# Theory: Dealing with modifiers



18 minutes 0 / 4 problems solved



Previously, you've learned how to find all the constructors, methods and fields that were declared in the class by using reflection, even if some of them are private. But what is the point of `Method`, `Field` and `Constructor` classes if you don't know how to use them for more complex actions, that are more difficult than finding out the name of the method? Actually, these classes can do a lot more, and you'll find out in the following topics.

We will start by learning how to deal with modifiers. As you probably have guessed, reflection allows us to find out what modifiers the fields and methods have as well.

## §1. Checking modifiers

Let's consider such a class:

```
class Item {
    public static final int maxItems = 100;
    public static int inStock = 19;

    private String name;
    protected int basePrice;

    public Item(String name, int basePrice) {
        this.name = name;

        this.basePrice = basePrice;

    }



    public String getName() {

        return name;

    }



    public int getPrice() {

        return (int) (basePrice * getMarkUp());

    }



    protected double getMarkUp() {

        double markUp = 0.1;

        // ... connecting to the remote server

        return 1 + markUp;

    }

}
```

As you can see, this class contains different types of fields: `private`, `protected`, `public`, `static` and `final`.

The `Modifier` class is designed to work with such modifiers.

You can get all modifiers by calling the `getModifiers()` method on a `Field`, `Method` or `Constructor` object.

Current topic:

Dealing with modifiers

Topic depends on:

✕ Retrieving Class instances

Topic is required for:

Manipulating fields and methods

Table of contents:

↑ Dealing with modifiers

§1. Checking modifiers

§2. Code example

§3. Conclusion

Feedback & Comments

In fact, this method returns a simple `int` value that represents all the modifiers: the information on them is contained inside the single number. To understand how this works, look at the table below. Each number can be viewed in binary as 32 different bits, each of them being either 0 or 1. Each bit's position is responsible for its own modifier, either being `true` or `false` to indicate the presence or absence of the modifier. The most right bit is for `public`, the second on the right is for `private`, the fourth on the right is for `static` and so on. Don't try to remember it, the table just illustrates the idea:

| Modifiers in source code | Modifiers as an integer |
|---|---|
| public | 00000000 ... 00000001 = 1 |
| private | 00000000 ... 00000010 = 2 |
| protected | 00000000 ... 00000100 = 4 |
| static | 00000000 ... 00001000 = 8 |
| final | 00000000 ... 00010000 = 16 |
| other modifiers | other bits |

The combination of the bits results in a combination of the modifiers. Look at the table below for an example:

| Modifiers in source code | Modifiers as an integer |
|---|---|
| public static | 00000000 ... 00001001 = 9 |
| private final | 00000000 ... 00010010 = 18 |
| protected static final | 00000000 ... 00011100 = 28 |

You don't really need to extract these bits to gather information about a particular modifier. The `Modifier` class has special static methods, such as `isPublic` or `isStatic`, which can check whether a field, method or constructor has a specific modifier.

Apart from access modifiers we have learned, Java has many other modifiers: `synchronized`, `volatile`, `transient`, `native`, `interface`, `abstract`, `strictfp`. Some of them are quite rarely used. In this topic, we will confine ourselves to modifiers from the table above.

## §2. Code example

Let's look at the following code:

```
Item item = new Item("apples", 500);

Class itemClass = item.getClass();
Field[] fields = itemClass.getDeclaredFields();

for (Field field : fields) {
    int modifiers = field.getModifiers();
    if (Modifier.isPublic(modifiers)) {
        System.out.print("public ");

    }

    if (Modifier.isProtected(modifiers)) {

        System.out.print("protected ");

    }

    if (Modifier.isPrivate(modifiers)) {

        System.out.print("private ");

    }

    if (Modifier.isStatic(modifiers)) {

        System.out.print("static ");

    }

    if (Modifier.isFinal(modifiers)) {

        System.out.print("final ");

    }



    System.out.print(field.getType() + " ");

    System.out.println(field.getName());

}
```

This code outputs modifiers of all the fields of the `Item` class, as well as the types of these fields and field names. The output is here:

```
public static final int maxItems
public static int inStock
private class java.lang.String name
protected int basePrice
```

As you can see, the output is as expected. You can also do the same with methods and constructors.

In this topic, we have learned how to get information about the modifiers of any object's fields, methods, and constructors while using reflection. We'll start applying it when dealing with fields and methods in the next topic.

## §3. Conclusion

In this topic you've learned:

- how to test if a method, field, or constructor has a specific modifier. For example, to test if it is `public` or not;
- how modifiers are stored internally.

You will need this knowledge at a higher level of using reflection like getting access to the fields of the object, as well as calling any of the object's methods.

Report a typo

**62** users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (3) Hints (0) Useful links (0) Show discussion