

Theory: Manipulating fields and methods

🕒 20 minutes 0 / 3 problems solved

Skip this topic

Start practicing

554 users solved this topic. Latest completion was about 18 hours ago.

For now, you must already have a basic understanding of the reflection. However, what we did in the previous topics was just a small part of what is possible. In this topic, we will learn how to get fields and their values, write data into fields and call methods by using reflection.

§1. Getting fields values

We will start by explaining how to get the values of particular fields. Let's revise such a class:

```
1 class Item {
2     public static final int maxItems = 100;
3     public static int inStock = 19;
4
5     private String name;
6     protected int basePrice;
7
8     public Item(String name, int basePrice) {
9         this.name = name;
10
11         this.basePrice = basePrice;
12     }
13
14     public String getName() {
15
16         return name;
17     }
18
19     public int getPrice() {
20
21         return (int) (basePrice * getMarkUp());
22     }
23
24     protected double getMarkUp() {
25
26         double markUp = 0.1;
27
28         // ... connecting to the remote server
29
30         return 1 + markUp;
31     }
32 }
```

With the help of the `Field` object we can find out the value of some field of the object. Basically, this is the main purpose of this class. It has a `get` method that takes one argument, which is an object whose field's value we want to get. Note that the `Field` object is not bound to any object of the `Item` class. That's why we need to pass the object directly as an argument. To get the value of the static field you can pass `null` as an argument.

Let's try to output the values of all fields of an `Item` object. At first, we create an object and get a `Class` object for it.

Current topic:

Manipulating fields and methods ...

Topic depends on:

✗ Dealing with modifiers ...

Topic is required for:

Detecting annotations ...

Creating instances ...

Table of contents:

1 Manipulating fields and methods

§1. Getting fields values

§2. Setting values to the fields

§3. Invoking methods

§4. When it works

Feedback & Comments

```

1  Item item = new Item("apples", 500);
2  Class itemClass = item.getClass();
3  Field[] fields = itemClass.getDeclaredFields();

```

Now let's try to use the `get` method for all its fields:

```

1  for (Field field : fields) {
2      System.out.println(field.getName() + " " + field.get(item));
3  }

```

And we get...

```

1  |
java.lang.IllegalAccessException: cannot access a member with modifiers "private"

```

Oops. Java is definitely right, you can't access private fields. Luckily, there's a very simple way around. Java checks if you can access this field, but you can change the accessibility just by calling `setAccessible(true)` method.

Let's improve the code a little:

```

1  for (Field field : fields) {
2      field.setAccessible(true);
3      System.out.println(field.getName() + " " + field.get(item));
4  }

```

And now we've got what we expected:

```

1  maxItems 100
2  inStock 19
3  name apples
4  basePrice 500

```

§2. Setting values to the fields

`Field`'s `set` method works the same way. It takes two arguments: an object and a new value for the field. Again, if you want to set a static field you can pass `null` as the first argument. Below is an example of calling the `set` method. What we try to do here is to set the value to itself:

```

1  for (Field field : fields) {
2      field.setAccessible(true);
3      field.set(item, field.get(item));
4      System.out.println(field.getName() + " " + field.get(item));
5  }

```

And there is another exception...

```

1  |
java.lang.IllegalAccessException: Can not set static final int field to java.lang.
Integer

```

The `final` fields in Java cannot be changed, it is true. But now there is no workaround for this: it would be an even bigger crime in the world of Java if someone changed a final field of an object.

To correct the code, we should make sure that the field is not final by checking its modifier with `isFinal()` method. Since the example above is somewhat useless, we'll also make it more sensible:

```

1  for (Field field : fields) {
2      field.setAccessible(true);
3
4      if (field.getType() == int.class && !Modifier.isFinal(field.getModifiers())) {
5          field.set(item, 0);
6      }
7  }

```

This code resets all non-final integer fields in the instance of some class to 0.

§3. Invoking methods

Invoking methods is similar, but this time the `invoke` method of the `Method` object is used. This method can take a different number of arguments: one more than the called method has. The first argument is the object whose method we want to call or, as you might expect, in case of static methods it's `null`. Take a look at the example:

```
1 Method[] methods = itemClass.getDeclaredMethods();
2 for (Method method : methods) {
3     method.setAccessible(true);
4     System.out.println(method.invoke(item));
5 }
```

All three methods had zero arguments, so `invoke` was called with only one argument. The output may differ because the elements in the returned array are not sorted and are not in any particular order. Here's one of the possible outputs:


```
1 apples
2 1.1
3 550
```

§4. When it works

We have covered a way of accessing information about methods and fields at runtime via reflection. After reading that you may have a question: why do you need to get methods and fields of a class while executing the program if you know all of them at the moment of compiling a program?

One of the typical problems that can be solved by using reflection is the **serialization** of objects. If some class does not implement the `Serializable` interface, it cannot be serialized without using reflection. With reflection, however, all the class fields, even private ones, become visible, so you can write them into an external file. When deserializing, you can read this file and restore all the fields, including private, in it. **Never change the values of private fields in all cases except for deserialization**, because this way there is a high probability to crash the program. To deserialize an object, you must first create an instance of it. You will learn how to do this in the following topics.

 Report a typo

58 users liked this theory.  didn't like it. What about you?



Start practicing

[Comments \(3\)](#)[Hints \(0\)](#)[Useful links \(0\)](#)[Show discussion](#)