Week 2 Lab

Classes and Objects

Objectives

In this week's lab, you will:

- design a multi-class program
- implement a multi-class Java program
- use Java arrays
- use a loop

Task 1. Preparation: UML Class diagram

You must draw a UML class diagram showing the system that you plan to implement in task 4. Note that this system will incorporate work you have done for Tasks 2 and 3, so read the entire lab sheet carefully.

This diagram should be quick and simple. It only needs to show the classes you plan to create and the relationships (associations) between them. You don't need to show attributes or methods. You are strongly encouraged to draw this diagram by hand and scan it in (or simply take a photo of it) for submission – you don't need to use a UML diagramming tool.

This must be submitted via the link on the FIT2099 Moodle page *at least 24 hours before the scheduled commencement of your lab class* so that your demonstrator can give you feedback on the quality of your design.

Task 2. Start the implementation

Even when you have a complete design, it is usually a good idea to implement and test your program bit by bit.[1]

Create a new Eclipse project called JavaUniversity.

There is a Java source file called `University.java` on Moodle. Download it and see if you can figure out how to add it to your Eclipse project. The source code is reproduced below:

```
1 public class University {
      public void printStatus() {
3         System.out.println("Welcome to Java University");
          System.out.println();
5         System.out.println("Thank you for using Java University");
      }
7 }
```

[1] This idea is taken to its logical conclusion in an approach called Test-Driven Development (TDD). We will not spend much time on TDD in this unit, but it will be discussed in other software engineering units

As you can see, all the class can currently do is print a welcome message.

Once you have added the class to your project, create a new `UniversityDriver` class. This class should contain only a `main(...)` method that creates a `University` object and calls its `printStatus` method.

Run your program and confirm that it works as expected.

While you are working on this task, your demonstrator will be circulating the room assisting students and providing feedback on your UML class diagrams. **Please ensure that you have had feedback from your demonstrator before proceeding to task 3.**

Task 3. Extending the implementation - adding a Unit class

The system needs a `Unit` class. Each `Unit` object must know its unit code (e.g. "FIT1234" and its name (e.g. "Advanced Bogosorts"). The `Unit` class should also have a `getUnitDescription()` method, that returns a string of the concatenated unit code and name, e.g. "FIT1234 Advanced Bogosorts".

To test your `Unit` class, add some code to `University.printStatus()` that

- creates a `Unit` object and
- prints its description using the `getUnitDescription()` method.

Confirm that your program is producing correct output before proceeding to the next task; if necessary consult your demonstrator.

Task 4. Arrays of `Units`

Now, we would like the system to be able to handle an arbitrary number of `Units`.

Modify the `University` class so that instead of containing a single Unit, it contains an *array* of units.[2]

Add two new methods, `createUnits()` and `displayUnits()`, to `University`.

- `createUnits()` must create three new `Unit` objects, add unit codes and descriptions to them, and store them in the array.
- `displayUnits()` must display the descriptions for the units, which you can generate by invoking `getUnitDescription()` on each of the units in the array. Try to use a loop rather than repeated code, and try to avoid hard-coding the array length into `displayUnits`.

If you need additional information about Java arrays and/or Java loops, Oracle has comprehensive documentation for Java at `http://docs.oracle.com/javase/8/`. Hint; have a look at the tutorials, particularly the "learning the language" tutorials.

Modify the `printStatus()` method to display the welcome message, call the `createUnits()` method, call the `displayUnits()` method, and then display the goodbye message.

Getting marked

Once you have completed this task, show your lab demonstrator to be marked. If you do not get this done by the end of this week's lab, you may submit it on Moodle **within 24 hours** – but make sure that your demonstrator has at least given you feedback on your design before you leave, so that you don't waste time implementing a poor design.

Note that to receive full marks, it is not sufficient merely to produce correct output. Your solution must implement the structure described above.

[2] Java offers you several other ways to store a collection of `Units`. For now, we'll keep it simple by just using an array.

Marking
This lab is worth 2% of your final mark for FIT2099. Marks will be granted as follows:

- 0 marks if you do not complete all tasks, or if your demonstrator has not seen and approved your design
- 1 mark if all tasks are completed but the design is poor or the implementation is flawed
- 2 marks if all tasks are complete, the design is good, and the implementation is correct