159.272 Programming Paradigms

Assignment 1

Object-oriented Programming in Java

Total marks: 15 marks
Due: 8pm Friday 20 April 2018

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!

Submit your work on Stream **by 8pm (Beijing time) Thursday 20th April 2017**. You must submit your final work using the stream submission system before the deadline (see stream for details).

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 (bitbucket or similar). Private is essential here to avoid plagiarism, we reserve the right to deduct marks from your submission if it is public

Assignment 1 Objectives

Apply the concepts learnt in part 2 of this paper, in particular:

1. core OO concepts (inheritance, encapsulation, object identity and equality)
2. explore and use an API from the standard library (such as java.io)
3. exception handling
4. unit testing
5. building user interfaces with swing

Tasks

Work **individually** to create the following program in Java using Eclipse. Create an Eclipse project *assignment1-<yourid>* , where *<yourid>* is replaced by your student id. Within this project, create a package *nz.ac.massey.cs159272.ass1.id<yourid>*. In this package, create the following classes (as explained in part 1-part4).

Part 1 Domain Model [4 marks]

1. Create the following classes in your package:
   i. **Course**, with properties for number and name (both **String**)
   ii. **Address**, with properties for town (**String**), street (**String**), post code (**String**) and house number (all **int**)
   iii. **Student**, with properties for name, first name, id (all **String**), dob (for "dateOfBirth", of type **java.util.Date**), course (of type **Course**) and address (of type **Address**)
2. All properties should be implemented using getters and setters, those can be generated
3. All classes should override **equals(Object)** and **hashCode()**, these methods must adhere to the usual contract between **equals** and **hashCode,** those can be generated
4. **Student** should have a **clone()** method that is a combination of *deep* and *shallow* clone: deep clone should be used for the **address** and **dob**, a shallow clone should be used for **course**

Part 2 Persistency [4 marks]

1. Create a class **StudentStorage** with the following three static methods:
2. **void save(java.util.Collection<Student>,java.io.File file) throws IOException** – saves a list of students to a binary file. Note that the data of referenced objects (**address** and **course**) should be saved as well. ***Please also read the hints at the end of the document****!*
3. **void save(java.util.Collection<Student>,String fileName) throws IOException** – saves a list of students to a binary file file with a given name.
4. **No code should be replicated (copied & pasted) between the two versions of save(..).**

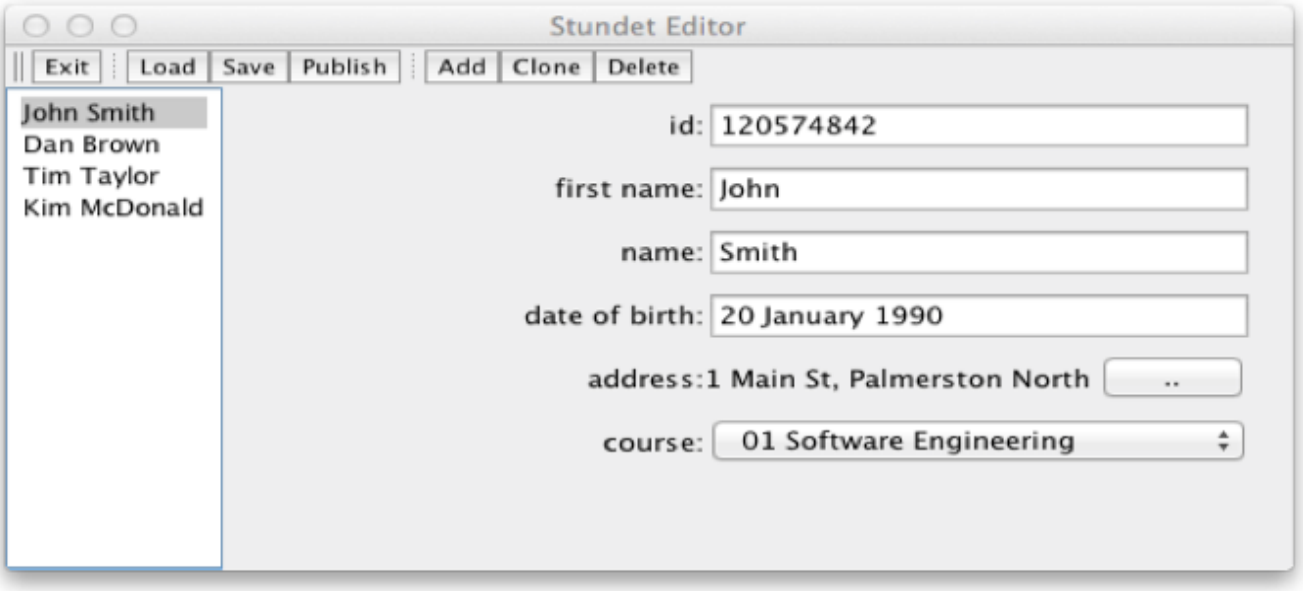
5. **java.util.Collection<Student> fetch(java.io.File file) throws IOException** – reads student data from a binary file.
6. **save(..)/fetch(..)** should preserve referential integrity: for instance, if two instances of **Student** share the same (==) course, then this should be preserved. I.e., when loading these two instances from a file, they should reference the same **Course** instance.

Part 3 Testing [4 marks]

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 **fetch(..)** 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. the roundtrip test should test whether **save(..)** and **fetch(..)** preserve referential integrity
4. write tests to check whether the exception declared by **fetch(..)** work as expected when a invalidfile name is used

Part 5 Graphical User Interface(GUI) [3 marks]

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
3. for instance, this user interface could look as follows:



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

4. a table (using **JTable**) based "spreadsheet-like" user interface is also acceptable. The save / load functions in the toolbar should save / load data to/from files that are selected using file selection dialogs that pop up when the respective buttons are pressed. You can use the class **javax.swing.JFileChooser** for this purpose.

Part 5 Bonus question: [1 mark]

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

Hints

1. you can use code and ideas from tutorials for the first three parts
2. for question 2, have a look at the following classes:
    1. java.io.ObjectOutputStream
    2. java.io.ObjectInputStream
3. plan how much time you want to invest to answer each question: 4 and 5 can take a lot of time, but will only give you 3-4 marks, it might be better to focus on 1-3.
4. see http://docs.oracle.com/javase/tutorial/deployment/jar/appman.html for instructions how to create executable jars (for the bonus questions).

How to submit

1. export the Eclipse project to a file *assign1-<yourid>.zip*, where <*yourid*> is replaced by your student id
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

   Upload this file to Stream. Once completed, make sure that you click Submit (**don't just save your work as a draft**!!).