LAB 1: I/O AND STRINGS

F27SG2 (5 MARKS)

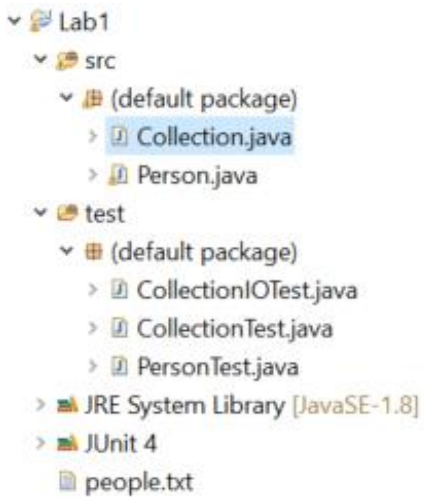
Download **Lab1.zip** from Vision. This can be found under:

Learning Materials -> Edinburgh -> Labs

Then import this zip archive as a new project in Eclipse. This can be done as follows:

File -> Import -> Existing Projects into Workspace

Then select the ZIP file you downloaded. Once you have imported the project is should look like the screenshot on the right (although the `JUnit 4' directory may be missing at this point)



First read the source code and make sure you understand what is going on and what is missing.

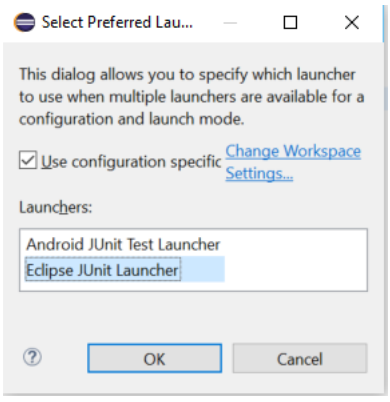
The project is organized as follows:

- The src directory contains all the source files
  - Person.java is a class representing in which an object of the class represent a person in our collection
  - Collection.java is the collection with relevant methods and fields
- The test directory contains the unit tests for the project.

1. WRITE JUNIT TESTS (2 MARKS)

When you have read the code then try to run the JUnit tests. To run a Junit you can do as follows:

- In the test directory, **right click** the file you would like to run (e.g. PersonTest.java)
- select **Run As**
- and then select **JUnit Test**.



You may here get a dialog box as shown on the right. Use the selections illlustrated there.

Some of the tests should fail. Your task is to complete the following JUnit test, with suitable values:

- **testSetAge()** of PersonTest.java. This test should ensure that when setAge(int age) of Person.java correctly sets the age field of the object to the given value. You will need to use getAge() in the test. See the tests for setting the first and last name in the same file.
- Complete the tests of **CollectionTest.java**. You should write a test for the maxAge() method of Collection.java. This method should return the maximum element of the collection. Note that maxAge() is not

yet implement so your test method will fail at this point (remember the test-driven development approach discussed in lecture 1). In the test you will need create some new Person objects and add them to the collection (using the addPerson method). You should write tests for the following scenarios:

- maxAge of an empty collection. In this case, assume that -1 is returned from the method.
- maxAge of a collection containing a single element
- maxAge of a collection of (at least) 3 elements

2. IMPLEMENT THE MAXAGE METHOD (1 MARK)

Implement the method

```
public int maxAge(){

    <your code>

 }
```

of Collection.java. The method should work of a collection of any size (even an empty collection), and all the tests from part 1 should work. Note that you will need to use a (for) loop that iterates through the collection.

3. IO – READ FROM FILE AND GENERATE COLLECTION (2 MARKS)

You should complete the method

```
public void readFromFile(String filename){

    <your code>

 }
```

where `<your code>` fills the collection with `Person` objects that you create by reading from the filename given by the `FROMFNAME` field of the collection (in the same order as in the file). You can assume that this file (which is `people.txt` at the top level of the project) has 200 lines, where each line has the following structure:

```
<last-name> ; <first-name> ; <height> ; <age>
```

For each line you need to create a new `Person` object where the `lastname` field of the object gets the value of `<last-name>`, the `firstname` field of the object gets the value of `<first-name>` and the `age` field get the value of `<age>`. You can insert this object using the provided `addPerson` method.

Hint! *To achieve this you can use the `FileReader` and `BufferedReader` which has been covered in the lecture. The `String.split` method (with ";" as separator) will also be useful to extract these elements (which returns an array of Strings), as will the `Integer.parseInt` method in order to convert the age to an integer. For a string `s`, you are recommended to use the `s.trim()`* method to remove white space before and after each string. Also *remember to handle the exceptions.*

The CollectionIOTest.java file provides a specification for this method in form of a JUnit test. You should make sure that this test succeeds.

ADDITIONAL CHALLENGE: SERIALISE THE COLLECTION (OPTIONAL)

We can also write objects to file directly. To do so, the object must be *serializable*, which in Java means that it must implement the `Serializable` interface (`java.io.Serializable`). First you need to update the `Person` class so that it implements this interface (and imports `java.io.Serializable`).

Then complete the method

```
public void writeObjects(){ <your code> }
```

which writes the `collection` array to a file (see lecture notes for details on how to do this). Next complete the method

```
public void readObjects(){ <your code> }
```

which reads the collection and stores it in the `collection` variable. This must use the same filename as the `writeObjects()` method. Remember to cast the object, i.e. the actual code for reading when the ObjectInputStream `ois` is created should be:

```
collection = (Person []) ois.readObject();
```