**Introduction :-**

* 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.

#### **How Java Works?**

* 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 stands for Java Development Kit. It contains Java Virtual Machine(JVM) and Java Runtime Environment(JRE).
* **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 Java Program

package com.company; // Groups classes

public class Main{ // Entrypoint into the application

public static void main(String[]args){

System.out.println(“Hello World”);

}

}

##### 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.

##### 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.

##### public static void main(String[]args){..} :

* + This is the main() method of our Java program.
  + Every Java program must contain the main() method.

##### 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.

#### 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

We can choose a name while declaring a Java variable if the following rules are followed:

* Must not begin with a digit. (E.g., 1arry is an invalid variable)
* 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.

#### Data Types

Data types in Java fall under the following categories

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

#### Primitive Data Types

Java 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 x 1038to 1.40239846 x 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 x 10308, 4.9406564584124654 x 10-324 |
| 7. char | 2 byte | 0 to 65,535 |
| 8. boolean | Depends on JVM | True or False |

# Getting User Input in Java

#### 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.

Copy

**Taking an integer input from the keyboard :**

Scanner S = new Scanner(System.in); //(Read from the keyboard)

int a = S.nextInt(); //(Method to read from the keyboard)

# Introduction to 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);

Copy

is same as :

String s = "Rushikesh";

Copy

* 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>";

Copy

##### Example :

String str = "Rushikesh";

# String Methods in Java

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

Some of the commonly used String methods are:

|  |  |
| --- | --- |
| **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”. [Notethat 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] |

String name = “Rushi”;

Copy

(Indexes of the above string are as follows: 0-R, 1-u, 2-s, 3-h, 4-i)

#### 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)

#### 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

# While Loops in Java

* 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.

/\*

while (Boolean condition)

{

// Statements -> This keeps executing as long as the condition is true.

}

\*/

##### Example :

int i=10;

while(i>0){

System.out.println(i);

i--;

}

#### 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.

##### Syntax :

/\* do {

//code

} while (condition); //Note this semicolon \*/

##### Example :

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.

# The for Loop in Java

#### 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.

##### Syntax :

/\* for (initialize; check\_bool\_expression; update){

//code;

} \*/

* **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.

##### Example :

for (i=7; i!=0; i--){

System.out.println(i);

}

The above for loop initializes the value of i=7 and keeps printing as well as decrementing the value of i till i do not get equals to 0.

**Introduction to 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

**Introduction to 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.

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

# For Each Loop in Java

/\* for (int element:Arr) {

Sout(element); //Prints all the elements

} \*/

# Methods in Java

* 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.

**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 :

#### 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



#### 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

# Variable Arguments (VarArgs) in Java

* In the previous tutorial, we discussed how we can [overload the methods in Java](https://codewithharry.com/videos/java-tutorials-for-beginners-32).
* Now, let's suppose you want to overload an "add" method. The "add" method will accept one argument for the first time and every time the number of arguments passed will be incremented by 1 till the number of arguments is equaled to 10.
* One approach to solve this problem is to overload the "add" method 10 times. But is it the optimal approach? What if I say that the number of arguments passed will be incremented by 1 till the number of arguments is equaled to 1000. Do you think that it is good practice to overload a method 1000 times?
* To solve this problem of method overloading, Variable Arguments(Varargs) were introduced with the release of JDK 5.
* With the help of Varargs, we do not need to overload the methods.

##### Syntax :

/\*

public static void foo(int … arr)

{

// arr is available here as int[] arr

}

\*/

Copy

* 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));

}

}

Copy

##### Output :

3

9

15

**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.

##### 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.

#### Four pillars of Object-Oriented-Programming Language :

#### 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.

#### 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.

#### **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.
  + A 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.

#### 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

# 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 :

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

#### 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.

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());

}

}

Copy

##### Output :

Shubham

1

# Constructors in Java

#### 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 CWH class.");

}

}

public class CWH\_constructors {

public static void main(String[] args) {

CWH obj1 = new CWH();

}

}

##### Output :

This is the default constructor of CWH class.

1. 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 "+ " "+ s);

}

}

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.

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);

}

}

Output :

The name of the first employee is : Shubham

The id of the first employee is : 1

The name of the second employee is : Harry

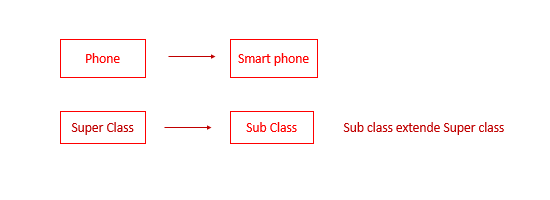
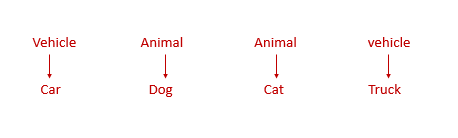
The id of the second employee is : 2

The salary of second employee is : 70000

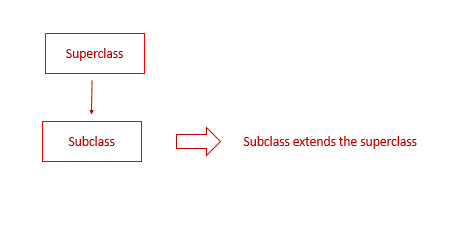
# Inheritance in Java

* 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

}

Copy

Example :

public class dog extends Animal {

// code

}

Copy

**Note:** [Java doesn't support multiple inheritances](https://codewithharry.com/videos/java-tutorials-for-beginners-56), i.e., two classes cannot be the superclass for a subclass.

**Quick quiz:**Create a class Animal and Derive another class dog from it

package com.company;

class Base{

public int x;

public int getX() {

return x;

}

public void setX(int x) {

System.out.println("I am in base and setting x now");

this.x = x;

}

public void printMe(){

System.out.println("I am a constructor");

}

}

class Derived extends Base{

public int y;

public int getY() {

return y;

}

public void setY(int y) {

this.y = y;

}

}

public class cwh\_45\_inheritance {

public static void main(String[] args) {

// Creating an Object of base class

Base b = new Base();

b.setX(4);

System.out.println(b.getX());

// Creating an object of derived class

Derived d = new Derived();

d.setY(43);

System.out.println(d.getY());

}

}

# Constructors in Inheritance in Java

#### 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 keyword in Java

#### 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 :

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 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();
* }
* }

Copy

##### Output :

I am method 1 of class A

I am method 1 of class A

Copy

* Now, let's see how we can override the meth1() for class B :
* class A{
* public void meth1(){
* System.out.println("I am method 1 of class A");
* }
* }
* class B extends A{
* @Override
* public void meth1(){
* System.out.println("I am method 1 of class B");
* }
* }
* public class CWH{
* public static void main(String[] args) {
* A a = new A();
* a.meth1();
* B b = new B();
* b.meth1();
* }
* }

Copy

##### Output :

I am method 1 of class A

I am method 1 of class B

# Abstract Class & Abstract Methods

#### 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:

public abstract class phone Model {

abstract void switch off ();

|| more code

}

Copy

* 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();

}

}

Copy

##### Output :

Turning on...

#### Abstract method :

* A method that is declared without implementation is known as the abstract method.
* An abstract method can only be used inside an abstract class.
* The body of the abstract method is provided by the class that inherits the abstract class in which the abstract method is present.
* In the above example, on() is the abstract method.

# Introduction to Interfaces

#### 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 Classes Vs Interfaces

|  |  |
| --- | --- |
| **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. Aabstract 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 inheritance is not supported in java?

#### Is multiple inheritance allowed in Java?

* Multiple inheritance faces problems when there exists a method with the same signature in both the superclasses.
* Due to such a problem, java does not support multiple inheritance directly, but the similar concept can be achieved using interfaces.
* A class can implement multiple interfaces and extend a class at the same time.

#### Some Important points :

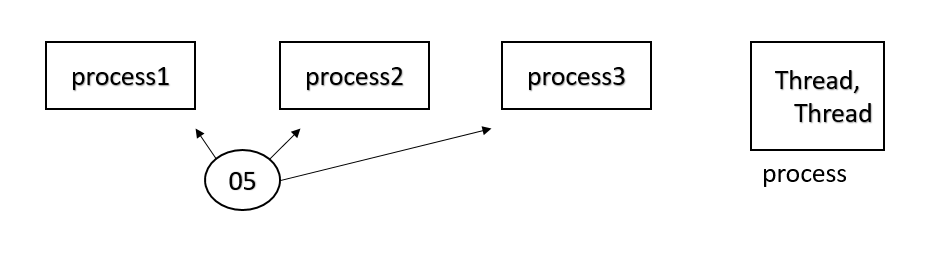
1. Interfaces in java are a bit like the class but with a significantly different.
2. An Interface can only have method signatures field and a default method.
3. The class implementing an interface needs to declare the methods ( not field )
4. You can create a reference of an interface but not the object
5. Interface methods are public by default

**Packages in Java**

* A package is used to group related classes.
* packages help in avoiding name conflicts

# Multithreading in Java

Multiprocessing and multithreading both are used to achive multitasking



#### **In a nut shell.......**

* threds use shared memory area
* threads = faster content switching
* Athread is light-weight where a process is heavyweight

   for example = Aword processor can have one thread running in foreground as an editor and another in the background auto saving the document !

#### **Creating a Threading**

   There are two ways to create a thread in java

1. By extending thread class
2. By implementing Runnable interface

# Creating a Thread by Extending Thread class

#### Multithreading In Java :

* Used to maximize the CPU utilization.
* We don't want our CPU to be in a free state; for example, Func1() comes into the memory and demands any input/output process. The CPU will need to wait for unit Func1() to complete its input/output operation in such a condition. But, while Func1() completes its I/O operation, the CPU is free and not executing any thread. So, the efficiency of the CPU is decreased in the absence of multithreading.
* In the case of multithreading, if a thread demands any I/O operation, then the CPU will let the thread perform its I/O operation, but it will start the execution of a new thread parallelly. So, in this case, two threads are executing at the same time.

#### Ways To Create A Thread In Java

1. By extending the thread class
2. By implementing Runnable interface

Let's see how we can create a thread by extending the thread class.

##### Extending Thread Class :

To create a thread using the thread class, we need to extend the thread class. Java's multithreading system is based on the thread class.

class MyThread extends Thread{

@Override

public void run(){

//code that we want to get executed on running the thread

}

}

Copy

* In the above code, we're first inheriting the Thread class and then overriding the run() method.
* The code you want to execute on the thread's execution goes inside the run() method.

class MyThread extends Thread{

@Override

public void run(){

int i =0;

while(i<40000){

System.out.println("My Cooking Thread is Running");

System.out.println("I am happy!");

i++;

}

}

}

public class cwh\_70 {

public static void main(String[] args) {

MyThread t1 = new MyThread();

t1.start();

}

}

Copy

In order to execute the thread, the start() method is used. start() is called on the object of the MyThread class. It automatically calls the run() method, and a new stack is provided to the thread. So, that's how you easily create threads by extending the thread class in Java.

# Creating a Java Thread Using Runnable Interface

In the previous tutorial, I told you that there are two ways to create a thread in java :

1. By Extending Thread Class
2. By implementing Runnable interface

We've already seen [how to create a thread by extending the thread class](https://codewithharry.com/videos/java-tutorials-for-beginners-71/java-tutorials-for-beginners-70). In this tutorial, we'll see how to create a Java thread by using a runnable interface.

#### Steps To Create A Java Thread Using Runnable Interface:

1. Create a class and implement the Runnable interface by using the implements keyword.
2. Override the run() method inside the implementer class.
3. Create an object of the implementer class in the main() method.
4. Instantiate the Thread class and pass the object to the Thread constructor.
5. Call start() on the thread. start()will call the run()method.

#### Example :

classs t1 implements Runnable{

@Override

public void run(){

System.out.println("Thread is running");

}

}

public class ClassName{

public static void main(String[] args) {

t1 obj1 = new t1();

Thread t = new Thread(obj1);

t.start();

}

}

Copy

1. class t1 is implementing the Runnable interface.
2. Overriding of the run() method is done inside the t1 class.
3. In the main() method, obj1, an object of the t1 class, is created.
4. The constructor of the Thread class accepts the Runnable instance as an argument, so obj1 is passed to the constructor of the Thread class.
5. Finally, the start()method is called on the thread that will call the run() method internally, and the thread's execution will begin.

#### Runnable Interface Vs Extending Thread Class :

Since we've discussed both the ways to create a thread in Java. There might be a question in your mind that should we use the Runnable interface or Thread class to implement a thread in Java. Let me answer this question for you. The Runnable interface is preferred over extending the Thread class because of the following reasons :

1. As multiple inheritance is not supported in Java, it is impossible to extend the Thread class if your class had already extended some other class.
2. While implementing Runnable, we do not modify or change the thread's behavior.
3. More memory is required while extending the Thread class because each thread creates a unique object.
4. Less memory is required while implementing Runnable because multiple threads share the same object.

#### **The Thread class**

Below are the commonly used constructors of the thread class:

1. Thread ( )
2. Thread ( string )
3. Thread ( Runnable r )
4. Thread ( Runnable r, String name )

package com.company;

class MyThr extends Thread{

public MyThr(String name){

super(name);

}

public void run(){

int i = 34;

System.out.println("Thank you");

// while(true){

// System.out.println("I am a thread");

// }

}

}

public class cwh\_73\_thread\_constructor {

public static void main(String[] args) {

MyThr t1 = new MyThr("Harry");

MyThr t2 = new MyThr("Ram Candr");

t1.start();

t2.start();

System.out.println("The id of the thread t is " + t1.getId());

System.out.println("The name of the thread t is " + t1.getName());

System.out.println("The id of the thread t is " + t2.getId());

System.out.println("The name of the thread t is " + t2.getName());

}

}

# Exceptions & Try-Catch Block in Java

#### Exceptions in Java

An exception is an event that occurs when a program is executed dissented the normal flow of instructions.

There are mainly two types of exceptions in java:

1) Checked exceptions - compile-time exceptions (Handle by the compiler)

2) Unchecked exceptions - Runtime exceptions

#### Commonly Occurring Exceptions

Following are few commonly occurring exceptions in java:

1) Null pointer exception

2) Arithmetic Exception

3) Array Index out of Bound exception

4) Illegal Argument Exception

5) Number Format Exception

package com.company;

public class cwh\_80\_try {

public static void main(String[] args) {

int a = 6000;

int b = 0;

// Without Try:

// int c = a / b;

// System.out.println("The result is " + c);

// With Try:

try {

int c = a / b;

System.out.println("The result is " + c);

}

catch(Exception e) {

System.out.println("We failed to divide. Reason: ");

System.out.println(e);

}

System.out.println("End of the program");

}

}

# Java Collections Framework

#### Collection Framework

A collection represents a group of object Java collections provide classes and Interfaces for us to be able to write code interfaces for us to be able to write code quickly and efficiently

Why do we need collections

We need collections for efficient storage and better manipulation of data in java

For ex: we use arrays to store integers but what if we want to

* Resize this array?
* Insert an element in between?
* Delete an elements in Array?
* Apply certain operations to change this array?

# Collections Hierarchy in Java

### How are collections available

Collections in java are available as class and interfaces Folling are few commonly used collections in java :

* ArrayList -> For variables size collections
* Set -> For distinct collection
* Stack-> A LIFO data structure
* HashMap -> For strong key - value pairs

Collections class is available in java util package collection class also provides static methods for sorting , searching etc.

**Functional Interfaces in Java**

* Difficulty Level : [Hard](https://www.geeksforgeeks.org/hard/)
* Last Updated : 16 Jan, 2022

Java has forever remained an Object-Oriented Programming language. By object-oriented programming language, we can declare that everything present in the Java programming language rotates throughout the Objects, except for some of the primitive data types and primitive methods for integrity and simplicity. There are no solely functions present in a programming language called Java. Functions in the Java programming language are part of a class, and if someone wants to use them, they have to use the class or object of the class to call any function.

A **functional interface** is an interface that contains only one abstract method. They can have only one functionality to exhibit. From Java 8 onwards, [lambda expressions](https://www.geeksforgeeks.org/lambda-expressions-java-8/) can be used to represent the instance of a functional interface. A functional interface can have any number of default methods. ***Runnable***, ***ActionListener***,***Comparable*** are some of the examples of functional interfaces.

Functional Interface is additionally recognized as **Single Abstract Method Interfaces**. In short, they are also known as **SAM interfaces**. Functional interfaces in Java are the new feature that provides users with the approach of fundamental programming.

Functional interfaces are included in Java SE 8 with Lambda expressions and Method references in order to make code more readable, clean, and straightforward. Functional interfaces are interfaces that ensure that they include precisely only one abstract method. Functional interfaces are used and executed by representing the interface with an **annotation called *@FunctionalInterface***. As described earlier, functional interfaces can contain only one abstract method. However, they can include any quantity of default and static methods.

In Functional interfaces, there is no need to use the abstract keyword as it is optional to use the abstract keyword because, by default, the method defined inside the interface is abstract only. We can also call Lambda expressions as the instance of functional interface.

Before Java 8, we had to create anonymous inner class objects or implement these interfaces.

* Java

|  |
| --- |
| // Java program to demonstrate functional interface    class Test {      public static void main(String args[])      {          // create anonymous inner class object          new Thread(new Runnable() {              @Override public void run()              {                  System.out.println("New thread created");              }          }).start();      }  } |

**Output**

New thread created

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

* JDBC-ODBC Bridge Driver,
* Native Driver,
* Network Protocol Driver, and

JPA

Thin DriverJPA is just a specification that facilitates object-relational mapping to manage relational data in Java applications. It provides a platform to work directly with objects instead of using SQL statements.

SPRING:-

The Spring framework can be considered as a collection of sub-frameworks, also called layers, such as Spring AOP. Spring Object-Relational Mapping (Spring ORM). Spring Web Flow, and Spring Web MVC. You can use any of these modules separately while constructing a Web application. The modules may also be grouped together to provide better functionalities in a Web application.