Java – Static Class, Block, Methods and Variables

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.

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.

```java
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();
    }
}
```
Output:

```
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.

```java
class JavaExample{
    static int num;
    static String mystr;
    static{
        num = 97;
        mystr = "Static keyword in Java";
        System.out.println("Static block");
    }
    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.

```java
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.

```java
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.

```java
class JavaExample{
```

```
    static int var1=77;
    String var2;                    ┌──────────────┐
                                    │   99         │
    public static void main(String args[])
    {                               └──────────────┘
        JavaExample ob1 = new JavaExample();
        JavaExample ob2 = new JavaExample();

         ob1.var1=88;
         ob1.var2="I'm Object1";
         ob2.var1=99;
         ob2.var2="I'm Object2";
         //JavaExample.var1;
         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
```

Java Static Methods

Static Methods can access class variables(static variables) without using object(instance) of the class, however non-static methods and non-static variables can only be accessed using objects.
Static methods can be accessed directly in static and non-static methods.
Syntax:
Static keyword followed by return type, followed by method name.

```
static return_type method_name();
```
Example 1: static method main is accessing static variables without object

```
class JavaExample{
   static int i = 10;
   static String s = "SCASTUDENTS";
   //This is a static method
   public static void main(String args[])
   {
       System.out.println("i:"+i);
       System.out.println("s:"+s);
   }
}
```
Output:

```
i:10
s:SCASTUDENTS
```

Example 2: Static method accessed directly in static and non-static method

```java
class JavaExample{
  static int i = 100;
  static String s = "SCASTUDENTS";
    static void display() //Static method
  {
     System.out.println("i:"+i);
     System.out.println("i:"+s);
  }
   void funcn() //non-static method
  {
          display(); //Static method called in non-static method
  }
   public static void main(String args[])  //static method
  {
          JavaExample obj = new JavaExample();
           obj.funcn(); //You need to have object to call this non-static method

           display(); //Static method called in another static method
    }
}
```
Output:

```
i:100
i:BCASTUDENTS
i:100
i:BCASTUDENTS
```

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

We will see these two points with the help of an example:

Static class Example

```java
class JavaExample{
   private static String str = "MY ADVANCED JAVA BOOK";

   //Static class
   static class MyNestedClass{
       String s="BCA STUDENTS";
        //non-static method
         public void disp() {
```

```
        /* If you make the str variable of outer class
         * non-static then you will get compilation error
         * because: a nested static class cannot access non-
         * static members of the outer class.
         */
        System.out.println(str);

      }
  }
  public static void main(String args[])
  {
      /* To create instance of nested class we didn't need the outer
       * class instance but for a regular nested class you would need
       * to create an instance of outer class first
       */
      JavaExample.MyNestedClass obj = new JavaExample.MyNestedClass();
      obj.disp();
  }
}
```

Output:

```
MY ADVANCED JAVA BOOK
```