# **Static class,Block,Variable,Method in Java.**

static keyword can be used with class, variable, method and block. Static members belong to the class instead of a specific instance, this means if you make a member static, you can access it without object. Let’s take an example to understand this:

Here we have a static method myMethod(), we can call this method without any object because when we make a member static it becomes class level. If we remove the static keyword and make it non-static then we must need to create an object of the class in order to call it.

Static members are common for all the instances(objects) of the class but non-static members are separate for each instance of class.

class SimpleStaticExample

{

// This is a static method

static void myMethod()

{

System.out.println("myMethod");

}

public static void main(String[] args)

{

/\* You can see that we are calling this

\* method without creating any object.

\*/

myMethod();

}

}

## Static Block

Static block is used for initializing the static variables.This block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which will execute in the same sequence in which they have been written into the program.

### Example 1: Single static block

As you can see that both the static variables were intialized before we accessed them in the main method.

class JavaExample{

static int num;

static String mystr;

static{

num = 97;

mystr = "Static keyword in Java";

}

public static void main(String args[])

{

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

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

}

}

Output:

Value of num: 97

Value of mystr: Static keyword in Java

### Example 2: Multiple Static blocks

Lets see how multiple static blocks work in Java. They execute in the given order which means the first static block executes before second static block. That’s the reason, values initialized by first block are overwritten by second block.

class JavaExample2{

static int num;

static String mystr;

//First Static block

static{

System.out.println("Static Block 1");

num = 68;

mystr = "Block1";

}

//Second static block

static{

System.out.println("Static Block 2");

num = 98;

mystr = "Block2";

}

public static void main(String args[])

{

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

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

}

}

**Output:**

Static Block 1

Static Block 2

Value of num: 98

Value of mystr: Block2

## Java Static Variables

A static variable is common to all the instances (or objects) of the class because it is a class level variable. In other words you can say that only a single copy of static variable is created and shared among all the instances of the class. Memory allocation for such variables only happens once when the class is loaded in the memory.  
Few Important Points:

* Static variables are also known as Class Variables.
* Unlike **non-static variables**, such variables can be accessed directly in static and non-static methods.

### Example 1: Static variables can be accessed directly in Static method

Here we have a static method disp() and two static variables var1 and var2. Both the variables are accessed directly in the static method.

class JavaExample3{

static int var1;

static String var2;

//This is a Static Method

static void disp(){

System.out.println("Var1 is: "+var1);

System.out.println("Var2 is: "+var2);

}

public static void main(String args[])

{

disp();

}

}

**Output:**

Var1 is: 0

Var2 is: null

### Example 2: Static variables are shared among all the instances of class

In this example, String variable is non-static and integer variable is Static. As you can see in the output that the non-static variable is different for both the objects but the static variable is shared among them, thats the reason the changes made to the static variable by object ob2 reflects in both the objects.

class JavaExample{

//Static integer variable

static int var1=77;

//non-static string variable

String var2;

public static void main(String args[])

{

JavaExample ob1 = new JavaExample();

JavaExample ob2 = new JavaExample();

/\* static variables can be accessed directly without

\* any instances. Just to demonstrate that static variables

\* are shared, I am accessing them using objects so that

\* we can check that the changes made to static variables

\* by one object, reflects when we access them using other

\* objects

\*/

//Assigning the value to static variable using object ob1

ob1.var1=88;

ob1.var2="I'm Object1";

/\* This will overwrite the value of var1 because var1 has a single

\* copy shared among both the objects.

\*/

ob2.var1=99;

ob2.var2="I'm Object2";

System.out.println("ob1 integer:"+ob1.var1);

System.out.println("ob1 String:"+ob1.var2);

System.out.println("ob2 integer:"+ob2.var1);

System.out.println("ob2 STring:"+ob2.var2);

}

}

Output:

ob1 integer:99

ob1 String:I'm Object1

ob2 integer:99

ob2 STring:I'm Object2

### **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.
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 creates an object first then call main() method that will lead the problem of extra memory allocation.

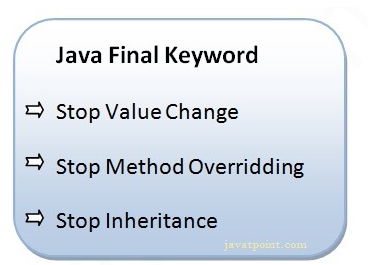
## Static Class

A class can be made **static** only if it is a nested class.

1. Nested static class doesn’t need reference of Outer class
2. A static class cannot access non-static members of the Outer class
3. Declare your class final - Prevents extension of the class since extending a static class makes no sense
4. Make the constructor private - Prevents instantiation by client code as it makes no sense to instantiate a static class
5. Make **all** the members and functions of the class static - Since the class cannot be instantiated no instance methods can be called or instance fields accessed
6. Note that the compiler will not prevent you from declaring an instance (non-static) member. The issue will only show up if you attempt to call the instance member

***top level classes:*** A java project can contain more than one top level classes in each java source file, one of the classes being named after the file name. There are only three options or keywords allowed in front of the top level classes, *public, abstract and final*.

***Inner classes:*** classes that are inside of a top level class are called inner classes, which is basically the concept of nested classes. **Inner classes can be static**. The idea making the inner classes static, is to take the advantage of instantiating the objects of inner classes without instantiating the object of the top level class. This is exactly the same way as the static methods and variables work inside of a top level class.



String Concept in Java

1. **How many objects are created in the following code?**

One of java technical interview questions that substitutes #4.

String s1="Hello";

String s2="Hello";

String s3="Hello";

The answer is “only one” because Java has a String Pool. When we create a String object using the new() operator, it creates a new object in heap memory. If we use String literal syntax, like in our example, it may return an existing object from the String pool, if it already exists.

1. **How many objects are created in the following code?**

String s = **new** String("Hello");

There are 2 objects. One is in string constant pool and the other in heap.

**What is Difference Between String, StringBuilder And StringBuffer Classes in Java ?**

There is one of the leader in top java interview questions.

First of all String is an Immutable class. That means you can’t modify its content once created. While StringBuffer and StringBuilder are mutable classes, so you can change them later. If we change content of String object, it creates a new string therefore it doesn’t modify the original one. That’s why the performance with StringBuffer is better than with String.

The main difference between StringBuffer and StringBuilder that StringBuffer’s methods are synchronized while StringBuilder’s are not.

**Is there any difference in String that was created using literal and with new() operator?**

It is. If we create string with new() operator it appears in heap and not added to the string pool. If you create String using literal it is created in String pool which exists in Perm area of heap.

**What is the difference between static binding and dynamic binding?**

The binding which can be resolved at compile time by compiler is called static or early binding. Binding of all the static, private and final methods is done at compile-time.

In Dynamic binding compiler can’t choose a method to be called. Overriding is a perfect example of dynamic binding. In overriding both parent and child classes have same method.