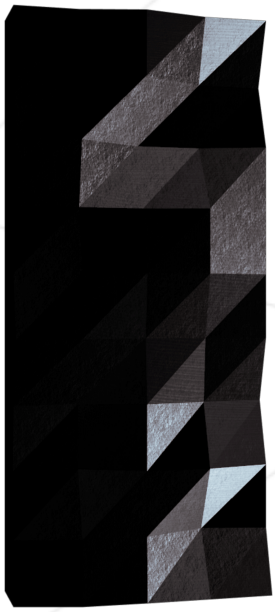

Module 07 – Piscine Java

Reflection

Summary: Today you will develop your own frameworks that use the reflection mechanism

Contents

Chapter I

Foreword

Reflection is a powerful mechanism that ensures the operation of frameworks (such as Spring or Hibernate). Knowledge of Java Reflection API operation principles guarantees the correct use of various technologies for implementing corporate systems.

Reflection tool enables to flexibly use class information during runtime, as well as dynamically change the state of objects without using this information in writing the source code.

One of reflection capabilities is modifying private field values from outside. We may ask then whether this contradicts the encapsulation principle, and the answer is no :)

Chapter II

Instructions

- Use this page as the only reference. Do not listen to any rumors and speculations about how to prepare your solution.
- Now there is only one Java version for you, 1.8. Make sure that compiler and interpreter of this version are installed on your machine.
- You can use IDE to write and debug the source code.
- The code is read more often than written. Read carefully the document where code formatting rules are given. When performing each task, make sure you follow the generally accepted Oracle standards
- Comments are not allowed in the source code of your solution. They make it difficult to read the code.
- Pay attention to the permissions of your files and directories.
- To be assessed, your solution must be in your GIT repository.
- Your solutions will be evaluated by your piscine mates.
- You should not leave in your directory any other file than those explicitly specified by the exercise instructions. It is recommended that you modify your .gitignore to avoid accidents.
- When you need to get precise output in your programs, it is forbidden to display a precalculated output instead of performing the exercise correctly.
- Have a question? Ask your neighbor on the right. Otherwise, try with your neighbor on the left.
- Your reference manual: mates / Internet / Google. And one more thing. There's an answer to any question you may have on Stackoverflow. Learn how to ask questions correctly.
- Read the examples carefully. They may require things that are not otherwise specified in the subject.
- Use "System.out" for output

- And may the Force be with you!
- Never leave that till tomorrow which you can do today ;)

Chapter III

Exercise 00 : Work with Classes

	Exercise 00
Work with Classes	
Turn-in directory : $ex00/$	
Files to turn in : Reflection-folder	
Allowed functions : All	

Now you need to implement a Maven project that interacts with classes of your application. We need to create at least two classes, each having:

- private fields (supported types are String, Integer, Double, Boolean, Long)
- public methods
- an empty constructor
- a constructor with a parameter
- toString() method

In this task, you do not need to implement get/set methods. Newly created classes must be located in a separate classes package (this package may be located in other packages). Let's assume that the application has User and Car classes. User class is described below:

```
public class User {
    private String firstName;
    private String lastName;
    private int height;

    public User() {
        this.firstName = "Default first name";
        this.lastName = "Default last name";
        this.height = 0;
    }

    public User(String firstName, String lastName, int height) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.height = height;
```

```
    }

    public int grow(int value) {
        this.height += value;
        return height;
    }

    @Override
    public String toString() {
        return new StringJoiner(", ", User.class.getSimpleName() + "[", "]")
                .add("firstName='" + firstName + "'")
                .add("lastName='" + lastName + "'")
                .add("height=" + height)
                .toString();
    }
}
```

The implemented application shall operate as follows:

- Provide information about a class in classes package.
- Enable a user to create objects of a specified class with specific field values.
- Display information about the created class object.
- Call class methods.

Example of program operation:

```
Classes:
User
Car
---------------------
Enter class name:
-> User
---------------------
fields :
    String firstName
    String lastName
    int height
methods:
    int grow(int)
---------------------
Let's create an object.
firstName:
-> UserName
lastName:
-> UserSurname
height:
-> 185
Object created: User[firstName='UserName', lastName='UserSurname', height=185]
---------------------
Enter name of the field for changing:
-> firstName
Enter String value:
-> Name
Object updated: User[firstName='Name', lastName='UserSurname', height=185]
---------------------
Enter name of the method for call:
-> grow(int)
Enter int value:
-> 10
Method returned:
195
```

- If a method contains more than one parameter, you need to set values for each one

- If the method has void type, a line with returned value information is not displayed
- In a program session, interaction only with a single class is possible; a single field of its object can be modified, and a single method can be called
- You may use throws operator.

Chapter IV

Exercise 01 : Annotations – SOURCE

	Exercise 01
Annotations – SOURCE	
Turn-in directory : $ex01/$	
Files to turn in : Annotations-folder	
Allowed functions : All	

Annotations allow to store metadata directly in the program code. Now your objective it to implement HtmlProcessor class (derived fromAbstractProcessor) that processes classes with special @HtmlForm and @Htmlnput annotations and generates HTML form code inside the target/classes folder after executing mvn clean compile command. Let's assume we have UserForm class:

```
@HtmlForm(fileName = "user_form.html", action = "/users", method = "post")
public class UserForm {
    @HtmlInput(type = "text", name = "first_name", placeholder = "Enter First Name")
    private String firstName;

    @HtmlInput(type = "text", name = "last_name", placeholder = "Enter Last Name")
    private String lastName;

    @HtmlInput(type = "password", name = "password", placeholder = "Enter Password")
    private String password;
}
```

Then, it shall be used as a base to generate "user_form.html" file with the following contents:

```
<form action = "/users" method = "post">
<input type = "text" name = "first_name" placeholder = "Enter First Name">
<input type = "text" name = "last_name" placeholder = "Enter Last Name">
<input type = "password" name = "password" placeholder = "Enter Password">
<input type = "submit" value = "Send">
</form>
```

- @HtmlForm and @HtmlInput annotations shall only be available during compilation.
- Project structure is at the developer's discretion.

- To handle annotations correctly, we recommend to use special settings of maven-compiler-plugin and auto-service dependency on com.google.auto.service.

Chapter V

Exercise 02 : ORM

	Exercise 02
ORM	
Turn-in directory : $ex02/$	
Files to turn in : ORM-folder	
Allowed functions : All	

We mentioned before that Hibernate ORM framework for databases is based on reflection. ORM concept allows to map relational links to object-oriented links automatically. This approach makes the application fully independent from DBMS. You need to implement a trivial version of such ORM framework.

Let's assume we have a set of model classes. Each class contains no dependencies on other classes, and its fields may only accept the following value types: String, Integer, Double, Boolean, Long. Let's specify a certain set of annotations for the class and its members, for example, User class:

```
@OrmEntity(table = "simple_user")
public class User {
    @OrmColumnId
    private Long id;
@OrmColumn(name = "first_name", length = 10)
private String firstName;
@OrmColumn(name = "first_name", length = 10)
private String lastName;
    @OrmColumn(name "age")
private Integer age;

    // setters /getters
}
```

OrmManager class developed by you shall generate and execute respective SQL code during initialization of all classes marked with @OrmEntity annotation. That code will contain CREATE TABLE command for creating a table with the name specified in the annotation. Each field of the class marked with @OrmColumn annotation becomes a column in this table. The field marked with @OrmColumnId annotation indicates that an auto increment identifier must be created. OrmManager shall also support the following set of operations (the respective SQL code in Runtime is also generated for each of them):

```
public void save(Object entity)
public void update(Object entity)
public <T> T findById(Long id, Class<T> aClass)
```

- OrmManager shall ensure the output of generated SQL onto the console during execution.
- In initialization, OrmManager shall remove created tables.
- Update method shall replace values in columns specified in the entity, even if object field value is null.