Java - elements of OOP (I)

Working environment setup

1. Download and unzip `lab01` source code
    1. Download `lab01.zip` from the course site (moodle)
    2. Unzip it (you get `lab01` directory)
    3. Move `lab01` to `programming-in-java` directory, i.e.,
        - `programming-in-java`
            - `lab00`
            - `lab01` <—
            - `gradle`
            - …
2. [`IntelliJ`] Add `lab01` module to the `programming-in-java` project
    1. In the *Project* window click `settings.gradle` file to open it
    2. Modify its content to the following:

        ```
        rootProject.name = 'programming-in-java'
        include 'lab00'
        include 'lab01'
        ```

    3. Save the file
    4. Click `Load Gradle Changes` (a small box in the top right corner)
    5. In the *Project* window, check if `lab00` and `lab01` have the same icons (i.e., they are both seen by *IntelliJ* as modules)
    6. In the *Gradle* window, click:
        - `programming-in-java`
            - `Tasks`
                - `verification`
                - `test`

        to run all the tests

0) JUnit 5

Look briefly at the JUnit 5 User Guide

1) Concepts of encapsulation, inheritence, and polymorphism

Analyse the source code in packages:

- `lst01_01` (encapsulation)
- `lst01_02` (inheritance)
- `lst01_03` (sub-type/inclusion polymorphism)

Exercises

1. Explain the concept of *encapsulation* and the way it is implemented in Java
2. Explain the following concepts:
    - mutator method (setter)
    - accessor method (getter)
3. Explain two different meanings/roles of:
    - `this`
    - `super`
4. Explain the concept of *inheritence* and the way it is implemented in Java
5. Explain the concept of *polymorphism*, name its three main kinds/forms, and explain the way they are implemented in Java
6. Explain the relationship between inheritance and sub-type/inclusion polymorphism
7. Read Composition vs. Inheritance: How to Choose?
8. In the analysed code identify *testable* methods and write a couple of unit tests for them (the IDE can help with it)

2) Static members (variables/constants and methods)

Analyse the source code in package `lst01_04`

Exercises

1. Explain the following concepts:
    - static variable (field/class member)
    - static constant
    - static method
2. Explain why *static constants* often have *public* visibility
3. Explain why static methods do not have access to instance members (methods and fields)
4. Give one example of a static method application

3) Constructors, factory methods, and singletons

Analyse the source code in packages:

- `lst01_05` (object initialisation process)
- `lst01_06` (order of constructor calls)
- `lst01_07` (simple factory method)
- `lst01_08` (singleton example implementations)

Exercises

1. Describe the *object initialisation process* for a class derived from the `Object` class (including default values for different types of fields/variables, static variables, static constants, anonymous static blocks, anonymous blocks, constructors)
2. For class `D9` from (defined in `ClassFamily.java`):
    1. draw the class (inheritence) diagram
    2. explain the seqence of the constructor calls
3. Compare capabilities of constructors and factory methods
4. Give at least two applications of the *singleton pattern*
5. Write a couple of unit test (JUnit 5) for singletons from `lst01_08`

4) Immutable objects/classes and Java Records

Analyse the source code in package `lst01_09`

Exercises

1. Explain a strategy for defining immutable objects
2. Compare the concepts of the *immutable object* and *immutable class*
3. Explain the advantages of *immutable objects*
4. Give at least two uses of the *Java Records*
5. Write a couple of unit tests to for `HelloImmutable` and `HelloJavaRecord`

5) Overriding hashCode, equals, and toString

Analyse the source code in package `lst01_10`

Exercises

1. Explain the difference between `==` operator and `equals` method in Java (consider primitive and reference types)
2. Explain the following formula o1.equals(o2) $\implies$ hasCode(o1) == hashCode(o2)
3. Familiarize yourself with the Java Object class
4. Explain the general contract of `hashCode` and `equals`
5. Generate JavaDOC documentation for the project (*hint*: `Tools > Generate JavaDoc`)

6) Push the commits to the remote repository