\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* \*\*\*\*\*\*\*\*OOPS java \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

JAVA programming start from main line by line like c.

Java – is an object oriented programming language.

What is object oriented – object oriented means everything and anything inside class.

Class A {

//Variable

//Method

}

* Java is guaranteed to be write ones and runs anywhere platform.
* Object oriented- in java everything is an object.
* Java can be easily extended since it is based on object model.
* Platform independent.
* Simple.
* **What is class in java**
* Class is template or container where everything is store.
* Class is template or blue print where variable and method are stored that describe the behavior /state that’s objects of this type supports.
* **Memory will assign when we create object**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **Variables**
* **Syntax :** datatype VariabeName = Value;
* A variable represent a piece of memory that can contain a data value.

**Variable**

**Global variable local variable**

1. **Instance variable 2static variable 1.Local variable**

**Eg. Class test {**

**int I =10 🡨----------- Instance variable**

**statc int j =20 🡨-------- static variable**

**void display(){**

**int k =30 🡨------------ local variable**

**}**

**}**

* **Instance Variable- i**nstance variable is declared inside class and outside the method.
* **Life** – within object**, when object is created instance variable is copied into object memory**
* **Static Variable -** static variable is declared inside class and outside the method with static keyword.
* Stored in static pool --- common for all object like global in c
* Scope/life - within the program
* Local variable declare inside the method

Public class variable{

Public static void main (String[] , args ){

Variable x = new variable(); // create object of class variable.

x. Display(); // calling display method of the class variable

}

Int number = 28; / instance varables

Flot flotting\_sub = 10.3f;

String name =”nitin”;

Double doble\_No = 34;

Char c=”N”;

Byte b=(byte)a;

Static char x= ’c’; //static variable

byte xx = (byte)c;

void Display(){

int d=a; // local variable

String name=”I am inside method”;

System.out.println(“Integer --- “” + number );

System.out.println(“Integer --- “” + doble\_No );

System.out.println(“Integer --- “” + c );

}

}

* **Scope of variable**
* The **scope** of a **variable** defines the section of the code in which the **variable** is visible.
* The lifetime of a **variable** refers to how long the **variable** exists before it is destroyed.

Public class variableScope{

Int a =10;

Static int b =3;

Void Add(){

Int x=2;

System.out.println(“a”); // a is an instance variable

System.out.println(“b”); // a is an static variable

}

**Static Void display(){**

**// System.out.println(“a”); // instance variable cant access inside static class**

**System.out.println(“b”); // static variable can access**

**}**

**s** variableScope(){ //this is a constructor

System.out.println(“a”);

System.out.println(“b”);

}

variableScope(int a, inr b){ //this is a parameterized constructor

this.a=a;

this.b=b;

}

}

* **Methods :**
* A **method** is one who perform problem specific task or some operation.(same like function in c language)
* USE – code re-usability
* A **method** is basically a behavior of class.
* A class can contain many methods
* In the **method** where the logic are written.
* Data is manipulate and all the action are executed.
* Syntax : <access specifier> <return type> <method or function name**>(parameter)**{

// body of method you can write your logic here (only this type of value you can return if you try to other value it give error)

Return <return value>

}

Float add(){

float sum = a+b;

return sum;

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Objects :** instance of class
* **Instance of class ?? ---** means separate copy of the class (templates)
* **Syntax : <className><objectName> = new <className>() 🡨--- “()” is a constructor see later**

The new operator instantiates a **class** by allocating memory for a new object and returning a reference to that memory. Note: The phrase "instantiating a **class**" means the same thing as "creating an object." When you create an object, you are creating an "**instance**" of a **class**, therefore "instantiating" a **class.**

Class objExample{ objExample a = new objExample ()

Int i=10;

Static int j = 20; instance of class objExample

Void Display(){

I=10

J=20

Display()

Add()

System.out.println(“……helo…”);

} a ---------🡪

Int add(){

Int x=10,y=20;

Int add=x+y;

Return add;

}

} When we create object a memory block is assign and the reference of that memory is assign to object name here “a” so the object ‘a’ contain the a copy of class objExample variable and methods.

I=10,J=20

Display()

Add()

1. objExample a = new objExample () a ---------------🡪

I=10,J=20

Display()

Add()

1. objExample b = new objExample () b---------------🡪

* a and b are instance (copy) of same class But having the different memory block.so the operation perform on one object is not valid for other.
* Using “ . ”operator you can access variable and method of the class
* E.G. a.i=30; a.Display();
* b. Add(); b.Display(); b.i=50;

**--For Every object separate memory block are created.**

**--the operation we perform using a object, The value is stored in memory block of that particular object only**

* *There is no header file in java like c language*
* *If all class file are in one package (folder)then any class file can create object of any class available on that PACKAGE.*

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* **Static keyword**
* This variable stored in static pool.
* When you create object that time all method & instance variable are copied in object memory block but static variable are not copied in objects memory but you can access this variable.
* This means when we create an object a new copy of instance variable is created in object block memory.

But

Static variable are created ones and stored in static pool not in object memory block,

* Here for static one memory block is assign to static variable and that is common for all the objects same like global variable in c.

**Class A{**

Y=20

Display()

(OBJECT MEMORY)

**Static int x =10; 1. Create object**

**Int y =20; A n=new A();**

**Void disly(){ x ----------------🡪**

**System.out.println(“hello’’);**

**} Static pool**

**} x=10;**

**2. Access: n.y=30; //allowed**

**n.x=40; 🡨--- this is not a good practice to access static variable**

1. **Corret way to access static variable:**

**ClassName.Static variable/method name**

**e.g. A.x=40;**

**Object memory**

**1. A a1= new A(); a1 ----🡪 X is common for all object**

Static pool

X=10

**2. A a2 = new A(); a2 ---🡪**

**3. A a3 =new A(); a3 --🡪**

**4 A a4 =new a(); a4--🡪**

**static** int y = 0; When a **variable** is declared with the keyword **static** , it's called a class **variable** . All instances share the same copy of the **variable**. A class **variable** can be accessed directly with the class, without the need to create a instance.

When a variable is declared with the keyword static, it’s called a class variable. All instances share the same copy of the variable. A class variable can be accessed directly with the class, without the need to create a instance.

The **static keyword** in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested 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)
3. Block
4. Nested class.

* 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.

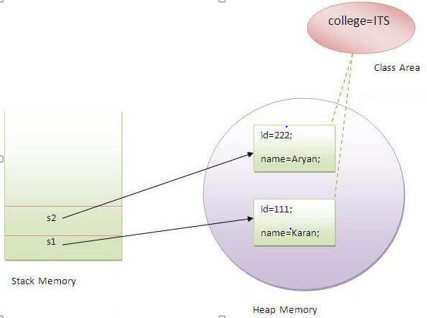
Advantages of static variable

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

### Example of static variable

//Java Program to demonstrate the use of static variable

**class** Student{



**int** rollno;//instance variable

   String name;

**static** String college ="ITS"; //static variable

   Student(**int** r, String n){    //constructor

   rollno = r;

   name = n;

   }

   //method to display the values

**void** display (){System.out.println(rollno+

" "+name+" "+college);}

}

//Test class to show the values of objects

**public** **class** TestStaticVariable1{

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

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

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

 //we can change the college of all objects by the single line of code   Heap memory

 //Student.college="BBDIT";

 s1.display();

 s2.display();

 }

}

Output:

111 Karan ITS

222 Aryan ITS

### Program of the counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable. If it is incremented, it won't reflect other objects. So each object will have the value 1 in the count variable.

//Java Program to demonstrate the use of an instance variable

//which get memory each time when we create an object of the class.

**class** Counter{

**int** count=0;//will get memory each time when the instance is created

Counter(){

count++;//incrementing value

System.out.println(count);

}

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

//Creating objects

Counter c1=**new** Counter();

Counter c2=**new** Counter();

Counter c3=**new** Counter();

}

}

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

Output:

1

1

1

### Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

//Java Program to illustrate the use of static variable which

//is shared with all objects.

**class** Counter2{

**static** **int** count=0;//will get memory only once and retain its value

Counter2(){

count++;//incrementing the value of static variable

System.out.println(count);

}

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

//creating objects

Counter2 c1=**new** Counter2();

Counter2 c2=**new** Counter2();

Counter2 c3=**new** Counter2();

}

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Counter2)

Output:

1

2

3

## **2) Java static method**

* **STATIC method: The above static concept is same for “static Method” also.**

If you apply static keyword with any method, it is known as static method.

* A static method belongs to the class rather than the object of a class.
* A static method can be invoked without the need for creating an instance of a class.
* A static method can access static data member and can change the value of it.

### Example of static method

//Java Program to demonstrate the use of a static method.

**class** Student{

**int** rollno;

     String name;

**static** String college = "ITS";

**static** **void** change(){      //static method to change the value of static variable

     college = "BBDIT";

     }

     Student(**int** r, String n){    //constructor to initialize the variable

     rollno = r;

     name = n;

     }

     //method to display values

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

}

**public** **class** TestStaticMethod{   //Test class to create and display the values of object

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

    Student.change();//calling change method

    //creating objects

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

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

    Student s3 = **new** Student(333,"Sonoo");

    //calling display method

    s1.display();

    s2.display();

    s3.display();

    }

}

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

Output:111 Karan BBDIT

222 Aryan BBDIT

333 Sonoo BBDIT

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

//Java Program to get the cube of a given number using the static method

**class** Calculate{

**static** **int** cube(**int** x){

**return** x\*x\*x;

  }

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

**int** result=Calculate.cube(5);

  System.out.println(result);

  }

}

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

Output:125

### Restrictions for the static method

**There are two main restrictions for the static method. They are:**

1. **The static method can not use non static data member or call non-static method directly.**

**WHY ?**

The simple reason behind this is that **Static data members** of parent class **can** be accessed (only if they are **not** overridden) but for instance(**non**-**static**) **data members** or **methods** we need their reference and so they **can** only be called through an object. ... A **non**-**static method** is dependent on the object.

1. **this and super cannot be used in static context.**

**WHY?**

Lets first understand what is static method and how it works, then we can easily conclude the answer of this question.

Whenever a method is declared as static then we should keep in mind these points:

* **A static method belongs to the class rather than object of a class.**
* A static method can be invoked without the need for creating an instance of a class.
* static method can access static data member and can change the value of it.

Now, **this**keyword in java is a **reference variable** that refers to the **current object**. Also the **super** keyword in java is a **reference variable** which is used to refer immediate **parent class object**.

So, we can say that **this**and **super** both keyword are **reference variable** that refers to some object. In other words these both keywords belong to instance of the class.

Whereas, **static method** belongs to the class than instance of the class. And so **static method** can’t access **this** and **super** keyword in java.

**class** A{

**int** a=40;//non static

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

  System.out.println(a);

 }

}

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

Output:Compile Time Error

### Q) Why is the Java main method static?

Ans) It is because the object is not required to call a static method. If it were a non-static method, JVM creates an object first then call main() method that will lead the problem of extra memory allocation.

1. he method is static because otherwise there would be ambiguity: which constructor should be called? Especially if your class looks like this:

public class JavaClass{

protected JavaClass(int x){}

public void main(String[] args){

}

}

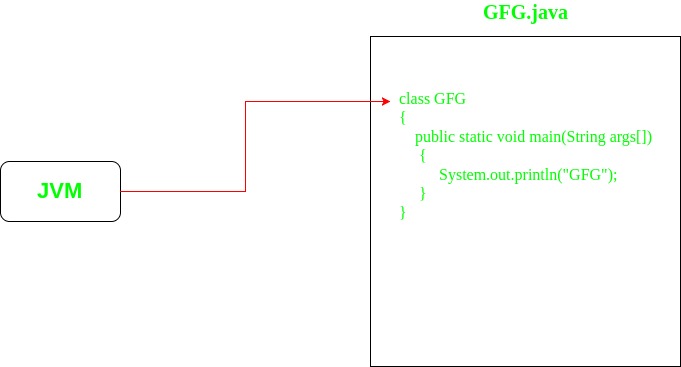
Should the JVM call new JavaClass(int)? What should it pass for x?

If not, should the JVM instantiate JavaClass without running any constructor method? I think it shouldn't, because that will special-case your entire class - sometimes you have an instance that hasn't been initialized, and you have to check for it in every method that could be called.

There are just too many edge cases and ambiguities for it to make sense for the JVM to have to instantiate a class before the entry point is called. That's why main is static.

2. Following points explain what is “static” in the main() method:

1. **main() method**: The main() method, in Java, is the entry point for the JVM(Java Virtual Machine) into the java program. JVM launches the java program by invoking the main() method.



1. [Static is a keyword](https://www.geeksforgeeks.org/static-keyword-java/). The role of adding static before any entity is to make that entity a class entity. It means that adding static before methods and variables make them class methods and class variables respectively, instead of instance methods and instance variables.  
   Hence, static methods and variables can be directly accessed with the help of Class, which means that there is no need to create objects in order to access static methods or variables.
2. // Making a function as static
3. static void func()
4. {}
5. // Making a variable as static
6. static int var;
7. [**Static methods:**](https://www.geeksforgeeks.org/static-methods-vs-instance-methods-java/) When a method is declared with static keyword, it is known as static method. As discussed above, any static member can be accessed before any objects of its class are created, and without reference to any object.
8. // Making a static function
9. class GfG
10. {
11. static void func()
12. {}
13. }
14. // Calling a static function
15. GfG.func();
16. **Static main() method:** When the static keyword is added in the function definition of main() method, then it is known as static main() method.
17. class GfG
18. {
19. // Making a static main function
20. public static void main(String[] args)
21. {}
22. }
23. **Need of static in main() method**: Since main() method is the entry point of any Java application, hence making the main() method as static is mandatory due to following reasons:
    * The static main() method makes it very clear for the JVM to call it **for launching the Java Application**. Otherwise, it would be required to specify the entry function for each Java application build, for the JVM to launch the application.
    * The method is static because otherwise there would be **ambiguity** which constructor should be called.

Example, if the class looks like this:

public class GfG{

protected GfG(int g){}

public void main(String[] args){

}

}

The JVM now enters an ambiguity state deciding whether it should call new GfG(int)? If yes, then what should it pass for g? If not, then should the JVM instantiate GfG without executing any constructor method?

There are just too many edge cases and ambiguities like this for it to make sense for the JVM to have to instantiate a class before the entry point is called. That’s why main is static.

* + The main() method is static because its convenient for the JDK. Consider a scenario where it’s not mandatory to make main() method static. Then in this case, that just makes it harder on various IDEs to auto-detect the ‘launchable’ classes in a project. Hence making it a convention to make the entry method ‘main()’ as ‘public static void main(String[] args)’ is convenient.

Thus, the main() method in java is declared as static so that the JVM can access it directly using its class name before object creation.

## **3) Java static block**

* Is used to initialize the static data member.
* It is executed before the main method at the time of classloading.

### Example of static block

**class** A2{

**static**{System.out.println("static block is invoked");}

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

   System.out.println("Hello main");

  }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=A2)

Output:static block is invoked

Hello main

### Q) Can we execute a program without main() method?

Ans) No, one of the ways was the static block, but it was possible till JDK 1.6. Since JDK 1.7, it is not possible to execute a java class without the main method.

**class** A3{

**static**{

  System.out.println("static block is invoked");

  System.exit(0);

  }

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=A3)

Output:

static block is invoked

Since JDK 1.7 and above, output would be:

Error: Main method not found in class A3, please define the main method as:

public static void main(String[] args)

or a JavaFX application class must extend javafx.application.Application

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Constructor:**

Constructor is a block of code that initializes the newly created object. A constructor resembles an instance method in java but it’s not a method as it doesn’t have a return type. In short constructor and method are differen. People often refer constructor as special type of method in Java.

Constructor has same name as the class and looks like this in a java code.

public class MyClass{

//This is the constructor

MyClass(){

}

..

}

* Note that the constructor name matches with the class name and it doesn’t have a return type.
* 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.
* Every time an object is created using the new() keyword, at least one constructor is called.

Student s = new student ( )

This always in small case This is called constructor

Without this we not able to create object

* Constructor get called/invoked when we create object.
* We can use this for setting value.

### 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
4. If user not provide constructor then JVM insert default constructor
5. If user gives non-arg constructor or parameterize constructor then JVM should not insert default constructor

## How does a constructor work

To understand the working of constructor, lets take an example. lets say we have a class MyClass.  
When we create the object of MyClass like this:

MyClass obj = new MyClass()

The **new keyword** here creates the object of class MyClass and invokes the constructor to initialize this newly created object.

You may get a little lost here as I have not shown you any initialization example, lets have a look at the code below:

### A simple constructor program in java

Here we have created an object obj of class Hello and then we displayed the instance variable nameof the object. As you can see that the output is BeginnersBook.com which is what we have passed to the name during initialization in constructor. This shows that when we created the object obj the constructor got invoked. In this example we have used **this keyword**, which refers to the current object, object obj in this example. We will cover this keyword in detail in the next tutorial.

public class Hello {

String name;

//Constructor

Hello(){

this.name = "hello I am inside constructor";

}

public static void main(String[] args) {

Hello obj = new Hello();

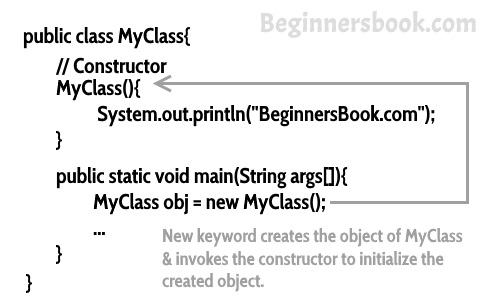
System.out.println(obj.name);

}

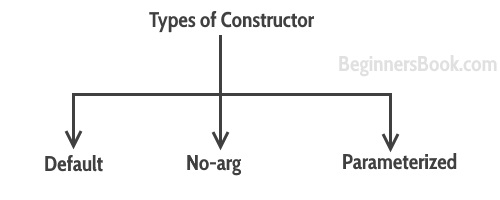
}

**Output:**

hello I am inside constructor



## Types of Constructors

There are three types of constructors: Default, No-arg constructor and Parameterized.  


### Default constructor

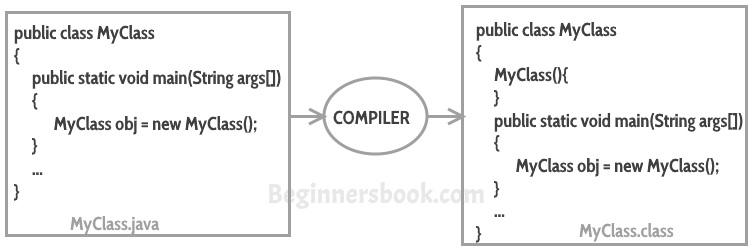
### Syntax : <class\_name>(){}

### If you do not implement any constructor in your class, Java compiler inserts a [default constructor](https://beginnersbook.com/2014/01/default-constructor-java-example/)in to your code on your behalf. This constructor is known as default constructor. You would not find it in your source code(the java file) as it would be inserted into the code during compilation and exists in .class file.

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



E.g.:



* If you implement any constructor then you no longer receive a default constructor from Java compiler.

### 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 of default constructor that displays the default values

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);}
9. **public** **static** **void** main(String args[]){
10. //creating objects
11. Student3 s1=**new** Student3();
12. Student3 s2=**new** Student3();
13. //displaying values of the object
14. s1.display();
15. s2.display();
16. }
17. }

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Student3)

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.

### no-arg constructor:

constructor with no arguments is known as **no-arg constructor**. The signature is same as default constructor, however body can have any code unlike default constructor where the body of the constructor is empty.

Although you may see some people claim that that default and no-arg constructor is same but in fact they are not, even if you write public Demo() { } in your class Demo it cannot be called default constructor since you have written the code of it.

#### Example: no-arg constructor

class Demo

{

public Demo()

{

System.out.println("This is a no argument constructor");

}

public static void main(String args[]) {

new Demo();

}

}

Output:  
This is a no argument constructor

### Parameterized Constructor

### Constructor with arguments(or you can say parameters) is known as [Parameterized constructor](https://beginnersbook.com/2014/01/parameterized-constructor-in-java-example/).

* 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.

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

**class** Student4{

**int** id;

    String name;

    Student4(**int** i,String n){ //creating a parameterized constructor

    id = i;

    name = n;

    }

**void** display(){System.out.println(id+" "+name);}   //method to display the values

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

   Student4 s1 = **new** Student4(111,"Karan");  //creating objects and passing values   Student4 s2 = **new** Student4(222,"Aryan");

   s1.display(); //calling method to display the values of object

    s2.display();

   }

}

Output:

111 Karan //the value is initialized with object creation

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 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.

//Java program to overload constructors

**class** Student5{

**int** id;

    String name;

**int** age;

    Student5(**int** i,String n){   //creating two arg constructor

    id = i;

    name = n;

    }

    Student5(**int** i,String n,**int** a){       //creating three arg constructor

    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");   compiler automatically select this

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

    s1.display();

    s2.display();

   }

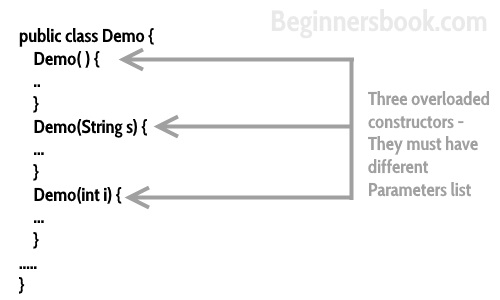
}

Output:

111 Karan 0

222 Aryan 25

## Constructor Overloading

Constructor overloading is a concept of having more than one constructor with different parameters list, in such a way so that each constructor performs a different task.  


## What if you implement only parameterized constructor in class

class Example3

{

private int var;

public Example3(int num)

{

var=num;

}

public int getValue()

{

return var;

}

public static void main(String args[])

{

Example3 myobj = new Example3();

System.out.println("value of var is: "+myobj.getValue());

}

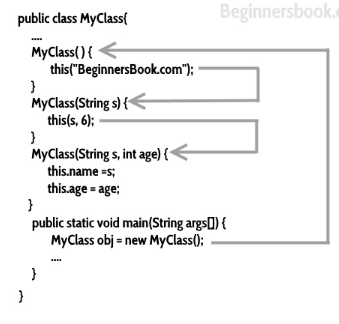
}

**Output**: It will throw a compilation error. The reason is, the statement Example3 myobj = new Example3() is invoking a default constructor which we don’t have in our program. when you don’t implement any constructor in your class, compiler inserts the default constructor into your code, however when you implement any constructor (in above example I have implemented parameterized constructor with int parameter), then you don’t receive the default constructor by compiler into your code.

If we remove the parameterized constructor from the above code then the program would run fine, because then compiler would insert the default constructor into your code.

## Constructor Chaining

* Calling a [constructor](https://beginnersbook.com/2013/03/constructors-in-java/) from the another constructor of same class is known as Constructor chaining. The real purpose of Constructor Chaining is that you can pass parameters through a bunch of different constructors, but only have the initialization done in a single place. This allows you to maintain your initializations from a single location, while providing multiple constructors to the user. If we don’t chain, and two different constructors require a specific parameter, you will have to initialize that parameter twice, and when the initialization changes, you’ll have to change it in every constructor, instead of just the one.
* As a rule, constructors with fewer arguments should call those with more



## Constructor Chaining Example

In this example, I have several constructors, one constructor is calling another constructor using **this keyword**.

this() should always be the first statement in **constructor** otherwise you will get this error message: **Exception in thread “main” java.lang.Error: Unresolved compilation problem: Constructor call must be the first statement in a constructor**

class Employee

{

public String empName;

public int empSalary;

public String address;

//default constructor of the class

public Employee()

{

//this will call the constructor with String param

this("Chaitanya");

}

public Employee(String name)

{

//call the constructor with (String, int) param

this(name, 120035);

}

public Employee(String name, int sal)

{

//call the constructor with (String, int, String) param

this(name, sal, "Gurgaon");

}

public Employee(String name, int sal, String addr)

{

this.empName=name;

this.empSalary=sal;

this.address=addr;

}

void disp() {

System.out.println("Employee Name: "+empName);

System.out.println("Employee Salary: "+empSalary);

System.out.println("Employee Address: "+address);

}

public static void main(String[] args)

{

Employee obj = new Employee();

obj.disp();

}

}

**Output:**

Employee Name: Chaitanya

Employee Salary: 12003

## **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.

//Java program to initialize the values from one object to another object.

**class** Student6{

**int** id;

    String name;

    Student6(**int** i,String n){      //constructor to initialize integer and string

    id = i;

    name = n;

    }

    //constructor to initialize another object

    Student6(Student6 s){   🡨----------------------it will take a object as aggument

    id = s.id;

    name =s.name;

    }

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

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

    Student6 s1 = **new** Student6(111,"Karan");

    Student6 s2 = **new** Student6(s1);

    s1.display();

    s2.display();

   }

}

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.

**class** Student7{

**int** id;

    String name;

    Student7(**int** i,String n){

    id = i;

    name = n;

    }

    Student7(){}

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

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

    Student7 s1 = **new** Student7(111,"Karan");

    Student7 s2 = **new** Student7();

    s2.id=s1.id;

    s2.name=s1.name;

    s1.display();

    s2.display();

   }

}

Output:

111 Karan

111 Karan

## **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. |



## Quick Recap

1. Every class has a constructor whether it’s a normal class or a abstract class.
2. Constructors are not methods and they don’t have any return type.
3. Constructor name should match with class name .
4. Constructor can use any access specifier, they can be declared as private also. Private constructors are possible in java but there scope is within the class only.
5. **Like constructors method can also have name same as class name, but still they have return type, though which we can identify them that they are methods not constructors.**
6. If you don’t implement any constructor within the class, compiler will do it for.
7. **this() and super() should be the first statement in the constructor code.** If you don’t mention them, compiler does it for you accordingly.
8. Constructor overloading is possible but overriding is not possible. Which means we can have overloaded constructor in our class but we can’t override a constructor.
9. Constructors can not be inherited.
10. If Super class doesn’t have a no-arg(default) constructor then compiler would not insert a default constructor in child class as it does in normal scenario.
11. Interfaces [do not have constructors](https://beginnersbook.com/2013/12/java-constructor-in-interface/).
12. Abstract class can have constructor and it gets invoked when a class, which implements interface, is instantiated. (i.e. object creation of concrete class).
13. A constructor can also invoke another constructor of the same class – By using this(). If you want to invoke a parameterized constructor then do it like this: **this(parameter list)**.

**More on Constructor:**

* [Private constructor](https://beginnersbook.com/2013/12/java-private-constructor-example/)
* [Can we have static constructors in Java?](https://beginnersbook.com/2013/05/static-constructor/)

## Difference between Constructor and Method

I know I should have mentioned it at the beginning of this guide but I wanted to cover everything in a flow. Hope you don’t mind :)

1. The purpose of constructor is to initialize the object of a class while the purpose of a method is to perform a task by executing java code.
2. Constructors cannot be abstract, final, static and synchronised while methods can be.
3. Constructors do not have return types while methods do.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

* **Conclusion**

**Public class text{**int count =0;

Text(){

Count++;

System.out.println(count);

}

Public static void main(){

Text a1= new text(); every object called constructor ()

Text a2= new text();

Text a3= new text();

}

**}**

**Every object have its copy in heap memory**

**Output**

1

**1 -------------🡪**

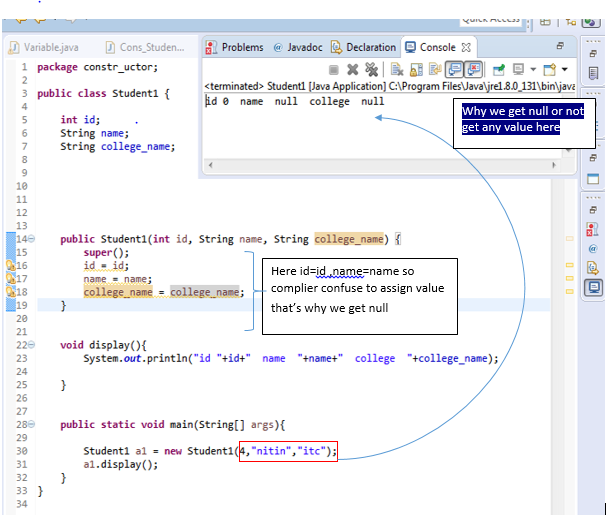
1

**1 -----------------------------------🡪**

**1 ----------------------------🡪**

1

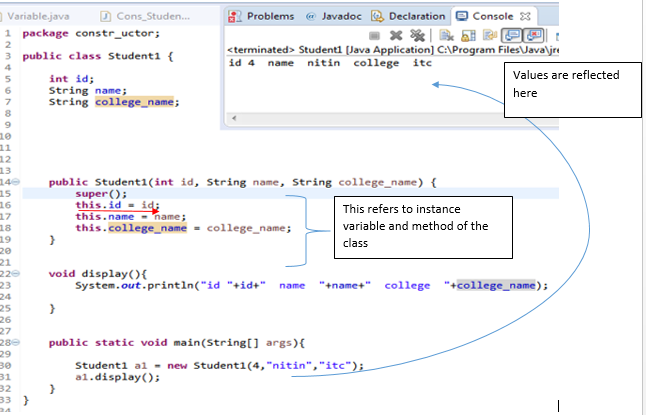
* **Here count is instance variable**
* **Every object create its separate memory block in heap memory**
* **The output is 1,1,1 because for every object complier make copy the instance variable.**
* **If we make count in above code as static then the output will be 1,2,3 because it ic common for all object**
* **This keyword**
* **What is problem without this**



**Here if variable name and argument name are same so compiler fail to assigning the value.**

**THIS PROBLEM IS SOLVE BY ”This” KEYWORD**

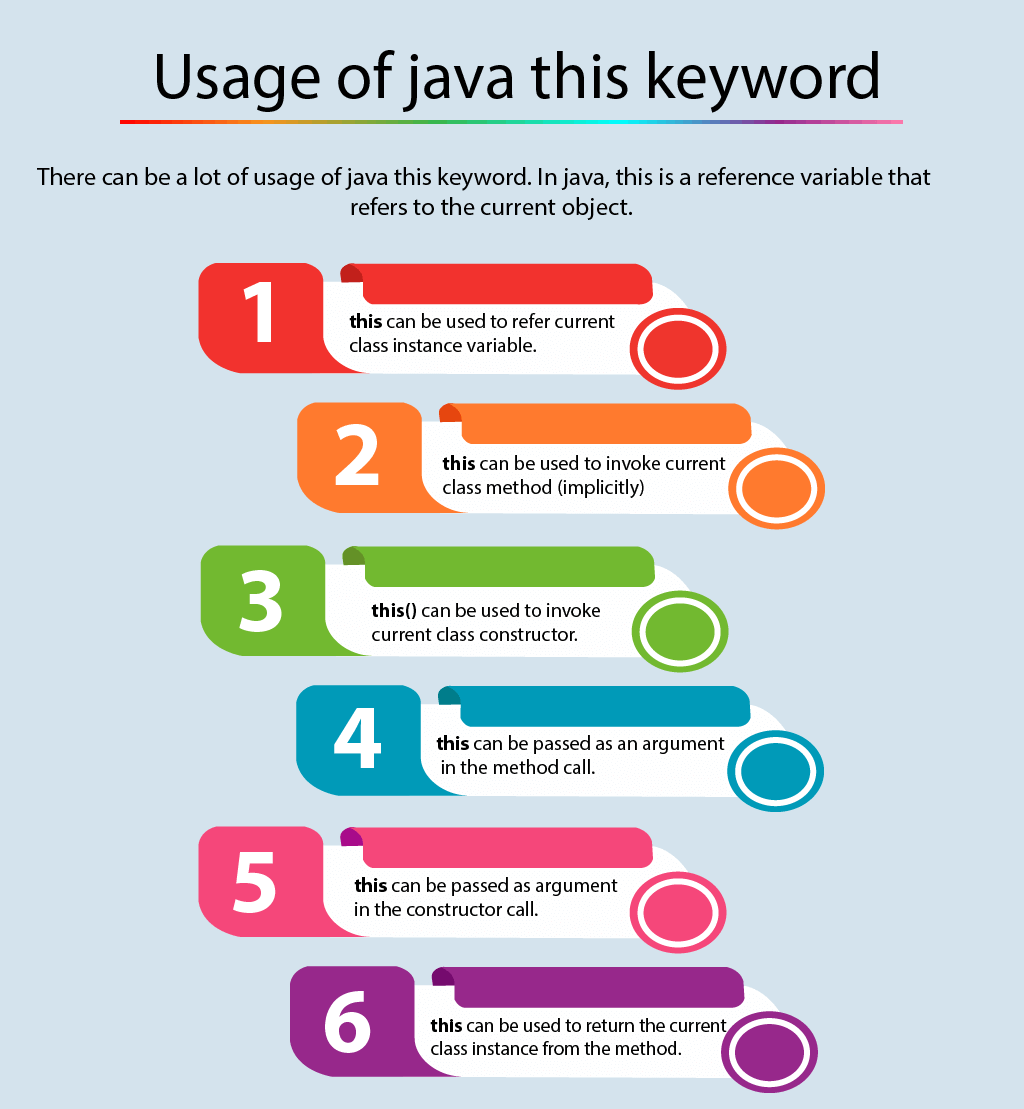
**RULE: “This” keyword only used for instance variable and methods**



* this’ is a reference variable that refers to the current object.



## **Usage of java this keyword**

1. this can be used to refer current **class instance variable**.
2. this can be used to invoke current **class method (implicitly)**
3. this() can be used to invoke **current class constructor**.
4. this can **be passed as an argument in the method call.**
5. this can be passed as argument in the constructor call.
6. this can be used to return the current class instance from the method

### 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.

|  |
| --- |
| /Java code for using 'this' keyword to  //refer current class instance variables  class Test  {      int a;      int b;        // Parameterized constructor      Test(int a, int b)      {          this.a = a;          this.b = b;      }        void display()      {          //Displaying value of variables a and b          System.out.println("a = " + a + "  b = " + b);      }        public static void main(String[] args)      {          Test object = new Test(10, 20);          object.display();      }  } |

Output:

a = 10 b = 20

If local variables(formal arguments) and instance variables are different, there is no need to use this keyword

**class** Student{

**int** rollno;

String name;

**float** fee;

Student(**int** r,String n,**float** f){

rollno=r;

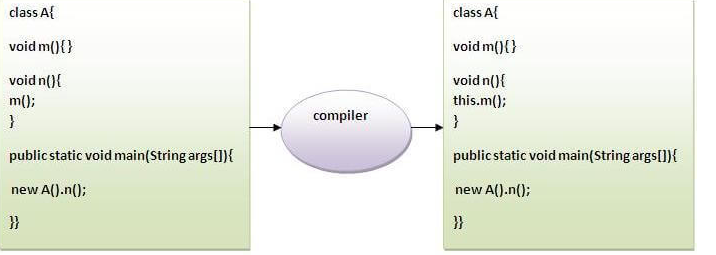
name=n;

fee=f;

}

### Using ‘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.



**class** A{

**void** m(){System.out.println("hello m");}

**void** n(){

System.out.println("hello n");

//m();//same as this.m()

**this**.m();

}

}

**class** TestThis4{

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

A a=**new** A();

a.n();

}}

Output:

hello n

hello m

### 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:**

**class** A{

A(){System.out.println("hello a");}

A(**int** x){

**this**();

System.out.println(x);

}

}

**class** TestThis5{

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

A a=**new** A(10);

}}

Output:

hello a

10

Or **Calling parameterized constructor from default constructor:**

|  |
| --- |
| // Java code for using this() to invoke current class constructor  class Test  {      int a;      int b;      Test()    //Default constructor      {          this(10, 20);          System.out.println("Inside  default constructor \n");      }      test(int a, int b) //Parameterized constructor      {          this.a = a;          this.b = b;          System.out.println("Inside parameterized constructor");      }      public static void main(String[] args)      {          Test object = new Test();      }  } |

Output:

Inside parameterized constructor

Inside default constructor

### Real usage of this() constructor call

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

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

**class** Student{

**int** rollno;

String name,course;

**float** fee;

Student(**int** rollno,String name,String course){

**this**.rollno=rollno;

**this**.name=name;

**this**.course=course;   // constructors

}

Student(**int** rollno,String name,String course,**float** fee){

**this**(rollno,name,course);**//reusing constructor**

**this**.fee=fee;

}

**void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}

}

**class** TestThis7{

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

Student s1=**new** Student(111,"ankit","java");

Student s2=**new** Student(112,"sumit","java",6000f);

s1.display();

s2.display();

}}

Output: null because not provided

111 ankit java null

112 sumit java 6000

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

**class** Student{

**int** rollno;

String name,course;

**float** fee;

Student(**int** rollno,String name,String course){

**this**.rollno=rollno;

**this**.name=name;

**this**.course=course;

}

Student(**int** rollno,String name,String course,**float** fee){

**this**.fee=fee;

**this**(rollno,name,course); //C.T.Error

}

**void** display(){System.out.println(rollno+" "+name+" "+course+" "+fee);}

}

**class** TestThis8{

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

Student s1=**new** Student(111,"ankit","java");

Student s2=**new** Student(112,"sumit","java",6000f);

s1.display();

s2.display();

}}

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

### this: to pass as an argument in the method or **Using ‘this’ keyword as method parameter**

**The this keyword can also be passed as an argument in the method. It is mainly used in the event handling.**

**class** S2{

**void** m(S2 obj){

  System.out.println("method is invoked");

  }

**void** p(){

  m(**this**);

  }

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

  S2 s1 = **new** S2();//  nothing happen

  s1.p();

  }

}

Output:

method is invoked

### Application of this that can be passed as an argument:

In event handling (or) in a situation where we have to provide reference of a class to another one. It is used to reuse one object in many methods.

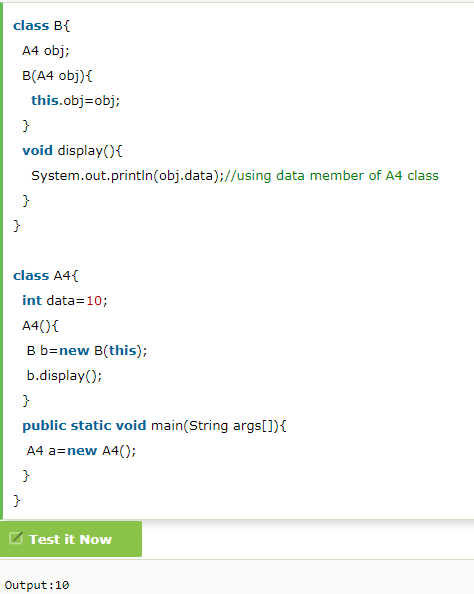
|  |
| --- |
| // Java code for using 'this'  // keyword as method parameter  class Test  {   int a;      int b;  **// Default constructor**      Test()      {   a = 10;          b = 20; }  **// Method that receives 'this' keyword as parameter**      void display(Test obj)      {  System.out.println("a = " + a + "  b = " + b);      }  **//** **Method that returns current class instance**      void get()      {  display(this);      }      public static void main(String[] args)      {   Test object = new Test();          object.get();      } } |

Output:

a = 10 b = 20

### this: to pass as argument in the constructor call

We can pass the this keyword in the constructor also. It is useful if we have to use one object in multiple classes. Let's see the example:



### this keyword can be used to return current class instance

We can return this keyword as an statement from the method. In such case, return type of the method must be the class type (non-primitive). Let's see the example:

### Syntax of this that can be returned as a statement

return\_type method\_name(){

**return** **this**;

}

### Example of this keyword that you return as a statement from the method

**class** A{

A getA(){

**return** **this**;

}

**void** msg(){System.out.println("Hello java");}

}

**class** Test1{

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

**new** A().getA().msg();

}

}

[**Test it Now**](http://www.javatpoint.com/opr/test.jsp?filename=Test1)

Output:

Hello java

### Proving this keyword

|  |
| --- |
| Let's prove that this keyword refers to the current class instance variable. In this program, we are printing the reference variable and this, output of both variables are same. |

**class** A5{

**void** m(){

System.out.println(**this**);//prints same reference ID

}

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

A5 obj=**new** A5();

System.out.println(obj);//prints the reference ID

obj.m();

}  }

Output:

A5@22b3ea59

A5@22b3ea59

**3. Using ‘this’ keyword to return the current class instance**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| //Java code for using 'this' keyword  //to return the current class instance  class Test  {      int a;      int b;        //Default constructor      Test()      {          a = 10;          b = 20;      }        //Method that returns current class instance      Test get()      {          return this;      }        //Displaying value of variables a and b      void display()      {          System.out.println("a = " + a + "  b = " + b);      }        public static void main(String[] args)      {          Test object = new Test();          object.get().display();      }  } |

Output:

a = 10 b = 20

**Using ‘this’ keyword to invoke current class method**

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Java code for using this to invoke current  // class method  class Test {        void display()      {          // calling fuction show()          this.show();           System.out.println("Inside display function");      }        void show() {          System.out.println("Inside show funcion");      }          public static void main(String args[]) {          Test t1 = new Test();          t1.display();      }  } |

Output :