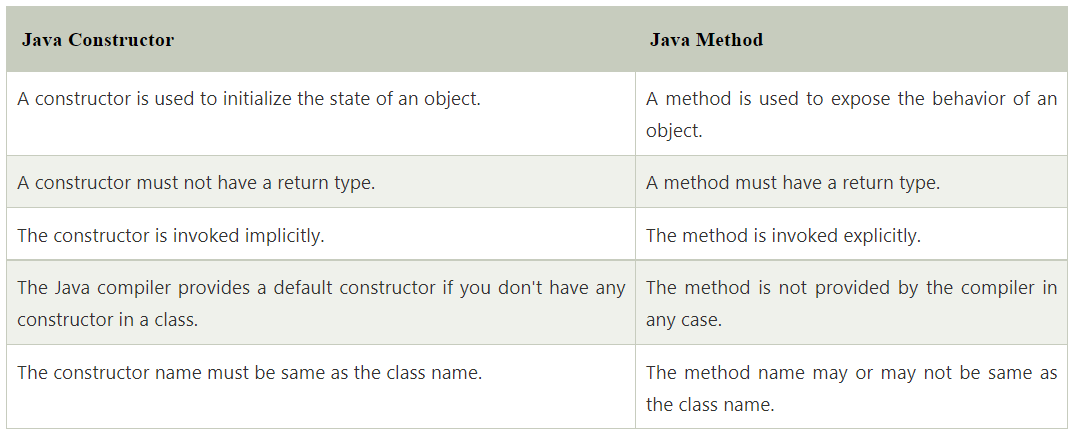
Rules for creating Java constructor

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

### Factory method

It is a method that returns an object to the class to which it belongs. All static methods are factory methods. For example, **NumberFormat obj = NumberFormat.getNumberInstance();**



### Does constructor return any value?

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

### Can constructor perform other tasks instead of initialization?

Yes, like object creation, starting a thread, calling a method, etc. You can perform any operation in the constructor as you perform in the method.

### Is there Constructor class in Java?

Yes.

### What is the purpose of Constructor class?

Java provides a Constructor class which can be used to get the internal information of a constructor in the class. It is found in the java.lang.reflect package.

**Static Keyword :**

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

Main advantage with static variable is that it is memory efficient. If we declare any variable as static it will allocate the memory only once at the time of class loading.

Java static property is shared to all obects.

## **Java 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 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.

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.
2. this and super cannot be used in static context.

### 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](https://www.javatpoint.com/jvm-java-virtual-machine) creates an object first then call main() method that will lead the problem of extra memory allocation.

## **Java static block**

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

1. **class** A2{
2. **static**{System.out.println("static block is invoked");}
3. **public** **static** **void** main(String args[]){
4. System.out.println("Hello main");
5. }
6. }

### 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](https://www.javatpoint.com/java-main-method).

1. **class** A3{
2. **static**{
3. System.out.println("static block is invoked");
4. System.exit(0);
5. }
6. }

## **Usage of Java super Keyword**

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

**Instance Initializer Block :**

**Instance Initializer block** is used to initialize the instance data member. It run each time when object of the class is created.

## **Rules for instance initializer block :**

|  |
| --- |
| There are mainly three rules for the instance initializer block. They are as follows: |

1. The instance initializer block is created when instance of the class is created.
2. The instance initializer block is invoked after the parent class constructor is invoked (i.e. after super() constructor call).
3. The instance initializer block comes in the order in which they appear.

### Q) What is blank or uninitialized final variable?

A final variable that is not initialized at the time of declaration is known as blank final variable.

If you want to create a variable that is initialized at the time of creating object and once initialized may not be changed, it is useful. For example PAN CARD number of an employee.

It can be initialized only in constructor.

### Q) Is final method inherited?

Ans) Yes, final method is inherited but you cannot override it. For Example:

1. **class** Bike{
2. **final** **void** run(){System.out.println("running...");}
3. }
4. **class** Honda2 **extends** Bike{
5. **public** **static** **void** main(String args[]){
6. **new** Honda2().run();
7. }
8. }

### static blank final variable

A static final variable that is not initialized at the time of declaration is known as static blank final variable. It can be initialized only in static block.

1. **class** A{
2. **static** **final** **int** data;//static blank final variable
3. **static**{ data=50;}
4. **public** **static** **void** main(String args[]){
5. System.out.println(A.data);
6. }
7. }

### What is final parameter?

If you declare any parameter as final, you cannot change the value of it.

1. **class** Bike11{
2. **int** cube(**final** **int** n){
3. n=n+2;//can't be changed as n is final
4. n\*n\*n;
5. }
6. **public** **static** **void** main(String args[]){
7. Bike11 b=**new** Bike11();
8. b.cube(5);
9. }
10. }

### Q) Can we declare a constructor final?

No, because constructor is never inherited.

**🡪by default interface variables are in java**

The variable in an interface is **public, static, and final** by default.

🡪**by default interface methods are in java**

Like regular interface methods, default methods are **implicitly public** — there's no need to specify the public modifier. Unlike regular interface methods, they are declared with the default keyword at the beginning of the method signature, and they provide an implementation.

#### **The Java compiler adds public and abstract keywords before the interface method. Moreover, it adds public, static and final keywords before data members.**

Since Java 8, we can have **default and static methods** in an interface.

Since Java 9, we can have **private methods** in an interface.

* can we override default method of **interface** in java ?

you can override a default method of an interface from the implementing class.

->can we override default method of **interface** in java 8 ?

Since Java8 static methods and default methods are introduced in interfaces. Unlike other abstract methods these are the methods can have a default implementation. If you have default method in an interface, **it is not mandatory to override (provide body) it in the classes** that are already implementing this interface.

🡪can we override static method of **interface** in java 8

**You cannot override the static method** of the interface; you can just access them using the name of the interface. If you try to override a static method of an interface by defining a similar method in the implementing interface, it will be considered as another (static) method of the class.

🡪can we override static method in java

Can we Override static methods in java? We can declare static methods with the same signature in the subclass, but it is not considered overriding as there won't be any run-time polymorphism. Hence the answer is **'No'**.