159.272 Programming Paradigms

Assignment 1

15% Weighting
Due 10 May 2020

Important: this is an individual assignment. You must not share your code with others, or use other's code (this will be plagiarism!). If we find two assignments that are copied from each other, both assignments will receive Zero marks.

Late submission: There is a penalty for late submissions. The penalty is 10% deducted from the total possible mark for every day delay in submission (one day late – 90%, two days – 80% etc).

You are expected to manage your source code, this includes making frequent backups. "The cat ate my source code" is not a valid excuse for late or missing submissions. Consider managing your code in a private repository (github, gitlab, bitbucket or similar). Set your repository to private (this is essential here to avoid plagiarism), we reserve the right to deduct marks from your submission if the repository is public.

Objectives

This assignment draws on material you have learned through the OOP section of the paper. In particular, you will need to apply the following principles to your code:

1. Object Persistence
2. Exception Handling
3. Unit Testing
4. Building User Interfaces

Tasks

In this assignment, you will create a simple student management system.

- Work individually to create the following program in Java using your IDE (Eclipse, intellij etc..).
- Create an Eclipse project. Inside, create a package *assignment1.*

Part 1 Domain Model **[3 mark]**

This section creates all the main classes you will need for this assignment.

1. create the following classes in your package (fields should be private):

a. **`Course`**, with properties for number and name (both **`String`**)
b. **`Address`**, with properties for town (**`String`**), street (**`String`**), post code (**`String`**) and house number (**`int`**)
c. **`Student`**, with properties for surname, first name, id (all **`String`**), dob (for "dateOfBirth", of type `java.time.LocalDat`**`e`**), course (of type **`Course`**) and address (of type **`Address`**)

2. Auto-generate getters, setters, equals and hashCode for all three classes. If two objects are equal, their hashCode should be the same - check that the generated code fulfils this contract.
3. **`Student`** should have a **`clone()`** method that is a combination of *deep* and *shallow* clone: deep clone should be used for the **`address`**, a shallow clone should be used for **`course`**

Note - LocalDate is immutable like a String. You can clone the Student date using:

```
clone.dob = this.dob
```

If you later change the dob on the clone, the original Student will not have its dob changed.

Part 2 Persistency **[4 marks]**

This section is focused on saving and loading the data we have so we can use it between runs of our programme.

Create a utility class **`StudentStorage`** with the following three **static** methods:

1. Create a class `StudentStorage` with the following three static methods:
   a. `void save(java.util.Collection<Student>,java.io.File file) throws IOException` **–** saves a list of students to a binary file and to CSV file, depending on what file format a user chooses (see part 4). Note that the data of referenced objects (address and course) should be saved as well. *Read the hints at the end of the document!*
   b. `void save(java.util.Collection<Student>,String fileName) throws IOException` **– saves a list of students to a binary file file with a given name.**
   c. No code should be replicated (copied & pasted) between the two versions of save(..).
   d. `java.util.Collection<Student> load(java.io.File file) throws IOException` **–** reads student data from a binary file.

The `save` methods should save a collection of students to a binary file with the supplied name. Do not copy paste code between the two versions of `save` - have one call the other.

The `load` method reads student data from the file and returns it (if you save students and then fetch them, you should get the original data back).

2. The above methods should preserve referential integrity. For example if two students share a course (i.e. the courses are at the same memory address), when loading the students back from a file, they should still reference the same course object, not two separate course objects (double check by comparing the memory locations of the courses on the fetched students in your Part 3 tests).

Part 3 Testing **[4 marks]**

This section tests that parts 1 and 2 are working as intended. Doing well in this section means probing that functionality well and considering any edge cases.

1. Write a class `TestCloningStudents` with JUnit test(s) to test the `clone()` method in `Student`.
2. Write a class `TestPersistency` with JUnit tests to test the `save(..)` and `load(..)` methods in `StudentStorage`. In particular, write tests that "round-trip" data: create a list of **`Student`** instances list1, save it, then load it from the file as list2, and compare list1 and list2.
3. Test whether `save(..)` and `load(..)` preserve referential integrity for the course objects (discussed in part 2)
4. Write tests to check whether the exception declared by `load(..)` work as expected when an invalid file name is used.

Hints

- Collections don't have a `get()` method. But you could cast them to Lists for the purposes of testing multiple objects against each other.
- `assertThrows(ClassCastException.class, () -> someMethod());` would check if a line of code produces a ClassCastException after `someMethod()` is called.
- Use the `@AfterClass` annotation to delete any files used over the tests and keep your directory clean!

Part 4 User Interface **[4 marks]**

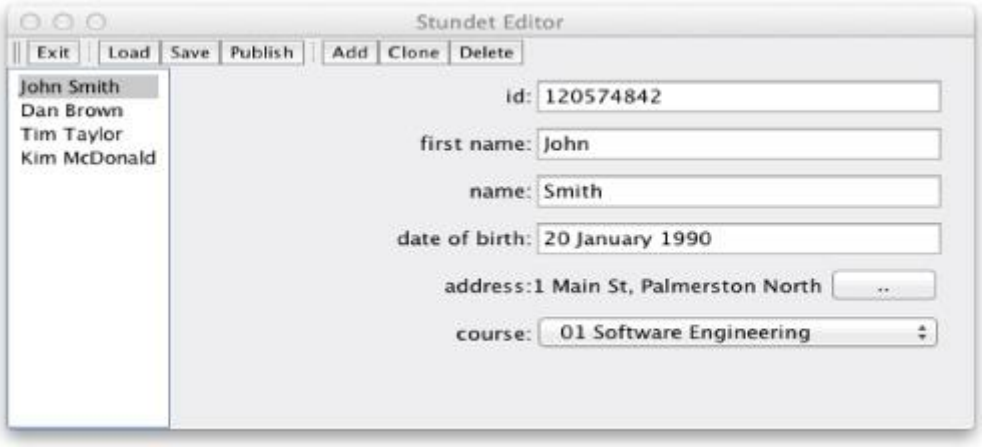
The GUI will be the main class of this assignment. It will be how you input information into the programme to create Students and manipulate them.
You may use either Swing or JavaFX for this GUI.

1. write a user interface StudentListEditor to edit lists of students, this is an executable class (a class with a main method) that when executed opens a window.
2. the user interface must show all instances of students represented in a file, and a form that can be used to edit a selected instance for instance. The user interface must have the following functionality:

    i. Show the students currently stored
    ii. Add new students.

iii. Delete students.
iv. Edit students.

an *example* of a user interface is shown below:



Note : you don't have to implement the publish or delete buttons !

3. Export the results to a file, you may choose any of the three file formats to use (.txt , .csv or .html). Feel free to implement this in multiple file formats if you wish. `javax.swing.JFileChooser` or `javafx.stage.FileChooser` can be helpful here.

There is no design requirement for this user interface - as long as it can perform the above functionality and has a sensible flow, it can get full marks. I would recommend using multiple scenes to keep implementation easier, particularly for showing the full information for students.

Part 5 (Bonus) Create a JAR **[Bonus: 1 mark]**

Create an executable java application *studenteditor.jar* with `StudentListEditor` as the main class.

Hint: See http://docs.oracle.com/javase/tutorial/deployment/jar/appman.html for instructions how to create executable jars.

Hints

- you can use code and ideas from tutorials for the first three parts
- for part 2, have a look at the following classes:

  java.io.ObjectOutputStream

  java.io.ObjectInputStream

3. see http://docs.oracle.com/javase/tutorial/deployment/jar/appman.html for instructions how to create executable jars (for the bonus questions).
4. Make your data classes implement Serializable (a 'tag' interface - you do not need to implement any methods, it's just there as a marker).

How to submit

1. Export the Eclipse project to a file ***assign1-<yourid>.zip***, where <*yourid*> is replaced by your student ID. Please use .zip and not another file extension.
2. Check that your zip file contains all project files and java sources by unzipping the file and inspecting its content before you submit , it is also recommended to re-import this as a project into an Eclipse workspace to check this (in Eclipse, use File > Import > General > Existing Projects into Workspace)
3. If you decide to do the bonus question (part 5), also include *studenteditor.jar* in the root (top) folder of this zip file.
4. Upload the zip file to stream.