Mini-Project: Instructions

Guillermo Román Díez

Programming Project

Escuela Técnica Superior de Ingenieros en Informática
Universidad Politécnica de Madrid

October 16, 2019

1 Norms

- The mini-project must be done in **pairs**. Pairs must be **registered** by using the Moodle form "*Mini-project: Pairs registration*" before **9th October 2019**. Note that the form include the URL of the GIT project, create you project structure before filling the form.
- The deadline for finishing the assignment is **Tuesday 5th November 2019**. Modifications done after this date will not be considered.
- A system for detecting plagiarisms will be used and the groups involved in plagiarism will be penalized with a failing grade. According to the UPM norms, other disciplinary measures can be studied and adopted in case of plagiarism.

2 Tools

The use of the following tools and libraries is mandatory:

- `Java` ≥ 1.8
- `GitLab` as GIT remote server
- `JUnit` 5 for test automation
- `Maven` as build tool
- `SonarQube` as quality tool

2.1 `GitLab`

Remember that the use of `GitLab` as remote GIT server is mandatory. The `GitLab` server of the subject is available at the URL:

`http://costa.ls.fi.upm.es/gitlab`

The `GitLab` project used to develop the assignment must fulfill the following requirements:

- The *name* of the project must be **mini-project**.
- The *Project visibility* must be **private**. If your code is accessible by other groups is your responsibility.

- Both members must have *maintainer* role for accessing the project (Settings - Members - ...).
- Email notifications are mandatory and must be sent to both users from the beginning of the project (Integrations - Emails on push - ...) .
- The username `@pproject` must have access to the project with *developer* role.
- In order to monitor the activity of the project, the email `programming.project@fi.upm.es` must be included in the notification emails from the beginning of the project.

Remember that both member must *push* their own changes to the remote server. Avoid committing all changes using the same user as it is a very bad practice.

The repository must include a README.md file to include relevant information to use the system, mainly a description of the interface and how to use the public methods of the program. Remember that the README file is written using `Markdown`[1] format.

2.2 Maven

The use of Maven as build tool is also mandatory and the file `pom.xml` file must be in the root directory of the project. The project must follow the standard directory layout of Maven. The details of this layout can be found at:

`https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html`

The `pom.xml` must include the dependencies needed to work with `JUnit` 5.20 and it must work without any manual intervention, that is, after cloning the GIT project the command `mvn test` must properly work passing all tests of the system.

2.3 JUnit

It is mandatory to write the testing classes and methods by using `JUnit 5.20` to ensure the correct behaviour of the program. Take a look to the *Testing* slides to read the good practices and recommendations in testing development. All tests must be automatically run by the `mvn test` command after cloning the project. Those tests that do not execute automatically by means of the `Maven` command will not be considered.

2.4 SonarQube

`SonarQube` will be used for checking the quality of the code. Remember that `SonarQube` include the notion of *quality gate* and the project must pass the quality gate to be evaluated. **Those projects that do not pass the quality gate of `SonarQube` server will no be evaluated**.

The URL of the `SonarQube` server used in the course is:

`http://costa.ls.fi.upm.es:9000/sonar/`

Remember that the initial username and the password of `SonarQube` server are your *número de matrícula* and your *DNI*.

The use of `SonarLint` Eclipse plugin is recommended to improve the efficiency during the development:

`https://www.sonarlint.org/`

Its installation instructions can be found in Eclipse Marketplace:

`https://marketplace.eclipse.org/content/sonarlint`

IMPORTANT: Do not bind the SonarLint plugin with the SonarQube server, it seems that the last version of SonarLint requires a most recent version on the SonarQube server and it does not wor fine.

[1] `https://www.markdownguide.org/basic-syntax/`

2.5 SonarQube and Maven

In order to automatically analyze the project with `SonarQube` and evaluate if the project passes the quality gate you can use `Maven` . To do so, first you have to update your `pom.xml` file with some plugins and set some properties of such file. You can find a `pom.xml` file with all required information in the Moodle of the course (Mini-project section). Dot not forget to update the properties with your SonarQube information:

```
<sonar.login>MATRICULA</sonar.login> <!-- UPDATE ME -->
<sonar.password>DNI</sonar.password> <!-- UPDATE ME -->
```

After preparing the `pom.xml` file, you can run the command

```
mvn verify
```

to automatically analyze the project with `SonarQube` . To check the results you can access to the `SonarQube` web interface and see whether the project passes the *quality gate.*

3 Managing Courses and Students

The goal of the practical assignment is the development of a Java program (without graphical user interface) to manage the students that are matriculated in the different courses offered by the university. Note that the program does not require any kind of graphical user interface, it is only needed to develop the classes required to implement the capabilities described below. The use of a Java *interface* defining all methods needed to interact with the program is strongly recommended and its *functional* description must be included in the `README.md` file.

The following list summarizes the requirements of the system that must be implemented:

1. A new course can be registered. The information associated to a course is its *code* (a numeric value), its *name* (a `String` value) and its *coordinator* (a `String` value). The following checks must be done:
    - The code, the name and the coordinator, cannot be blank.
2. A new student can be registered. The information associated to a student is its *identification number* (a numeric value), its *name* (a `String` value) and its *email address* (a `String` value). The following checks must be done:
    - The identification number, the name and the email cannot be blank.
    - The format of the email address must be correct, that is, it must contain the character `'@'` and it cannot end with the character `'.'`.
3. A student can be enrolled in a course, by using the identification number of the student and the course code. The following restrictions must be considered and an proper exception must be thrown when they are not satisfied:
    - The student must be already registered in the system.
    - The course must be already registered in the system.
    - A course could have, at most, 50 students matriculated.
    - A student cannot be enrolled in the same course twice.
4. Given a course code, the system must return the list of its matriculated students, returning its identification number, name and email. The list must be sorted by the identification number.

5. A student can cancel its enrollment in a course. The system must check that the student is registered in the system and matriculated in the course and throws an exception when it does not happen.
6. A course can be restarted and this operation must remove all students matriculated in the course.
7. The user can obtain a list of all users registered in the system, including its name, email and identification number. The list must be ordered by its identification number.
8. The user can obtain a list of all courses, sorted by their code.
9. Exceptions should be thrown when the pre-conditions of the operations do not hold.