T5 - Java Seminar

T-JAV-500

Day 09

Annotations & Reflection

3.0

EPITECH.

Day 09

language: Java

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.

Today, we will focus on two other Java's specificities: **annotations** and **reflection**.

Annotations allow the developers to attach information or even action to a number of things (classes, methods, variables, ...) in an elegant way.

You already used **@Override** annotation in the 6th exercise of the 4th day.

Reflection is a mechanism that allows you to inspect the content of a class or object at runtime.
An example of tool using these features is **Javadoc**, a tool used to generate a documentation of a Java code. It heavily depends on annotations to get information about classes, methods and variables.

We encourage you to try to write some "Javadoc compatible" comments and see for yourself the power of it!

To see the result, you can use the command `javadoc YourClassFile.java`

Exercise 01

File to hand in: ./Inspector.java

Create a class named **Inspector** with:

- an attribute **inspectedClass** of type `Class<T>`
- a constructor taking the inspected class in parameter
- a public method **displayInformations** that will display some informations about the class to inspect (see the example).

For example:

```
Inspector<Number> inspector = new Inspector<>(Number.class);
inspector.displayInformations();
```

```
Terminal
~/T-JAV-500> java Example
Information of the "java.lang.Number" class:
Superclass: java.lang.Object
6 methods:
- byteValue
- shortValue
- intValue
- longValue
- floatValue
- doubleValue
1 fields:
- serialVersionUID
```

Yes, the `Class` class use a generics, so be smart about how you store the field.

For methods and fields we only want the **declared** one. That means the one directly declared by the class (and not his parents).

Exercise 02

File to hand in: ./Inspector.java

Add a **createInstance** method to the previously created `Inspector` class.
It must create a new instance of the **inspectedClass** using the default constructor and return it.
Every exception that may occur must be rethrown by **createInstance**.

EXERCISE 03

File to hand in: ./Test.java

We will create a *very* small test framework using annotations.
The first step is to create an annotation that will be used to mark the methods to call for the testing.

Create an annotation called **Test** with the following properties:

- it must be available at runtime,
- we should only be able to use it on methods,
- it should have two fields:
    - **name**: a string containing the name of the test
    - **enabled**: a boolean indicating if the test is enabled or not (true by default).

EXERCISE 04

Files to hand in: ./Test.java
./TestRunner.java

Create a class named **TestRunner** with a **runTests** method.
This method takes a class in parameter and executes every method of this class that is annotated with **@Test**.

runTests should only execute methods where **@Test** is present and its property **enabled** is true.

It should also display the name of the executed test before running the method.

Be smart and combine concepts you have seen up to this point.

EXERCISE 05

Files to hand in: ./Before.java
./After.java
./BeforeClass.java
./AfterClass.java

Create four new annotations `Before`, `After`, `BeforeClass`, `AfterClass`, that must be available at runtime for methods only.
They do not have any parameters.

These annotations don't do anything so far.

Exercise 06

Files to hand in: ./Before.java
./After.java
./BeforeClass.java
./AfterClass.java
./Test.java
./TestRunner.java

Change your public method **runTests** of **TestRunner** to add these features:

- before each test, you must execute the method annotated with **@Before** (if such a method exists),
- after each test, you must execute the method annotated with **@After** (if such a method exists),
- before any test is executed, you must execute the method annotated with **@BeforeClass** (if such a method exists),
- after any test is executed, you must execute the method annotated with **@AfterClass** (if such a method exists).

Congratulations, you have a way to launch test sequences.

Going further

You have now created a basic Test Framework (even if you can't tell for yourself if a test is valid or not).
To see what more you could do or just to read further informations about a real Test Framework, have a look at JUnit.