# Java Constructors

A constructor in Java is similar to a method that is invoked when an [object](https://www.programiz.com/java-programming/class-objects#objects) of the [class](https://www.programiz.com/java-programming/class-objects) is created.

Unlike [Java methods](https://www.programiz.com/java-programming/methods), a constructor has the same name as that of the class and does not have any return type. For example,

class Test {

Test() {

// constructor body

}

}

Here, Test() is a constructor. It has the same name as that of the class and doesn't have a return type.

## Example: Java Constructor

class Main {

private String name;

// constructor

Main() {

System.out.println("Constructor Called:");

name = "Programiz";

}

public static void main(String[] args) {

// constructor is invoked while

// creating an object of the Main class

Main obj = new Main();

System.out.println("The name is " + obj.name);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Constructor Called:

The name is Programiz

In the above example, we have created a constructor named Main().

Inside the constructor, we are initializing the value of the name [variable](https://www.programiz.com/java-programming/variables-literals).

Notice the statement creating an object of the Main class.

Main obj = new Main();

Here, when the object is created, the Main() constructor is called. And the value of the name variable is initialized.

Hence, the program prints the value of the name variables as Programiz.

### Types of Constructor

In Java, constructors can be divided into three types:

1. No-Arg Constructor
2. Parameterized Constructor
3. Default Constructor

## 1. Java No-Arg Constructors

Similar to methods, a Java constructor may or may not have any parameters (arguments).

If a constructor does not accept any parameters, it is known as a no-argument constructor. For example,

private Constructor() {

// body of the constructor

}

### Example: Java Private No-arg Constructor

class Main {

int i;

// constructor with no parameter

private Main() {

i = 5;

System.out.println("Constructor is called");

}

public static void main(String[] args) {

// calling the constructor without any parameter

Main obj = new Main();

System.out.println("Value of i: " + obj.i);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**:

Constructor is called

Value of i: 5

In the above example, we have created a constructor Main().

Here, the constructor does not accept any parameters. Hence, it is known as a no-arg constructor.

**Notice that we have declared the constructor as private.**

Once a constructor is declared private, it cannot be accessed from outside the class.

So, creating objects from outside the class is prohibited using the private constructor.

Here, we are creating the object inside the same class.

Hence, the program is able to access the constructor. To learn more, visit [Java Implement Private Constructor](https://www.programiz.com/java-programming/examples/private-constructor-implementation).

However, if we want to create objects outside the class, then we need to declare the constructor as public.

## Example: Java Public no-arg Constructors

class Company {

String name;

// public constructor

public Company() {

name = "Programiz";

}

}

class Main {

public static void main(String[] args) {

// object is created in another class

Company obj = new Company();

System.out.println("Company name = " + obj.name);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Company name = Programiz

## 2. Java Parameterized Constructor

A Java constructor can also accept one or more parameters. Such constructors are known as parameterized constructors (constructors with parameters).

### Example: Parameterized Constructor

class Main {

String languages;

// constructor accepting single value

Main(String lang) {

languages = lang;

System.out.println(languages + " Programming Language");

}

public static void main(String[] args) {

// call constructor by passing a single value

Main obj1 = new Main("Java");

Main obj2 = new Main("Python");

Main obj3 = new Main("C");

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Java Programming Language

Python Programming Language

C Programming Language

In the above example, we have created a constructor named Main().

Here, the constructor takes a single parameter. Notice the expression:

Main obj1 = new Main("Java");

Here, we are passing the single value to the constructor.

Based on the argument passed, the language variable is initialized inside the constructor.

## 3. Java Default Constructor

If we do not create any constructor, the Java compiler automatically creates a no-arg constructor during the execution of the program.

This constructor is called the default constructor.

### Example: Default Constructor

class Main {

int a;

boolean b;

public static void main(String[] args) {

// calls default constructor

Main obj = new Main();

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Default Value:

a = 0

b = false

Here, we haven't created any constructors.

Hence, the Java compiler automatically creates the default constructor.

The default constructor initializes any uninitialized instance variables with default values.

|  |  |
| --- | --- |
| Type | Default Value |
| boolean | false |
| byte | **0** |
| short | **0** |
| int | **0** |
| long | **0L** |
| char | \u0000 |
| float | **0.0f** |

890

System.out.println("Default Value:");

System.out.println("a = " + obj.a);

System.out.println("b = " + obj.b);

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Default Value:

a = 0

b = false

## Important Notes on Java Constructors

* Constructors are invoked implicitly when you instantiate objects.
* The two rules for creating a constructor are:  
  1. The name of the constructor should be the same as the class.  
  2. A Java constructor must not have a return type.
* If a class doesn't have a constructor, the Java compiler automatically creates a **default constructor** during run-time. The default constructor initializes instance variables with default values. For example, the int variable will be initialized to 0
* Constructor types:  
  **No-Arg Constructor** - a constructor that does not accept any arguments  
  **Parameterized constructor** - a constructor that accepts arguments  
  **Default Constructor** - a constructor that is automatically created by the Java compiler if it is not explicitly defined.
* A constructor cannot be abstract or static or final.
* A constructor can be overloaded but can not be overridden.

# Java static keyword:

The **static keyword** in [Java](https://www.javatpoint.com/java-tutorial)

is used for memory management mainly. We can apply static keyword with [variables](https://www.javatpoint.com/java-variables)

, methods, blocks and [nested classes](https://www.javatpoint.com/java-inner-class)

. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)



## **1) Java static variable**

If you declare any variable as static, it is known as a static variable.

* The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
* The static variable gets memory only once in the class area at the time of class loading.

### **Advantages of static variable**

It makes your program **memory efficient** (i.e., it saves memory).

### **Example of static variable**

1. //Java Program to demonstrate the use of static variable
2. **class** Student{
3. **int** r;//instance variable
4. String n;
5. **static** String college ="ITS";//static variable
6. //constructor
7. Student(**int** r, String n){
8. This.r= r;
9. This.n = n;
10. }
11. //method to display the values
12. **void** display (){
13. System.out.println(rollno+" "+name+" "+college);
14. }
15. }
16. //Test class to show the values of objects
17. **public** **class** TestStaticVariable1{
18. **public** **static** **void** main(String args[]){

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

1. Student s2 = **new** Student(222,"Aryan");
2. //we can change the college of all objects by the single line of code
3. //Student.college="BBDIT";
4. s1.display();
5. s2.display();
6. }
7. }

**[Test it Now](https://www.javatpoint.com/opr/test.jsp?filename=TestStaticVariable1" \t "_blank)**

Output:

111 Karan ITS

222 Aryan ITS

### **Another example of a static method that performs a normal calculation**

1. //Java Program to get the cube of a given number using the static method
3. **class** Calculate{
4. **static** **int** cube(**int** x){
5. **return** x\*x\*x;
6. }
8. **public** **static** **void** main(String args[]){
9. **int** result=Calculate.cube(5);
10. System.out.println(result);
11. }
12. }

**[Test it Now](https://www.javatpoint.com/opr/test.jsp?filename=Calculate" \t "_blank)**

Output:125

# 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.



## **Usage of Java this keyword**

Here is given the 6 usage of java this keyword.

1. [this can be used to refer current class instance variable.](https://www.javatpoint.com/this1)
2. [this can be used to invoke current class method (implicitly)](https://www.javatpoint.com/this2)
3. [this() can be used to invoke current class constructor.](https://www.javatpoint.com/this3)
4. [this can be passed as an argument in the method call.](https://www.javatpoint.com/this4)
5. [this can be passed as argument in the constructor call.](https://www.javatpoint.com/this5)
6. [this can be used to return the current class instance from the method.](https://www.javatpoint.com/this6)

**Suggestion:** If you are beginner to java, lookup only three usages of this keyword.

# Usage of Java this keyword

#### **Solution of the above problem by this keyword**

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. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis2)

**Output:**

111 ankit 5000.0

112 sumit 6000.0

#### **Program where this keyword is not required**

1. **class** Student{
2. **int** rollno;
3. String name;
4. **float** fee;
5. Student(**int** r,String n,**float** f){
6. rollno=r;
7. name=n;
8. fee=f;
9. }
10. **void** display(){System.out.println(rollno+" "+name+" "+fee);}
11. }
13. **Public class** TestThis3{
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. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis3)

**Output:**

111 ankit 5000.0

112 sumit 6000.0

### **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. Let's see the example



1. **class** A{
2. **void** m(){
3. System.out.println("hello m");
4. }
5. **void** n(){
6. System.out.println("hello n");
7. //m();//same as this.m()
8. **this**.m();
9. }
10. }
11. **class** TestThis4{
12. **public** **static** **void** main(String args[]){
13. A a=**new** A();
14. a.n();
15. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis4)

**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(){
3. System.out.println("hello a");
4. }
5. A(**int** x){
6. **this**();
7. System.out.println(x);
8. }
9. }
10. **class** TestThis5{
11. **public** **static** **void** main(String args[]){
12. A a=**new** A(10);
13. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis5)

**Output:**

hello a

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. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis6)

**Output:**

ADVERTISEMENT

ADVERTISEMENT

5

hello a

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

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(){
15. System.out.println(rollno+" "+name+" "+course+" "+fee);
16. }
17. }
18. **class** TestThis7{
19. **public** **static** **void** main(String args[]){
20. Student s1=**new** Student(111,"ankit","java");
21. Student s2=**new** Student(112,"sumit","java",6000f);
22. s1.display();
23. s2.display();
24. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis7)

**Output:**

111 ankit java 0.0

112 sumit java 6000.0

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

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){
11. **this**.rollno=rollno;
12. **this**.name=name;
13. **this**.course=course;
14. }
15. Student(**int** rollno,String name,String course,**float** fee){
16. **this**.fee=fee;
17. **this**(rollno,name,course);//C.T.Error
18. }
19. **void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}
20. }
21. **class** TestThis8{
22. **public** **static** **void** main(String args[]){
23. Student s1=**new** Student(111,"ankit","java");
24. Student s2=**new** Student(112,"sumit","java",6000f);
25. s1.display();
26. s2.display();
27. }}

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=TestThis8)

**Output:**

ADVERTISEMENT

ADVERTISEMENT

Compile Time Error: Call to this must be first statement in constructor

### **4) this: to pass as an argument in the method**

The this keyword can also be passed as an argument in the method. It is mainly used in the event handling. Let's see the example:

1. **class** S2{
2. **void** m(S2 obj){
3. System.out.println("method is invoked");
4. }
5. **void** p(){
6. m(**this**);
7. }
8. **public** **static** **void** main(String args[]){
9. S2 s1 = **new** S2();
10. s1.p();
11. }
12. }

[**Test it Now**](https://www.javatpoint.com/opr/test.jsp?filename=S2)

**Output:**

method is invoked