What is a static keyword in Java?

In Java, if we want to access class members, we must first create an instance of the class. But there will be situations where we want to access class members without creating any variables.

In those situations, we can use the static keyword in Java. If we want to access class members without creating an instance of the class, we need to declare the class members static.

The Math class in Java has almost all of its members static. So, we can access its members without creating instances of the Math class.

Example:

```
public class Main {
   public static void main( String[] args ) {
      // accessing the methods of the Math class
      System.out.println("Absolute value of -12 =  " + Math.abs(-12));
      System.out.println("Value of PI = " + Math.PI);
      System.out.println("Value of E = " + Math.E);
      System.out.println("2^2 = " + Math.pow(2,2));
   }
}
```

Static Methods

Static methods are also called class methods. It is because a static method belongs to the class rather than the object of a class.

And we can invoke static methods directly using the class name.

Example:

```
class Test {
   // static method inside the Test class
   public static void method() {...}
}

class Main {
   // invoking the static method
   Test.method();
```

```
}
```

Here, we can see that the static method can be accessed directly from other classes using the class name.

In every Java program, we have declared the main method static. It is because to run the program the JVM should be able to invoke the main method during the initial phase where no objects exist in the memory.

Example 1: Java static and non-static Methods

```
class StaticTest {
    // non-static method
    int multiply(int a, int b){
        return a * b;
    }
    // static method
    static int add(int a, int b){
        return a + b;
    }
}

public class Main {
  public static void main( String[] args ) {
        // create an instance of the StaticTest class
        StaticTest st = new StaticTest();
        // call the nonstatic method
        System.out.println(" 2 * 2 = " + st.multiply(2,2));
        // call the static method
        System.out.println(" 2 + 3 = " + StaticTest.add(2,3));
  }
}
```

Static Variables

In Java, when we create objects of a class, then every object will have its own copy of all the variables of the class.

Example:

```
class Test {
    // regular variable
   int age;
}

class Main {
    // create instances of Test
    Test test1 = new Test();
    Test test2 = new Test();
}
```

Here, both the objects test1 and test2 will have separate copies of the variable age. And, they are different from each other.

However, if we declare a variable static, all objects of the class share the same static variable. It is because like static methods, static variables are also associated with the class. And, we don't need to create objects of the class to access the static variables.

Example:

```
class Test {
    // static variable
    static int age;
}
class Main {
    // access the static variable
    Test.age = 20;
}
```

Here, we can see that we are accessing the static variable from the other class using the class name.

Example 2: Java static and non-static Variables

```
class Test {
  // static variable
  static int max = 10;
  // non-static variable
  int min = 5;
}

public class Main {
  public static void main(String[] args) {
    Test obj = new Test();
    // access the non-static variable
    System.out.println("min + 1 = " + (obj.min + 1));
    // access the static variable
    System.out.println("max + 1 = " + (Test.max + 1));
  }
}
```

Note: Static variables are rarely used in Java. Instead, the static constants are used. These static constants are defined by static final keyword and represented in uppercase.

Access static Variables and Methods within the Class

We are accessing the static variable from another class. Hence, we have used the class name to access it. However, if we want to access the static member from inside the class, it can be accessed directly.

Example:

```
public class Main {
  // static variable
  static int age;
  // static method
  static void display() {
    System.out.println("Static Method");
```

```
  }
  public static void main(String[] args) {
    // access the static variable
    age = 30;
    System.out.println("Age is " + age);
    // access the static method
    display();
  }
}
```

Here, we are able to access the static variable and method directly without using the class name. It is because static variables and methods are by default public. And, since we are accessing from the same class, we don't have to specify the class name.

Static Blocks

In Java, static blocks are used to initialize the static variables.

syntax:

```
static {
  // variable initialization
}
```

Example:

```
class Test {
  // static variable
  static int age;
  // static block
  static {
    age = 23;
  }
}
```

- The static block is executed only once when the class is loaded in memory. The class is loaded if either the object of the class is requested in code or the static members are requested in code.
- A class can have multiple static blocks and each static block is executed in the same sequence in which they have been written in a program.

Example: Use of static block in java

```
class Main {
  // static variables
  static int a = 23;
  static int b;
  static int max;
  // static blocks
  static {
     System.out.println("First Static block.");
     b = a * 4;
  }
  static {
     System.out.println("Second Static block.");
     max = 30;
  }

  // static method
  static void display() {
     System.out.println("a = " + a);
     System.out.println("b = " + b);
     System.out.println("max = " + max);
  }

  public static void main(String args[]) {
     // calling the static method
     display();
  }
```

}

Nested Static Class

In Java, we can declare a class inside another class. Such classes are known as nested classes.

Nested classes are of 2 types:

- Static Nested Classes
- Non-static Nested Classes

Example:

```
class OuterClass {
   // static nested class
   static class NestedClass {...}
   // non-static nested class
   class InnerClass {...}
}
```