The purpose of this assignment is to practice the use of the reflection API.

Exercise 1 – Hierarchies

Open the `hierarchies/` folder in you IDE. It contains the skeleton of the main class to implement and some support classes to be used for this exercise. The goal is to write 3 utility methods that provide the class hierarchy of any java class using Java's reflection API.

Write a class that implements the following three methods:

1) **`getPathToObject(String className)`**
   that returns a list of Strings representing all the classes in the class hierarchy, from *className* to *Object* (e.g. [F1Car, RacingCar, Car, Vehicle, java.lang.Object]).
2) **`getPathToClass(String startClassName, String endClassName)`**
   that returns a list of Strings representing all the classes in the class hierarchy from *startClassName* to *endClassName*; if the path does not exist an empty list should be returned.
3) **`getCommonAncestor(String className0, String className1)`**
   that returns the class name of the common superclass between *className0* and *className1*.

You also have to implement a method that tests all 3 methods using the provided example classes.

Exercise 2 – Class encapsulator

Open the `CodeGen/Target.java` file in your IDE. The goal of this exercise is to generate the source code of an encapsulated version of the given Target class using reflection.
The program should therefore:

- use reflection to read all fields of the Target class
- generate for each field:
  - the private field with its value initialization
  - the corresponding getter and setter methods, combining the field's name and camelCase naming conventions

The resulting source code of the encapsulated class should be printed to the console.

Example:

Target class

```
public class Target {
    int theAnswer = 42;
    public String hello = "world";
}
```

Desired output to console

```
public class Target {
    private int theAnswer = 42;
    private String hello = "world";

    public int getTheAnswer() {
        return theAnswer;
    }

    public String getHello() {
        return hello;
    }

    public void setTheAnswer(int theAnswer) {
        this.theAnswer = theAnswer;
    }

    public void setHello(String hello) {
        this.hello = hello;
    }
}
```

Exercise 3 – Class markdown documentation

Open the `markdown/DocumentationHelper.java` file in your IDE and use it as a starting point for implementing a program that helps generating a template for documenting all **public constructors and methods** of a java class in markdown format. Make sure to consider also the parameters.
A possible template for the desired output could be:

```
# ClassName

## Constructors

### `Constructor()`
TODO describe constructor

### `Constructor(float, float)`

Parameters:
- `float arg0`: TODO describe parameter
- `float arg1`: TODO describe parameter

TODO describe constructor

## Methods

### `int method(float) `

Parameters:
- `float arg0`: TODO describe parameter

TODO describe method

...
```

A possible output of the *Coordinate* class provided in the *DocumentationHelper* class could be:

```
# `markdown.Coordinate`

## Constructor(s)

### `markdown.Coordinate()`

TODO describe constructor

### `markdown.Coordinate(float, float)`

#### Parameters:
 - `float arg0`: TODO describe parameter.
 - `float arg1`: TODO describe parameter.

TODO describe constructor

## Methods(s)

### `float getLat()`

TODO describe method

### `float getLon()`

TODO describe method

### `java.lang.String toString()`

TODO describe method

### `double distance(markdown.Coordinate)`

#### Parameters:
 - `markdown.Coordinate arg0`: TODO describe parameter.

TODO describe method
```