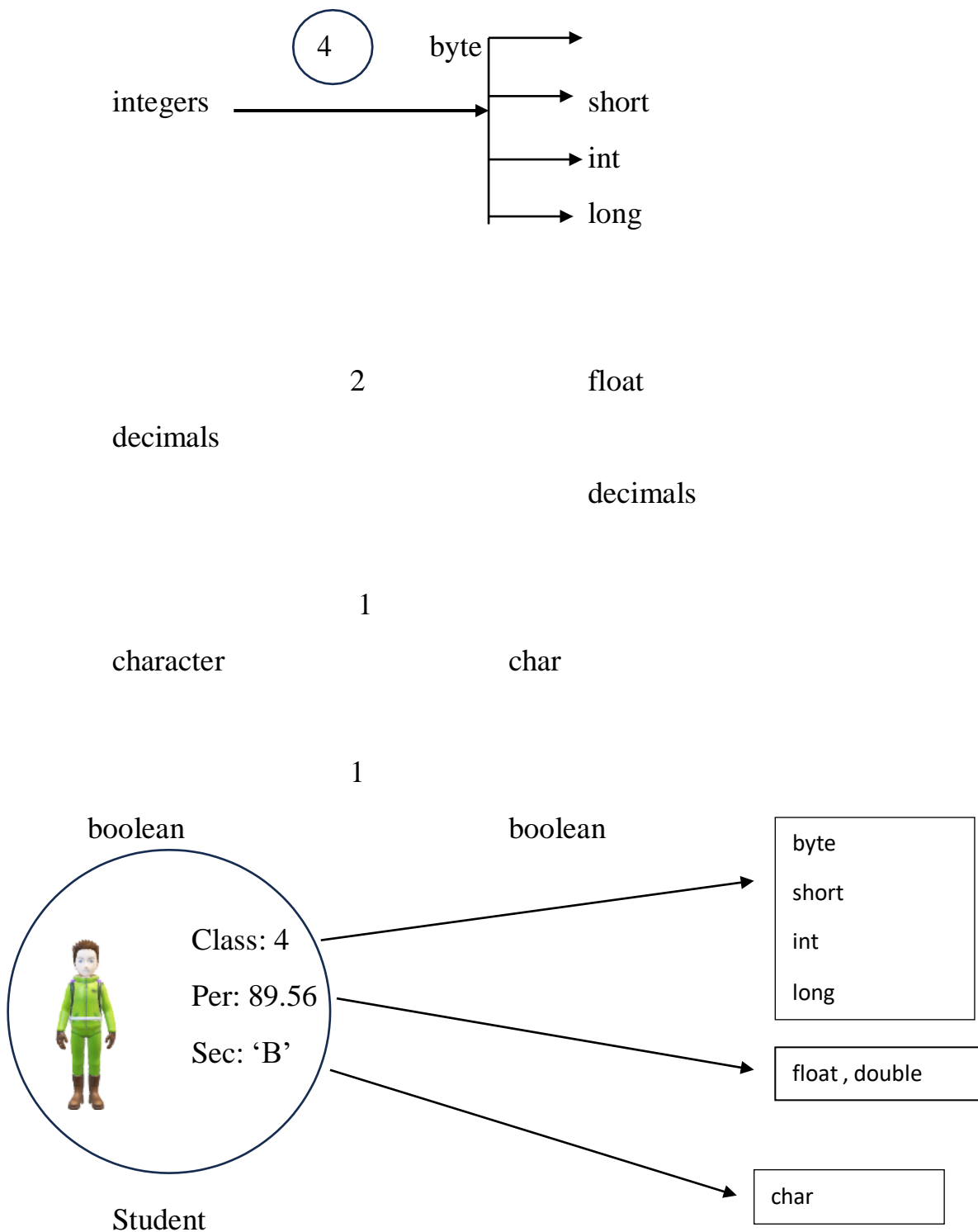
Primitive datatypes are used to create primitive variables to store primitive data or values such as number, character or boolean

Primitive datatypes cannot be created we have predefined datatypes

Primitive values		Primitive data types	Default values	Size
Number	Integers (whole numbers) +ve to -ve	byte	0	1bytes
		short	0	2bytes
		int	0	4bytes
		long	0 l/L	8bytes
	Floating values	float	0.0 f/F	4bytes
		double	0.0 d/D	8bytes
	Character	char	/u0000	2bytes
	Boolean	boolean	false	1bit

Numberdatatype in increasing order of the capacity

byte<short<int<long<float<double



Class can be stored either in byte ,short,int.long

Percentage can be stored in either float or double

Sec can be stored in char

## Non primitive datatypes

The datatypes which is used to create a non primitive variables which can store non primitive data is called as non primitive datatypes.

Every class name is a non primitive datatypes

```
class Book{  
}
```

Book is a non primitive datatype

Book a ;

a is the variable of Book type(non primitive datatype)

eg 2

```
class Instagram  
{  
}
```

Instagram is a non primitive datatype

Instagram i;

i is the variable of Instagram type(non primitive datatype)

null is the default value for all non primitive datatypes

## Chapter 4 : Introduction to operators

Operators

Predefined symbol

It is used to execute a task

Eg

(+) addition

(-) subtraction



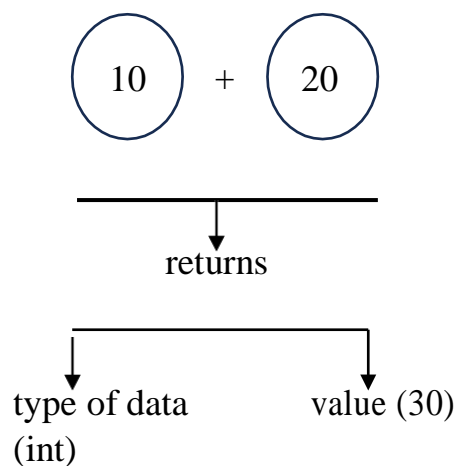
Operand: Operand is the data on which the operators execute the task.

Operator: An operator is a predefined symbols which performs a task on the given operand

### Characterstics of operators

- 1) It returns the result after the execution

Eg 10+20





$10 + 'a'$

An integer and a character can be added because character cannot be recognized by jvm it converts to ascii values of characters and based on the higher data type it stores

## **2)Precedence(priority)**

- ➔ Precedence is the priority given for every operators
- ➔ Precedence is used when an expression contains more than one operators
- ➔ Precedence is used only when we have complex expression
- ➔ It is the priority of operators used for evaluation expressions

Eg  $10 + 20 * 3$ ;

Here the multiplication is performed first and then addition is performed

Because precedence of multiplication is higher than addition

So the answer is 70

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Left to Right
13	++ -- + - ! ~ ( type )	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5		Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3		Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= % =	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

*Larger number means higher precedence.*

### 3) **Associativity:** Direction of order of execution

Left to Right

Right to left

Ex  $10 \oplus 20 \oplus 30$

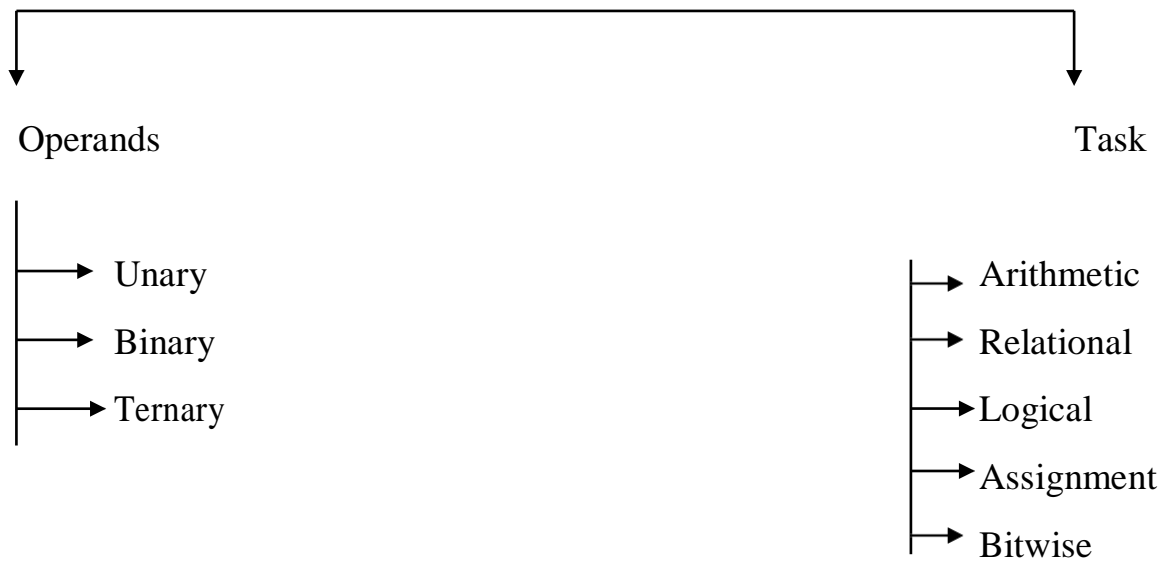


$10+20 \rightarrow 30+30$

$30+30 \rightarrow 60$

## Types of Operators

### Operators



**Unary:** Operator which accepts single values and performs operations

**Binary:** Operator which performs task on two data's at a time.

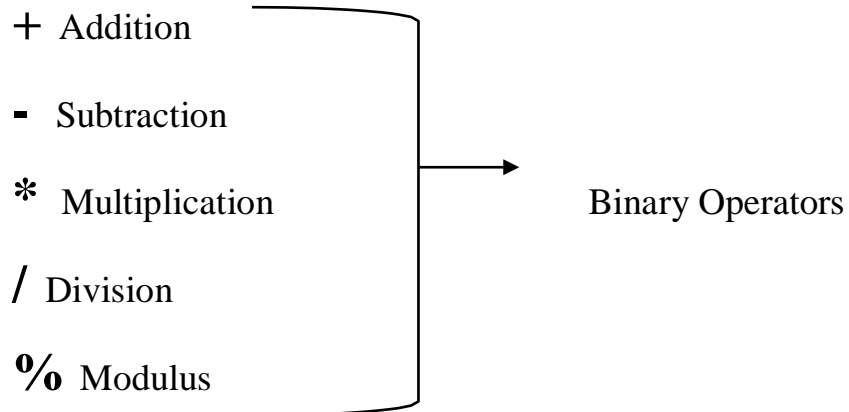
Eg

Addition operator

$10 + 20;$

**Ternary Operator:** Operator which accepts three values at a time.

## Arithmetic Operator



$+$  has two important behaviours

1) Addition

2) Concatination(merging)

1) when any one of the operand is String then it behaves like a concatenation operator

2) In all the other cases it behaves like an addition operator.

Eg  $10 + 20 \rightarrow 30$

$10 + \text{"Hello"} \rightarrow 10\text{Hello}$

## Mutlification operator(\*)

Multiplicaiton operator is allowed only with number data and character literals any other literal is not allowed.

$10 * 2 \rightarrow 20$

$\text{'a'} * 2 \rightarrow 194$

$\text{"a"} * 2 \rightarrow \text{C.T.E}$

$\text{true} * 2 \rightarrow \text{C.T.E}$

Divisionn operators returns quotient

## **Modulus operators( %)**

% → divides and returns remainder as the output

3 % 10 → 3/10 → 3

10 % 3 → 10/3 → 1

In both the cases after dividing the quotient is not taken instead the remainder is taken.

→  $m \% n$

Case 1: if  $m$  is  $< n$  o/p:  $m$

Case 2: if  $m == n$  o/p: 0

Case 3: if  $m < n$

1)  $m$  is multiple of  $n$  → o/p: 0

2)  $m$  is not multiple of  $n$  o/p:  $\{1, 2, 3 \dots n\}$

## **Increment Operator**

Represented as ++

Is used to update the variables by increasing the value by 1.

int a;

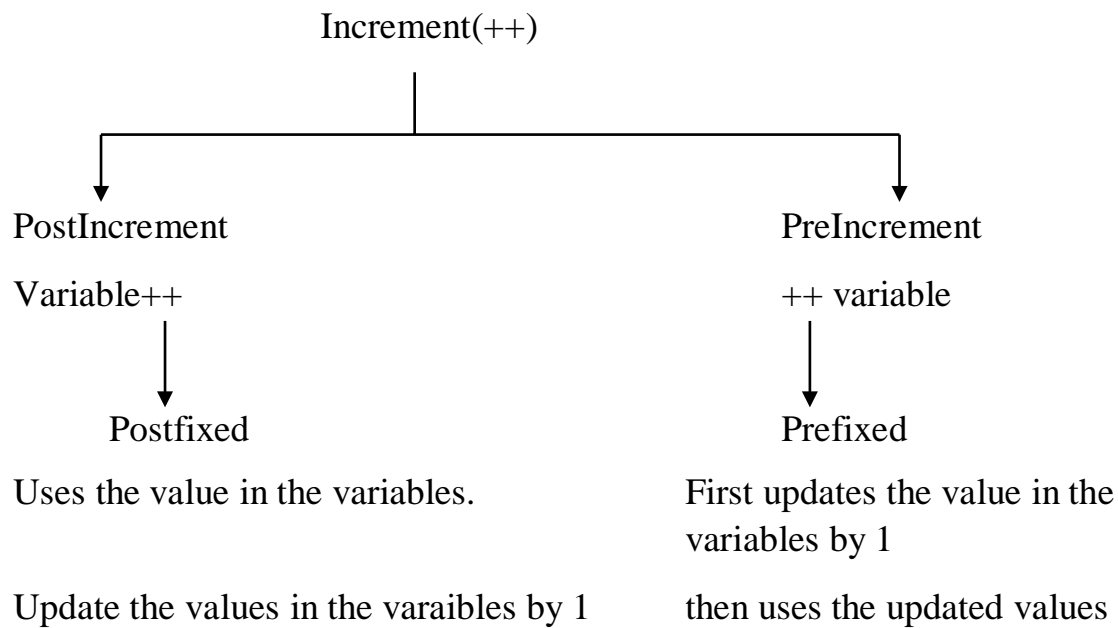
a=10;

a++; → a = a+1;

10+1;

a = 11

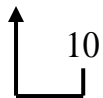
it will 1 to the data present inside the variables



Case 1:

```
int a=10;
```

```
int b= a ++;
```



a ----> ~~10~~ 11

b ----> 10

```
S.O.Pl(a);
```

```
S.O.Pl(b);
```

Here the value 10 is copied first and then updates the value by 1.

Post increment operators will substitute the existing data present inside the variables later updates the variables.

First substitute the data and then update the variables

PreIncrement operator first update the value and then substitutes the value to the variables.

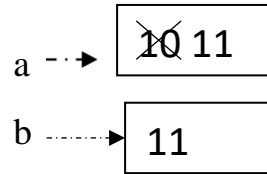
### Case 1:

Int a=10;

Int b=++a;

S.O.Plh(a);

S.O.Plh(b);



Preincrement operator first updates the values in the variables by 1

Then uses the updated values.

### **Decrement operator**

→ represented by - -

It is used to decrement the variables by 1;

### **Types of Decrement operator**

#### **Post Decrement :**

Uses the existing values and

then decrements the value by 1

#### **Pre Decrement**

Decrements the value by 1

Then uses the updated values

NOTE: we can only use the increment and decrement's with variables

We cannot use with values directly.

## **Relational Opertors**

Relational Opertors are used to compare the values

We can only relational operators to compare the values nothing can be done else.

1) It is a binary Operator

Return type of relational opertor is always boolean

## **Types of Relationsl operators**

- 1) Equality opertor → compares the two values are same or not
- 2) Not equlaity operator → Compares the two values are different or not
- 3) Greater than opertor → returns true is the leftside is higher than the Right side.
- 4) Lesser than operator → returns true is the leftside is lesser than the Right side.
- 5) Greater than operator → returns true is the leftside is higher than or equal to the Right side.
- 6) Lesser than operaor. ==> returns true is the leftside is lesser than or equal to the Right side.



Operator				
==	10==10	'a'=='a'	10==20	true==true
	True	true	false	true
!=	10!=10	'a'!='a'	10!=20	true!=true
	False	false	true	false
>	10>20	20>10	'a'>'b'	
	False	true	false	
<	10<20	20<10	'a'<'b'	
	True	false	true	
>=	10>=10	'a'>='a'	10>=20	
	True	true	false	
<=	10<=10	'a'<='a'	10<=20	
	True	true	true	

### **Conditonal Opertors**

Condtional operator is called as ternary operator as it takes three data to process.

Syntax

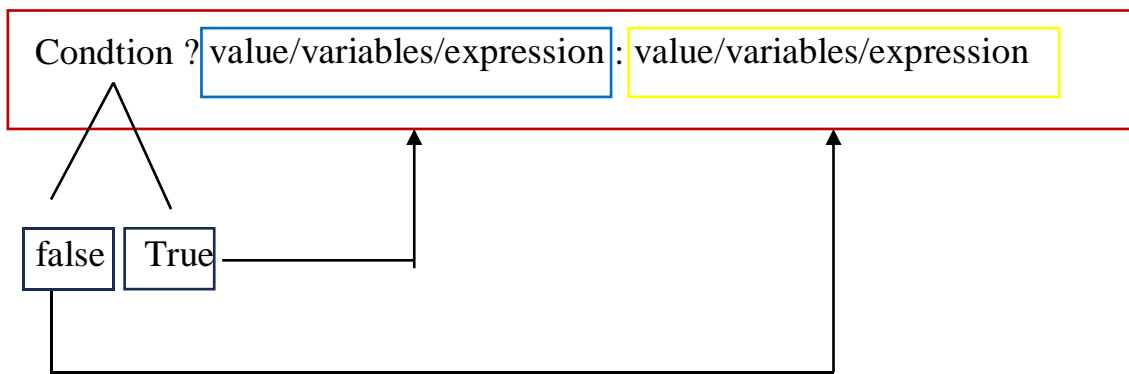
Operand1 ? Operand2 : Operand3

In operand1 we should write the condition

What is a condtion

Any expresssion who's output is boolean is called as condition

In operand2 and operand3 we can write values/variables/expressions



If the condition is true then the operand2 is executed

If the condition is false then the operand2 is skipped and operand3 is executed

We can directly use the conditional operator in print statement

Or we can store the result in a container and use it later

Eg

```
int a=10
```

```
int b=20;
```

```
int res=(a>b) ? a : b;
```

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

## Logical Operators

1) Logical And ( **&&** )

2) Logical Or ( **||** )

3) Logical Not ( **!** )

1) Logical And operator

- It is a binary operator
- It works only on boolean data
- Return type is always boolean
- We use only if both the conditions to be satisfied

- 1) Returns true if both the conditions are true
- 2) Returns false if any of the conditions is false.

Eg  $(100 > 200) \ \&\& \ (100 == 100)$

false && true → false

eg 2  $(100 == 100) \ \&\& \ (100 < 200)$

true && true → true

## 2) Logical OR operator( || )

Logical operator : Logical or Operator is used to merge two multiple conditions in a situation where one of the conditions should be satisfied not all.

- It is a binary operator
- It works only on boolean data
- Return type is always boolean
- We use only if the any one of the conditions to be satisfied

Eg  $(100 > 200) \ || \ (100 == 100)$

false || true → true

eg 2  $(100 == 100) \ || \ (100 < 200)$

true || true → true

eg 3  $(100 == 100) \ || \ (100 < 100)$

true || false → true

eg 4  $(100 > 100) \ || \ (100 < 50)$

false || false → false

### 3) Logical Not Operator ( ! )

- It is a unary operator
- It works only on boolean data
- Return type is always boolean
- It will negate the boolean values

Eg

1) !true → false

2) !false → true

3) !10 → C.T.E // operators can't be used directly on values

4) !10==10 → C.T.E // operators can't be used directly on values

5) !(10==10) → !(true) → false

## **TypeCasting**

Type → datatype → type of data

Casting → converting

Conversion of datatype from one to another is called as type casting

10                      20

Int                      double

The process of converting from one datatype into another datatype is called as typecasting

## TypeCasting

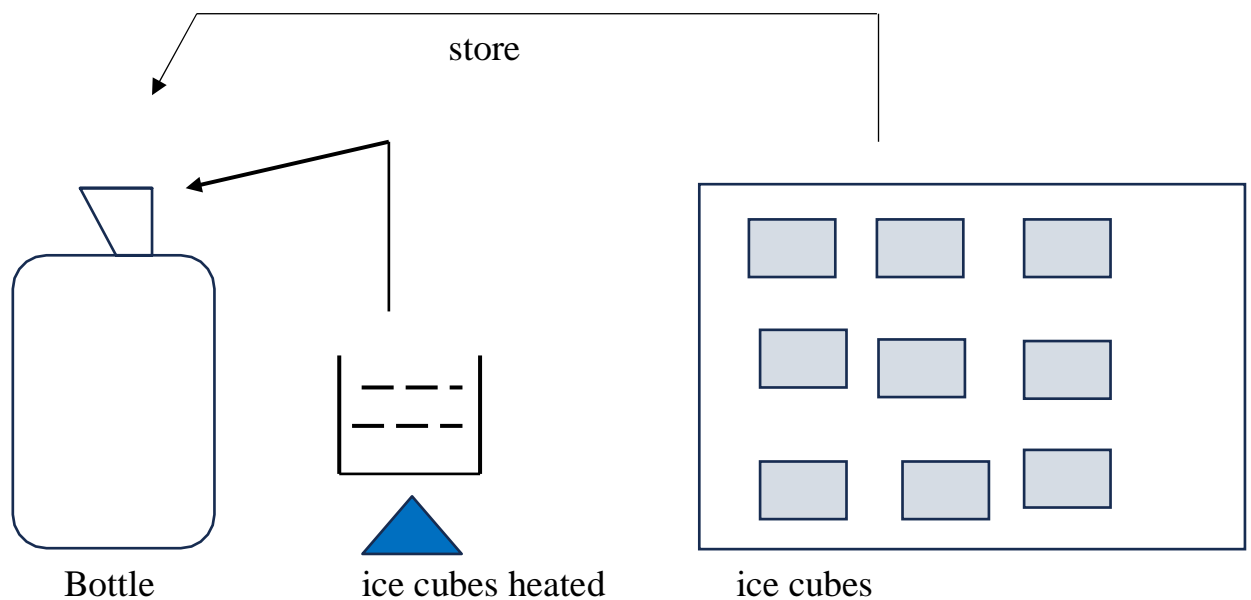


Who can do type casting

- 1) programmer
- 2) Compiler

If a programmer does the typecasting we call it as **Explicit Typecasting**.

If a compiler does the type casting we call it is as **Implicit typecasting**.



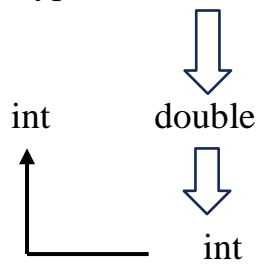
In this example the ice cubes cannot be stored directly into the water bottle because they are solid in state and bottle can store only liquid forms of data

So in order to store the ice cubes we are heating it so that the ice cubes are changed to water i.e from solid to liquid state

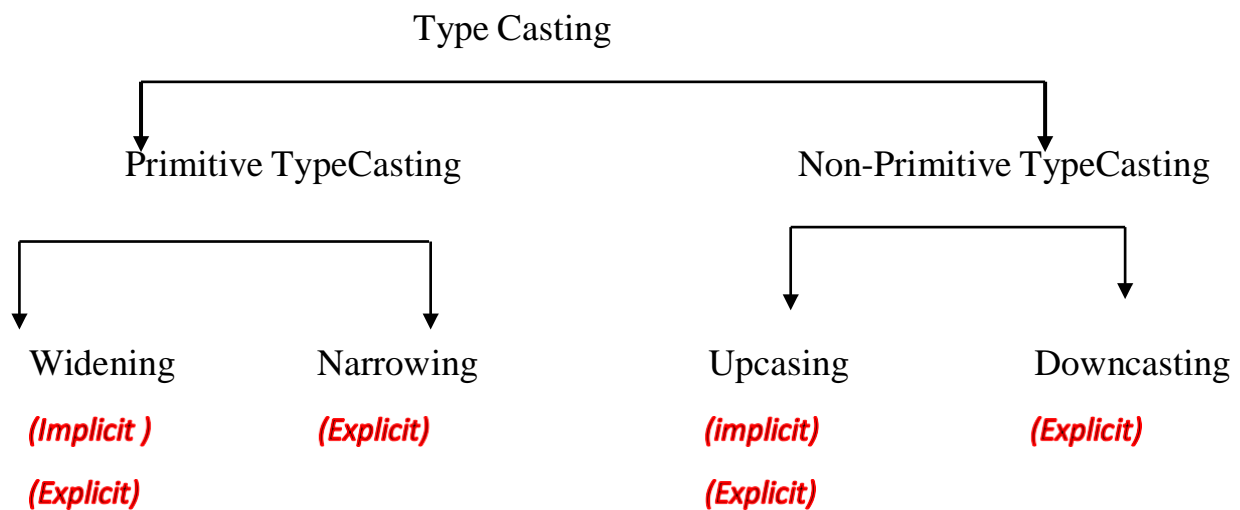
This is called as typecasting.

We can only convert the type of data but not the container

Type: int a=24.5;



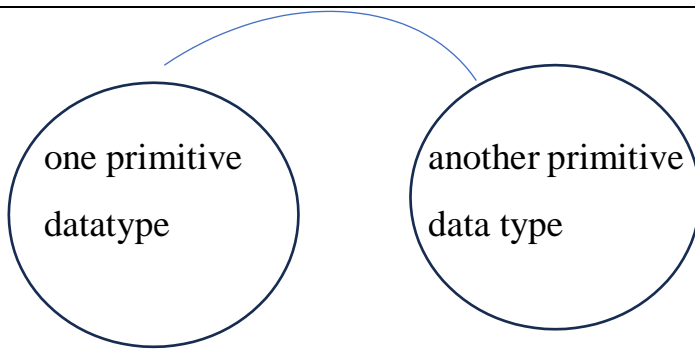
## **Types of TypeCasting**



### **Primitive type casting**

Primitive types

- 1.byte
- 2.short
- 3.int
- 4.long
- 5.float
- 6.double
- 7.char
- 8.boolean



The process of converting one primitive data type to another primitive datatype is called as Primitive typecasting.

### **Types of primitive typecasting**

- 1)widening
- 2)Narrowing

### **Widening**

The process of Converting a smaller or lower primitive data type into larger primitive datatype is called as Widening

```
int a =10;
```

```
double d;
```

```
d=a;
```

```
S.O.Pl(d)
```

```
o/p
```

```
10.0
```

Here integer datatype is converted into double datatype

### **Narrowing**

The process of Converting a Larger or higher primitive data type into lower primitive datatype is called as narrowing.

During narrowing there is a possibility of data loss

Since data loss is there compiler will not type cast.

Since compiler will not do it programmer has to do it we call it as explicit typecasting

### **Explicit type casting**

Type cast operator ( )

Syntax:

(type)values/variable/expression

Double a=10.23;

Int b=a;//C.T.E;

Int b=(int)a;

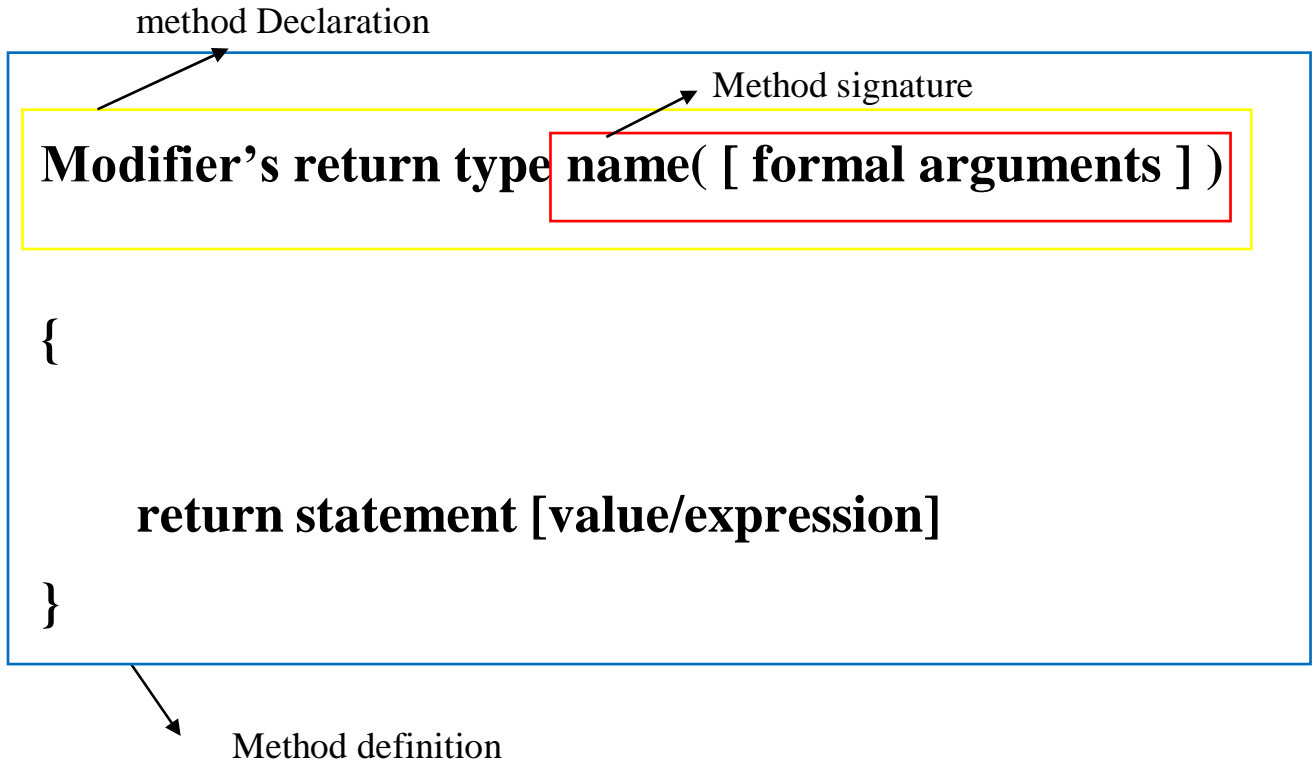
Here we are telling the compiler to convert from double datatype to integer datatype even though there is data loss



## Chapter 5 Methods

It is a block of instructions used to perform some specific task

Syntax



### **Method signature**

Name followed by formal arguments is called as method signature

Method signature

Method name + formal arguments

### **Method declaration**

Modifiers + return type + method signatures

### **Method definition**

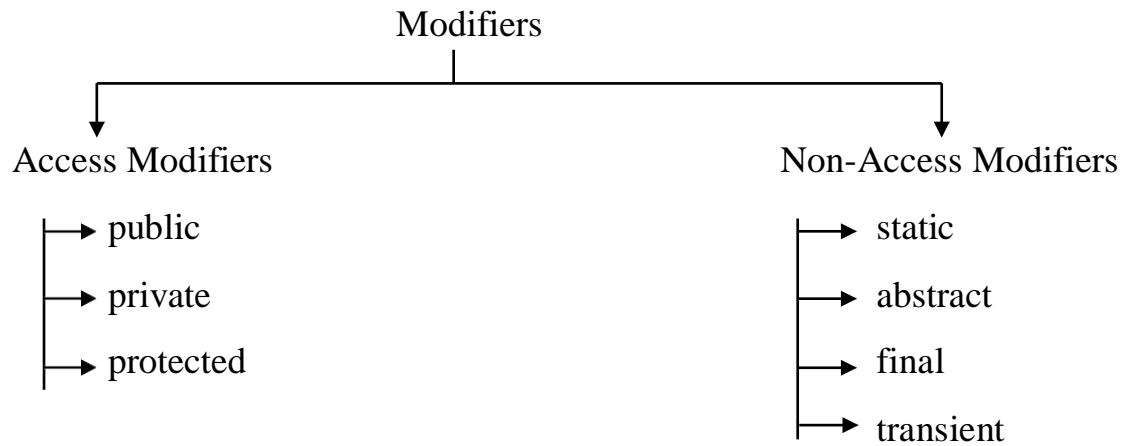
Method declaration+ method body

We can only create a method inside a class

# **Modifiers**

They are keyword

Used to change the behaviours of members of class



default is not a modifier it is a keyword

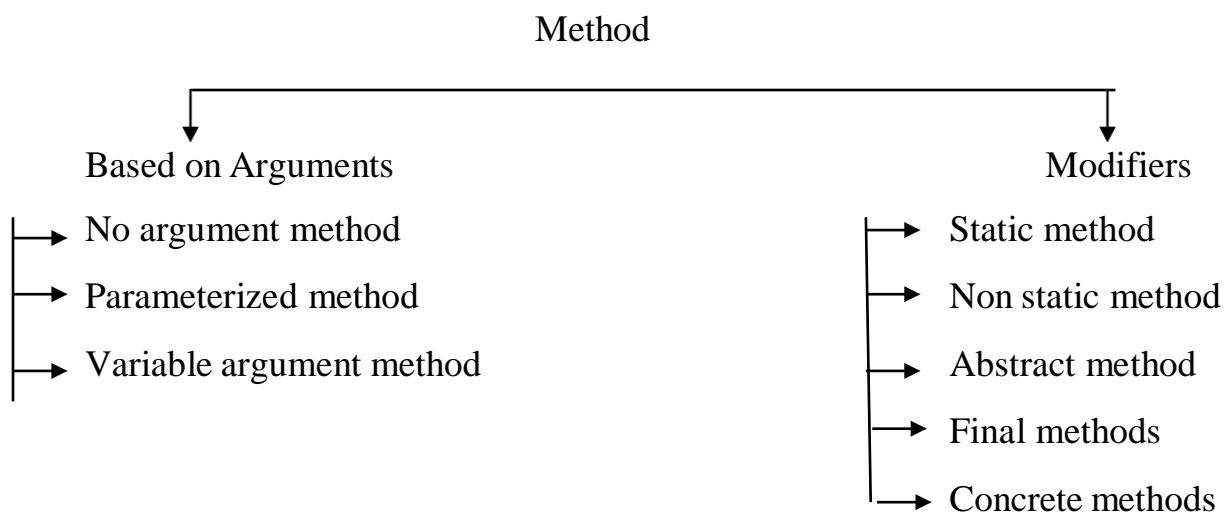
## **Return type**

Return type is the data type which tells what type of data is given back to caller after execution of method is completed

1) primitive type 8 primitive datatypes

2) non primitive types [arrays, classes, objects...)

## **Types of methods**



## **Characteristics of a method**

- 1) Method gets executed only when they are called
- 2) Methods can be executed any no of times

Call a method

Method signature is required to call a method

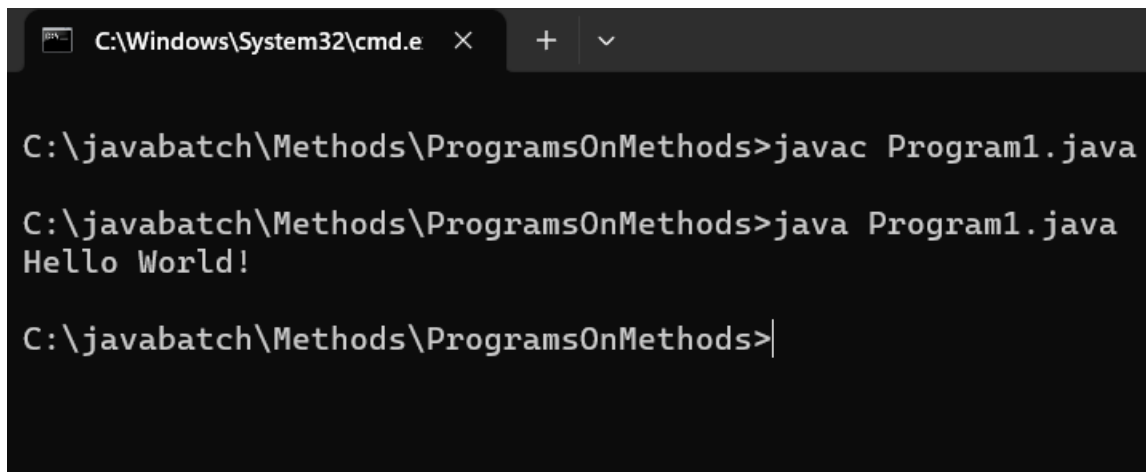
### **Syntax**

name(formal arguments)

We call this as method call statements

Here arguments are optional

```
class Program1
{
    public static void main(String[] args)
    {
        test();
    }
    public static void test()
    {
        System.out.println("Hello World!");
    }
}
```



```
C:\Windows\System32\cmd.e  X  +  v

C:\javabatch\Methods\ProgramsOnMethods>javac Program1.java

C:\javabatch\Methods\ProgramsOnMethods>java Program1.java
Hello World!

C:\javabatch\Methods\ProgramsOnMethods>|
```

When method call statements is executed

- 1)execution of current method is paused
- 2)control is transferred from caller to called method
- 3)Execution of called method starts from beginning
- 4)Once the execution is completed control returns to the caller
- 5)execution of caller resumes

## **No arguments method**

A method declared without formal arguments is called as no argument's method

Advantages

Easy to call the method

name()

No complications are involved

Disadvantages

Caller of the method cannot pass data to the method

Eg:

```
public static void add()
```

```
{
```

```
int a,b
```

```
a=10;
```

```
b=20;  
Int res=a+b;  
System.out.println(res);  
}
```

When you call the method add()

You cannot pass your own data so each time you get the output as 30

## **Parameterized method**

A method having formal argument's is called parameterized method

**Formal arguments** : The variables declared or created in the method declaration statements is called as formal arguments

Note: formal arguments are local variables of the method

Advantage: caller can pass data

Disadvantage

Caller cannot call without passing data

To call a parameterized method

Syntax

Name(value/expression)

The data passed in the method call statement is called as actual arguments

### **Rules to call a parameterized methods**

- 1) The no of formal arguments and no of actual arguments are same
- 2) The type of corresponding actual and formal arguments should be same

Ex signature demo(int)

demo() ; C.T.E

demo(20); Successful

demo('a') Successful because implicit type casting is happened

demo(20.0) C.T.E because of differ in actual and formal arguments

```
public static void demo(int a)
{
    _____
    _____
    _____
}
```

## Return data

When ever the caller calls the method the called method has the ability to return the data after execution

The data returned by the method after execution is called as return data

**Step1:** provide the return type

**Step2:** use return statement with value/expression

eg

```
Public static void add(int a, int b)
```

```
{
```

```
int res= a + b;
```

```
return res;
```

```
}
```

## To consume data returned by the method

The data returned by the method can be stored or can be used in an expression any other statement

```

1 package Methods;
2
3 public class Program6 {
4     public static void main(String[] args) {
5         System.out.println("main starts");
6         System.out.println(test(10,20)); //use in an expression
7         int data=test(40,50); //storing in the variables or container
8         System.out.println(data); //printing the values stored in the container
9         System.out.println("main ends");
10    }
11
12    public static int test(int a,int b)
13    {
14        int result=a+b;
15        return result;
16    }
17
18 }
19

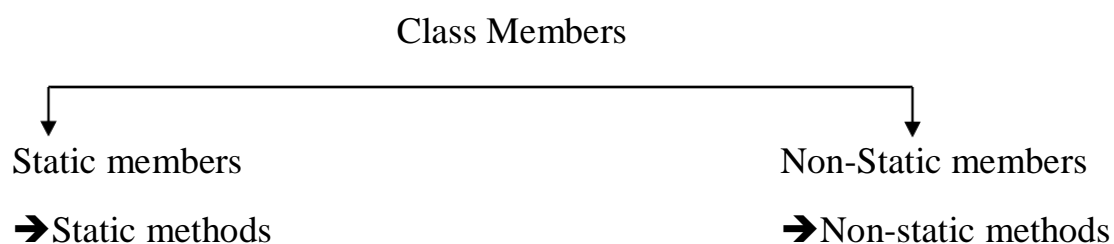
```

We can use method as an expression whenever the return type is anything other than void

## Rules to use return statement

- 1) return statement is mandatory if return type of the method is anything other than void.
- 2) If the return type is void, return statement is optional.
- 3) return statement in Java can return only one value.
- 4) return statement should be the last statement of the block.

Members of the class



Class Appl{

```
public static void demo()  
{  
}  

```

➔ Static method

```
public void test()  
{  
}  

```

➔ Non-Static method

}

To access the static method of the class inside the same class

Two ways

1)using method signature(directly)

2)using classname as reference

```
package Methods;  
  
public class Program8 {  
    public static void demo()  
    {  
        System.out.println("java");  
    }  
  
    public static void main(String[] args) {  
        demo();//using method signature  
        Program8.demo();//using class name as reference  
    }  
}
```

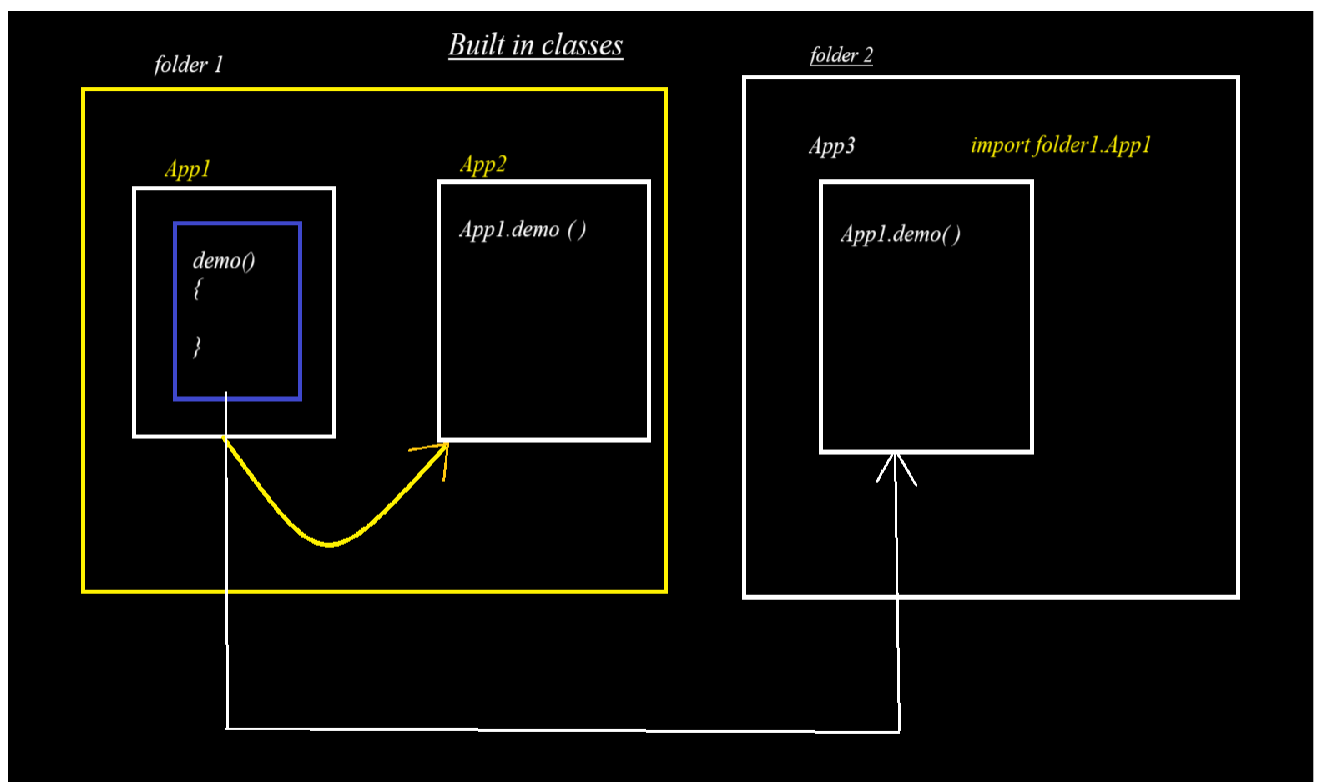


## To use static method of the class inside another class

1) Using class name as reference

```
1 package Methods;  
2  
3 public class Program9 {  
4  
5     public static void main(String[] args) {  
6         Program8.demo();  
7     }  
8  
9 }  
10
```

### Built In class



To call one method inside another folder2 we need to import the folder1 in to the second folder so we can access that method

So to import we have **import** keyword

To import the method we need to specify the class in which it is present in

And in which folder that class is present in

For eg:

**import java.lang.Math** → Fully Qualified Class Name

**import java.util.Scanner**

## **To create a non Static method**

A method which is declared without prefixing the keyword static is called as non -static method.

Eg :

Class App

```
{  
public void demo( )  
    {  
    }  
}
```

We cannot call non static method directly inside main method

With method signature or class name as reference

## **Steps to call a non static method**

Step 1: Create an instance(object) for class.

Syntax:

**Classname variable=new Classname();**

New is an operator which creates a block of memory inside the heap area for non-static method and returns the address.

And the address is stored in variables.

Non static variables are stored in the object to use non static variables we need the address.

Step 2: Call the method with the help of object reference

Syntax

**reference\_variables. methodcall( );**

```
1 package Methods;
2
3 public class Program11 {
4     public void demo()
5     {
6         System.out.println("non static method");
7     }
8     public static void main(String[] args) {
9         Program11 obj=new Program11();//object is created
10        obj.demo();//access with object reference
11    }
12
13 }
```

## Dynamic read

Taking the data directly from the user without the programmer explicitly giving the data during the execution of the application

To take data directly from the user and store it we have inbuilt methods

Such as nextInt( ),nextDouble( ), nextByte( )

Sl No	Return type	Method signature	Description
1	Byte	nextByte()	Returns byte data
2	short	nextShort()	Returns short data
3	Int	nextInt()	Returns int data
4	long	nextLong()	Returns long data
5	float	nextFloat()	Returns float data
6	double	nextDouble()	Returns double data
7	boolean	nextBoolean()	Returns Boolean data
8	String	next( )	Return string data(one word or upto space)
9		nextLine( )	Returns String
10		next().charAt(0)	Returns character data

And all these methods are present in Scanner class

Which is in turn present in another folder so we need to import that

The fully qualified class name for the given importing scanner class

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

**step1**: import Scanner class from java.util folder

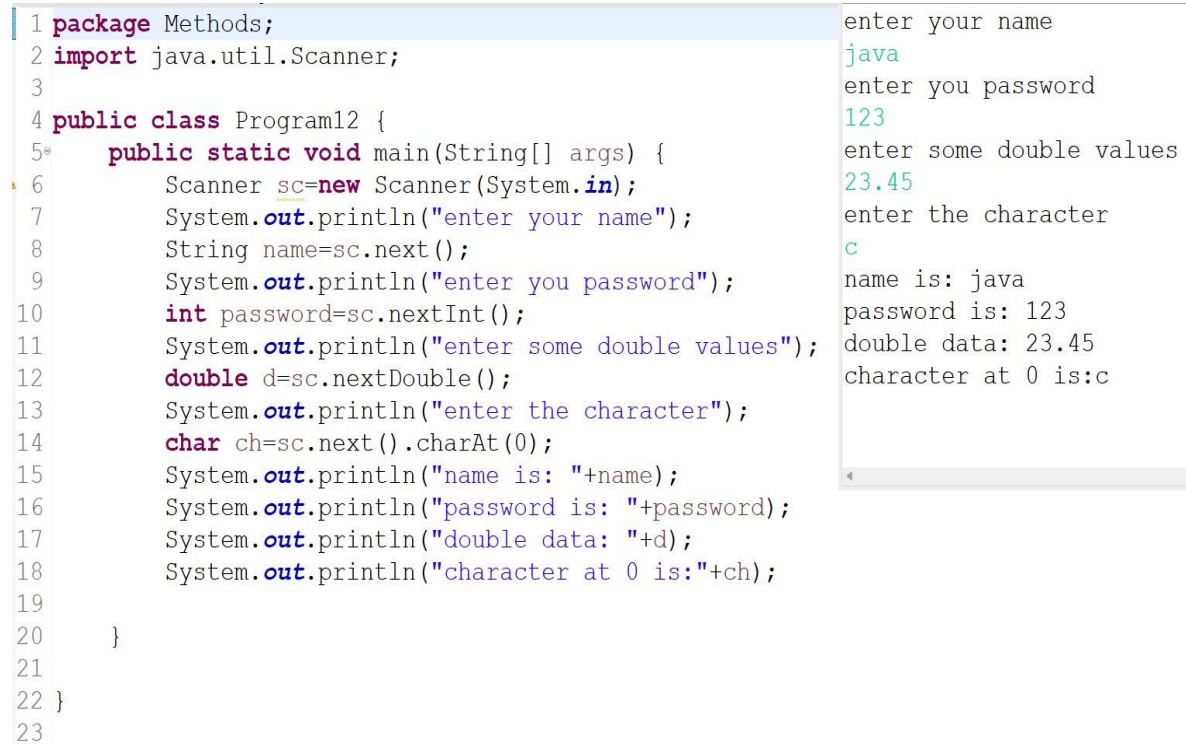
eg: import java.util.Scanner;

**step 2:** create an object of scanner and initialize it with system.in

Scanner sc=new Scanner(System.in);

**step 3:** Request for data by calling a method.

sc.nextInt();



The screenshot shows a Java IDE with a code editor on the left and a console window on the right. The code in the editor is as follows:

```
1 package Methods;
2 import java.util.Scanner;
3
4 public class Program12 {
5     public static void main(String[] args) {
6         Scanner sc=new Scanner(System.in);
7         System.out.println("enter your name");
8         String name=sc.next();
9         System.out.println("enter you password");
10        int password=sc.nextInt();
11        System.out.println("enter some double values");
12        double d=sc.nextDouble();
13        System.out.println("enter the character");
14        char ch=sc.next().charAt(0);
15        System.out.println("name is: "+name);
16        System.out.println("password is: "+password);
17        System.out.println("double data: "+d);
18        System.out.println("character at 0 is:"+ch);
19    }
20 }
21
22 }
23
```

The console window on the right displays the following output:

```
enter your name
java
enter you password
123
enter some double values
23.45
enter the character
c
name is: java
password is: 123
double data: 23.45
character at 0 is:c
```

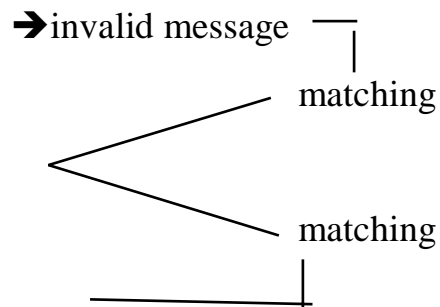
## Chapter 6 Decision Statements

It helps the programmer to decide whether the block of instructions to be executed or skipped

S1: open Facebook

S2: Enter UserID

S3: Enter Password



password has some rules

- 1)one uppercase letter
- 2)one number
- 3)one special character
- 4) Minimum 8 characters

If the rules are met only the we will be redirected to home page else

We get the invalid message

### Types of Decision statements

- 1)if
- 2)if else
- 3)else if ladder
- 4)nested if
- 5)switch

# **If**

It is a keyword

It is a decision making statement

Syntax

if (condition)

Statement;

If only one statements needs to be executed

if (condition)

{

Statements1;

Statements 2;

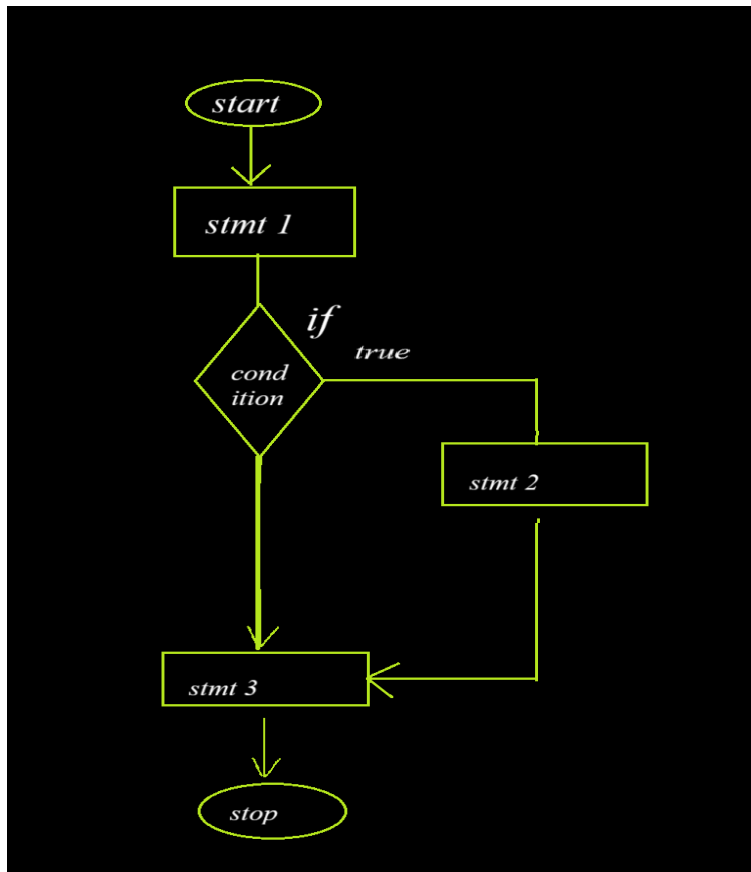
.

Statements n;

}

if multiple statements  
needs to be executed

**Flow Diagram**



```
1 package DecisionStatements;
2
3 import java.util.Scanner;
4
5 public class Program1 {
6     public static void main(String[] args) {
7         Scanner sc=new Scanner(System.in);
8         System.out.println("main starts");
9         System.out.println("enter the 1st data");
10        int a=sc.nextInt();
11        System.out.println("enter the 2nd data");
12        int b=sc.nextInt();
13        if(a>b)
14            System.out.println("a is greater");
15
16        System.out.println("main ends");
17    }
18 }
19
20
```

```
<terminated> Program1 (6) [Java Application] C:\Users\vinay\p2\pool\plugins\org.
main starts
enter the 1st data
20
enter the 2nd data
10
a is greater
main ends
```



## **If Else Decision Statement**

If else

if is a keyword and else is a keyword.

### **Syntax**

```
if(condition)
```

```
{
```

```
Statement1;
```

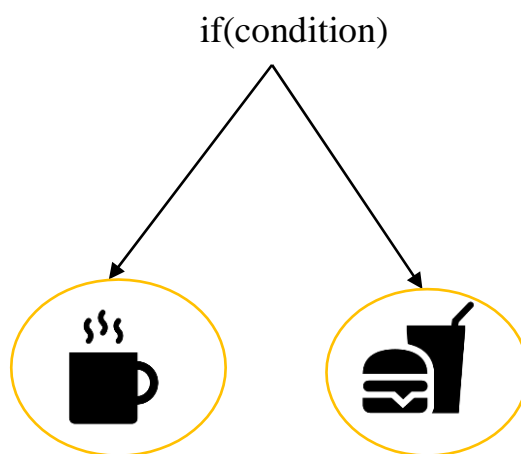
```
}
```

```
else
```

```
{
```

```
Statement 2;
```

```
}
```

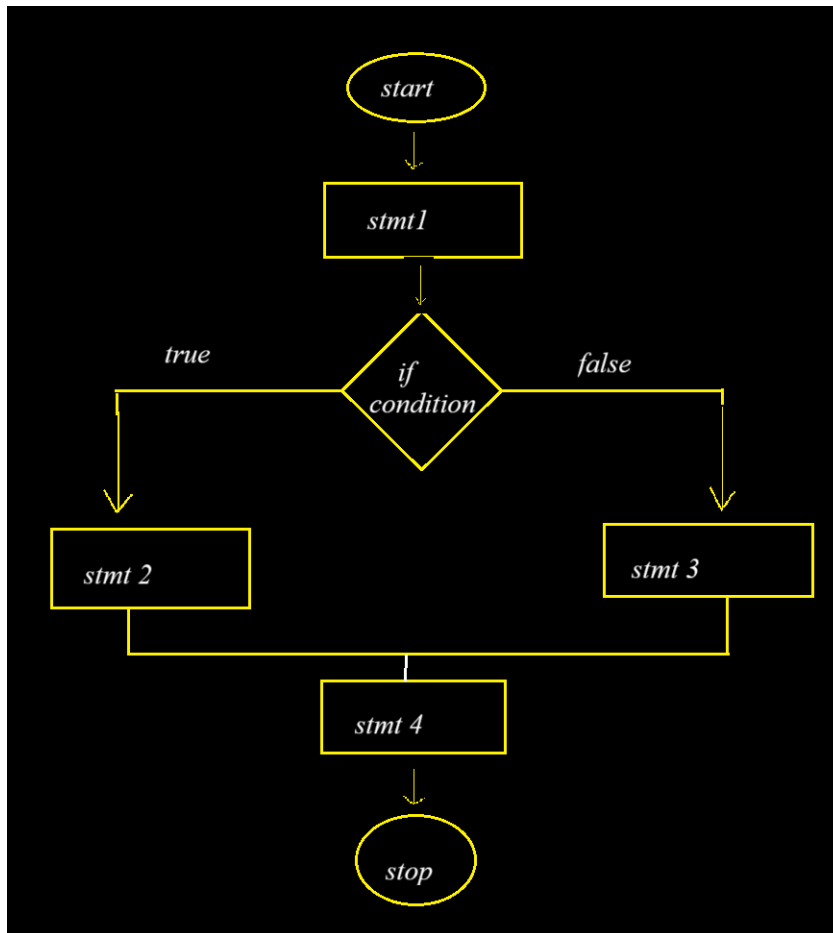


If a person is hungry he visits a restaurant

His first choice is coffee if coffee is available then he takes coffee

but if coffee is not there then only he will take burger.

**Flow Diagram :**



```
1 package DecisionStatements;
2
3 import java.util.Scanner;
4
5 public class Program2 {
6     public static void main(String[] args) {
7         Scanner sc=new Scanner(System.in);
8         System.out.println("enter your first preference");
9         int n1=sc.nextInt();
10        System.out.println("enter your second preference");
11        int n2=sc.nextInt();
12        if(n1>n2)
13            System.out.println(n1+" is greater");
14        else
15            System.out.println(n2+" is greater");
16    }
17 }
18 }
19
```

enter your first preference  
10  
enter your second preference  
5  
10 is greater

## **Else if ladder**

When we have multiple options then we go for if else ladder

- ➔ else block should be placed at the end of the ladder
- ➔ We can use else if ladder without using else block
- ➔ Only one block can be executed.

## **Syntax**

if(condition)

```
{  
    Statement;  
}
```

else if(condition)

```
{  
    Statement;  
}
```

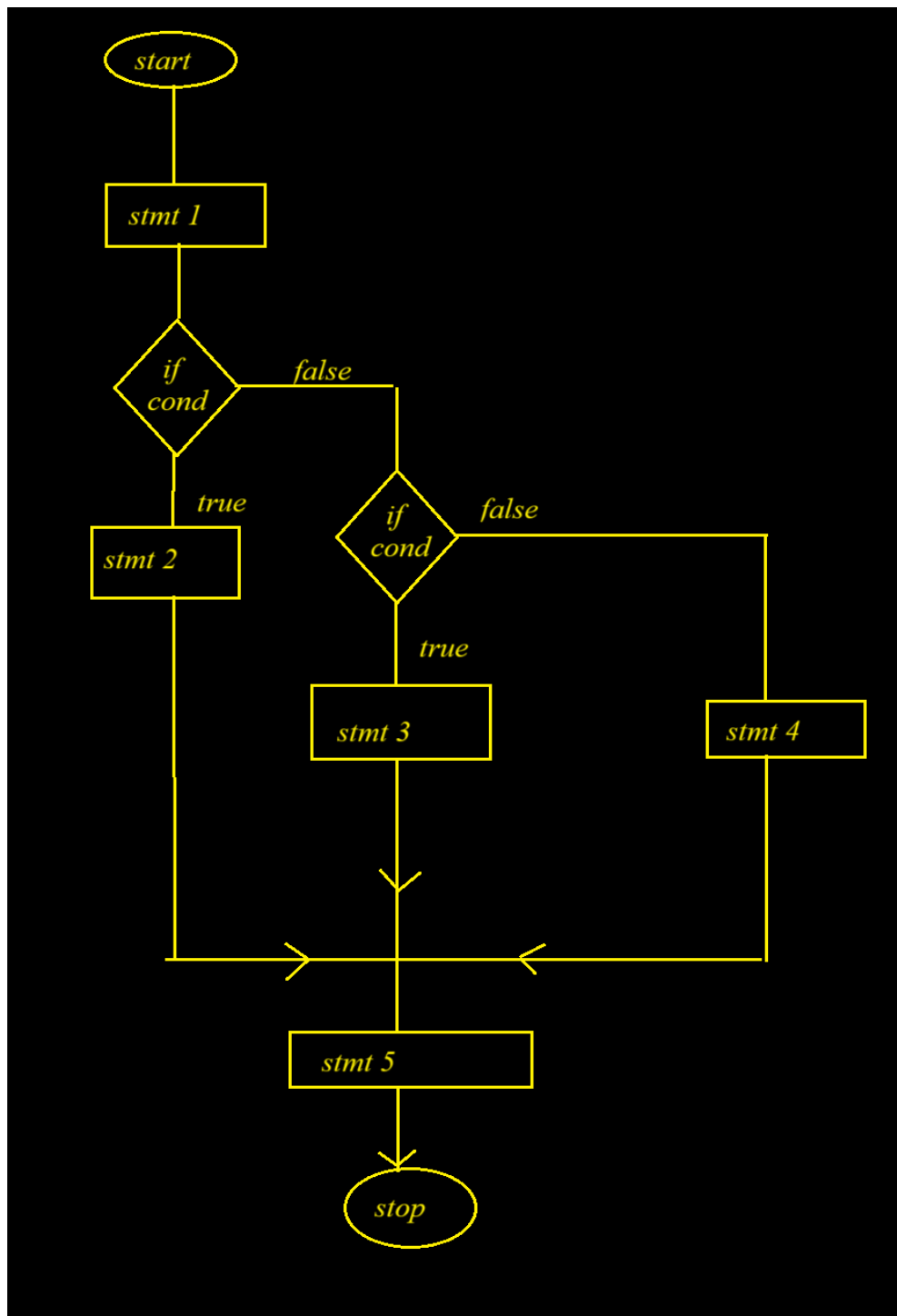
else if(condition)

```
{  
    Statement;  
}
```

else

```
{  
    Statement;  
}
```

- \* If first checks the if condition if the condition is true then statements present in that block gets executed
- \* If first condition is false then it checks the else if condition if the condition is true the block gets executed
- \* If else if condition is also false then else block gets executed
- \* Else block is the last block that gets executed and it gets executed only when all the conditions are false.



```
1 package DecisionStatements;
2
3 public class Program3 {
4     public static void main(String[] args) {
5         System.out.println("main starts");
6         int a=10;
7         int b=20;
8         if(a==b)
9             System.out.println("a and b are equal");
10        else if(a>b)
11            System.out.println("a is greater");
12        else
13            System.out.println("b is greater");
14
15        System.out.println("main ends");
16    }
17 }
18
```

```
main starts
b is greater
main ends
```

# **Switch**

Syntax

Switch(value/expression/)

```
{  
case value/expression:
```

```
    {  
        Stmts;  
        break;  
    }
```

```
case value/expression:
```

```
    {  
        Stmts;  
        break;  
    }
```

```
.  
.   
.
```

```
default:
```

```
    {  
    }
```

```
}
```

1. Variables are not allowed in cases in switch
2. We can use only byte short int char string datatypes

Other than that we cannot use .

## **Rules**

1. We can pass variables for switch but not for cases.

2. We cannot give condition in switch statements

3. If break is not given after each cases it will execute other cases also.

```
1 package DecisionStatements;
2
3 import java.util.Scanner;
4
5 public class Program4 {
6     public static void main(String[] args) {
7         Scanner sc=new Scanner(System.in);
8         System.out.println("enter the value");
9         int n=sc.nextInt();
10        switch(n)
11        {
12            case 1:
13            {
14                System.out.println("case 1");
15                break;
16            }
17            case 2:
18            {
19                System.out.println("case 2");
20                break;
21            }
22            case 3:
23            {
24                System.out.println("case 3");
25                break;
26            }
27            default:
28            {
29                System.out.println("default");
30            }
31        }
32    }
33 }
```

enter the value

1

case 1

## **Chapter 7 Looping statements**

Loop is a statement in java it helps the programmer to execute a set of instructions repeatedly multiple times

### **Types of loops**

1. While
2. For
3. Do while
4. For each

Case 1:

To print one star

We use

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

To print five stars

We use

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

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

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

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

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

But the same instructions have to be repeated to print the \*

So instead of that we want to write the instructions only once and execute the

Same instructions multiple times



## **While loop**

While: It is a keyword used as a loop statement

### **Syntax:**

```
While(condition)
```

```
    Statements;
```

If only statement is there then braces are optional

```
While(condition)
```

```
{
```

```
    Statement1;
```

```
    Statement2
```

```
    .
```

```
    .
```

```
    Statement n;
```

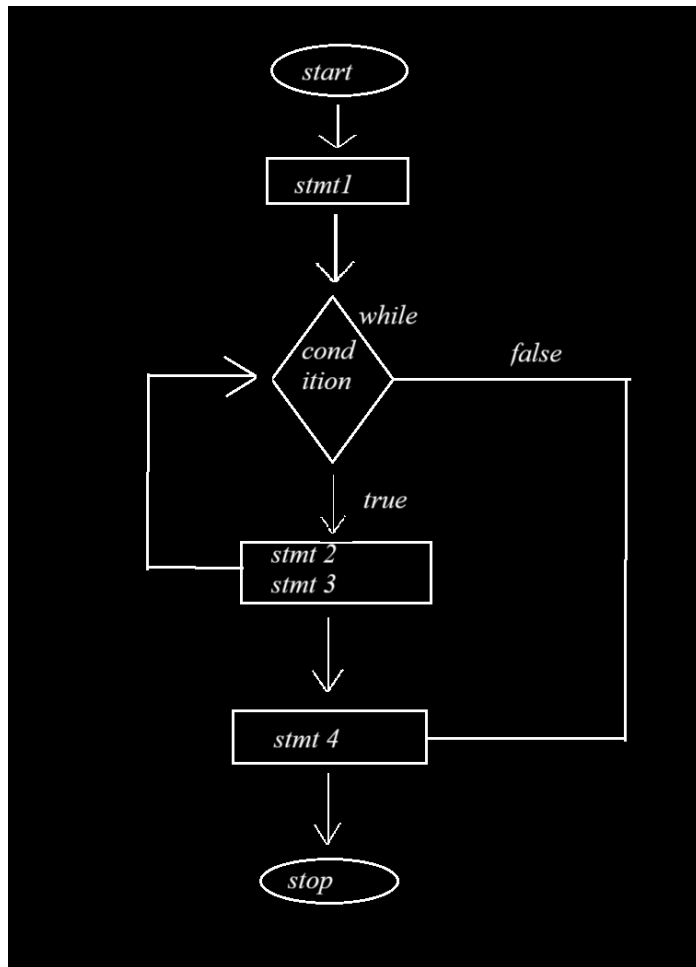
```
}
```

Condition is used in while statement to control the no of repeatations

If condition is true loop block will be executed multiple times

If condition is false loop block will be stopped.

### **Flow Diagram:**



```
1 package Loopingstatements;
2
3 public class Program1 {
4     public static void main(String[] args)
5     {
6         int count=0;//intilaization
7         System.out.println("main starts");
8         while(count<5)//condition
9         {
10             System.out.println("*");
11             count++;//updation
12         }
13         System.out.println("main ends");
14     }
15 }
16
```

```
main starts
*
*
*
*
*
main ends
```

## For Loop

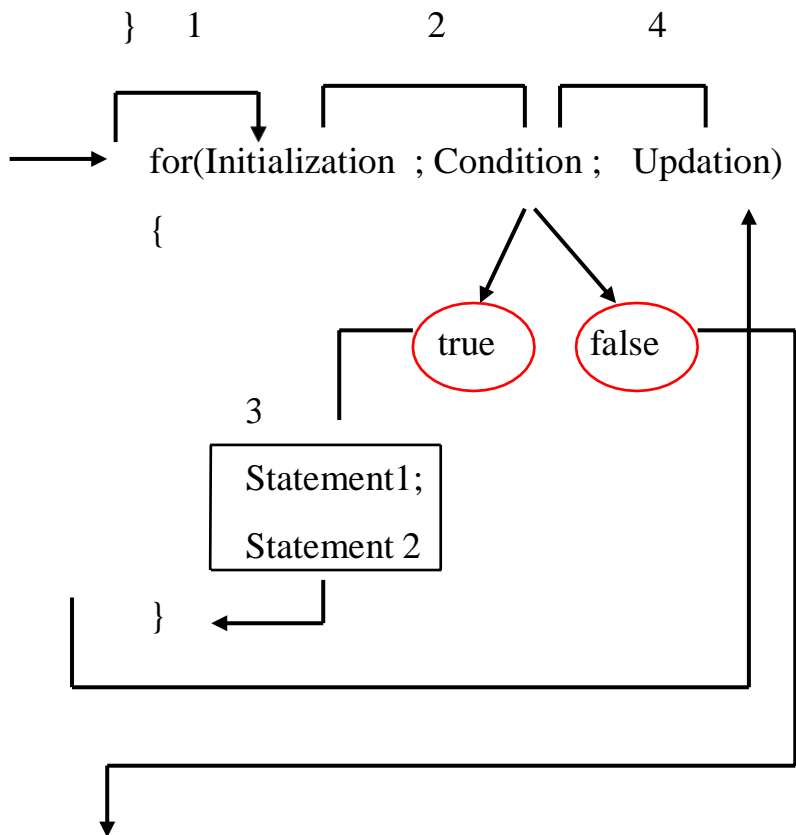
For: it is a keyword used as looping statement

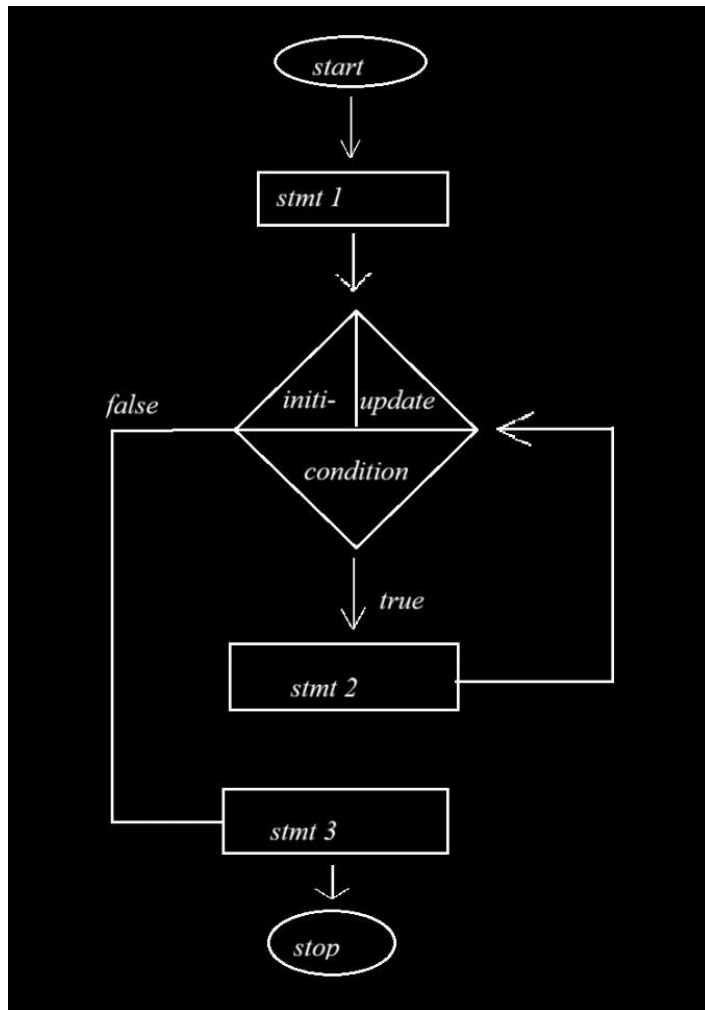
### **Syntax:**

For(initialization; conditions; updation)

```
{  
    Statements 1;  
    Statements 2;  
    .  
    .  
    Statements n;  
}
```

### **Flow Diagram :**





```

1 package Loopingstatements;
2
3 public class Program2 {
4     public static void main(String[] args) {
5         System.out.println("main starts");
6         for(int i=0;i<5;i++)
7         {
8             System.out.println("java");
9         }
10        System.out.println("main ends");
11    }
12 }
13

```

main starts  
java  
java  
java  
java  
java  
main ends

## **do while**

do is a keyword,

While is a keyword we have two keywords

It is a loop statement in java.

1)do

2)while

### **Syntax**

do

{

Statement 1;

Statement 2;

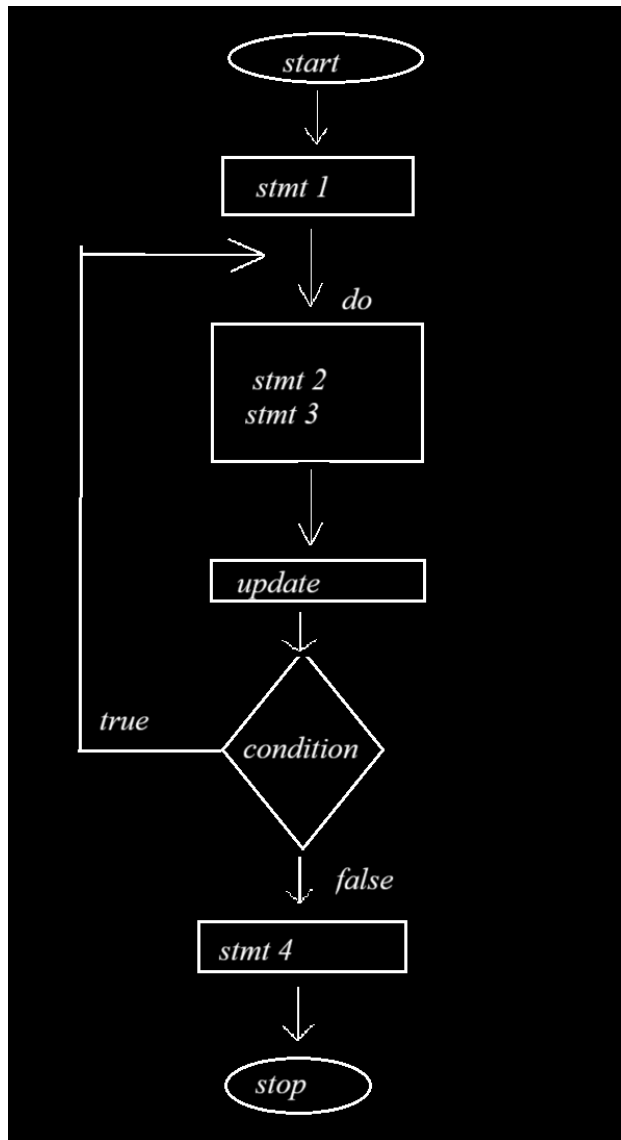
.

.

}

While(condition);

**Flow Diagram:**



```
1 package Loopingstatements;  
2  
3 public class Program3 {  
4     public static void main(String[] args) {  
5         System.out.println("main starts");  
6         int count=0;  
7         do {  
8             System.out.println("java");  
9             System.out.println("jdk 21");  
10            count++;  
11        }while(count<1);  
12        System.out.println("main ends");  
13    }  
14 }
```

```
main starts  
java  
jdk 21  
main ends
```

## Nested Loops

Writing a loop statement inside another loop statement is known as nested loop

```
1 package Loopingstatements;
2
3 public class Program4 {
4     public static void main(String[] args) {
5         System.out.println("main starts");
6         for(int i=0;i<5;i++)
7         {
8             for(int j=0;j<5;j++)
9             {
10                System.out.print(" * ");
11            }
12            System.out.println();
13        }
14        System.out.println("main ends");
15    }
16 }
```

main starts  
\* \* \* \* \*  
\* \* \* \* \*  
\* \* \* \* \*  
\* \* \* \* \*  
main ends

## Chapter 8 Arrays

It is a continuous block of memory which is used to store multiple values of same datatype

Array is a block of memory logically broken in to small blocks of memory. Arrays doesn't have name but it has memory address.

0x1

V1	V2	V3	V4
----	----	----	----

### Characteristics of array

1) Arrays are fixed in size.

Once the array is created with specific size it cannot be increased or decreased.

Size=4

10	150	200	350
----	-----	-----	-----

Size=6

10	20	85	96	102	550
----	----	----	----	-----	-----

2) It is a homogenous collection of data.

**Note:** we provide datatype at the declaration type.

10	20		
----	----	--	--



Int

int

double ✗

When we want to display what type of data to be stored in Array.

Then at the time of declaration we need to specify the datatype.

3) To access the data present in the array

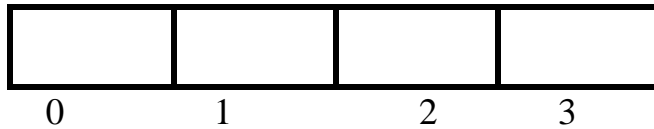
We need two things



1) **Array address/reference**: address of array.

2) **index** : It is a positive integer which always starts from 0 up to size -1.

0X1



## To create array using reference variables

To store or fetch the data from the array reference is required

To store reference we need variables

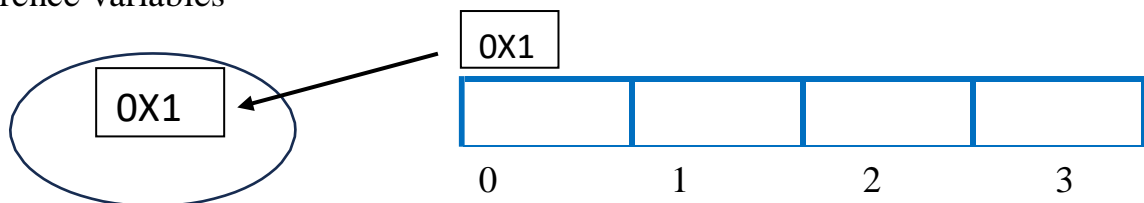
Syntax;

Datatype[ ] identifiers;

Datatype identifiers[ ];

Array Reference Variables

Variables which is used to store the reference of an array is known as array reference variables



## To create an array using new

**Syntax:**

**new datatype[size]**

`new int[5];`

0	0	0	0
---	---	---	---

new is an operator which is used to perform some operation

new operator returns the reference of array.

It creates an array of given type for the given size

Initialize with the default value.

Default values depends on the datatypes

It returns the reference of array.

## Syntax to create an array

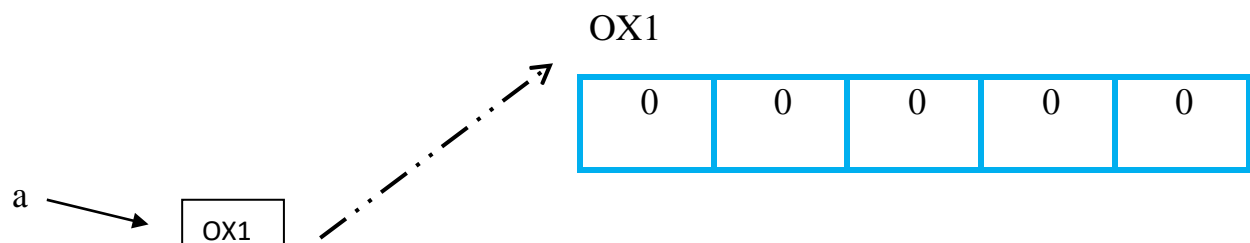
**Datatype[ ] variable\_name=new Datatype[size];**

Ex

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

A variables will be created of name a

And next an array object will be created of size(5) then new operator returns the address of the object created and will be stored in the reference variable.



Program

```
class Program1
{
    public static void main(String[] args)
    {
        int[] a = new int[5];
        System.out.println(a);
    }
}
```

## Length of array

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

In java to find the size of the array we have variable called as length

**Length** is a variable which stores the size of the array

For every array we declare in array we have a variable called as length.

### Syntax

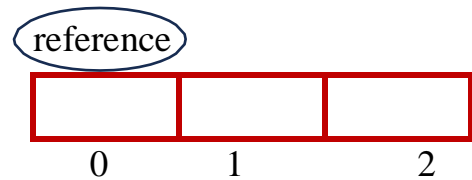
**Array\_reference.length**

```
class Program1
{
    public static void main(String[] args)
    {
        int[] a = new int[5];
        System.out.println(a.length);
    }
}
```

## To store and access array

### Syntax to store

*Array\_reference[index]=value/variable/expression;*



Ex: `int[] a=new int[5];`

`a[0]=10;`

`a[1]=10+20;`

`int b=30;`

`a[2]=b;`

`a[3]=b+20;`

`a[4]=70;`

## To access or fetch data from array

To fetch data from array

### Syntax

*Array\_index[index];*

To access or fetch the nth elements present in array

### Syntax

`Array_ref[n-1];`

To print 2<sup>nd</sup> element

`System.out.println(a[1]);`

Find the sum of first and last element

`Int sum=a[0]+a[1];`

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

## **Array Index Out Of Bounds Exception**

```
S.o.pln(a[0]); //10
```

```
S.o.pln(a[1]); //10
```

```
S.o.pln(a[2]); //10
```

```
S.o.pln(a[3]); ➔ R.T.E
```

0X1	
0	10
1	20
2	30

Compiler will only check whether the syntax and semantics are proper or not.

Only if the syntax is wrong it will give you the error.

But during execution the index 4 is not present inside array which is beyond the range of array hence we get array out of bounds exception.

`ArrayIndexOutOfBoundsException`.

We get this exception in 3 scenarios

1) if index < 0

2) if index > 0

3) if index == length

## **Create an array without new keyword.**

Declaration and initialization statement.

Syntax:

```
Datatype[ ] variables = { values1, values2, .....value n};
```

This type of array should be created only when the data to be entered is known by user.

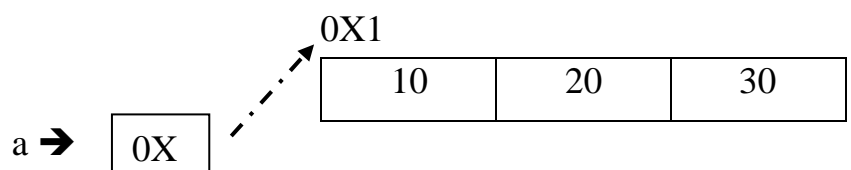
Here size is not mentioned but the no of values entered becomes the size of array.

Program

```
Class Demo{  
    Public static void main(String[] args)  
    {  
        int[] a={10,20,30};  
        System.out.println(a);  
        System.out.println(a.length);  
    }  
}
```

## **To access array elements using loop statements**

```
int[] a={10,20,30};  
System.out.println(a[0]);  
System.out.println(a[1]);  
System.out.println(a[2]);
```



```

class Program1
{
public static void main(String[] args)
    {
        int[] a={ 10,20,30,40};
        //printing all elements at once
        for(int i=0;i<a.length;i++)
            {
                System.out.println(a[i])
            }
    }
}

```

## **Dynamic initialization of array**

Until now the programmer used to give the values and size of the array

Not we take the sizes and data from the user

We call this as Dynamic initialization of array.

We can take dynamic values from users using Scanner class.

eg

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

```

public class Demo {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
        System.out.println("enter the size of array");
        int[] arr=new int[sc.nextInt()];
        //insert elements into array
        for(int i=0;i<arr.length;i++)
            {
                System.out.println("enter the "+i+" value");
                a[i]=sc.nextInt();
            }
    }
}

```

```

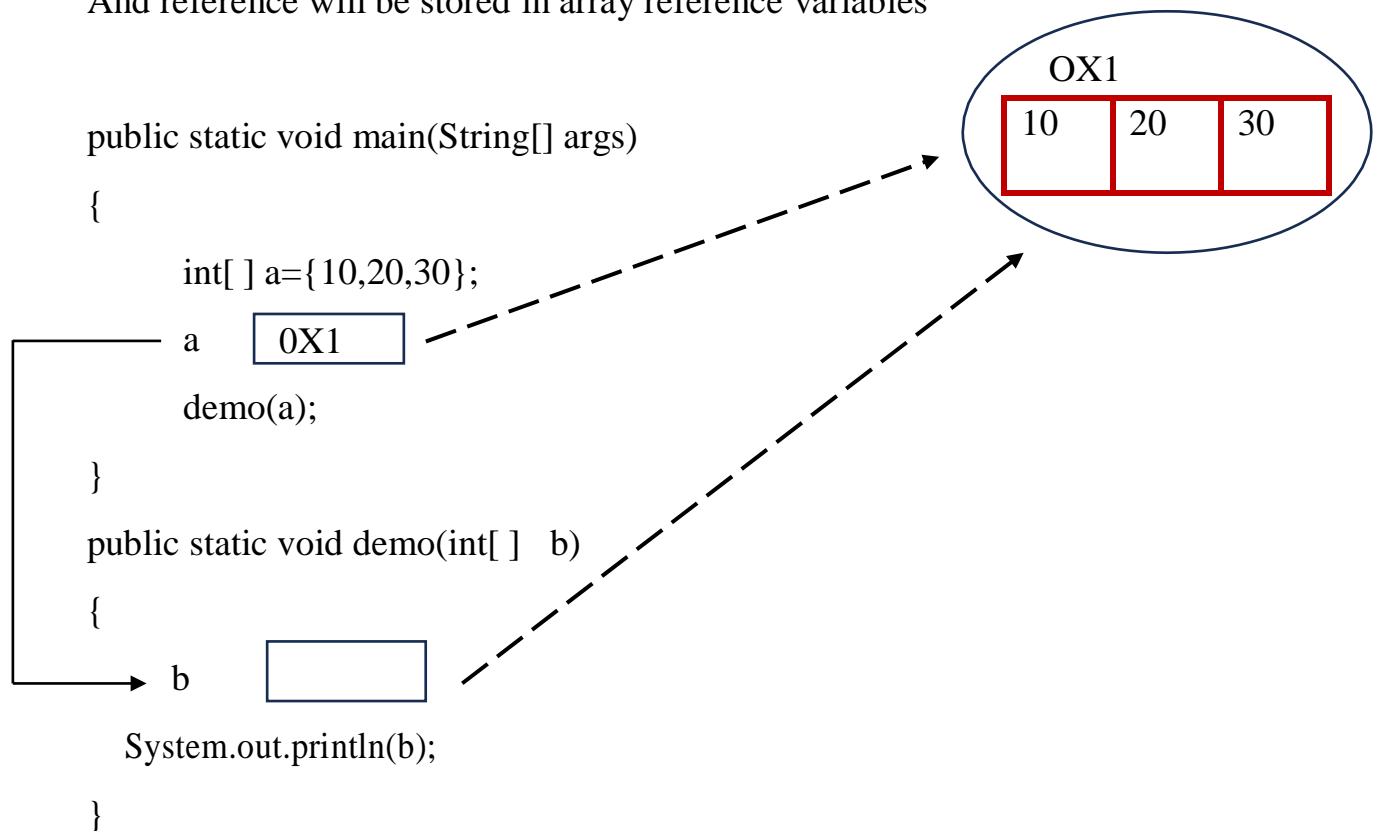
    }
    //display the elements
    System.out.println("elements in array are");
    for(int i=0;i<arr.length;i++)
    {
        System.out.print(arr[i]+" ");
    }
}

```

## To design a method which can receive array

Whenever we create an array a block of memory will be created in heap area

And reference will be stored in array reference variables



Now the address in variable a is copied to variable b which is present in demo( )

In this program the variable a holds the address of the created array so

If the demo( ) wants to access the array then it must have the reference

But the reference is stored in a variable which is present in local block called main method



So we should design a method which can accept the array reference as the arguments only then we can pass the reference of the array to other method

Here we have designed a method called as demo which has formal arguments of `int[]` array type so this method can accept the array reference of `int[]` array.