Introduction to Jamva

[Ig @corporatebilla]

I hope my notes will add some value to your life 😸

**Happy Coding **

Note: Some resources are taken from CodeWithHarry

Love u Harry Bhai

- Java is one of the most popular programming languages because it is used in various tech fields like app development, web development, client-server applications, etc.
- Java is an object-oriented programming language developed by Sun Microsystems of the USA in 1991.
- It was originally called Oak by James Goslin. He was one of the inventors of Java.
- Java = Purely Object-Oriented.

Working of Jamva

- The source code in Java is first compiled into the bytecode.
- Then the Java Virtual Machine(JVM) compiles the bytecode to the machine code.

- JDK – Java Development Kit = Collection of tools used for developing and running java programs.
- JRE – Java Runtime Environment = Helps in executing programs developed in JAVA.

Basic Structure of a Jamva Program: Understanding our First Jamva Hemlo World Program

```
package com.company; // Groups classes
public class Main{  // Entrypoint into the application
    public static void main(String[]args){
        System.out.println(“Hello World”);
    }
}
```

Working of the "Hello World" program shown above :

1. package com.company :
    - Packages are used to group the related classes.

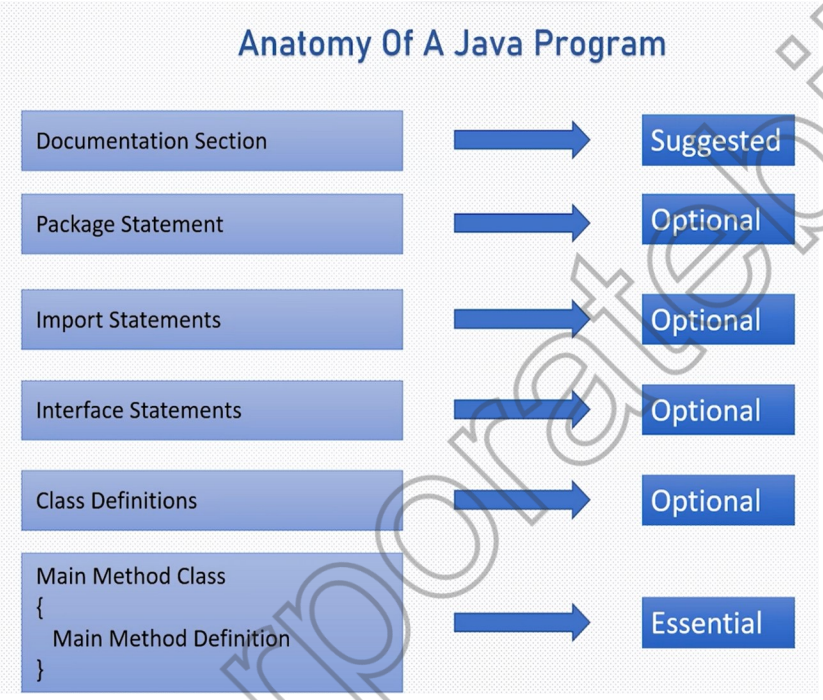
- The "Package" keyword is used to create packages in Java.
- Here, com.company is the name of our package.

2. public class Main :
   - In Java, every program must contain a class.
   - The filename and name of the class should be the same.
   - Here, we've created a class named "Main".
   - It is the entry point to the application.
3. public static void main(String[]args){..} :
   - This is the main() method of our Java program.
   - Every Java program must contain the main() method.
4. System.out.println("Hello World"):
   - The above code is used to display the output on the screen.
   - Anything passed inside the inverted commas is printed on the screen as plain text.

Naming Conventions

- For classes, we use PascalConvention. The first and Subsequent characters from a word are capital letters (uppercase).
  Example: Main, MyScanner, MyEmployee, CodeWithHarry
- For functions and variables, we use camelCaseConvention. Here the first character is lowercase, and the

subsequent characters are uppercase like myScanner, myMarks, CodeWithHarry



Variables and Damta Types

Just like we have some rules that we follow to speak English (the grammar), we have some rules to follow while writing a Java program. This set of these rules is called syntax. It's like Vocabulary and Grammar of Java.

Variables

- A variable is a container that stores a value.
- This value can be changed during the execution of the program.
- Example: int number = 8; (Here, int is a data type, the number is the variable name, and 8 is the value it contains/stores).

- Rules for declaring a variable name:
  - Must not begin with a digit. (E.g., 1arry is an invalid variable)
  - The name is case-sensitive. (Harry and harry are different)
  - Should not be a keyword (like Void).
  - White space is not allowed. (int Code With Harry is invalid)
  - Can contain alphabets, $character, _character, and digits if the other conditions are met.

Reserved keymwords in Jamva

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default

double	do	else	enum	extends	false
final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long
native	new	null	package	private	protected
public	return	short	static	strictfp	super
switch	synchronized	this	throw	throws	transient
true	try	void	volatile	while	

Damta Types

Data types in Java fall under the following categories

1. Primitive Data Types (Intrinsic)
2. Non-Primitive Data Types (Derived)

Primitive Damta Types

Jamva is statically typed, i.e., variables must be declared before use.
Java supports 8 primitive data types:

Data Type	Size	Value Range
1. Byte	1 byte	-128 to 127
2. short	1 byte	-32,768 to 32,767
3. int	2 byte	-2,147,483,648 to 2,147,483,647
4. float	4 byte	$3.40282347 \times 10^{38}$ to $1.40239846 \times 10^{-45}$
5. long	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
6. double	8 byte	$1.7976931348623157 \times 10^{308}$, $4.9406564584124654 \times 10^{-324}$
7. char	2 byte	0 to 65,535
8. boolean	Depends on JVM	True or False

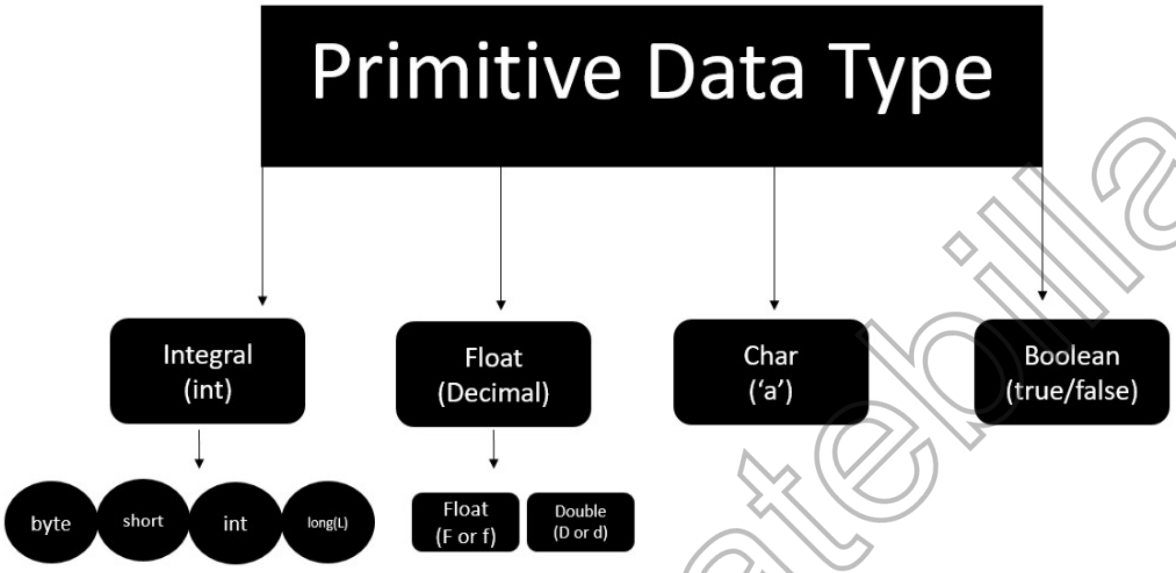
Primitive Type Keyword

FacingIssuesO

Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined*	true or false	false

How to choose damta types for our variables

In order to choose the data type, we first need to find the type of data we want to store. After that, we need to analyze the min & max value we might use.



Literals

A constant value that can be assigned to the variable is called as a literal.

- 101 – Integer literal
- 10.1f – float literal
- 10.1 – double literal (default type for decimals)
- 'A' – character literal
- true – Boolean literal
- "Harry" – String literal

Gemtting user Input

Reading data from the Keyboard :

- Scanner class of java.util package is used to take input from the user's keyboard.The Scanner class has many methods for taking input from the user depending upon the type of input. To use any of the methods of the Scanner class, first, we need to create an object of the Scanner class as shown in the below example :

```
import java.util.Scanner;  // Importing  the Scanner class
Scanner sc = new Scanner(System.in);  //Creating an object named "sc" of the Scanner class.
int a = S.nextInt();  //(Method to read from the keyboard)
```

Note: we use nextLine() to take string as input

If we use next() then as default it'll take string as input

Omperators

Types of operators:

- Arithmetic operators
  - Arithmetic operators are used to perform mathematical operations such as addition, division, etc on expressions.
  - Arithmetic operators cannot work with Booleans.
  - % operator can work on floats and doubles.
- Comparison operators

- As the name suggests, these operators are used to compare two operands.

- Logical operators
  - These operators determine the logic in an expression containing two or more values or variables.
- Bitwise operators
  - These operators perform the operations on every bit of a number.

```java
package com.company;

public class CWH_Ch2_Operators {
    public static void main(String[] args) {
        // 1. Arithmetic Operators
        int a = 4;
        // int b = 6 % a; // Modulo Operator
        // 4.8%1.1 --> Returns Decimal Remainder

        // 2. Assignment Operators
        int b = 9;
        b *= 3;
        System.out.println(b);

        // 3. Comparison Operators
        // System.out.println(64<6);

        // 4. Logical Operators
        // System.out.println(64>5 && 64>98);
        System.out.println(64>5 || 64>98);

        // 5. Bitwise Operators
        System.out.println(2&3);
        //        10
        //        11
        //        ----
        //        10
    }
}
```

Precedence and Associavity

Precedence: The operators are applied and evaluated based on precedence. For example, (+, -) has less precedence compared to (*, /). Hence * and / are evaluated first.

In case we like to change this order, we use parenthesis ().

Associativity: tells the direction of the execution of operators. It can either be left to right or vice versa.

/ * -> L to R

+ - -> L to R

++, = -> R to L

	Operator	Associativity	Precedence
()	Function call	Left-to-Right	Highest 14
[]	Array subscript		
.	Dot (Member of structure)		
->	Arrow (Member of structure)		
!	Logical NOT	Right-to-Left	13
~	One's-complement		
-	Unary minus (Negation)		
++	Increment		
--	Decrement		
&	Address-of		
*	Indirection		
(type)	Cast		
sizeof	Sizeof		
*	Multiplication	Left-to-Right	12
/	Division		
%	Modulus (Remainder)		
+	Addition	Left-to-Right	11
-	Subtraction		
<<	Left-shift	Left-to-Right	10
>>	Right-shift		
<	Less than	Left-to-Right	8
<=	Less than or equal to		
>	Greater than		
>=	Greater than or equal to		
==	Equal to	Left-to-Right	8
!=	Not equal to		
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
|	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
||	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, +=	Assignment operators	Right-to-Left	1
*=, etc.			
,	Comma	Left-to-Right	Lowest 0

Resulting data type after arithmetic operation

- Result = byte + short -> integer
- Result = short + integer -> integer
- Result = long + float -> float
- Result = integer + float -> float
- Result = character + integer -> integer
- Result = character + short -> integer
- Result = long + double -> double
- Result = float + double -> double

Increment and Decrement operators

- a++, ++a (Increment Operators)
- a--, --a (Decrement Operators)

These will operate on all data types except Booleans.
Quick Quiz: Try increment and decrement operators on a Java variable

- a++ -> first use the value and then increment
- ++a -> first increment the value then use it

Strings

- A string is a sequence of characters.
- Strings are objects that represent a char array. For example :

```
char[] str = {'H','A','R','R','Y'};
String s = new String(str);
```

is same as :

```
String s = "Harry";
```

- Strings are immutable and cannot be changed.
- java.lang.String class is used to create a String object.
- The string is a class but can be used as a data type.
  Syntax of strings in Java :

```
String <String_name> = "<sequence_of_string>";
```

Example :

```
 String str = "CodeWithHarry";
```

In the above example, str is a reference, and "CodeWithHarrt" is an object.

Different ways to create a string in Java :

In Java, strings can be created in two ways :

1. By using string literal
2. By using the new

Creating String using String literal :

```
String s1= "String literal"
```

We use double quotes("") to create string using string literal. Before creating a new string instance, JVM verifies if the same string is already present in the string pool or not. If it is already present, then JVM returns a reference to the pooled instance otherwise, a new string instance is created.

Creating Stri

```
String s=new S
```

When we create a

- System.out.format()

```
System.out.printf("%c",ch)
```

- %d for int
- %f for float
- %c for char
- %s for string

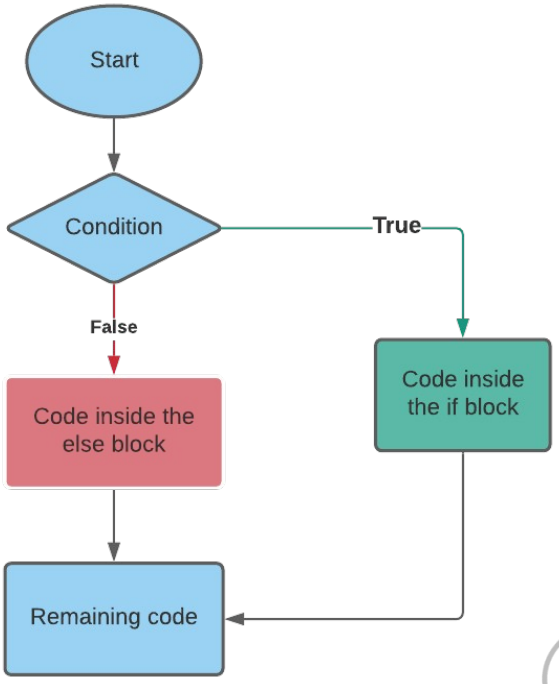
Strig Methods

String Methods operate on Java Strings. They can be used to find the length of the string, convert to lowercase, etc.

Method	Description
1. length()	Returns the length of String name. (5 in this case)
2. toLowerCase()	Converts all the characters of the string to the lower case letters.
3. toUpperCase()	Converts all the characters of the string to the upper case letters.
4. trim()	Returns a new String after removing all the leading and trailing spaces from the original string.
5. substring(int start)	Returns a substring from start to the end. Substring(3) returns "ry". [Note that indexing starts from 0]
6. substring(int start, int end)	Returns a substring from the start index to the end index. The start index is included, and the end is excluded.
7. replace('r', 'p')	Returns a new string after replacing r with p. Happy is returned in this case. (This method takes char as argument)
8. startsWith("Ha")	Returns true if the name starts with the string "Ha". (True in this case)
9. endsWith("ry")	Returns true if the name ends with the string "ry". (True in this case)
10. charAt(2)	Returns the character at a given index position. (r in this case)
11. indexOf("s")	Returns the index of the first occurrence of the specified character in the given string.
12. lastIndexOf("r")	Returns the last index of the specified character from the given string. (3 in this case)
13. equals("Harry")	Returns true if the given string is equal to "Harry" false otherwise [Case sensitive]
14.equalsIgnoreCase("harry")	Returns true if two strings are equal, ignoring the case of characters.

Escape Sequence Characters :

- The sequence of characters after backslash '\' = Escape Sequence Characters
- Escape Sequence Characters consist of more than one character but represent one character when used within the strings.
- Examples: \n (newline), \t (tab), \' (single quote), \\ (backslash), etc.



If-else Statement

Decision-making instructions in Java

- If-Else Statement
- Switch Statement

If-Else Statement

Syntax of If-else statement in Java :

```
/* if (condition-to-be-checked) {
    statements-if-condition-true;
}
else {
    statements-if-condition-false;
} */
```

```
/* if (condition1) {

            //Statements;

else if {

            // Statements;

}

else {

            //Statements

} */
```

If-else ladder :

- Instead of using multiple if statements, we can also use else if along with if thus forming an if-else-if-else ladder.
- Using such kind of logic reduces indents.
- Last else is executed only if all the conditions fail.

Relational and Logical Operators

Relational Operators in Java :

Relational operators are used to evaluate conditions (true or false) inside the if statements. Some examples of relational operators are:

- == (equals)
- >= (greater than or equals to)
- > (greater than)
- < (less than)
- <= (less than or equals to)
- != (not equals)

Note: '=' is used for an assignment whereas '==' is used for equality check. The condition can be either true or false.

Logical Operators :

- Logical operators are used to provide logic to our Java programs.
- There are three types of logical operators in Java :
- && - AND
- || - OR
- ! – NOT

AND Operator :

Evaluates to true if both the conditions are true.

- Y && Y = Y
- Y && N = N
- N && Y = N
- N && N = N

Convention: # Y – True and N - False

OR Operator :

Evaluates to true when at least one of the conditions is true.

- Y || Y = Y
- Y || N = Y
- N || Y = Y
- N || N = N

Convention: # Y – True and N - False

NOT Operator :

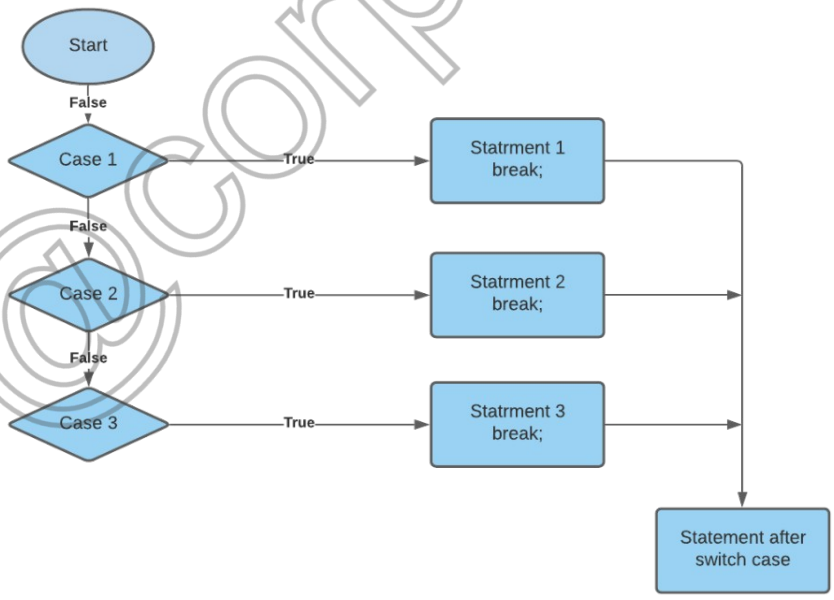
Negates the given logic (true becomes false and vice-versa)

- !Y = N
- !N = Y

Switch Case

Switch Case-Control Instruction

- Switch-Case is used when we have to make a choice between the number of alternatives for a given variable.
- Var can be an integer, character, or string in Java.
- Every switch case must contain a default case. The default case is executed when all the other cases are false.
- Never forget to include the break statement after every switch case otherwise the switch case will not terminate.



```
switch (var) {
    case "Shubham" -> {
        System.out.println("You are going to become an Adult!");
        System.out.println("You are going to become an Adult!");
        System.out.println("You are going to become an Adult!");
    }
    case "Saurabh" -> System.out.println("You are going to join a Job!");
    case "Vishaka" -> System.out.println("You are going to get retired!");
    default -> System.out.println("Enjoy Your life!");
}
System.out.println("Thanks for using my Java Code!");
```

Loops

- In programming languages, loops are used to execute a particular statement/set of instructions again and again.
- The execution of the loop starts when some conditions become true.
- For example, print 1 to 1000, print multiplication table of 7, etc.
- Loops make it easy for us to tell the computer that a given set of instructions need to be executed repeatedly.

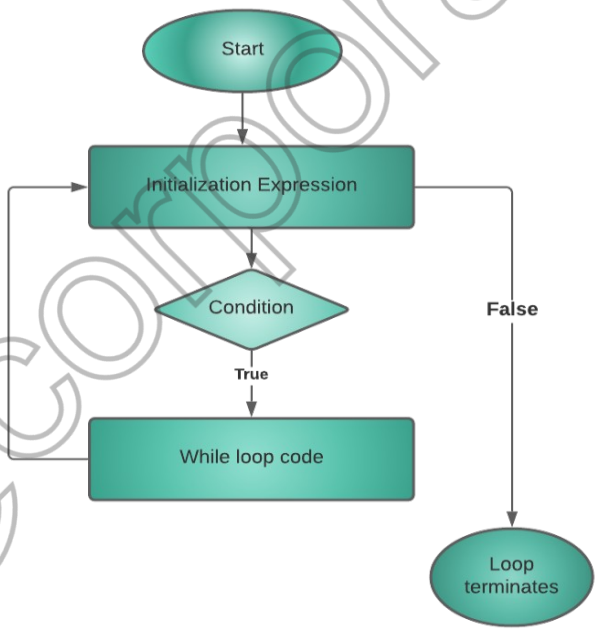
Types of Loops :

Primarily, there are three types of loops in Java:

1. While loop
2. do-while loop
3. for loop

While loops :

- The while loop in Java is used when we need to execute a block of code again and again based on a given boolean condition.
- Use a while loop if the exact number of iterations is not known.
- If the condition never becomes false, the while loop keeps getting executed. Such a loop is known as an infinite loop.



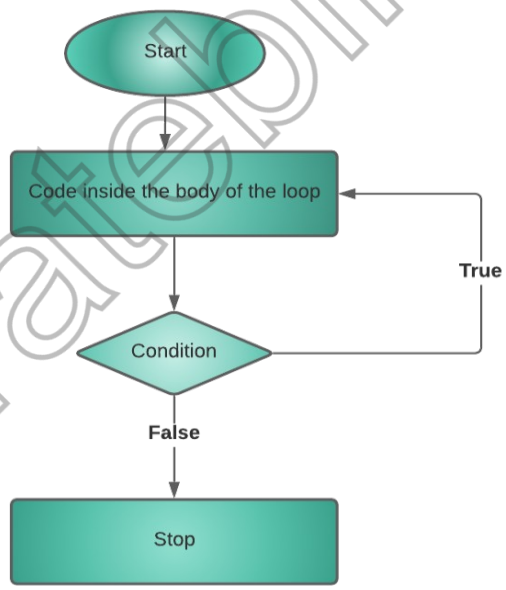
Do-while loop:

- Do- while loop is similar to a while loop except for the

fact that it is guaranteed to execute at least once.

- Use a do-while loop when the exact number of iterations is unknown, but you need to execute a code block at least once.
- After executing a part of a program for once, the rest of the code gets executed on the basis of a given boolean condition.

```
int i=1;
do{
System.out.println(i);
i++;
}while(i<=10);
```

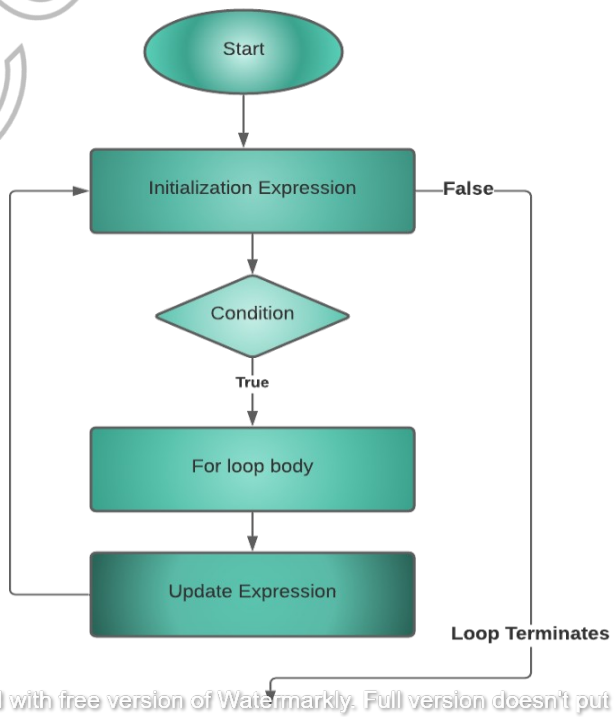


Difference Between while loop and do-while loop :

- while – checks the condition & executes the code.
- do-while – executes the code at least once and then checks the condition. Because of this reason, the code in the do-while loop executes at least once, even if the condition fails.

For loop:

- For loop in java is used to iterate a block of code multiple times.
- Use for loop only when the exact number of iterations needed is already known to you.
- Initializer: Initializes the value of a variable. This part is executed only once.
- check_bool_expression: The code inside the for loop is executed only when this condition returns true.
- update: Updates the value of the initial variable.



Break and Continue

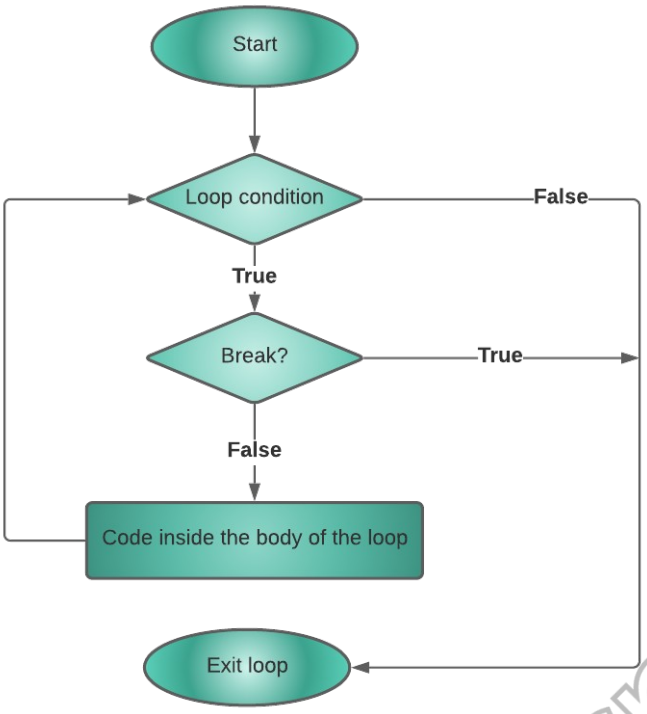
Break statement :

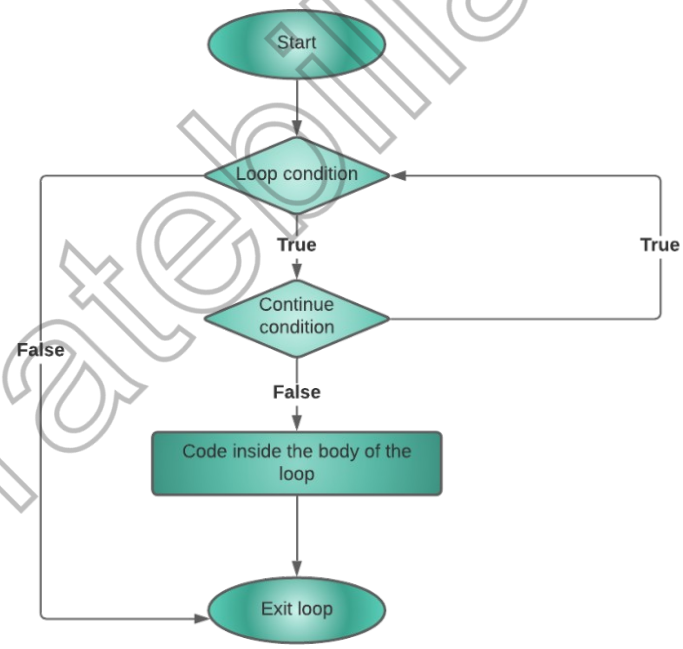
- The break statement is used to exit the loop irrespective of whether the condition is true or false.
- Whenever a 'break' is encountered inside the loop, the control is sent outside the loop.

Continue statement :

- The continue statement is used to immediately move to the next iteration of the loop.

- The control is taken to the next iteration thus skipping everything below 'continue' inside the loop for that iteration.



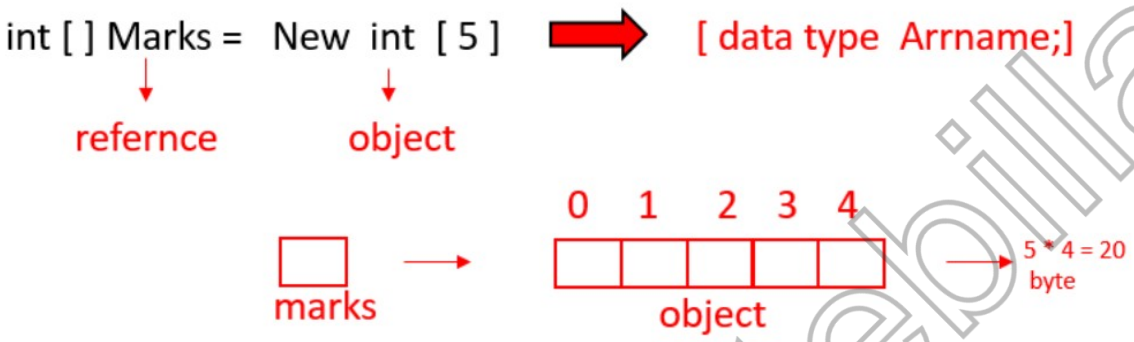


Arrays

- An array is a collection of similar types of data having contiguous memory allocation.
- The indexing of the array starts from 0., i.e 1st element will be stored at the 0th index, 2nd element at 1st index, 3rd at 2nd index, and so on.
- The size of the array can not be increased at run time therefore we can store only a fixed size of elements in

array.

- Use Case: Storing marks of 5 students



Accessing Array Elements :

Array elements can be accessed as follows,

```
/* marks[0] = 100       //Note that index starts from 0
marks[1] = 70
.
.
marks[4] = 98 */
```

So in a nut shell, this is how array works:

1. int[] marks; //Declaration!
2. marks = new int[5]; //Memory allocation!

2. int[] marks = new int[5]; //Declaration + Memory allocation!
3. int[] marks = {100,70,80,71,98} // Declare + Initialize!

Note : Array indices start from 0 and go till (n-1) where n is the size of the array.

```
marks.length //Gives 5 if marks is a reference to an array with 5 elements
```

Displaying an Array :

An array can be displayed using a for loop:

```
for (int i=0; i<marks.length; i++)
{
        Sout(marks[i]);    //Array Traversal
}
```

For Each Loop in Java

- For each loop is an enhanced version of for loop.
- It travels each element of the data structure one by one.
- Note that you can not skip any element in for loop and it is also not possible to traverse elements in reverse order with the help of for each loop.
- It increases the readability of the code.
- If you just want to simply traverse an array from start to end then it is recommended to use for each loop.

```
/* for (int element:Arr) {
          Sout(element);    //Prints all the elements
} */
```

Syntax :

Example :

```
class CWH_forEachLoop{
  public static void main(String args[]){
   //declaring an array
   int arr[]={1,2,3,3,4,5};
   //traversing the array with for-each loop
   for(int i:arr){
     System.out.println(i);
   }
 }
}
```


Output :

```java
// Displaying the Array (for loop)
System.out.println("Printing using for loop");
for(int i=0;i<marks.length;i++){
    System.out.println(marks[i]);
}

// Quick Quiz: Displaying the Array in Reverse order (for loop)
System.out.println("Printing using for loop in reverse order");
for(int i=marks.length -1;i>=0;i--){
    System.out.println(marks[i]);
}

// Quick Quiz: Displaying the Array (for-each loop)
System.out.println("Printing using for-each loop");
for(int element: marks){
    System.out.println(element);
}
```

Multidimensional 2-D Array

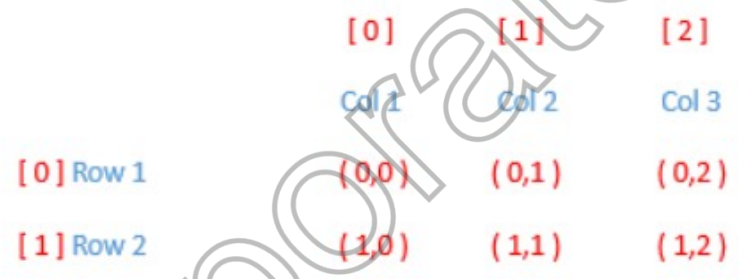
A 2-D array can be created as follows:

```
int [][] flats = new int[2][3]          //A 2-D array of 2 rows + 3 columns
```

We can add elements to this array as follows

```
flats[0][0] = 100
flats[0][1] = 101
flats[0][2] = 102
// … & so on!
```

This 2-D array can be visualized as follows:



Methods

- Sometimes our program grows in size, and we want to separate the logic of the main method from the other methods.

- For instance, if we calculate the average of a number pair 5 times, we can use methods to avoid repeating the logic. [DRY – Don't Repeat Yourself]

Syntax of a Method

A method is a function written inside a class. Since Java is an object-oriented language, we need to write the method inside some class.

Syntax of a method :

```
returnType nameOfMethod() {
//Method body
}
```

The following method returns the sum of two numbers

```
int mySum(int a, int b) {
    int c = a+b;
    return c;   //Return value
}
```

- In the above method, int is the return data type of the mySum function.
- mySum takes two parameters: int a and int b.
- The sum of two values integer values(a and b) is stored in another integer value named 'c'.
- mySum returns c.

Calling a Method :

A method can be called by creating an object of the class in which the method exists followed by the method call:

```
Calc obj = new Calc(); //Object Creation

obj.mySum(a , b); //Method call upon an object
```

The values from the method call (a and b) are copied to the a and b of the function mySum. Thus even if we modify the values a and b inside the method, the values in the main method will not change.

Void return type :

When we don't want our method to return anything, we use void as the return type.

Static keyword :

- The static keyword is used to associate a method of a given class with the class rather than the object.
- You can call a static method without creating an instance of the class.
- In Java, the main() method is static, so that JVM can call the main() method directly without allocating any extra memory for object creation.
- All the objects share the static method in a class.

Process of method invocation in Java :

Consider the method Sum of the calculate class as given in the below code :

```
class calculate{
     int sum(int a,int b){
        return a+b;
    }
```

The method is called like this:

```
class calculate{
     int sum(int a,int b){
        return a+b;
    }

public static void main(String[] args) {

        calculate obj = new calculate();
        int c = obj.sum(5,4);
        System.out.println(c);
    }
}
```

Output :

```
9
```

- Inside the main() method, we've created an object of the calculate class.
- obj is the name of the calculate class.
- Then, we've invoked the sum method and passed 5 and 4 as arguments.

Note: In the case of Arrays, the reference is passed. The same is the case for object passing to methods.

Java Tutorial: Method Overloading in Java

- In Java, it is possible for a class to contain two or more methods with the same name but with different parameters. Such methods are called Overloaded methods.
- Method overloading is used to increase the readability of the program.

```
void foo()
void foo(int a) //Overloaded function foo
int foo(int a, int b)
```

Ways to perform method overloading :

In Java, method overloading can be performed by two ways listed below :

1. By changing the return type of the different methods
2. By changing the number of arguments accepted by the method

Now, let's have an example to understand the above ways of method overloading :

a. By changing the return type :

- In the below example, we've created a class named calculate.
- In the calculate class, we've two methods with the same name i.e. multiply
- These two methods are overloaded because they have the same name but their return is different.
- The return type of 1st method is int while the return type of the other method is double.

```
class calculate{
     int multiply(int a,int b){
        return a*b;
    }
    double multiply(double a,double b){
         return  a*b;
    }

public static void main(String[] args) {

        calculate obj = new calculate();
        int c = obj.multiply(5,4);
        double d = obj.multiply(5.1,4.2);
        System.out.println("Mutiply method : returns integer : " + c);
        System.out.println("Mutiply method : returns double : " +  d);

}
}
```

Output :

```
Mutiply method : returns integer : 20
Mutiply method : returns double : 21.419999999999998
```

b. By changing the number of arguments passed :

- Again, we've created two methods with the same name i.e. multiply
- The return type of both the methods is int.
- But, the first method 2 arguments and the other method accepts 3 arguments.

Example :

```
class calculate{
     int multiply(int a,int b){
        return a*b;
    }
    int multiply(int a,int b,int c){
         return  a*b*c;
    }

public static void main(String[] args) {

        calculate obj = new calculate();
        int c = obj.multiply(5,4);
        int d = obj.multiply(5,4,3);
        System.out.println(c);
        System.out.println(d);

}
}
```

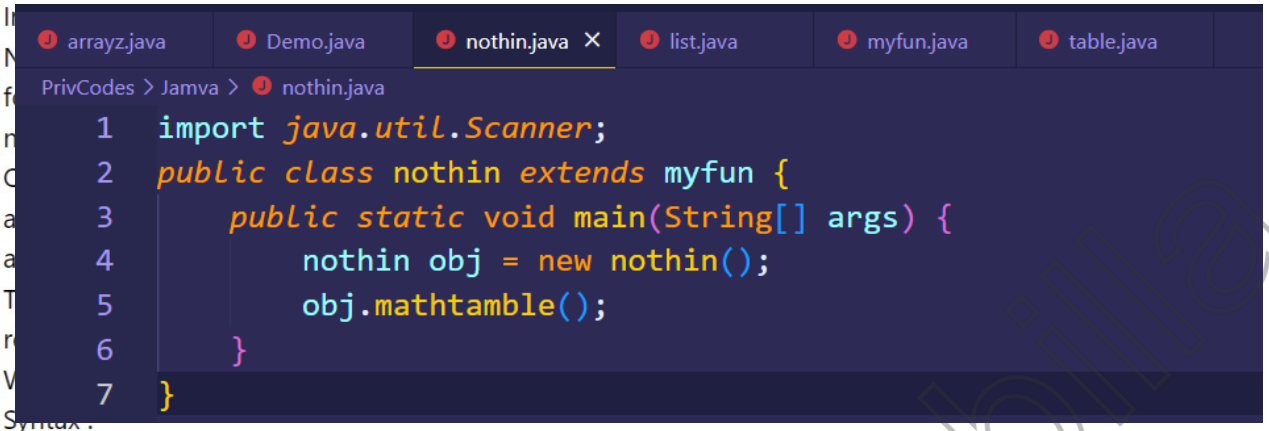
Output :

```
20
60
```

Note: Method overloading cannot be performed by changing the return type of methods.

Java Tutorial: Variable Arguments (VarArgs) in Java

- I
- N
  f
  n
- C
  a
  a
- T
  r
- V



Syntax :

```
/*
public static void foo(int … arr)
{
// arr is available here as int[] arr
}
*/
```

- foo can be called with zero or more arguments like this:
  - foo(7)
  - foo(7,8,9)
  - foo(1,2,7,8,9)

Example of Varargs In Java :

```
class calculate {

    static int add(int ...arr){
        int result = 0;
        for (int a : arr){
            result = result + a;
        }
        return result;
}

public static void main(String[] args){
    System.out.println(add(1,2));
    System.out.println(add(2,3,4));
    System.out.println(add(4,5,6));
}
}
```

Output :

```
3
9
15
```

OOPS

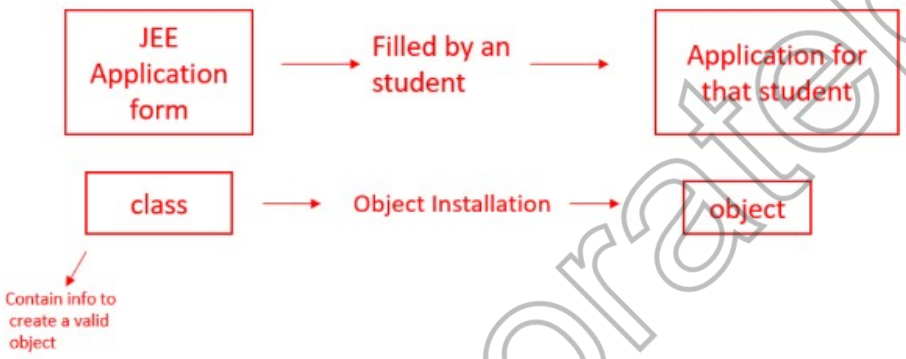
Java Tutorial: Introduction to Object Oriented Programming

- Object-Oriented Programming tries to map code instructions with real-world, making the code short and easier to understand.
- With the help of OOPs, we try to implement real-world entities such as object, inheritance, abstraction, etc.
- OOPs helps us to follow the DRY(Don't Repeat Yourself) approach of programming, which in turn increases the reusability of the code.

Two most important aspects of OOPs - Classes & Objects :

Class :

- A class is a blueprint for creating objects.
- Classes do not consume any space in the memory.
- Objects inherit methods and variables from the class.
- It is a logical component.



Objects :

- An object is an instantiation of a class. When a class is defined, a template (info) is defined.
- Every object has some address, and it occupies some space in the memory.
- It is a physical entity.

Take a look at the below example to get a better understanding of objects and classes :

How to model a problem in OOPs

We identify the following:

- Noun - Class - Employee
- Adjective - Attributes - name, age, salary
- Verb - Methods - getSalary(), increment()

This is all for this tutorial. We will do a detailed discussion on every aspect of OOPs in further tutorials.

Four pillars of Object-Oriented-Programming Language:

1. Abstraction:

- Let's suppose you want to turn on the bulb in your room. What do you do to switch on the bulb. You simply press the button and the light bulb turns on. Right? Notice that here you're only concerned with your final result, i.e., turning on the light bulb. You do not care about the circuit of the bulb or how current flows through the bulb. The point here is that you press the switch, the bulb turns on! You don't know how the bulb turned on/how the circuit is made because all these details are hidden from you. This phenomenon is known as abstraction.
- More formally, data abstraction is the way through which only the essential info is shown to the user, and all the internal details remain hidden from the user.
  Example :

→ Use this phone without bothering about how it was made

2. Polymorphism:

- One entity many forms.
- The word polymorphism comprises two words, poly which means many, and morph, which means forms.
- In OOPs, polymorphism is the property that helps to perform a single task in different ways.
- Let us consider a real-life example of polymorphism. A

woman at the same time can be a mother, wife, sister, daughter, etc. Here, a woman is an entity having different forms.

- Let's take another example, a smartphone can work like a camera as well as like a calculator. So, you can see the a smartphone is an entity having different forms. Also :

Laptop is a single entity with Wifi + Speaker + storage in a single box!

3. Encapsulation:

- The act of putting various components together (in a capsule).
- In java, the variables and methods are the components that are wrapped inside a single unit named class.
- All the methods and variables of a class remain hidden from any other class.
- An automatic cold drink vending machine is an example of encapsulation.
- Cold drinks inside the machine are data that is wrapped inside a single unit cold drink vending machine.

4.Inheritance:

- The act of deriving new things from existing things.
- In Java, one class can acquire all the properties and behaviours of other some other class
- The class which inherits some other class is known as child class or sub class.
- The class which is inherited is known as parent class or super class.
- Inheritance helps us to write more efficient code because it increases the reusablity of the code.
- Example :
- Rickshaw → E-Rickshaw
- Phone → Smart Phone

Java Tutorial: Access modifiers, getters & setters in Java

Access Modifiers

Access Modifiers specify where a property/method is accessible. There are four types of access modifiers in java :

1. private
2. default
3. protected
4. public

Access Modifier	within class	within package	outside package by subclass only	outside package
public	Y	Y	Y	Y
protected	Y	Y	Y	N
Default	Y	Y	N	N
private	Y	N	N	N

From the above table, notice that the private access modifier can only be accessed within the class. So, let's try to access private modifiers outside the class :

```java
class Employee {

    private int id;
    private  String name;

}

public  class CWH {
    public static void main(String[] args) {
        Employee emp1 = new Employee();
        emp1.id = 3;
        emp1.name = "Shubham";

    }
}
```

Output :

```
java: id has private access in Employee
```

You can see that the above code produces an error that we're trying to access a private variable outside the class. So, is there any way by which we can access the private access modifiers outside the class? The answer is Yes! We can access the private access modifiers outside the class with the help of getters and setters.

Getters and Setters :

- Getter ➳ Returns the value [accessors]
- setter ➳ Sets / updates the value [mutators]

In the below code, we've created total 4 methods:

1. setName(): The argument passed to this method is assigned to the private variable name.
2. getName(): The method returns the value set by the setName() method.
3. setId(): The integer argument passed to this method is assigned to the private variable id.
4. getId(): This method returns the value set by the setId() method.

```java
class Employee {

    private int id;
    private  String name;

    public String getName(){
        return name;
    }
    public void setName(String n){
        name = n;
    }
    public void setId(int i){
        id = i;
    }
    public int getId(){
        return id;
    }
}

public  class CWH {
    public static void main(String[] args) {
        Employee emp1 = new Employee();

        emp1.setName("Shubham");
        System.out.println(emp1.getName());
        emp1.setId(1);
        System.out.println(emp1.getId());

    }
}
```

```
Shubham
1
```

Constructors

Constructors in Java :

- Constructors are similar to methods,, but they are used to initialize an object.
- Constructors do not have any return type(not even void).
- Every time we create an object by using the new() keyword, a constructor is called.
- If we do not create a constructor by ourself, then the default constructor(created by Java compiler) is called.

Rules for creating a Constructor :

1. The class name and constructor name should be the same.
2. It must have no explicit return type.
3. It can not be abstract, static, final, and synchronized.

Types of Constructors in Java :

There are two types of constructors in Java :

1. **Defaut constructor :** A constructor with 0 parameters is known as default constructor.
   Syntax :

```
<class_name>(){
//code to be executed on the execution of the constructor
}
```

Example :

```
class CWH {
    CWH(){
        System.out.println("This is the default constructor of CW
    }

}
public class CWH_constructors {
    public static void main(String[] args) {
        CWH obj1 = new CWH();

    }
}
```

In the above code, CWH() is the constructor of class CWH The CWH() constructor is invoked automatically with the creation of object ob1.

2. **Paramerterized constructor** : A constructor with some specified number of parameters is known as a parameterized constructor.
   Syntax :

```
<class-name>(<data-type> param1, <data-type> param2,.....){
//code to be executed on the invocation of the constructor
}
```

Example :

```
class CWH {
    CWH(String s, int b){

        System.out.println("This is the " +b+ "th video of "+ " "
    }

}
public class CWH_constructors {
    public static void main(String[] args) {
        CWH obj1 = new CWH("CodeWithHarry Java Playlist",42);

    }
}
```

Output :

```
This is the 42th video of  CodeWithHarry Java Playlist
```

In the above example, CWH() constructor accepts two parameters i.e., string s and int b.

Constructor Overloading in Java :

Just like methods, constructors can also be overloaded in Java. We can overload the Employe constructor like below:

```
public Employee (String n)
           name = n;
         }
```

Note:

1. Constructors can take parameters without being overloaded
2. There can be more than two overloaded constructors

Let's take an example to understand the concept of constructor overloading.

Example :

In the below example, the class Employee has a constructor named Employee(). It takes two argument,i.e., string s & int i. The same constructor is overloaded and then it accepts three arguments i.e., string s, int i & int salary.

```java
class Employee {
// First constructor
    Employee(String s, int i){
        System.out.println("The name of the first employee is : " + s);
        System.out.println("The id of the first employee is : " + i);
    }
//    Constructor overloaded
    Employee(String s, int i, int salary){
        System.out.println("The name of the second employee is : " + s);
        System.out.println("The id of the second employee is : " + i);
        System.out.println("The salary of second employee is : " + salary);
    }

}
public class CWH_constructors {
    public static void main(String[] args) {
        Employee shubham = new Employee("Shubham",1);
        Employee harry = new Employee("Harry",2,70000);



    }
}
```

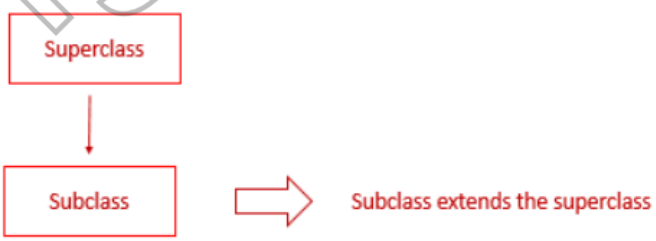
```
The name of the first employee is : Shubham
The id of the first employee is : 1
The name of the second employee is : Harry
The id of the second employee is : 2
The salary of second employee is : 70000
```

Inheritance

- You might have heard people saying your nose is similar to your father or mother. Or, more formally, we can say that you've inherited the genes from your parents due to which you look similar to them.
- The same phenomenon of inheritance is also valid in programming.
- In Java, one class can easily inherit the attributes and methods from some other class. This mechanism of acquiring objects and properties from some other class is known as inheritance in Java.
- Inheritance is used to borrow properties & methods from an existing class.
- Inheritance helps us create classes based on existing classes, which increases the code's reusability.
  Examples :

Important terminologies used in Inheritance :

1. Parent class/superclass: The class from which a class inherits methods and attributes is known as parent class.
2. Child class/sub-class: The class that inherits some other class's methods and attributes is known as child class.



Extends keyword in inheritance :

- The **extends** keyword is used to inherit a subclass from a superclass.
  Syntax :

```
class Subclass-name extends Superclass-name
{
   //methods and fields
}
```

  Example :

```
public class dog extends Animal {
              // code
                 }
```

Note: Java doesn't support multiple inheritances, i.e., two classes cannot be the superclass for a subclass.

Constructors in Inheritance:

When a drived class is extended from the base class, the constructor of the base class is executed first followed by the constructor of the derived class. For the following Inheritance hierarchy , the constructors are executed in the order:

1. C1- Parent
2. C2 - Child
3. C3 - Grandchild

Constructors during constructor overloading :

- When there are multiple constructors in the parent class, the constructor without any parameters is called from the child class.
- If we want to call the constructor with parameters from the parent class, we can use the super keyword.
- super(a, b) calls the constructor from the parent class which takes 2 variables

This and Super Keywords

this keyword in Java :

- this is a way for us to reference an object of the class which is being created/referenced.
- It is used to call the default constructor of the same class.
- **this** keyword eliminates the confusion between the parameters and the class attributes with the same name. Take a look at the example given below :

```
class cwh{
    int x;

//    getter of x
    public  int getX(){
        return  x;
    }

    // Constructor with a parameter
    cwh(int x) {
        x = x;
    }

    // Call the constructor
    public static void main(String[] args) {
        cwh obj1 = new cwh(65);
        System.out.println(obj1.getX());

    }
}
```

Output :

```
0
```

- In the above example, the expected output is 65 because we've passed x=65 to the constructor of the cwh class. But the compiler fails to differentiate between the parameter 'x' & class attribute 'x.' Therefore, it returns 0.
- Now, let's see how we can handle this situation with the help of this keyword. Take a look at the below code :

```java
class cwh{
    int x;

//    getter of x
    public  int getX(){
        return  x;
    }

    // Constructor with a parameter
    cwh(int x) {
        this.x = x;
    }

    // Call the constructor
    public static void main(String[] args) {
        cwh obj1 = new cwh(65);
        System.out.println(obj1.getX());

    }
}
```

Output :

```
65
```

Now, you can see that we've got the desired output

Super keyword

- A reference variable used to refer immediate parent class object.
- It can be used to refer immediate parent class instance variable.
- It can be used to invoke the parent class method.

Method Omverridimg

Method Overriding in Java:

- If the child class implements the same method present in the parent class again, it is know as method overriding.
- Method overriding helps us to classify a behavior that is specific to the child class.
- The subclass can override the method of the parent class only when the method is not declared as final.
- Example :
- In the below code, we've created two classes: class A & class B.
- Class B is inheriting class A.
- In the main() method, we've created one object for both classes. We're running the meth1() method on class A and B objects separately, but the output is the same because the meth1() is defined in the parent class, i.e., class A.

```
class A{
    public void meth1(){
        System.out.println("I am method 1 of class A");
    }
}

class B extends A{

}
public class CWH{
    public static void main(String[] args) {
        A a = new A();
        a.meth1();

        B b = new B();
        b.meth1();
    }
}
```

Output :

```
I am method 1 of class A
I am method 1 of class A
```

```
        B b = new B();
        b.meth1();
    }
}
```

Output :

```
I am method 1 of class A
I am method 1 of class B
```

Dynamic Methomd Dispatch

- Dynamic method dispatch is also known as run time polymorphism.
- It is the process through which a call to an overridden method is resolved at runtime.
- This technique is used to resolve a call to an overridden method at runtime rather than compile time.
- To properly understand Dynamic method dispatch in Java, it is important to understand the concept of upcasting because dynamic method dispatch is based on upcasting.

Upcasting :

- It is a technique in which a superclass reference variable refers to the object of the subclass.
  Example :

```
class Animal{}
class Dog extends Animal{}
```

```
Animal a=new Dog();//upcasting
```

In the above example, we've created two classes, named Animal(superclass) & Dog(subclass). While creating the object 'a', we've taken the reference variable of the parent class(Animal), and the object created is of child class(Dog).

Example to demonstrate the use of Dynamic method dispatch :

- In the below code, we've created two classes: **Phone & SmartPhone**.
- The **Phone** is the parent class and the **SmartPhone** is the child class.
- The method **on()** of the parent class is overridden inside the child class.
- Inside the main() method, we've created an object **obj** of the **Smartphone()** class by taking the reference of the **Phone()** class.
- When **obj.on()** will be executed, it will call the **on()** method of the **SmartPhone()** class because the reference variable obj is pointing towards the object of class **SmartPhone()**.

Abstract Class and Abstract Method

What does Abstract mean?

Abstract in English means existing in through or as an idea without concrete existence.

Abstract class :

- An abstract class cannot be instantiated.
- Java requires us to extend it if we want to access it.
- It can include abstract and non-abstract methods.
- If a class includes abstract methods, then the class itself must be declared abstract, as in:

Abstract

```
public abstract class phone Model {
            abstract void switch off ();
```

- Abstract class are used when we want to achieve security & abstraction(hide certain details & show only necessary details to the user)
  Example :

```
abstract class Phone{
  abstract void on();
}
class SmartPhone extends Phone{
void run(){
System.out.println("Turning on...");
}
public static void main(String args[]){
 Phone obj = new SmartPhone();
 obj.on();
}
}
```

Output :

```
Turning on...
```

Introduction to Interface

Interfaces in Java :

- Just like a class in java is a collection of the related methods, an interface in java is a collection of abstract methods.
- The interface is one more way to achieve abstraction in Java.
- An interface may also contain constants, default methods, and static methods.
- All the methods inside an interface must have empty bodies except default methods and static methods.
- We use the **interface** keyword to declare an interface.
- There is no need to write **abstract** keyword before declaring methods in an interface because an interface is implicitly abstract.
- An interface cannot contain a constructor (as it cannot be used to create objects)
- In order to implement an interface, java requires a class to use the **implement** keyword.

Example to demonstrate Interface in Java :

```
interface Bicycle {
            void apply brake ( int decrement );
            void  speed up ( int increment );
            }

class Avon cycle implements Bicycle {
            int speed = 7 ;
             void apply brake ( int decrement ) {
                    speed = speed - decrement ;
                        }
        void speedup ( int increment ){
                speed = speed + increment ;
                   }
```

Abstract Class vs Interface

Abstract class	Interface
1. It can contain abstract and non-abstract method	It can only contain abstract methods. We do not need to use the "abstract" keyword in interface methods because the interface is implicitly abstract.
2. abstract keyword is used to declare an abstract class.	interface keyword is used to declare an interface.
3. A sub-class extends the abstract class by using the "extends" keyword.	The "implements" keyword is used to implement an interface.
4. A abstract class in Java can have class members like private, protected, etc.	Members of a Java interface are public by default.
5. Abstract class doesn't support multiple inheritance.	Multiple inheritance is achieved in Java by using the interface.

Why multiple inheritamce is not sumpported in jamva

□□□□□Hamppy Ending□□□□□