# WEB PROGRAMMING

Academic year 2023/2024


## ASSIGNMENT 1: INTRODUCTION TO JAVA

In this assignment we will be introduced to Java development, also incorporating design patterns to follow the best practices in software development, which are the hallmark of good professionals. In a team of four people, which will remain the same during the academic year, we will make use of the usual elements of the language, as well as console and file I/O, property file management and design pattern development. Please note the following **normative aspects** concerning this assignment:

- A report in PDF format (maximum 2 pages) will be submitted, explaining and justifying the design and implementation decisions taken by the team, as well as the list of sources consulted, and an analysis of the difficulties encountered.
- The code must follow good naming practices (variables, packages, etc.) and be documented using Javadoc at class and method level:

  https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
- The proposal of exercises has a margin of decision by the team. As these decisions are expected to be different for each exercise team, they should be justified accordingly in the report.
- Only one member of the team will submit the assignment through Moodle. The person who submits this assignment will do the same with the other assignments. A ZIP file must be uploaded to the Moodle task with the Eclipse project (preferably) or Java sources of the assignment, the executables of each exercise (JAR) and the report in the root of the file. The submitted ZIP file must be named as follows:

  GM5_<*login*>.zip, where <*login*> is the UCO *nick* of the team member who submit the assignment.

Not complying with the delivery rules (content, naming, etc.) will imply the consideration of non-submitted assignment. The due date will be announced in the corresponding Moodle task. Likewise, late submission will result in the assignment not being graded.

**IMPORTANT**: No late submissions or submission received by other channel rather than Moodle (e.g., email) will be accepted.

## INTRODUCTION TO THE COURSE PROJECT

The three assignments of the course will be aimed at the development of a web application for the management summer camps. This management will focus on allowing camp registrations for different age groups, as well as the organisation of the activities that form part of the camp. In each assignment, a series of exercises will be proposed so that, incrementally, the classes and artefacts (files, database, etc.) necessary to develop the project can be built. The division into exercises will facilitate the organisation of the work during the practical sessions, but the student should keep in mind that they all form part of a single project.

## EXERCISE 1

In this exercise, the Java classes needed to represent the concepts of the application domain will be developed. Specifically, the following classes must be designed and implemented:

**Class Participant**: A class representing a person attending the summer camp. The class must contain the following attributes:

- Person identifier, using integer type.
- First name and surname, using string type.
- Date of birth (to manage age groups).
- A boolean value to specify if special attention is require or not.

As for methods, the class should provide the following ones:

- Empty constructor (without parameters).
- Parameterised constructor, whose parameters are all the attributes of the class.
- get/set methods for all attributes.
- toString method to print the information of the participant.

**Class Monitor**: This class represents a person who supervises the activities taking place in the camp. The class must contain the following attributes:

- Identifier of the person, using integer type.
- First name and surname, using string type.
- Whether or not he/she is a special educator (type Boolean), to indicate if he/she is eligible for camps where attendees requiring special attention participate.

Also, the class should implement the following methods:

- Empty constructor (without parameters).
- Parameterised constructor, whose parameters are all the attributes of the class.
- get/set methods for all attributes.

- toString method to print the monitor's information.

**Class Activity**: This class represents an activity that is part of the organisation of a camp. The class must contain the following attributes:

- Name of the activity, which shall be unique, of type string.
- Educational level, using an enumeration, to designate the recommended age group among children (4-6 years), youth (7-12 years) and teenager (13-17 years).
- Timetable of the activity, to indicate whether it takes place in the morning or in the afternoon.
- Maximum number of participants in the activity, as an integer value.
- Number of monitors required.
- Monitor(s) assigned to the activity (see Monitor class).

Also, the class should provide the following methods:

- Empty constructor (without parameters).
- Parameterised constructor, whose parameters are all the attributes of the class except the list of monitors, which shall be initialised empty.
- get/set methods for all attributes.
- toString method to print the activity information.
- Method *associateMonitor*, which adds a monitor to the list of monitors if the maximum number of monitors required has not been reached.

**Class Camp**: This class represents the organisation of a set of activities for a limited period of time. The class must contain the following attributes:

- Camp identifier, of type integer.
- Start and end dates of the camp, of type date.
- Educational level, equal to those defined for the activities.
- Maximum number of participants, of type integer.
- List of activities that make up the camp (see Activity class).
- Monitor(s) responsible for the camp (see Monitor class).

In addition, the class must provide the following methods:

- Empty constructor (without parameters).
- Parameterised constructor, whose parameters are all the attributes of the class except the lists of activities and the responsible monitor.
- get/set methods for all attributes.
- toString method that prints the camp information.

- Method *associateActivity*, which allows to add an activity to the camp if the activity is of the same educational level as the camp.
- Method *associateMonitor*, which allows to establish the responsible monitor among those who are in charge of one of the activities that make up the camp.
- Method *associateSpecialMonitor*, which allows the establishment of a second monitor responsible among those identified as special attention monitors. This monitor cannot be associated with any of the activities that make up the camp, as it is a reinforcement that will support all of them.

*Help*: To manage dates, we recommend reading the official Java API documentation and more specifically, classes Date and DateFormat.

## EXERCISE 2

This exercise deals with the definition and implementation of the camp registration management, using the Factory pattern for its creation. The first step will be the creation of the registration class, following the following guidelines:

**Class Registration:** This class represents the registration of a participant to a camp. For each registration, some general information must be stored:

- Identifier of the participant registering, which must already exist.
- Identifier of the camp being registered, which must exist.
- Date on which the registration was made, using a date type.
- Price of the registration, expressed in euros, using a float type.

On the other hand, there are different types of registration:

- Full registration: The participant registers for the camp on a full-time basis, i.e., for all the activities included in the camp (morning and afternoon).
- Partial registration: The participant registers for the camp on a part-time basis, i.e., for the activities in the morning only.

In terms of registration (and its types), the following methods should be considered:

- Empty constructor (without parameters).
- get/set methods for all attributes.
- toString method to print the registration information, which may require specialisation depending on the type of registration.

To create the registrations, a factory pattern must be used, so that it is possible to create two types of registrations in two different modalities:

- Early registration: The registration is made at least 15 days in advance, and allows the possibility to cancel it, regardless of whether it is full or partial.
- Late registration: Registration is made less than 15 days in advance and at least 48 hours before the start of the camp. It does not allow for cancellation, and is applicable to both full and partial registrations.

## EXERCISE 3

Using the previous classes, three manager classes must be created to implement the functionalities that will be available in the main program.

**Participant Manager:** This class will be in charge of managing the information of the camp participants. It must provide the following functionalities:

- To register a participant, checking that he/she is not previously registered.
- Modify the information of a participant.
- List the currently registered participants.

**Camp Manager:** This class will be the one to manage the organisation of the camps, including their activities and monitors. Specifically, it must implement the following functionalities:

- Create activities, monitors and camps. A camp is composed of a fixed number of morning and afternoon activities, so that the registration of the camp participants implies their registration to all the activities that compose it (in full or partial modality).
- Associating monitors to activities, considering the restrictions on the number of monitors.
- Associating activities to camps, considering the conditions of educational level. That is, a "children" level camp can only contain "children" level activities. This does not imply that the same activity (e.g., "basketball") cannot exist at more than one level ("children basketball" and "youth basketball").
- Associate monitors to camps, among those who are assigned to the activities that make up the camp at any given time. That is, if an activity is removed from the camp, its monitor cannot be listed as the monitor responsible if he/she is not assigned to another activity in the same camp.
- Associating special attention monitors to camps, among those who have such a role and who are not listed as being assigned to camp activities. The association of the special attention monitor is only necessary when at least one participant with special needs is registered in the camp.

**Registration Manager:** This class shall be in charge of the management of the registrations, and shall use the factory pattern when deemed necessary. The following are the minimum functionalities expected and the restrictions to be taken into account:

- Allow registrations in early and late registration modes. It must be ensured that the same participant cannot register for the same full and partial camp. Likewise, if you are already registered in the early registration mode, you cannot register late if you have not cancelled before.
- The price of a booking is established according to the number of activities that make up the booking. Each full camp has an initial cost of 300€ for accommodation and meals. This cost is reduced to €100 if it is only partial, for meals only. To this must be added a cost of €20 per activity.
- If the participant has special needs, this must be stated at the time of registration to associate a special attention monitor to the camp, if he/she does not have one. There is no extra cost involved.
- The manager must allow you to consult the camps that have not yet started and have places available, i.e., those for which registration is possible.

To test the functionality of the different managers, a Java program will be developed with a text interface (console I/O) that offers menus to access the different functionalities related to participants, camps and registrations. The organisation of different menus of options according to the type of manager is recommended, as well as the pre-loading of information from one or several text files. These files should be updated at the end of the programme, so that the students can work with lists during the program execution.

Structure the code appropriately to separate the data-related classes, the data management operations, and the console user interface. The paths to the data files should be indicated in a properties file within the Eclipse project, avoiding the use of absolute paths. The properties file will be accessed from the Java program using the *java.util.Properties* class (example available in Moodle). For more information:

- Class: https://docs.oracle.com/javase/8/docs/api/java/util/Properties.html
- Tutorial: https://www.baeldung.com/java-properties

For the implementation of the console input, it is recommended to use the class *java.util.Scanner*. https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html

Remember that apart from the source code, the program must be delivered as a JAR executable. Also include a README.txt text file explaining the parameters required for the program execution. Tutorial: https://www.baeldung.com/java-create-jar