

Lab Exercise 4

CS 2334

September 17, 2015

Introduction

With this lab, we consider the relationships between objects. Think about the records kept for this university. It is important that information is stored for every student and every teacher. These “objects” have different information, so they each need a unique class, but students and teachers are both people, so some information is common. We can reduce redundancy by using class inheritance.

For this laboratory, you will create classes for person, student, instructor, course and university. Think about how all these objects are related. We have provided a complete UML diagram showing these relationships, along with a set of basic methods for each object class. Your task is to implement these classes as they are shown.

Learning Objectives

By the end of this laboratory exercise, you should be able to:

1. Read and understand UML diagrams
2. Implement classes from a given specification
3. Create inheritance relationships
4. Understand the difference between *is-a* and *has-a* relationships
5. Develop testing procedures for your own code

Proper Academic Conduct

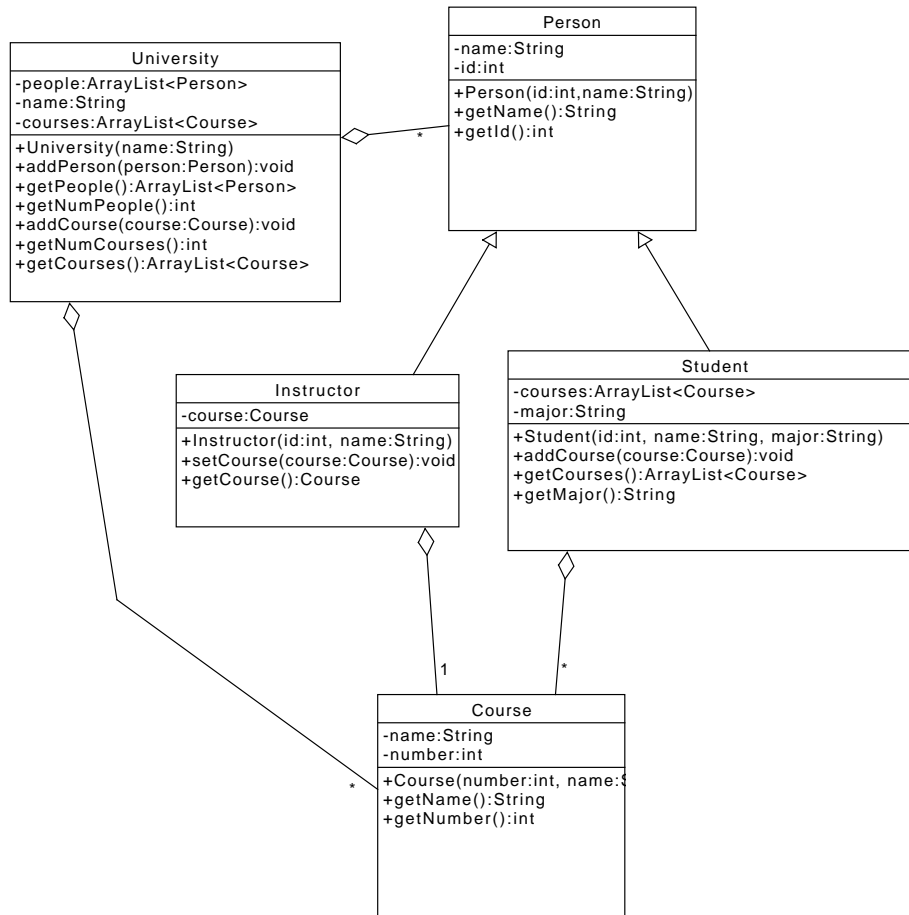
This lab is to be done individually. Do not look at or discuss solutions with anyone other than the instructor or the TAs. Do not copy or look at specific solutions from the net.

Preparation

1. Import the existing lab4 implementation into your eclipse workspace.
 - (a) Download the lab4 implementation:
`http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab4/lab4-initial.zip`
 - (b) In Eclipse, select *File/Import*
 - (c) Select *General/Existing projects into workspace*. Click *Next*
 - (d) Select *Select archive file*. Browse to the lab4-initial.zip file. Click *Finish*
2. Note that you are given only an incomplete test class to start with. You may examine this if you wish, but you will most likely want to start by creating new classes

Representing People and Courses at a University

Below is the UML representation of a set of classes that represent the people affiliated with a university and the courses offered by the university. Your task will be to implement this set of classes and an associated set of Junit test procedures.



The line from `Instructor` to `Person` indicates that `Instructor` *is-a* `Person`. You should use inheritance for this relationship. Likewise, for the `Student` and `Person` classes.

The line from `Instructor` to `Course` indicates that `Instructor` *has-a* `Course`. Observe that there is a `Course` attribute (instance variable) in the middle section of the `Instructor` box. Other than including this attribute, there is nothing special to be

done for a has-a relationship. This also goes for Student and Course, University and Course, and University and Person.

The number “1” on the line from Instructor to Course indicates that an Instructor can (for this lab) teach only one course. The “*” on the other relationships indicates that a class instance can have any number of instances of the other class (think ArrayList).

Lab 4: Specific Instructions

1. Create a new Java class for each object class described in the UML diagram
 - Be sure that the class name is exactly as shown
 - You must use the default package, meaning that the package field must be left blank
2. Implement the attributes and methods for each class
 - Use the same spelling for instance variables and method names as shown in the UML
 - Do not add functionality to the classes beyond what has been specified
 - Don't forget to document as you go!
3. Use the *Lab4Test* class and JUnit to thoroughly test all of your code
 - You need to convince yourself that everything is working properly
 - Much of the necessary testing is already covered for you, but it is incomplete. It is up to you to figure out which classes/methods are missing from the tests. Keep in mind that we have our own tests that we will use for grading.

Final Steps

1. Generate Javadoc using Eclipse.
 - Select *Project/Generate Javadoc...*
 - Make sure that your project is selected, as are the Person, Student, Instructor, Course, University, and Lab4Test classes (check for these individually!)

- Select *Private* visibility
 - Use the default destination folder
 - Click *Finish*
2. Open the *lab4/doc/index.html* file using your favorite web browser or Eclipse (double clicking in the package explorer will open the web page). Check to make sure that that all of your classes are listed and that all of your documented methods have the necessary documentation.
 3. If you complete the above instructions during lab, you may have your implementation checked by one of the TAs.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59pm on Friday, September 18th.
- Prepare your submission file:
 1. Select the project in the *Package Explorer* window.
 2. Right-click. Select *Export*
 3. Select *General/Archive File*
 4. Expand your project and verify that both the *src* and *doc* folders are selected, as well as all of their contents
 5. Enter the archive file name: *lab4.zip* (note that you may want to browse to a different destination folder)
 6. Select *Save in zip format*
 7. Click *Finish*
- Submit your zip file to the lab4 folder on D2L.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 10 points

- (10) The program is properly formatted (including indentation, curly brace and semicolon locations).
- (5) There is one problem with program formatting.
- (0) The program is not properly formatted.

Data types and method calls: 15 points

- (15) The program is using proper data types and method calls.
- (10) There is one error in data type or method call selection.
- (5) There are two errors in data type or method call selection.
- (0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

- (10) All of the required methods are implemented.
- (7) A component of one required method is missing.
- (4) A method is not implemented or components are missing from two methods.
- (0) Two or more required methods are not implemented or components from three or more methods are missing.

Proper Execution: 30 points

Output: 15 points

- (15) The program passes all tests.
- (10) The program fails one test.
- (5) The program fails two tests.
- (0) The program fails three or more tests.

Execution: 15 points

- (15) The program executes with no errors.
- (8) The program executes, but there is one minor error.
- (0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

- (5) The java file contains all of the required documentation elements at the top of the file.
- (3) The java file is missing one of the required documentation elements.
- (2) The java file is missing two of the required documentation elements.
- (0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

- (10) Every method contains all of the required documentation elements ahead of the method prototype.
- (7) The method documentation is missing one of the required documentation elements.
- (3) The method documentation is missing two of the required documentation elements.
- (0) The method documentation is missing more than two of the required documentation elements.

Inline Documentation: 10 points

- (10) Every method contains appropriate inline documentation.
- (7) There is one missing or incorrect line of inline documentation.
- (3) There are two missing or incorrect lines of inline documentation.
- (0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

- (10) The correct zip file name is used and has the correct contents.
- (5) The correct zip file name is used, but one required component is missing.
- (0) An incorrect zip file name is used or more than one required component is missing.