#### **1) Local Variable**

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

#### **2) Instance Variable**

A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as [static](https://www.javatpoint.com/static-keyword-in-java).

It is called an instance variable because its value is instance-specific and is not shared among instances.

#### **3) Static variable**

A variable that is declared as static is called a static variable. It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class. Memory allocation for static variables happens only once when the class is loaded in the memory.

Constructors in Java

In [Java](https://www.javatpoint.com/java-tutorial)

, a constructor is a block of codes similar to the method. It is called when an instance of the [class](https://www.javatpoint.com/object-and-class-in-java)

is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

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

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

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

#### **We can use [access modifiers](https://www.javatpoint.com/access-modifiers)**

#### **while declaring a constructor. It controls the object creation. In other words, we can have private, protected, public or default constructor in Java.**

## **Types of Java constructors**

There are two types of constructors in Java:

1. Default constructor (no-arg constructor)
2. Parameterized constructor



## **Java Default Constructor**

A constructor is called "Default Constructor" when it doesn't have any parameter

1. //Java Program to create and call a default constructor
2. **class** Bike1{
3. //creating a default constructor
4. Bike1(){System.out.println("Bike is created");}
5. //main method
6. **public** **static** **void** main(String args[]){
7. //calling a default constructor
8. Bike1 b=**new** Bike1();
9. }
10. }

Output :

Bike is created

#### **If there is no constructor in a class, compiler automatically creates a default constructor.**



**What is the purpose of a default constructor?**

The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Example :

1. //Let us see another example of default constructor
2. //which displays the default values
3. **class** Student3{
4. **int** id;
5. String name;
6. //method to display the value of id and name
7. **void** display(){System.out.println(id+" "+name);}
8. **public** **static** **void** main(String args[]){
9. //creating objects
10. Student3 s1=**new** Student3();
11. Student3 s2=**new** Student3();
12. //displaying values of the object
13. s1.display();
14. s2.display();
15. }
16. }

Output :

0 null

0 null

**Explanation:**In the above class,you are not creating any constructor so compiler provides you a default constructor. Here 0 and null values are provided by default constructor

### **Java Parameterized Constructor**

A constructor which has a specific number of parameters is called a parameterized constructor.

### **Why use the parameterized constructor?**

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

### **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.

1. //Java Program to demonstrate the use of the parameterized constructor.
2. **class** Student4{
3. **int** id;
4. String name;
5. //creating a parameterized constructor
6. Student4(**int** i,String n){
7. id = i;
8. name = n;
9. }
10. //method to display the values
11. **void** display(){System.out.println(id+" "+name);}
13. **public** **static** **void** main(String args[]){
14. //creating objects and passing values
15. Student4 s1 = **new** Student4(111,"Karan");
16. Student4 s2 = **new** Student4(222,"Aryan");
17. //calling method to display the values of object
18. s1.display();
19. s2.display();
20. }
21. }

Output

111 Karan

222 Aryan

## **Constructor Overloading in Java**

In Java, a constructor is just like a method but without return type. It can also be overloaded like Java methods.

Constructor [overloading in Java](https://www.javatpoint.com/method-overloading-in-java)

is a technique of having more than one constructor with different parameter lists. They are arranged in a way that each constructor performs a different task. They are differentiated by the compiler by the number of parameters in the list and their types.

1. //Java program to overload constructors
2. **class** Student5{
3. **int** id;
4. String name;
5. **int** age;
6. //creating two arg constructor
7. Student5(**int** i,String n){
8. id = i;
9. name = n;
10. }
11. //creating three arg constructor
12. Student5(**int** i,String n,**int** a){
13. id = i;
14. name = n;
15. age=a;
16. }
17. **void** display(){System.out.println(id+" "+name+" "+age);}
19. **public** **static** **void** main(String args[]){
20. Student5 s1 = **new** Student5(111,"Karan");
21. Student5 s2 = **new** Student5(222,"Aryan",25);
22. s1.display();
23. s2.display();
24. }
25. }

Output

111 Karan 0

222 Aryan 25

|  |
| --- |
|  |



## **Java Copy Constructor**

There is no copy constructor in Java. However, we can copy the values from one object to another like copy constructor in C++.

There are many ways to copy the values of one object into another in Java. They are:

* By constructor
* By assigning the values of one object into another
* By clone() method of Object class

In this example, we are going to copy the values of one object into another using Java constructor.

1. //Java program to initialize the values from one object to another object.
2. **class** Student6{
3. **int** id;
4. String name;
5. //constructor to initialize integer and string
6. Student6(**int** i,String n){
7. id = i;
8. name = n;
9. }
10. //constructor to initialize another object
11. Student6(Student6 s){
12. id = s.id;
13. name =s.name;
14. }
15. **void** display(){System.out.println(id+" "+name);}
17. **public** **static** **void** main(String args[]){
18. Student6 s1 = **new** Student6(111,"Karan");
19. Student6 s2 = **new** Student6(s1);
20. s1.display();
21. s2.display();
22. }
23. }

Output

111 Karan

111 Karan

## **Copying values without constructor**

We can copy the values of one object into another by assigning the objects values to another object. In this case, there is no need to create the constructor.

1. **class** Student7{
2. **int** id;
3. String name;
4. Student7(**int** i,String n){
5. id = i;
6. name = n;
7. }
8. Student7(){}
9. **void** display(){System.out.println(id+" "+name);}
11. **public** **static** **void** main(String args[]){
12. Student7 s1 = **new** Student7(111,"Karan");
13. Student7 s2 = **new** Student7();
14. s2.id=s1.id;
15. s2.name=s1.name;
16. s1.display();
17. s2.display();
18. }
19. }

Output:

111 Karan

111 Karan

# this keyword in Java

There can be a lot of usage of **Java this keyword**. In Java, this is a **reference variable** that refers to the current object.



.



### **1) this: to refer current class instance variable**

The this keyword can be used to refer current class instance variable. If there is ambiguity between the instance variables and parameters, this keyword resolves the problem of ambiguity.

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** rollno,String name,**float** fee){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.fee=fee;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
13. **class** TestThis2{
14. **public** **static** **void** main(String args[]){
15. Student s1=**new** Student(111,"ankit",5000f);
16. Student s2=**new** Student(112,"sumit",6000f);
17. s1.display();
18. s2.display();
19. }}

Output

111 Karan 5000.0

112 Sumit 6000.0

#### **It is better approach to use meaningful names for variables. So we use same name for instance variables and parameters in real time, and always use this keyword.**

### **2) this: to invoke current class method**

You may invoke the method of the current class by using the this keyword. If you don't use the this keyword, compiler automatically adds this keyword while invoking the method.

1. **class** A{
2. **void** m(){System.out.println("hello m");}
3. **void** n(){
4. System.out.println("hello n");
5. //m();//same as this.m()
6. **this**.m();
7. }
8. }
9. **class** TestThis4{
10. **public** **static** **void** main(String args[]){
11. A a=**new** A();
12. a.n();
13. }}

Output

hello n

hello m

### **3) this() : to invoke current class constructor**

The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

**Calling default constructor from parameterized constructor:**

1. **class** A{
2. A(){System.out.println("hello a");}
3. A(**int** x){
4. **this**();
5. System.out.println(x);
6. }
7. }
8. **class** TestThis5{
9. **public** **static** **void** main(String args[]){
10. A a=**new** A(10);
11. }}

Output

hello n

10

**Calling parameterized constructor from default constructor:**

1. **class** A{
2. A(){
3. **this**(5);
4. System.out.println("hello a");
5. }
6. A(**int** x){
7. System.out.println(x);
8. }
9. }
10. **class** TestThis6{
11. **public** **static** **void** main(String args[]){
12. A a=**new** A();
13. }}

Output

5

hello n

### **Real usage of this() constructor call**

The this() constructor call should be used to reuse the constructor from the constructor. It maintains the chain between the constructors i.e. it is used for constructor chaining. Let's see the example given below that displays the actual use of this keyword.

1. **class** Student{
2. **int** rollno;
3. String name,course;
4. **float** fee;
5. Student(**int** rollno,String name,String course){
6. **this**.rollno=rollno;
7. **this**.name=name;
8. **this**.course=course;
9. }
10. Student(**int** rollno,String name,String course,**float** fee){
11. **this**(rollno,name,course);//reusing constructor
12. **this**.fee=fee;
13. }
14. **void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
15. }
16. **class** TestThis7{
17. **public** **static** **void** main(String args[]){
18. Student s1=**new** Student(111,"ankit","java");
19. Student s2=**new** Student(112,"sumit","java",6000f);
20. s1.display();
21. s2.display();
22. }}

111 ankit java 0.0

112 sumit java 6000.0

#### **Call to this() must be the first statement in constructor.**