*static* is a non-access modifier in Java which is applicable for the following:

1. blocks
2. variables
3. methods
4. nested classes

To create a static member(block,variable,method,nested class), precede its declaration with the keyword static. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object

**Static block** gets **executed** when a class is loaded into JVM. While init **block** gets copied into the **Constructor** whose object will be created and runs **before** creation of object. **static block** -> Initialization **block** -> and finally **Constructor**

**Static blocks**

If you need to do computation in order to initialize your **static variables**, you can declare a static block that gets executed exactly once, when the class is first loaded

class Test

{

    // static variable

    static int a = 10;

    static int b;

    // static block

    static {

        System.out.println("Static block initialized.");

        b = a \* 4;

    }

    public static void main(String[] args)

    {

       System.out.println("from main");

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

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

    }

}

**Static variables**

When a variable is declared as static, then a single copy of variable is created and shared among all objects at class level. Static variables are, essentially, global variables. All instances of the class share the same static variable.

**Important points for static variables :-**

* We can create static variables at class-level only. See [here](https://www.geeksforgeeks.org/g-fact-47/)
* static block and static variables are executed in order they are present in a program.

Below is the java program to demonstrate that static block and static variables are executed in order they are present in a program.

|  |
| --- |
| // java program to demonstrate execution  // of static blocks and variables  class Test  {      // static variable      static int a = m1();        // static block      static {          System.out.println("Inside static block");      }        // static method      static int m1() {          System.out.println("from m1");          return 20;      }        // static method(main !!)      public static void main(String[] args)      {         System.out.println("Value of a : "+a);         System.out.println("from main");      }      } |

Output:

from m1

Inside static block

Value of a : 20

from main

**Static methods**

When a method is declared with *static* keyword, it is known as static method. The most common example of a static method is *main( )* method.As discussed above, Any static member can be accessed before any objects of its class are created, and without reference to any object.Methods declared as static have several restrictions:

* They can only directly call other static methods.
* They can only directly access static data.
* They cannot refer to [this](https://www.geeksforgeeks.org/this-reference-in-java/) or [super](https://www.geeksforgeeks.org/super-keyword/) in any way.

Below is the java program to demonstrate restrictions on static methods.

**When to use static variables and methods?**

Use the static variable for the property that is common to all objects. For example, in class Student, all students shares the same college name. Use static methods for changing static variables.

**Static nested classes :**We can not declare top-level class with a static modifier, but can declare [nested classes](https://www.geeksforgeeks.org/nested-classes-java/) as static. Such type of classes are called Nested static classes.

Java allows a class to be defined within another class. These are called **Nested Classes**. The class in which the nested class is defined is known as the **Outer Class**. Unlike top level classes, **Inner classes can be Static**. Non-static nested classes are also known as **Inner classes**.

**What are the differences between static and non-static nested classes?**  
The following are major differences between static nested classes and inner classes.

1. A static nested class may be instantiated without instantiating its outer class.
2. Inner classes can access both static and non-static members of the outer class. A static class can access only the static members of the outer class

Instance Initialization Block (IIB) in Java

In a Java program, operations can be performed on methods, constructors and initialization blocks. Instance Initialization Blocks or IIB are used to initialize instance variables. IIBs are executed before constructors. They run each time when object of the class is created.

* Initialization blocks are executed whenever the class is initialized and before constructors are invoked.
* They are typically placed above the constructors within braces.
* It is not at all necessary to include them in your classes.

|  |
| --- |
| // Java program to illustrate  // Instance Initialization Block  class GfG  {      // Instance Initialization Block      {          System.out.println("IIB block");      }        // Constructor of GfG class      GfG()      {          System.out.println("Constructor Called");      }      public static void main(String[] args)      {          GfG a = new GfG();      }  } |

Output :

IIB block

Constructor Called

**Multiple Instance Initialization Blocks in a Program**

We can also have multiple IIBs in a single class. If compiler finds multiple IIBs, then they all are executed from top to bottom i.e. the IIB which is written at top will be executed first.

**Static Block: Execute code once for all objects of a class.**

​

**Initializer block: Execute some code for every object.**