

# CS-115 Lab Class

The lab classes consist of tasks that are designed to be completed during the lab class each week. If you do not complete the tasks during the lab class then you should do them at home and have them ready to be marked off in the next lab class.

Each week you will be given a new lab class sheet with new tasks. You should aim to get all tasks completed and marked off by staff in the lab class. You can only get the lab sheet signed off in the lab classes (not by email, etc.).

**Full marks will be awarded if the tasks are marked off by staff during the session they are handed out; or if they are marked off by staff during the following session. Half marks will be awarded if the tasks are signed off two weeks late. Marks will not be awarded if the tasks are more than two weeks late.**

The lab work as a whole is worth 10% of the module.

The challenge tasks are optional and are not worth extra marks. However, it is a good idea to attempt these as completing them will help strengthen your programming and problem solving skills. You may need to do a little research to find out how to complete the challenge tasks.

## CS-115 Lab Class 1

**27/02/2020**

As we are going to be working with multiple classes your programs will consist of more than one file. You should store each program in its own project in Eclipse.

In CS-115, your code should follow the coding conventions set out by the module. You will not be signed off in the labs if your code does not follow these conventions (see end of lab). In the real world, the company you work for will set coding conventions to ensure that team members can read code written by other team members. They will be different to the 115 conventions, but quite similar if the language is Java. **Read the end of these instructions for details.**

### ☐ Task 1.1

Create a class to model a student. This class should be named **Student**. A student has the following attributes:

- first name.
- surname.

Your student class should support the following operations:

- Get first name – returns the first name of a student.
- Get surname – returns the surname of a student.
- Set first name – sets the first name of a student.
- Set surname – sets the surname of a student.
- Get full name – return the concatenation of the first name and the surname of a student.

Now we need somewhere to put the main method. It does not really fit into the student class (yes, technically it can, but it simply feels wrong). Create a new class named **Main** to hold your main method.

In the `main` method, you should create a student or two and call the various methods to demonstrate the implementation of the `Student` class works correctly.

Note: we never instantiate the `Main` class – it is simply the “driver” for the program. This class constructs and manipulates *Student* objects. Likewise we never run the `Student` class.

## ☐ Task 1.2

Students should always have a name. The first name and surname should be set when the student is created. Create a constructor for the `Student` class that takes as arguments a first name and a surname.

## ☐ Task 1.3

Extend your student class to also store a coursework and test mark (between 0 and 100).

Add a method named `averageMark` which calculates (and returns) the average of the coursework mark and the test mark (add them both up and divide by 2 – watch out for integer division). This method should return a `double` as it’s possible that the average mark is not an integer.

Test your implementation by calling these methods from your `Main` method.

## ☐ Task 1.4

In your `main` method create an array that holds three different `Student` objects.

## ☐ Challenge Task 1.5

Extend your `Student` class to have another attribute to hold the student’s lab partner. Thus, student objects will hold references to other student objects. You should provide a setter and a getter for this attribute. Demonstrate it works using your `Main` method.

## Coding Conventions

Labs won’t be signed off if you severely violate these conventions. Also, you will lose marks on the assignment for violating these conventions. In the real world, companies care about this and will enforce a coding convention.

1. All class names will begin with a capital letter (e.g. `MyClass` and not `myClass`).
2. All variables/attributes and methods (except the constructor) will begin with a small letter and be camel case (e.g. `iLoveJava` and not `ILoveJava` or `i_love_java`)
3. All constants will be all caps with underscores. (e.g. `I_STILL_LOVE_JAVA` and not `iStillLoveJava` or `ISTILLOVEJAVA`)
4. Constants can be public or private. Attributes will never be public. Attributes will only be accessed through methods.
5. All of the constants and attributes of a class always appear at the top of the class. They are followed by all of the methods.
6. Binary operators will have spaces around them (e.g. `i + j` and not `i+j`).

7. Unary operators will never have a space around them (e.g. `i++` and not `i ++`).
8. Every block of code (encased in `'{'` and `'}'`) will be indented consistently with the left edge of the contents of that block aligned: classes, loops, if statements, case statements, methods and others. You can use tabs or spaces, but try not to use a mixture of the two.
9. Every block of a loop and if statement will always require `'{'` and `'}'`. You will not omit them.
10. The `'('` and `')'` characters in statements will be flush to the beginning and end of the statement. (e.g. `if (x == 0) {` and not `if ( x == 0 ) {`)
11. Methods and classes will be documented using the javadoc style you learn in CS-135. You are expected to use this style for the assignments. There will be some flexibility here, but at the very least you must always comment your methods by stating what it does, all of its parameters, and its return type. Always comment your classes stating what the purpose of the class is.
12. Complex bits of code in a method should have their own inline comment and be placed in a block separated by blank lines around it. You can, and will, lose marks for overcommenting. Overcommenting occurs when you have comments that do not add to the explanation of the code (e.g. `int i = 0; //initialising i to zero.`)