Laboratory Sheet 4

This Lab Sheet contains material primarily based on Lectures 7 – 8.

The deadline for Moodle submission is 17:00 on Thursday 20 October.

Aims and objectives

- Further practice with object-oriented modelling in Java
- Gain experience of **refactoring** – reusing and modifying existing code
- Gain experience of creating new classes from scratch without starter code
- Particular focus on subclasses, abstract classes, and overriding methods
- Practice making use of exceptions

Set Up

1. Download Laboratory4.zip from Moodle and launch Eclipse.
2. In Eclipse, select File → Import ... to launch the import wizard – this will be used to import the starter projects into your workspace. Then select General → Existing Projects into Workspace and click Next.
3. Choose Select archive file and then browse to the location where you downloaded Laboratory4.zip. Select Laboratory4.zip and click Open.
4. You should now see a new project in the Projects window: Submission4_1. Ensure that the checkbox beside this project is selected, and then press Finish.
5. In the *Package Explorer view* (on the left of the Eclipse window), you should now see an icon representing the new project (in addition to any projects that were there before).

Please refer to the lab sheet from Laboratory 1 for additional details on how to open and run the code in the new project.

Submission material

Preparatory work for these programming exercises, prior to your scheduled lab session, is expected and essential to enable you to submit satisfactory solutions.

Unit Tests: Unit tests will be supplied separately for this lab, along with instructions for adding the tests into your project and running them. **Note that the tests may not test all aspects of the program's behaviour – passing all of the tests does not mean that your code is perfect (although failing a test will definitely indicate a problem to be investigated).** You should also read through the specification and make sure that you have implemented everything correctly before submitting.

Submission

This exercise builds on the material you submitted for Laboratory 3, so it might be worth referring back to your work on that lab before beginning this one.

Recall that the StudentRecord class developed in Laboratory 3 represents a (simplified) record of a current student: name, student ID, degree programme and year, along with information about fee payments. Your task in this lab is to **refactor** the StudentRecord class into a set of classes that are able to represent both current and former students: the information that is common to all student types will be retained in the StudentRecord class, which will be made **abstract**, while other information that is relevant only to one of the two student types will be moved to the appropriate **concrete** (i.e., non-abstract) subclass.

The following sections describe the modifications and additions that must be made as part of this refactoring task. **Please read through the whole specification before beginning and make sure that you understand what is involved.**

Subclasses and Fields

You must make the StudentRecord class abstract, and then create two concrete subclasses of StudentRecord: CurrentStudent, which will contain all fields and methods relevant to current students, and FormerStudent, which will contain all fields and methods relevant to former students. You should then move the members of StudentRecord into the appropriate location and add members as necessary. The fields and methods that should be in each class after the refactoring are as follows.

- StudentRecord should contain fields for the student number, student name, and degree programme, along with the relevant **getters** and **setters** (use the same getters and setters that are in the original StudentRecord class)
- CurrentStudent should contain fields for the year of study, tuition amount, and total payment fields, along with the relevant getters and setters and the `makePayment()` method (again, use the same getters and setters as in the original StudentRecord)
- FormerStudent should contain fields for the graduation year (an int) and the degree class (a String), along with getter methods for both (this class should have getters only)

Note that the fields and methods from the existing StudentRecord class will be split between the revised StudentRecord class and the new CurrentStudent class, while the fields and methods in FormerStudent will all be new.

To create a new class in Eclipse: right click on the project name and choose New – Class. Then fill in the class name in the Name field and the superclass (if any) in the Superclass field. If the superclass is abstract, you can also tick the box beside "Inherited abstract methods" to create initial versions of all abstract methods; you can also add the methods later, so this is not necessary. Press "Finish" and the new class will be added to your project.

Constructors

The constructors for each class should reflect the fields that are included in or inherited by each class, as follows:

- `public StudentRecord(String studentName, String degreeProgramme)`
- `public CurrentStudent(String studentName, String degreeProgramme, int yearOfStudy, double tuitionAmount)`
- `public FormerStudent(String studentName, String degreeProgramme, int graduationYear, String degreeClass)`

The constructors for the subclasses should call the superclass constructor with appropriate arguments using the `super` keyword, and then initialize the additional fields in the subclass.

toString() and getDetails()

The next piece of the refactoring is to modify the `toString()` method in StudentRecord so that it produces an appropriate string representation of both types of student records. You should do this as follows:

- Add an abstract `getDetails()` method to the parent StudentRecord class – this method will be implemented in the subclasses to provide string representations of the fields specific to each subclass. **The method should have a `protected` access modifier.**
- Modify the `StudentRecord.toString()` method so that it returns a string consisting of the student ID, the student name, followed by the result of calling `getDetails()`
- Implement `getDetails()` in each subclass to return a string representation of only the fields specific to each subclass – that is, CurrentStudent should return a string representation of the year of study, tuition amount, and current payments, while FormerStudent should return a string representation of the year of graduation and the degree class.

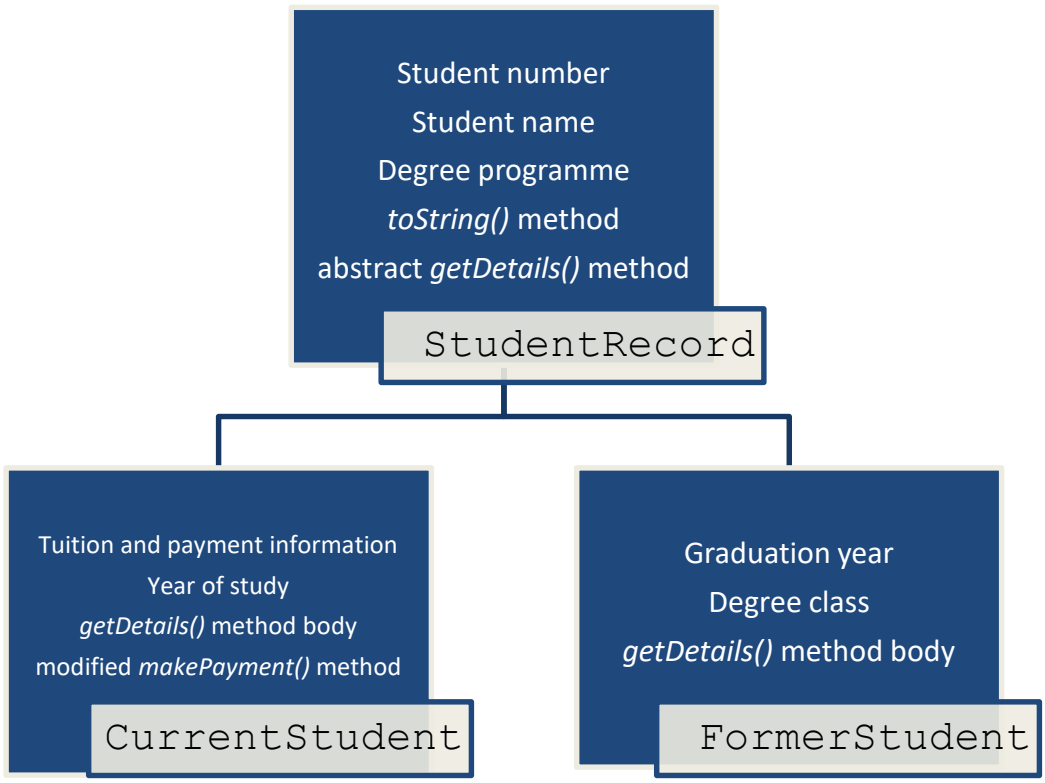
Adding an Exception

The final step in the refactoring process is to modify the `makePayment()` method. In its current version, it returns true/false depending on whether the amount is valid or invalid. You should change the method as follows:

- Change the return type to `void`
- If the payment amount is invalid, the method should throw an IllegalArgumentException with an appropriate message.

Diagram

The following diagram summarises the classes that you should have after the refactoring is done, and the fields and methods that should be included in each.



Development process

The initial Submission4_1 project contains two files:

- StudentRecord.java – a sample solution to Laboratory 3
- StudentRecordRunner.java – a class with only a main method, which you can use to test the behavior of the classes that you develop in this lab. The current version of the main method creates several student records of different types and calls methods on them. You can add, remove, and change the calls in this method to experiment with the code that you are developing if you wish – note that you will not be submitting this file so it does not matter what changes you make.
    - Note that, as you will not have yet completed the refactoring, StudentRecordRunner.java will show errors and will not run when you first open it – **this is expected.**

You should carry out all of the steps described in the preceding section to refactor the code. Here are some hints and suggestions.

1. The refactoring process will probably "break" the code as you carry out the above steps – for example, if you move some fields but do not yet move the methods that refer to those fields, you will see a lot of errors. You can use these errors to help you decide what edits to make next – but please make sure that the class design in the end product is the same as the diagram on the preceding page.
2. The sample code from the tutorial on 14 October (posted on Moodle) may be useful to you as an example of abstract classes, inheritance, and exception throwing.

How to submit

You should submit your work before the deadline no matter whether the programs are fully working or not.

When you are ready to submit, go to the JOOSE2 moodle site. Click on Laboratory 4 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code ...\Submission4_1\ and drag *only* the three Java files StudentRecord.java, CurrentStudent.java, FormerStudent.java into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java files, not your class file.** Then click the blue save changes button. Check the three .java file is uploaded to the system. Then click submit assignment and fill in the non-plagiarism declaration. Your tutor will inspect your file and return feedback to you via moodle.

Guidelines for Mark Scheme

A1: All methods completed, with correct types and behaviour. All JUnit tests pass.

B1: Majority of methods completed, with correct types and behaviour. Some JUnit tests pass.

C1: Some methods attempted, possible problems with types and behaviour. Code may not compile.

D1: Minimal effort.

For this assignment (and all future ones), we will also be considering code style – comments, formatting, variable names, etc – in addition to correctness. You will be deducted some marks (e.g., from an A1 to and A3) if your code style is not appropriate. Please check with the tutors and demonstrators during your lab session if you are uncertain about style, or look at the BankAccount class from Laboratory 3 for an example.