EPITECH. TECHNOLOGY

JAVA SEMINAR

DAY 09 - ANNOTATIONS & REFLECTION

JAVA SEMINAR

Contents

Today, we will focus on two other Java's specificities:

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

✓ **Reflection** is a mechanism that allows you to inspect the content of a class or object at runtime.

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

Javadoc is a tool that uses these features and generates a documentation of Java code.
It heavily depends on annotations to get information about classes, methods and variables.

We encourage you to write some *Javadoc compatible* comments.
To see its powerful results, you can use the command `javadoc YourClassFile.java`

Exercise 01

Delivery: `./Inspector.java`

Create a `Inspector` class with:

- ✓ an attribute `inspectedClass` of type `Class<T>` ;
- ✓ a constructor taking the inspected class in parameter ;
- ✓ a public method `displayInformations` that displays some info about the class to inspect.

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
```

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 its parents).

Exercise 02

Delivery: `./Inspector.java`

Add a `createInstance` method to the previously created class that:

- ✓ creates a new instance of the `inspectedClass` using the default constructor ;
- ✓ returns it.

Every exception that may occur must be rethrown by `createInstance`.

Exercise 03

Delivery: `./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

Delivery: `./Test.java`, `./TestRunner.java`

Create a class named `TestRunner` with a `runTests` method that:

- ✓ takes a class in parameter ;
- ✓ 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

Delivery: `./Before.java`, `./After.java`, `./BeforeClass.java`, `./AfterClass.java`

Create four new annotations:

- ✓ `Before` ;
- ✓ `After` ;
- ✓ `BeforeClass` ;
- ✓ `AfterClass`.

Each annotation must be available at runtime for methods only.

They do not have any parameters.

These annotations don't do anything so far.

Exercise 06

Delivery: `./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` ;
- ✓ after each test, you must execute the method (*) annotated with `@After` ;
- ✓ before any test is executed, you must execute the method (*) annotated with `@BeforeClass` ;
- ✓ 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 created a basic Test Framework.

Even if you can't tell for yourself if a test is valid or not!

Have a look at JUnit to dig deeper a real Test Framework and what it can do.

v 3.1