# Laboratory Sheet 3

**This Lab Sheet contains material primarily based on Lectures 5 – 6.**

**The deadline for Moodle submission is 17:00 on Thursday 13 October.**

## Aims and objectives

- Reinforcement of the basic concepts of classes, objects, methods, and fields in Java
- Design and implementation of a simple class

## Set Up

1. Download Laboratory3.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 Laboratory3.zip. Select Laboratory3.zip and click Open.
4. You should now see a new project in the Projects window: Submission3_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**: I have supplied some simple test cases to check that your code is working properly. When you create each of the projects, you should see *two* source code files – one is the skeleton Java source code for you to fill in the details. The other file contains the Unit tests to check your programs are working properly.

To execute the program:
right click on StudentRecord -> run as Java application
right click on StudentRecordTest -> run as JUnit test

## Submission

You are to design and implement a class `StudentRecord` representing a (simplified) student record. A student record includes the student's **ID number**; details of the student's **name**, **degree programme**, and **year of study**; as well as details of their **financial status**. Two aspects of a student's financial status are recorded: the total **tuition amount** that they are liable to pay, and the amount of money that they have paid so far (their **total payments)**. The student's **balance** can be computed by subtracting the total payments from their tuition amount – if the student has paid their entire tuition amount, then their balance is zero.

I have provided a file StudentRecord.java which contains only the declaration of the StudentRecord class and an empty body. You should fill in the class body by implementing all of the fields and methods described below.

The relevant properties should be included in the `StudentRecord` class by defining the following `private` fields:

- `studentNumber` (an `int`)
- `studentName` (a `String`)
- `degreeProgramme` (a `String`)
- `yearOfStudy` (an `int`)
- `tuitionAmount` (a `double`)
- `totalPayments` (a `double`)

There should a single `public` constructor method that takes an `studentName`, a `degreeProgramme`, a `yearOfStudy`, and a `tuitionAmount`, i.e.

```
public StudentRecord(String studentName, String degreeProgramme,
    int yearOfStudy, double tuitionAmount)
```

For all created `StudentRecord` objects, each object should have a unique `studentNumber`. All `StudentRecord` objects are initialized with a `totalPayments` value of 0.00.

You must define an instance method to compute and return the balance, as follows:

```
public double getBalance()
```

This method should compute the balance (i.e., the difference between the `tuitionAmount` and the `totalPayments` values) and return it.

You must also define an instance method to make a payment, as follows:

```
public boolean makePayment(double amount)
```

This method should first check that the proposed payment is valid, that is:

- The `amount` should be greater than zero, and
- The `amount` should be less than or equal to the remaining balance

If one of the above conditions is violated, then the method should make no changes to the state and return false. If the proposed payment is valid, then the `totalPayments` value should be increased by the given `amount` and the method should return true.

Please also define relevant getter methods for all `StudentRecord` fields:

- `public int getStudentNumber()`
- `public String getStudentName()`
- `public String getDegreeProgramme()`
- `public int getYearOfStudy()`
- `public double getTuitionAmount()`
- `public double getTotalPayments()`

along with setter methods for two fields:

- `public void setDegreeProgramme(String degreeProgramme)`
- `public void setYearOfStudy(int yearOfStudy)`

Also override the inherited `toString` method, to display the values of the six object fields. The format is up to you; however, you must be sure to include at least the student ID, name, degree programme, year of study, along with the two pieces of financial information.

Sums of money (e.g., the `tuitionAmount` and `totalPayments` values) should be displayed in a conventional way e.g. `£1,820.00`. You can obtain a `NumberFormat` object which can be used to format currency values like this:

```
java.text.NumberFormat cf = java.text.NumberFormat.getCurrencyInstance(
      java.util.Locale.getDefault());
```

You can then use the `NumberFormat` like this as part of your `toString()` method (where `value` is a double representing the currency value):

```
String currencyString = cf.format(value);
```

If you would prefer to use a different mechanism, you should ensure that the output matches.

Use the JUnit test driver class, `StudentRecordTest`, for this class to check that your code works. The test driver creates a small number of student records and carries out some operations on them, providing a systematic test of the methods. It also checks that you have defined the correct fields and methods with the correct accessibility modifiers (public/private).

## Hints and tips

1. The sample BankAccount class gives a sample of most of the things that you will need to implement in your StudentRecord class.
2. If you want to test your code yourself in addition to using the provided test cases, you can add a `main()` method to your StudentRecord class to allow it to be run in Eclipse; you can then create, modify, and print out student record objects. There is an example of this process in the BankAccount class. But if you do this, **please remember to remove the `main()` method before submitting your code** as it is not part of the specification for the StudentRecord class.
3. Make sure that none of your methods produce any output through `System.out.println` or similar mechanisms. You are implementing a class that will (in theory) be used as part of a larger system, so output is not appropriate. The JUnit tests will also catch this situation.
4. Once you have defined the fields of your class, Eclipse is able to automatically generate many of the methods that you will need for this assignment based on those fields. For example, try right clicking on the source file and choosing "Source – Generate getters and setters …", "Source – Generate constructor using fields", or "Source – generate toString(). Experiment and see what it generates!
    a. **If you do this, you **MUST** read through the automatically generated code to be sure that it behaves as required – any automatically generated code should be thought of as a starting point only.**

## 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 3 Submission. Click 'Add Submission'. Open Windows Explorer and browse to the folder that contains your Java source code ...\Laboratory2\Submission3_1\ and drag *only* the single Java file StudentRecord.java into the drag-n-drop area on the moodle submission page. **Your markers only want to read your java file, not your class file.** Then click the blue save changes button. Check the single .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 for an example.