Java → Object-oriented programming → Reflection → Reflection basics

Theory: Reflection basics

22 minutes 0 / 5 problems solved

Skip this topic Start practicing

1259 users solved this topic. Latest completion was about 9 hours ago.

§1. Introduction

Reflection is one of the most powerful features in Java. Reflection is the process of **accessing** and **modifying** the application at runtime. It means you can get and modify classes and its members such as constructors, fields, and methods in runtime.

§2. java.lang.reflect package

Java Reflection is implemented by the `java.lang.reflect` package. Although `java.lang.reflect` package includes many interfaces, classes, and exceptions, there are only four classes that you need to know at this level. These classes are:

- **Field**: you can use it to get and modify name, value, datatype and access modifier of a variable.
- **Method**: you can use it to get and modify name, return type, parameter types, access modifier, and exception type of a method.
- **Constructor**: you can use it to get and modify name, parameter types and access modifier of a constructor.
- **Modifier**: you can use it to get information about a particular access modifier.

§3. java.lang.Class

There is another important point. You can't just achieve reflection only with the Reflect package that we've mentioned above. Reflect package can give you information about a field, method or constructor of a class, but first you have to take field list, method list, and constructor list.

This is possible with `java.lang.Class` class and its static `forName()` method. When you pass the name of any class to the `forName()` method, it returns a **Class** object that includes information about this class.

The `java.lang.Class` also has several methods that you can use to get **attributes** (fields, methods, constructors) of the particular class you passed to `forName()` method. Here are some of those methods:

- `forName(String ClassName)`
- `getConstructors()`
- `getDeclaredConstructors()`
- `getFields()`
- `getDeclaredFields()`
- `getMethods()`
- `getDeclaredMethods()`

There are two important things to know about these methods.

First, each of these methods except `forName()`, which we have already discussed, returns an array of objects from `java.lang.reflect` classes. For example, `getFields()` returns an array of objects from `java.lang.reflect.Field` class. After that, you can use methods from `java.lang.reflect` package to get further information about **constructors**, **fields**, and **methods**.

Second, `getConstructors()`, `getFields()` and `getMethods()` return only public constructors, fields and methods from the class represented by the **Class** object. These methods also return **inherited** public fields and methods from **superclasses**.

Similarly, `getDeclaredConstructors()`, `getDeclaredFields()`, `getDeclaredMethods()` return all the constructors, fields and methods from the class represented by the **Class** object. These methods do **not** return inherited fields and methods from the superclasses.

Current topic:

Reflection basics

Topic depends on:

✓ Inheritance Stage 7

Topic is required for:

Retrieving Class instances

Generics and Reflection

Usually, you can see developers use declared methods more often than non-declared methods. You will understand these things better with a code sample. Let's try out a practical example now.

§4. Coding examples

Suppose that you have a class called `Student`. It has three public fields, one protected field, and a private field. It also has a default constructor and a public constructor. `Student` class also has a private method and a public method.

```
public class Student {
    public String firstName;
    public String lastName;
    public int age;
    protected String phoneNumber;
    private String accountNumber;

    Student(){
        System.out.println("This is default Constructor");

    }



    public Student(String firstName, String lastName){

        this.firstName= firstName;

        this.lastName= lastName;

        System.out.println("This is public Constructor");

    }



    private String sanitizeAccountNumber(String accountNumber){


System.out.println("This is a private method to sanitize account number");

        //code to sanitize accountNumber goes here.

        return accountNumber;

    }



    public void setAccountNumber(String accountNumber){

        accountNumber = sanitizeAccountNumber(accountNumber);

        this.accountNumber = accountNumber;

    }

}
```

Reflection process usually has three steps:

1. Get a `java.lang.Class` object of the class using the `forName()` method. In this case, the class we want to reflect is `Student`.

```
Class student = Class.forName("Student");
```

2. Get the class attributes as an array. In this case, we are interested in fields, constructors, and methods.

```
Constructor[] declaredConstructors = student.getDeclaredConstructors();
Constructor[] constructors = student.getConstructors();
Field[] declaredFields = student.getDeclaredFields();
Field[] fields = student.getFields();
Method[] declaredMethods = student.getDeclaredMethods();
Method[] methods = student.getMethods();
```

3. Get the information about class attributes and use it. In this case, we are going to retrieve names of constructors, fields, and methods and print them.

```
for(Constructor dc : declaredConstructors) {
    System.out.println("Declared Constructor " + dc.getName());
}
for (Constructor c : constructors) {
    System.out.println("Constructor " + c.getName());
}
for (Field df : declaredFields) {
    System.out.println("Declared Field " + df.getName());
}
for (Field f : fields) {
    System.out.println("Field " + f.getName());
}
for (Method dm : declaredMethods) {
    System.out.println("Declared Method " + dm.getName());
}
for (Method m : methods) {
    System.out.println("Method " + m.getName());
}
```

You can write these three sections inside the `main()` method and run this code.

§5. Explaining the output

When you run the code above you will get a list of constructors, fields, and methods:

Table of contents:

```
Declared Constructor Student
Declared Constructor Student
Constructor Student
Declared Field firstName
Declared Field lastName
Declared Field age
Declared Field phoneNumber
Declared Field accountNumber
Field firstName
Field lastName
Field age
Declared Method sanitizeAccountNumber
Declared Method setAccountNumber
Method setAccountNumber
Method wait
Method wait
Method wait
Method equals
Method toString
Method hashCode
Method getClass
Method notify
Method notifyAll
```

You can see that `getDeclaredConstructor()` has returned both constructors of the `Student` class while `getConstructors()` has returned only the public constructor. Likewise, `getDeclaredFields()` has returned all fields of the `Student` class while `getFields()` has returned only public fields.

Finally, we print the methods of the `Student` class. As expected, `getDeclaredMethods()` has returned both methods. Now the interesting part is that `getMethods()` has returned some other methods other than `setAccountNumber()` we've expected. If you remember, in one of our previous topics, we've mentioned that `java.lang.Object` class is the **superclass** of all the classes we create. `Object` class has **nine** public methods and all classes we create inherit those methods. That's why you can see nine extra methods in the output.

§6. Summary

Reflection is a way to get information about or modify fields, methods, and constructors of a class. `java.lang.reflect` package and `java.lang.Class` class are essential in Java reflection.

There are three steps in the Java reflection process:

- Get the Class object of the class that you want to reflect on.
- Get the attributes of the class you want to reflect on as a list or array using `java.lang.Class` methods.
- Get information about the particular attribute you got in the second step using the `java.lang.reflect` package.

Reflection is a complicated concept that requires some knowledge of **JVM** and **Java internal processes**. Anyhow, we believe the information we've provided in this topic will help you start using reflection in your projects.

Report a typo

115 users liked this theory. 0 didn't like it. **What about you?**

Start practicing

Comments (8) Hints (0) Useful links (1) Show discussion