### **Constructor in Java:**

Constructor in java is a special type of method that is used to initialize the object. Java constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rule for construction creation:

* Constructor name must be same as its class name
* Constructor must have no explicit return type.

There are two types of constructors:

* Default constructor (no-arg constructor) JVM
* Parameterized constructor

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

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists. The compiler differentiates these constructors by taking into account the number of parameters in the list and their type.

1. **Does constructor return any value?**

Yes, that is current class instance (You cannot use return type yet it returns a value).

Constructor is not inherited.

1. **What is constructor?**

Constructor is just like a method that is used to initialize the state of an object. It is invoked at the time of object creation.

The default constructor provides the default values to the objects. The java compiler creates a default constructor only if there is no constructor in the class.

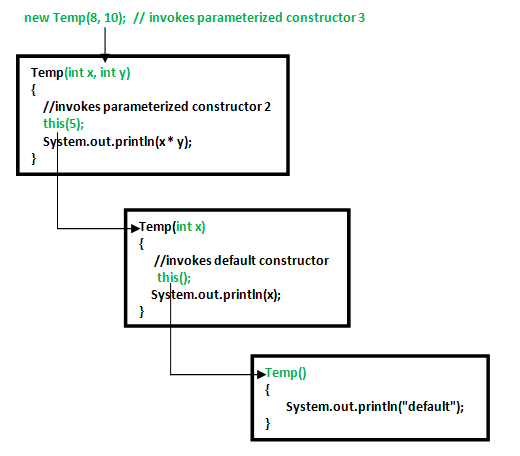
**Is constructor inherited?**

No, constructor is not inherited.

1. **How I can call one constructor in another in same class?**

Constructor chaining is the process of calling one constructor from another constructor with respect to current object.  
Constructor chaining can be done in two ways:

* **Within same class**: It can be done using this() keyword for constructors in same class
* **From base class**: by using super () keyword to call constructor from the base class.



|  |
| --- |
| **package** demo;  // Java program to illustrate Constructor Chaining  // within same class Using this() keyword  **public** **class** Temp {  // default constructor 1  // default constructor will call another constructor  // using this keyword from same class  Temp() {  // calls constructor 2  **this**(5);  System.***out***.println("The Default constructor");  }  // parameterized constructor 2  Temp(**int** x) {  // calls constructor 3  **this**(5, 15);  System.***out***.println(x);  }  // parameterized constructor 3  Temp(**int** x, **int** y) {  System.***out***.println(x \* y);  }  **public** **static** **void** main(String args[]) {  // invokes default constructor first  **new** Temp();  }  }  Output:  75  5  The Default constructor |
| **package** code;  **class** A {  A(**int** i) {  **this**();  System.***out***.print("B");  }  A() {  **this**(5);  System.***out***.print("A");  }  }  **class** Ex6 {  **public** **static** **void** main(String args[]) {  A a = **new** A();  }  }  Output:  Compilation error  Recursive constructor invocation A(int) |

**Rules of constructor chaining:**

* This () expression should always be the first line of the constructor.
* There should be at-least be one constructor without the this () keyword (constructor 3 in above example).
* Constructor chaining can be achieved in any order.

### **Method vs Constructor.**

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

### **This keyword in java:**

Below uses of this keyword in java:

* Invoke current class constructor
* Invoke current class method
* Return the current class object
* Pass an argument in the method call
* Pass an argument in the constructor call

|  |
| --- |
| **package** core\_java\_p3;  **class** Bank {  **private** String id;  **private** String name;  **private** String address;  **public** Bank(String id, String name, String address) {  **this**();  **this**.id = id;  **this**.name = name;  **this**.address = address;  }  **public** Bank() {  System.***out***.println("Default constructor in java");  }    **public** **void** display() {  **this**.add();  System.***out***.println("Id of student=>"+id);  System.***out***.println("Name of student=>"+name);  System.***out***.println("Address of student=>"+address);  }    **public** **void** add() {  System.***out***.println("add method in java");  }  }  **public** **class** ThiKeyword {  **public** **static** **void** main(String[] args) {  Bank b1=**new** Bank("11","Jeevan","Pune");  b1.display();    }  }  Output:  Default constructor in java  add method in java  Id of student=>11  Name of student=>Jeevan  Address of student=>Pune |

### **Static keyword in java:**

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 instance of the class.

Static variable is used to refer the common property of all objects (that is not unique for each object) e.g. company name of employees, college name of students etc.

Static variable gets memory only once in class area at the time of class loading.

**Static final variable initialized only in static block.**

**Can we override static method?**

No, you can't override the static method because they are the part of class not object.

The static can be:

* variable (also known as class variable)
* method (also known as class method)
* block
* nested class

**Static method:**

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

* 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.

|  |
| --- |
| There are two main restrictions for the static method. They are: |
| * The static method cannot use non static data member or call non-static method directly. * this and super cannot be used in static context. | |

**Java static block:**

Is used to initialize the static data member.

It is executed before main method at the time of class loading.

Static variable/method can be inheritated it means we can access method or variable by using the subclass name.

### **What is difference between static (class) method and non static/instance method?**

|  |  |
| --- | --- |
| **Static/ class method** | **Instance method** |
| A method i.e. declared as static is known as static method. | A method i.e. not declared as static is known as instance method |
| Object is not required to call static method. | Object is required to call instance methods |
| Non-static (instance) members cannot be accessed in static context (static method, static block and static nested class) directly. | Static and non-static variables both can be accessed in instance methods. |
| It is not possible to override the static method. | Instance method can be overridden. |
| Public static int add(int x, int y) | Public int add(int x, int y) |

**Static methods/variables are inherited but not overridden below example shows. If we declare the variable in static method then it will allow but if you declare the variable as instance variable then it will give compile time error.**

**By default variable is public static final.**

|  |
| --- |
| **package** com.test;  **interface** A1 {  **int** ***x*** = 20;  }  **class** A2 **implements** A1 {  **int** p;  **public** **static** **int** *y* = 30;  **public** **static** **void** display() {  System.***out***.println("Hello from A2");  }  **public** **static** **void** add() {  **int** x = 20;  **int** y = 30;  System.***out***.println("addition of the x and y--->" + (x + y));    }  }  **class** A3 **extends** A2 {  **public** **static** **int** *z* = 30;  **public** **static** **void** display() {  System.***out***.println("Hello from A3");  }  }  **public** **class** TestStatic **extends** A3 {  **public** **static** **void** main(String[] args) {  // method of the A2 is inheritated.  A2 a = **new** A3();  a.*display*();  // static method add from A2 is inheritated.  A3 b = **new** A3();  b.*add*();  System.***out***.println("x variable of the A1 " + A3.***x***);  System.***out***.println("Y variable of the A2 " + A3.*y*);  System.***out***.println("Z variable of the A2 " + A3.*z*);  }  }  Output:  Hello from A2  addition of the x and y--->50  x variable of the A1 20  Y variable of the A2 30  Z variable of the A2 30 |

Below code give you compile time error because of the super class contains the static method and same method got overridden in subclass which is not static then it will give you compile time error. Vice versa also give the compile time error.

Static method cannot be overridden. Static method is class level method.

Static methods provide the class level copy. Because static method is bound with class wheras instance method is bound with object. Static belongs to class area and instance belongs to heap area.

|  |
| --- |
| **package** code;  **class** Abc{  **public** **void** add()  {  System.***out***.println("Abc class");  }  }  **class** Xyz **extends** Abc{  **public** **static** **void** add()  {  System.***out***.println("Welcome");  }  }  **public** **class** ClassAssert {  **public** **static** **void** main(String[] args) {  Abc abc=**new** Bcz();  abc.add();  }  }  Output:  Compile time error |

**Vice versa**

|  |
| --- |
| **package** code;  **class** Abc{  **public** **static** **void** add()  {  System.***out***.println("Abc class");  }  }  **class** Xyz **extends** Abc{  **public** **void** add()  {  System.***out***.println("Welcome");  }  }  **public** **class** ClassAssert {  **public** **static** **void** main(String[] args) {  Abc abc=**new** Bcz();  abc.*add*();  }  }  **Output:**  compile time error |

### **Static import in java:**

In Java, static import concept is introduced in 1.5 version. With the help of static import, we can access the static members of a class directly without class name or any object. For Example: we always use sqrt() method of Math class by using Math class i.e. **Math.sqrt()**, but by using static import we can access sqrt() method directly.

**Example:**

|  |
| --- |
| **package** core\_java\_p3;  //syntax of static variable import as below  //import static package\_name.class\_name.static\_variable;  **import** **static** java.lang.Math.\*;  **import** **static** java.lang.System.***out***;  **public** **class** StaticImport {  **public** **static** **void** main(String[] args) {  ***out***.println("math function"+*sqrt*(10));  }  }  Output:  math function3.1622776601683795 |

### **Volatile keyword:**

The Java volatile keyword cannot be used with method or class and it can only be used with a variable.  
The value of this variable will **never be cached thread-locally**: all reads and writes will go straight to "main memory".

The Java volatile keyword is used to mark a Java variable as "being stored in main memory". More precisely that means, that every read of a volatile variable will be read from the computer's main memory, and not from the CPU cache, and that every write to a volatile variable will be written to main memory, and not just to the CPU cache.