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 *fields* (a.k.a. attributes) in Java classes
- use *methods* (a.k.a. operations) in Java classes

In the assessed lab exercises this semester, you will create a simple framework for an information system that keeps track of personnel and resource allocations for an organization. That's a bit boring, so we will customize our system for the needs of a particular kind of client: evil geniuses and supervillains.

Each lab will build on the previous one. Each week, the object-oriented concepts introduced in lectures will be applied in the lab exercise. You will also be expected to apply the principles of good design and implementation you will learn in lectures and readings.

Task 1. UML Class diagram

You must draw a UML class diagram showing the system that you plan to implement, and show it to your demonstrator *before you start coding*. Note that this system will incorporate work you have done in multiple tasks, 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 in UML*) between them. You don't need to show fields or methods. You are strongly encouraged to draw this diagram by hand, either on paper or on a whiteboard[1].

Remember that you need to do the readings on Moodle before the lectures and labs each week. This week there are readings about UML class diagrams.

Even if you don't finish the lab today, **you must get your completed design in UML approved by your demonstrator before you leave.** This will mean you can work at home without coding up a poorly-designed solution.

Task 2. Start the implementation

Even when you have a complete design, it is usually a good idea to implement and test your program a small amount at a time. We will start by creating a class that models locations inside the supervillain's lair.[2]

[1] If you choose to use the whiteboard in your classroom, take a photo so you can refer to it later.

[2] 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 Service, but it will be discussed in other software engineering Departments

Create a new Java project called `LairManagementSystem`. Create a class in this project called `Lair`. Type in the source code listed below:[3]

```
public class Lair {

    public void printStatus() {
        System.out.println("Welcome␣to␣the␣Supervillain's␣Lair␣Management␣
            System.");
        System.out.println();
        System.out.println();
        System.out.println("Good-bye.␣Thank␣you␣for␣using␣the␣Supervillain'
            s␣Lair␣Management␣System.");
    }
}
```

As you can see, all the class currently does is print welcome and good-bye messages.

Once you have added this class to your project, create a new `LairDriver` class. This class should contain only a `main(...)` method[4] that creates a `Lair` object and calls its `printStatus()` method.

Run your program and confirm that it works as expected.

Task 3. Extending the implementation: adding a `LairLocation` class

Any lair worthy of the title will contain many locations: shark tanks, lava pits, supercomputer rooms, etc. These locations will eventually be filled with traps and evil minions, but we'll leave those out for the time being. To store information about these locations, we will need a `LairLocation` class. As you saw in lectures, this class is a template for creating `LairLocation` objects at run-time. Each `LairLocation` object must know its name (e.g. "Shark Tank" and a brief description (e.g. "Full of sharks with laser beams on their heads"). The `LairLocation` class should also have a `getLairLocationDescription()` method, that returns a string consisting of the location's concatenated name and description separated by a colon and a space, e.g. "Shark Tank: Full of sharks with laser beams on their heads".

To test your `LairLocation` class, add some code to `Lair.printStatus()` that

- creates a `LairLocation` object and
- prints its description using the `getLairLocationDescription()` method.

Confirm that your program is working correctly before proceeding to the next task. If necessary consult your demonstrator.

Task 4. Multiple `LairLocations`

The system needs to be able to manage multiple `LairLocations`. Modify `Lair` so that it has three fields, each of type `LairLocation`. Add two new methods, `createRooms()` and `displayRooms()`, to `Lair`.

- `createRooms()` must create three `LairLocation` objects and store references to them in the fields described above. You can choose whatever names and descriptions you like[5]
- `displayRooms()` must display the descriptions of the three `LairLocation` objects to the screen using the `getLairLocationDescription()` method of `LairLocation`.

[3] Do **not** cut-and-paste from the PDF!

[4] If you're using Eclipse, you saw how to use the class wizard to create a class with a `main(...)` method last week.

[5] If you need inspiration, consult `https://tvtropes.org/pmwiki/pmwiki.php/Main/SupervillainLair` or your favourite James Bond movie.

Modify the `printStatus()` method of `Lair` to display the welcome message, call the `createRooms()` method, call the `displayRooms()` method, and then display the goodbye message.

Tips

That's it for this week's lab. You can now have your demonstrator look it over. In the meantime, here are a few tips to bear in mind.

Keep your code

The code you create in each lab is the starting point for the following week's lab. Be sure to keep it! You should create a copy of your previous week's project when starting the following week's one, so that you always have a copy of the state of your project as it was at the end of the week. This is particularly important if you have not yet been marked for the previous week!

Finished early?

Some students in this unit have used Java or similar languages before. They should find this lab very easy. If you're one of them and have finished early, we encourage you to stay around and help your fellow students understand the material. Our goal is for *everyone* to complete these lab tasks successfully.

Getting ready for next week

Using three hard-coded attributes is a terrible way for the `Lair` to keep track of its `LairLocation`s. We only did it this way as a simple way of introducing you to basic attributes and methods in this early lab.

Why is this an example of bad design? Can you think of a better way of doing it?

Getting marked

Once you have completed these tasks, call your lab demonstrator to be marked. Your demonstrator will ask you to do a "walkthrough" of your program, in which you explain what each part does. Note that to receive full marks, **it is not sufficient merely to produce correct output**. Your solution must implement all structures specified above *exactly* — as software engineers we must follow specifications precisely. If you have not done so, your demonstrator will ask you to fix your program so that it does (if you want to get full marks).

If you do not get this finished by the end of this week's lab, you can complete it during the following week and get it marked at the start of your next lab — 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 or incorrect design. That will be the last chance to get marks for the exercise. You cannot get marked for an exercise more than one week after the lab for which it was set.

Marking

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

- 0 marks if you do not attempt the tasks, or if you could only complete them once given the solution
- 1 mark if some tasks are incomplete, if the design or the implementation is flawed, or if you needed to see a significant part of the solution to complete them
- 2 marks if you were able to complete all tasks independently, with good design and correct implementation