**Object orientd programing lang-**

**(C, c++,) (oops)🡪java ,python, .net…..**

**Distributed app**

**JSE**

**JEE**

**JME**

**Platform inde**

**Client side server side**

**JAVA 🡪FRONT-END(UI), BACK-END DATABASE**

**jvm**

**JSE,JME,JEE**

**Platform dependent**

**“ independent**

**Client server**

**Software ->coll’t of program**

**1.system s/w : platform dependent**

**2.Application s/w: ms office,games,galler,**

**3.Internet s/w: 1.watup ,insta**

**Intel s.w**

**Mother board**

**G.byte Mother board :-G.B s/w**

**Java Keywords**

**Data types**

Integer

Char

Double

Boolean

String

Float

A list of Java keywords or reserved words are given below:

1. [**abstract**](https://www.javatpoint.com/abstract-keyword-in-java)**:** Java abstract keyword is used to declare abstract class. Abstract class can provide the implementation of interface. It can have abstract and non-abstract

methods.

Abstract void demo();

1. [**boolean:**](https://www.javatpoint.com/boolean-keyword-in-java) Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
2. [**break**](https://www.javatpoint.com/java-break)**:** Java break keyword is used to break loop or switch statement. It breaks the current flow of the program at specified condition.
3. [**byte**](https://www.javatpoint.com/byte-keyword-in-java)**:** Java byte keyword is used to declare a variable that can hold an 8-bit data values.
4. [**case**](https://www.javatpoint.com/case-keyword-in-java)**:** Java case keyword is used to with the switch statements to mark blocks of text.
5. [**catch**](https://www.javatpoint.com/try-catch-block)**:** Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
6. [**char**](https://www.javatpoint.com/char-keyword-in-java)**:** Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
7. [**class**](https://www.javatpoint.com/class-keyword-in-java)**:** Java class keyword is used to declare a class.
8. [**continue**](https://www.javatpoint.com/java-continue)**:** Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
9. [**default**](https://www.javatpoint.com/default-keyword-in-java)**:** Java default keyword is used to specify the default block of code in a switch statement.
10. [**do**](https://www.javatpoint.com/java-do-while-loop)**:** Java do keyword is used in control statement to declare a loop. It can iterate a part of the program several times.
11. [**double**](https://www.javatpoint.com/double-keyword-in-java)**:** Java double keyword is used to declare a variable that can hold a 64-bit floating-point numbers.
12. [**else**](https://www.javatpoint.com/java-if-else)**:** Java else keyword is used to indicate the alternative branches in an if statement.
13. [**enum**](https://www.javatpoint.com/enum-in-java)**:** Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
14. [**extends**](https://www.javatpoint.com/inheritance-in-java)**:** Java extends keyword is used to indicate that a class is derived from another class or interface.
15. [**final**](https://www.javatpoint.com/final-keyword)**:** Java final keyword is used to indicate that a variable holds a constant value. It is applied with a variable. It is used to restrict the user.
16. [**finally**](https://www.javatpoint.com/finally-block-in-exception-handling)**:** Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether exception is handled or not.
17. [**float**](https://www.javatpoint.com/float-keyword-in-java)**:** Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
18. [**for**](https://www.javatpoint.com/java-for-loop)**:** Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some conditions become true. If the number of iteration is fixed, it is recommended to use for loop.
19. [**if**](https://www.javatpoint.com/java-if-else)**:** Java if keyword tests the condition. It executes the if block if condition is true.
20. [**implements**](https://www.javatpoint.com/interface-in-java)**:** Java implements keyword is used to implement an interface.
21. [**import**](https://www.javatpoint.com/package)**:** Java import keyword makes classes and interfaces available and accessible to the current source code.
22. [**instanceof**](https://www.javatpoint.com/downcasting-with-instanceof-operator)**:** Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
23. [**int**](https://www.javatpoint.com/int-keyword-in-java)**:** Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
24. [**interface**](https://www.javatpoint.com/interface-in-java)**:** Java interface keyword is used to declare an interface. It can have only abstract methods.
25. [**long**](https://www.javatpoint.com/long-keyword-in-java)**:** Java long keyword is used to declare a variable that can hold a 64-bit integer.
26. **native:** Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
27. [**new**](https://www.javatpoint.com/new-keyword-in-java)**:** Java new keyword is used to create new objects.
28. [**null**](https://www.javatpoint.com/null-keyword-in-java)**:** Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.

Extends

implements

class A{

}

Class B extends A{

}

Class --extends--- class

Class -implements- interface//interface ---implements—class

Class calssname{

}

Interface ---extends--interface

Interface A{

}

Class B implements A{

}

c-extends-c

c-implements-I/f

I/f -implemnets---clas

I/f --extends---I/f

1. [**package**](https://www.javatpoint.com/package)**:** Java package keyword is used to declare a Java package that includes the classes.
2. [**private**](https://www.javatpoint.com/private-keyword-in-java)**:** Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
3. [**protected**](https://www.javatpoint.com/protected-keyword-in-java)**:** Java protected keyword is an access modifier. It can be accessible within package and outside the package but through inheritance only. It can't be applied on the class.
4. [**public**](https://www.javatpoint.com/public-keyword-in-java)**:** Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
5. [**return**](https://www.javatpoint.com/return-keyword-in-java)**:** Java return keyword is used to return from a method when its execution is complete.
6. [**short**](https://www.javatpoint.com/short-keyword-in-java)**:** Java short keyword is used to declare a variable that can hold a 16-bit integer.
7. [**static**](https://www.javatpoint.com/static-keyword-in-java)**:** Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is used for memory management mainly.
8. [**strictfp**](https://www.javatpoint.com/strictfp-keyword)**:** Java strictfp is used to restrict the floating-point calculations to ensure portability.
9. [**super**](https://www.javatpoint.com/super-keyword)**:** Java super keyword is a reference variable that is used to refer parent class object. It can be used to invoke immediate parent class method.
10. [**switch**](https://www.javatpoint.com/java-switch)**:** The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.

1byte =4 bits

Static Void demo()

{

Static int a=10; variable

Doesnt have return stmt;

}

Int demo()

{

It only integer

Return 0;

}

1. [**synchronized**](https://www.javatpoint.com/synchronization-in-java)**:** Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.
2. [**this**](https://www.javatpoint.com/this-keyword)**:** Java this keyword can be used to refer the current object in a method or constructor.
3. [**throw**](https://www.javatpoint.com/throw-keyword)**:** The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exception. It is followed by an instance.
4. [**throws**](https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws)**:** The Java throws keyword is used to declare an exception. Checked exception can be propagated with throws.
5. [**transient**](https://www.javatpoint.com/transient-keyword)**:** Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
6. [**try**](https://www.javatpoint.com/try-catch-block)**:** Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.

try{

}

finally()

{

}

1. **void:** Java void keyword is used to specify that a method does not have a return value.
2. [**volatile**](https://www.javatpoint.com/volatile-keyword-in-java)**:** Java volatile keyword is used to indicate that a variable may change asynchronously.
3. [**while**](https://www.javatpoint.com/java-while-loop)**:** Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Loops

For

Nested for loop

While

Do-while

## Control Statements

* if statement
* if-else statement
* else-if ladder
* nested if statement

**Syntax**

**If(a>b) T/F**

**{**

**----**

**---**

**}//**

**Int a=20,b=10;**

**If(a>b)-->t/F**

**{**

**System.out.println(“a is big”);**

**}**

**if**(condition){

//code to be executed

}

 if-else statement

**Syntax:**

**if**(condition){

//code if condition is true

}**else**{

//code if condition is false

}

else-if ladder statement **Syntax:**

**if**(condition1){

//code to be executed if condition1 is true

}

**else** **if**(condition2){

//code to be executed if condition2 is true

}

**else** **if**(condition3){

//code to be executed if condition3 is true

}

...

else

{

//code to **else**be executed if all the conditions are false

}

nested if statement.

**Syntax:**

**if**(condition){

     //code to be executed

**if**(condition){

             //code to be executed

    }

}

If ()T/F

{

If()T/f

{----

}

Else

{

}

}

Else

{

}

**Switch syntax**

**Switch()**

**{**

**Case 1:**

**Break;**

**switch**(expression){

**case** value1:

 //code to be executed;

**break**;  //optional

**case** value2:

 //code to be executed;

**break**;  //optional

......

**default**:

 code to be executed **if** all cases are not matched;

}

|  |  |  |  |
| --- | --- | --- | --- |
| **Comparison** | **for loop** | **while loop** | **do while loop** |
| Introduction | The Java for loop is a control flow statement that iterates a part of the [programs](https://www.javatpoint.com/java-programs) multiple times. | The Java while loop is a control flow statement that executes a part of the programs repeatedly on the basis of given boolean condition. | The Java do while loop is a control flow statement that executes a part of the programs at least once and the further execution depends upon the given boolean condition. |
| When to use | If the number of iteration is fixed, it is recommended to use for loop. | If the number of iteration is not fixed, it is recommended to use while loop. | If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use the do-while loop. |
| Syntax | for(init;condition;incr/decr){  // code to be executed  } | while(condition){  //code to be executed  } | do{  //code to be executed  }while(condition); |
| Example | //for loop  for(int i=1;i<=10;i++){  System.out.println(i);  } | //while loop  int i=1;  while(i<=10){  System.out.println(i);  i++;  } | //do-while loop  int i=1;  do{  System.out.println(i);  i++;  }while(i<=10); |

OOPs (Object-Oriented Programming System)

Object means a real-world entity such as a pen, chair, table, computer, watch, etc.

**Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:

* [Object](https://www.javatpoint.com/object-and-class-in-java)
* Class
* [Inheritance](https://www.javatpoint.com/inheritance-in-java)
* [Polymorphism](https://www.javatpoint.com/runtime-polymorphism-in-java)
* [Abstraction](https://www.javatpoint.com/abstract-class-in-java)
* [Encapsulation](https://www.javatpoint.com/encapsulation)

## Object

**An object is an instance of a class.** A class is a template or blueprint from which objects are created.

So, an object is the instance(result) of a class.

**Object Definitions:**

* An object is *a real-world entity*.
* An object is *a runtime entity*.
* The object is *an entity which has state and behavior*.
* The object is *an instance of a class*.

Any entity that has state and behavior is known as an object.

For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.

**Example:** A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

## Class

Collection of objects is called class. It is a logical entity.

A class in Java can contain:

* **Fields**
* **Methods**
* **Constructors**
* **Blocks**
* **Nested class and interface**

A class can also be defined as a blueprint from which you can create an individual object.

Class doesn't consume any space.

### Inheritance

When one object acquires all the properties and behaviors of a parent object, it is known as inheritance.

It provides code reusability. It is used to achieve runtime polymorphism.



### Polymorphism

If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

In Java, we use method overloading and method overriding to achieve polymorphism.

Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

#### Abstraction

Hiding internal details and showing functionality is known as abstraction. For example phone call, we don't know the internal processing.

In Java, we use abstract class and interface to achieve abstraction.



### Encapsulation

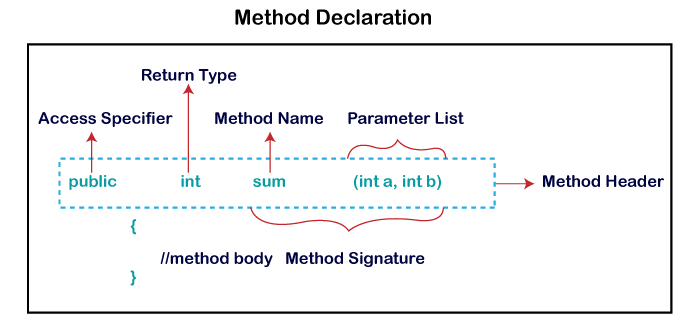
Binding (or wrapping) code and data together into a single unit are known as encapsulation.

For example, a capsule, it is wrapped with different medicines.

A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data members are private here.

## What is a method in Java?

A **method** is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.



# public

# private

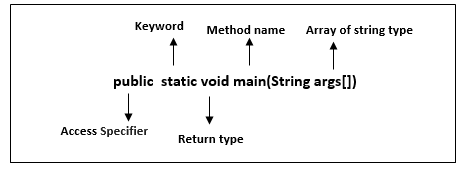
protected

default

# 

# Java main() method

The main() is the starting point for JVM to start execution of a Java program. Without the main() method, JVM will not execute the program. The syntax of the main() method is:



**public:** It is an access specifier. We should use a public keyword before the main() method so that JVM can identify the execution point of the program. If we use private, protected, and default before the main() method, it will not be visible to JVM.

**static:** You can make a method static by using the keyword static. We should call the main() method without creating an object. Static methods are the method which invokes without creating the objects, so we do not need any object to call the main() method.

**void:** In Java, every method has the return type. Void keyword acknowledges the compiler that main() method does not return any value.

**main():** It is a default signature which is predefined in the JVM. It is called by JVM to execute a program line by line and end the execution after completion of this method. We can also overload the main() method.

**String args[]:** The main() method also accepts some data from the user. It accepts a group of strings, which is called a string array. It is used to hold the command line arguments in the form of string values.

## Types of Method

There are two types of methods in Java:

* Predefined Method
* User-defined Method

### Predefined Method

In Java, predefined methods are the method that is already defined in the Java class libraries is known as predefined methods. It is also known as the **standard library method** or **built-in method**. We can directly use these methods just by calling them in the program at any point. Some pre-defined methods are **length(), equals(), compareTo(), sqrt(),** etc. When we call any of the predefined methods in our program, a series of codes related to the corresponding method runs in the background that is already stored in the library.

Each and every predefined method is defined inside a class. Such as **print()** method is defined in the **java.io.PrintStream** class. It prints the statement that we write inside the method. For example, **print("Java")**, it prints Java on the console.

Let's see an example of the predefined method.

**Demo.java**

1. **public** **class** Demo
2. {
3. **public** **static** **void** main(String[] args)
4. {
5. // using the max() method of Math class
6. System.out.print("The maximum number is: " + Math.max(9,7));
7. }
8. }

### User-defined Method

The method written by the user or programmer is known as **a user-defined** method. These methods are modified according to the requirement

Let's create a user defined method that checks the number is even or odd. First, we will define the method.

1. //user defined method
2. **public** **static** **void** findEvenOdd(**int** num)
3. {
4. //method body
5. **if**(num%2==0)
6. System.out.println(num+" is even");
7. **else**
8. System.out.println(num+" is odd");
9. }

**Constructors in Java**

Every time an object is created using the new() keyword, at least one constructor is called.

Class A{

Void m1(){

}

A(){

}

Class B{

Public static void main(string arg[]){

A obj = new A();

Obj.m1();

}

}

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

**Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

### Rules for creating Java constructor

There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors

There are two types of constructors in Java:

Default constructor (no-arg constructor)

Parameterized constructor

//Java Program to create and call a default constructor

**class** Bike1{

//creating a default constructor

Bike1(){System.out.println("Bike is created");}

/main method

**public** **static** **void** main(String args[]){

//calling a default constructor

Bike1 b=**new** Bike1();

}

}

### Example of parameterized constructor

In this example, we have created the constructor of Student class that have two parameters. We can have any number of parameters in the constructor.

//Java Program to demonstrate the use of the parameterized constructor.

**class** Student4{

**int** id;

    String name;

    //creating a parameterized constructor

   Student4(**int** i,String n){

   id = i;

    name = n;

    }

    //method to display the values

**void** display(){System.out.println(id+" "+name);}

**public** **static** **void** main(String args[]){

    //creating objects and passing values

    Student4 s1 = **new** Student4(111,"Karan");

    Student4 s2 = **new** Student4(222,"Aryan");

    //calling method to display the values of object

    s1.display();

    s2.display();

   }

}

### Example of Constructor Overloading

//Java program to overload constructors

**class** Student5{

**int** id;

    String name;

**int** age;

    //creating two arg constructor

    Student5(**int** i,String n){

    id = i;

    name = n;

    }

    //creating three arg constructor

    Student5(**int** i,String n,**int** a){

    id = i;

    name = n;

    age=a;

    }

**void** display(){System.out.println(id+" "+name+" "+age);}

**public** **static** **void** main(String args[]){

    Student5 s1 = **new** Student5(111,"Karan");

    Student5 s2 = **new** Student5(222,"Aryan",25);

    s1.display();

    s2.display();

   }

}

Difference between constructor and method in Java

There are many differences between constructors and methods. They are given below.

|  |  |
| --- | --- |
| **Java Constructor** | **Java Method** |
| A constructor is used to initialize the state of an object. | A method is used to expose the behavior of an object. |
| A constructor must not have a return type. | A method must have a return type. |
| The constructor is invoked implicitly. | The method is invoked explicitly. |
| The Java compiler provides a default constructor if you don't have any constructor in a class. | The method is not provided by the compiler in any case. |
| The constructor name must be same as the class name. | The method name may or may not be same as the class name. |

class multiplestaticblocks

{

public static void main(string[]args);

{

system.out.println("main");

}

static

{

system.out.println("SB2");

}

static

{

system.out.println("SB1");

}

}