L39

Thursday, February 3, 2022 11:55 PM

Static fields, Static methods, Static classes

Static fields

Q1. In a Java class, the fields which are marked as static are called **static fields** and those that are **not marked as static** are called as **instance fields** or simply **fields**. For example, in the below code:

```
class A {
        static int a;
        int b;
}
```

a is a **static field**, while b is an **instance field**. Meaning, if we create two or three instances of class A as given below:

```
A a1 = new A();
A a2 = new A();
A a3 = new A();
```

All the three instances a1, a2 and a3 will have their individual copies of **instance field** b. However, all the three instances will share a single copy of the **static field** a.

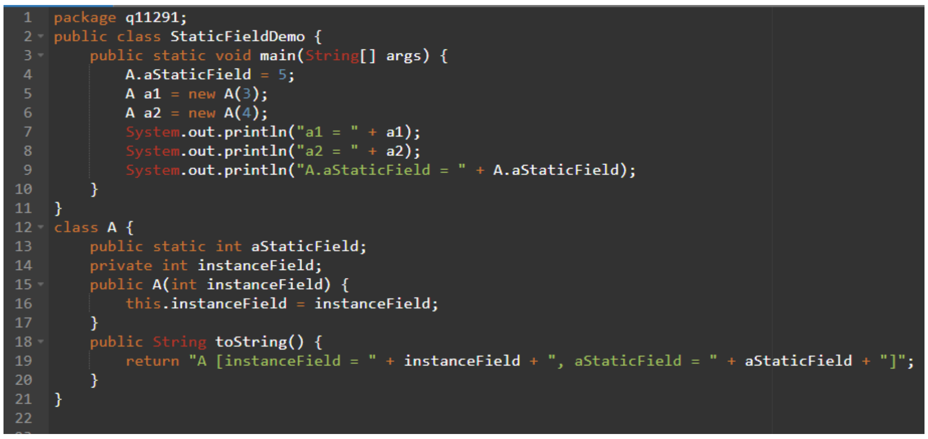
Static fields can be initalized in static initializer blocks.

Instance fields are accessed using the dot . operator on the references of a class. However, the static fields are accessed using the dot . operator directly on the class name and not references (as shown in the below StaticFieldDemo class).

See and retype the below code to understand the difference between static and instance fields.

After execution, in the output you will notice that the value of aStaticField is the same in both a1 and a2, while the values stored in instanceField differ.

Also note how the static field aStaticField is directly accessed using the class name in the statement: A.aStaticField = 5;.



```java
package q11291;
public class StaticFieldDemo {
    public static void main(String[] args) {
        A.aStaticField = 5;
        A a1 = new A(3);
        A a2 = new A(4);
        System.out.println("a1 = " + a1);
        System.out.println("a2 = " + a2);
        System.out.println("A.aStaticField = " + A.aStaticField);
    }
}
class A {
    public static int aStaticField;
    private int instanceField;
    public A(int instanceField) {
        this.instanceField = instanceField;
    }
    public String toString() {
        return "A [instanceField = " + instanceField + ", aStaticField = " + aStaticField + "]";
    }
}
```

Static methods

Q1. In a Java class, the methods which are marked as static are called **static methods** and those that are **not marked as static** are called as **instance methods** or simply **methods**.

If you remember, the main() method is always marked as static so that the **JVM** can directly invoke it without creating an instance of the class which is being executed.

In other words, a method which is marked as static can be called or invoked directly without creating an instance of that **class**.

For example, in the below code:

```
class A {
        static int a;
        int b;
        public static int getA() {
                return a;
        }
        public int getB() {
                return b;
        }
}
```

getA() is a **static method**, while getB() is an **instance method**.

Static methods can access only static fields and other static members like methods and cannot access instance members. If a static method is trying to access an instance member (field or a method), the compiler will throw an error saying the instance field/method cannot be accessed from a static context.

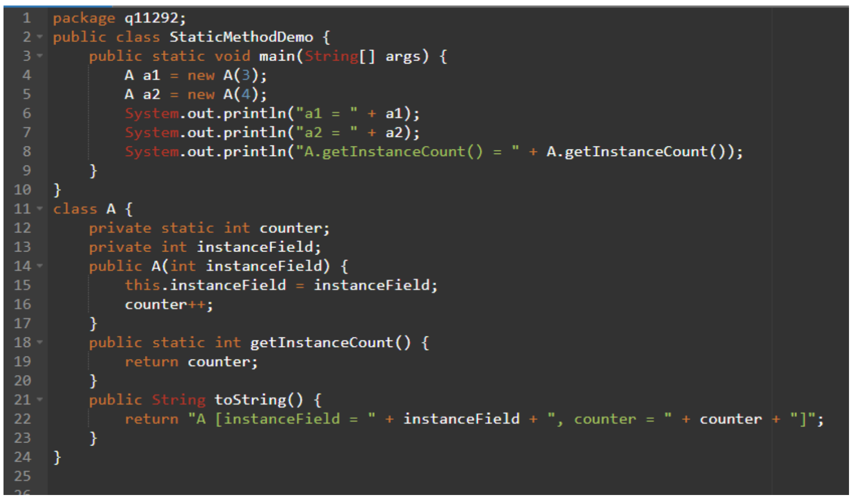
Just like **static fields**, **static methods** too are accessed using the dot . operator on the class name and not on the references like instance methods.

See and retype the below code to understand the usage of static method.

After execution, in the output you will notice that the value of A.getInstanceCount() is the same in both a1 and a2, because both the references a1 and a2 share a single copy of the static field named counter.

In the first invocation of the constructor A() for a1, counter is incremented from its default value 0 to 1. During the second invocation of the constructor for a2, the value of the static field counter is incremented from 1 to 2. Since the increment happens twice on the counter, we see the counter value as 2, when we print a1 using System.out.println("a1 = " + a1);.

Also note how the static method getInstanceCount() is directly accessed using the class name in the statement: System.out.println("A.getInstanceCount() = " + A.getInstanceCount());.



```java
package q11292;
public class StaticMethodDemo {
    public static void main(String[] args) {
        A a1 = new A(3);
        A a2 = new A(4);
        System.out.println("a1 = " + a1);
        System.out.println("a2 = " + a2);
        System.out.println("A.getInstanceCount() = " + A.getInstanceCount());
    }
}
class A {
    private static int counter;
    private int instanceField;
    public A(int instanceField) {
        this.instanceField = instanceField;
        counter++;
    }
    public static int getInstanceCount() {
        return counter;
    }
    public String toString() {
        return "A [instanceField = " + instanceField + ", counter = " + counter + "]";
    }
}
```

Static classes

Q1. A Java class which is marked as static is called a static class.

A static class can be written only inside another class. For example:

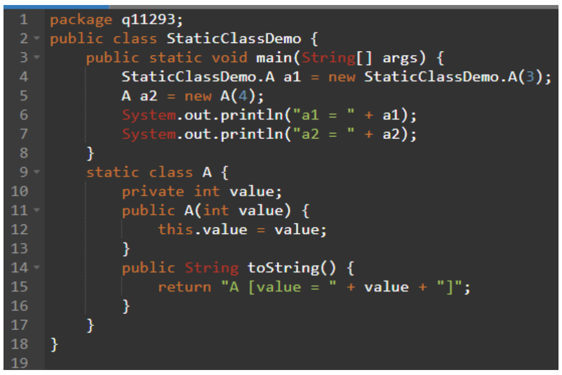
```
class A {
        static class B {
                private int b;
                public B(int b) {
                        this.b = b;
                }
        }
}
```

The static class B is called a **static nested class** since it is nested inside class A.

Just like static fields and static methods, **static nested classes** can be accessed using their other class name. For example instances of class B can be created as below:

```
A.B b = new B();
```

See and retype the below code to understand the syntax of static classes. Since the static nested class A is inside StaticClassDemo, in the main method, we can directly refer A or refer using the context of the outer class as StaticClassDemo.A.



```java
package q11293;
public class StaticClassDemo {
    public static void main(String[] args) {
        StaticClassDemo.A a1 = new StaticClassDemo.A(3);
        A a2 = new A(4);
        System.out.println("a1 = " + a1);
        System.out.println("a2 = " + a2);
    }
    static class A {
        private int value;
        public A(int value) {
            this.value = value;
        }
        public String toString() {
            return "A [value = " + value + "]";
        }
    }
}
```