


Theory: Creating instances

 13 minutes

0 / 3 problems solved

Skip this topic

Start practicing

444 users solved this topic. Latest completion was 1 day ago.

In previous topics, we've learned how to control the fields of an object and invoke methods of a class with reflection. Reflection can also be used to create instances of a class, and in this topic, we will learn how to do so.

\$1. Acquiring a constructor

Let's take a look at the following code:

```
1  class Item {
2
3      private String name;
4      protected int basePrice;
5
6      public Item() {
7          this("default");
8      }
9
10
11     public Item(String name) {
12
13         this(name, 0);
14     }
15
16
17     public Item(String name, int basePrice) {
18
19         this.name = name;
20
21         this.basePrice = basePrice;
22     }
23
24
25     public String getName() {
26
27         return name;
28     }
29
30
31     public int getPrice() {
32
33         return (int) (basePrice * getMarkUp());
34     }
35
36
37     protected double getMarkUp() {
38
39         double markUp = 0.1;
40
41         // ... connecting to the remote server
42
43         return 1 + markUp;
44     }
45 }
```

As you can see, there are three constructors and one of them is empty.

Current topic:

[Creating instances](#) ...

Topic depends on:

✗ [Manipulating fields and methods](#) ...

In Java, any object must be created by using a constructor. Reflection is no exception: it also requires a constructor to create an instance of a class. Luckily, there is a `Constructor` class that can do this.

Since a class can have multiple constructors you should pick one first. There are two ways to find a suitable constructor:

1. Calling the `getDeclaredConstructors()` method from the `Class` object, and then find a suitable constructor among the existing ones.
2. If you know what types you need to call the constructor with or you need to simply call the constructor without arguments, you can use the `getDeclaredConstructor(...)` method and pass objects of the `Class` type there indicating types of parameters the constructor should have. If you need a constructor without arguments, you don't need to pass anything there.

Once the appropriate constructor has been found, the `newInstance(...)` method should be called to create an instance of the class.

There's also another way to create an object by using the `newInstance()` method on the `Class` object. For example, to create an object of the `Item` class, you could write `Item.class.newInstance()`. Now, this method is deprecated, so it is better to use the other methods discussed above.

§2. Getting constructor parameters

Let's try to use these methods to create an instance of an object. First of all, let's try to output all the constructors and their arguments:

```
1  Class itemClass = Item.class;
2  Constructor[] constructors = itemClass.getDeclaredConstructors();
3
4  for (Constructor constructor : constructors) {
5      Class[] params = constructor.getParameterTypes();
6      System.out.println("Constructor:");
7      if (params.length == 0) {
8          System.out.println("No params");
9          continue;
10     }
11
12     for (Class param : params) {
13         System.out.println(param);
14     }
15
16     System.out.println();
17 }
18 }
```

As you can see, you need to call `getParameterTypes()` method to get the constructor arguments.

The code above prints the following:

```
1  Constructor:
2  class java.lang.String
3  int
4
5  Constructor:
6  class java.lang.String
7
8  Constructor:
9  No params
```

Everything is as expected. There are three constructors: with zero, one and two arguments.

§3. Creating an instance

Since you must know the types of arguments to create an instance, it is always easier to use the `getDeclaredConstructor(...)` method. Below is an example of creating an instance using a constructor with two arguments:

```
1 |
Constructor constructor = itemClass.getDeclaredConstructor(String.class, int.class
);
2 |   Object instance = constructor.newInstance("orange", 990);
3 |
4 |   System.out.println(instance.getClass().getSimpleName());
```

Note that the constructor returns an `Object`, but in this case, it is actually an `Item` object. You can verify this by running the code above, and it will output the following:

```
1 | Item
```

And calling all the methods of this object will also work. Let's invoke the `getName()` method.

```
1 | Method getName = instance.getClass().getDeclaredMethod("getName");
2 | System.out.println(getName.invoke(instance));
```

And you'll get the following output:


```
1 | orange
```

Which indeed means that you are working with the `Item` object.

§4. Conclusion

From this topic, you should know how to use reflection to create instances: first, we get `constructor` object for that class that we want to use; second, we call `newInstance()` on it, passing the arguments. You may also encounter the `newInstance()` method with a similar functions, that is now deprecated.

 Report a typo

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



Start practicing

Table of contents:

- [↑ Creating instances](#)
- [§1. Acquiring a constructor](#)
- [§2. Getting constructor parameters](#)
- [§3. Creating an instance](#)
- [§4. Conclusion](#)
- [Feedback & Comments](#)