Lab Exercise 4: Inheritance
CS 2334

September 15, 2016

Introduction

With this lab, we consider relationships between objects. Think about the records that we keep for every item used to compute your grade in this course. Each of these *GradedItems* has a name, a date and a grade. However, depending on the type of the GradedItem, other information might need to be stored, such as a description of a project or the name of a file used for testing your project. We will use class inheritance in this lab to capture the fact that different item types have some properties in common, but differ in other properties.

For this laboratory, you will create classes for graded items, exams assignments and a list of graded items. Think about how all these objects are related. We have provided a complete UML diagram showing these relationships. Your task is to implement these classes as they are shown and a complete set of unit tests.

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.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.zip file. Click *Finish*

   (e) In *Package Explorer*: Select *lab4*

   (f) Right click, select *Build Path/Configure Build Path*

   (g) Select *Java Build Path*

   (h) Click on the *Libraries* tab

   (i) Click *Add Library*

   (j) Select *JRE System Library*. Click *Next*

   (k) Select *Workspace default JRE*. Click *Finish*

   (l) Click *Add Library*

   (m) Select *Junit*. Click *Next*

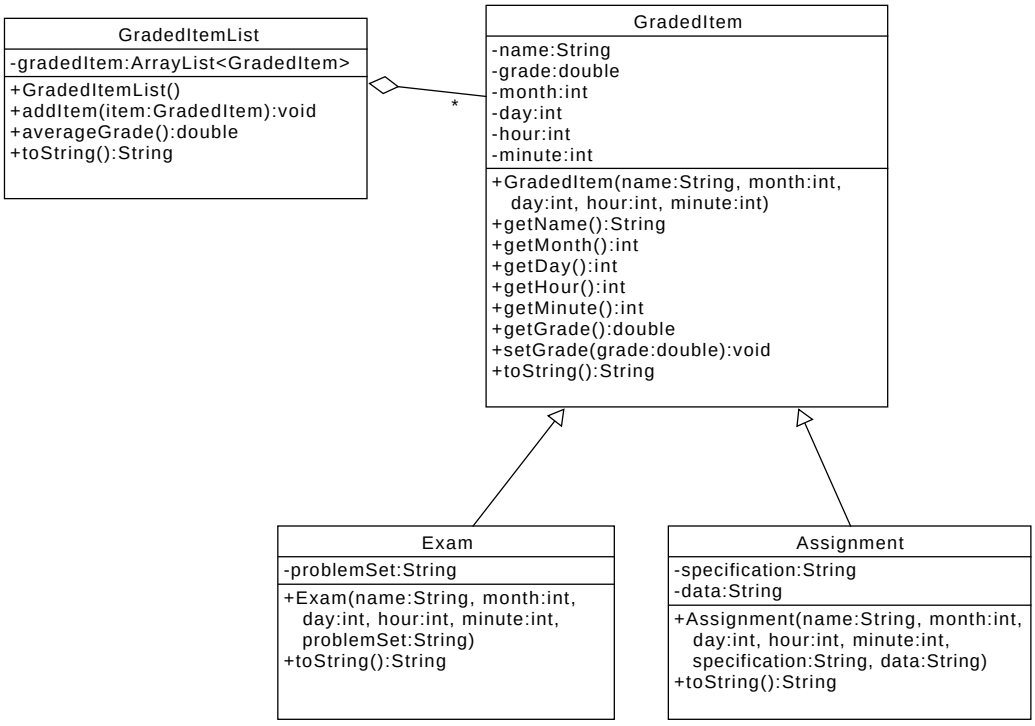
   (n) Select *JUnit4*. Click *Finish*

   (o) Click *Apply* (if Eclipse shows this button) and then *OK*

   (p) Should one or more source files be flagged in the *Package Explorer* as having errors (declared package "" does not match expected package "src"): add a space to the class file, save it, close the window and reopen the window.

Representing Graded Items

Below is the UML representation of a set of classes that represent items that are used to compute your course grade. Your task is to implement this set of classes and the associated set of Junit tests.



The line from Exam to GradedItem indicates that Exam *is-a* GradedItem. You should use inheritance to capture this relationship. Likewise, for the Assignment and GradedItem classes.

The line from GradedItemList to GradedItem indicates that GradedItemList *has-a* GradedItem. Observe that there is a GradedItem attribute (instance variable) in the middle section of the GradedItemList box. Other than including this attribute, there is nothing special to be done for a has-a relationship. The "*" on this relationship indicates that the GradedItemList can have any number of GradedItem objects. This fact is captured in the GradedItemList through the use of the ArrayList of GradedItems (this is the meaning of the ArrayList<GradedItem> notation in the type definition).

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 when you create the class
2. Implement the designated attributes and methods for each class
   - Use the same spelling and visibility 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. GradedItem.toString(): for a graded item by the name of "Final", a month of 12, day of 13, hour of 8, minutes of zero and a grade of 0.78, toString() must return a String in the following format:

```
"Final (date: 12-13 at 08:00): grade = 0.78"
```

   Note that the quotes are not part of the String. Also note that each of month, day, hour and minute must be represented using exactly two digits, and that the grade must have exactly two digits following the decimal point.
4. Exam.toString(): for the same final exam as above and a problemSet of "final.pdf", toString() must return a String in the following format:

```
"Final (date: 12-13 at 08:00): grade = 0.78: problem set = final.pdf"
```

   Note that this toString() method must make use of the toString() method provided by GradedItem.
5. Assignment.toString(): for an assignment called "Lab 1", a specification of "lab1.pdf" and data of "calendar.csv", toString() must return a String in the following format:

```
"Lab 1 (date: 08-26 at 23:59): grade = 0.50: specification = lab1.pdf; data ↩
    source = calendar.csv"
```

Note that the arrow on the right-hand side of the box is not part of the returned String (there is one space between "data" and "source") and that this String occupies exactly one line. Alsoe note that this toString() method must make use of the toString() method provided by GradedItem.

6. GradedItemList.toString() must return a String that is a concatenation of all of the individual items in the list, with a newline character ("\n") after each item. The order of the items must be the same order as they were added to the GradedItemList.

7. Create one or more JUnit classes to thoroughly test all of your code

   - You need to convince yourself that all of your primary classes are covered and that they are working properly
   - We provide a small example test class. However, this test class is incomplete
   - We will test your code with a separate set of unit tests

Final Steps

1. Generate Javadoc using Eclipse.

   - Select *Project/Generate Javadoc...*
   - Make sure that your project is selected, as are the Driver, Pokemon, PokemonTeam classes
   - 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 both of your classes are listed and that all of your documented methods have the necessary documentation.

Submission Instructions

- All required components (source code and compiled documentation) are due at 11:59:00pm on Friday, September 16th.
- Method 1: Submit through Eclipse
  1. From the *Window* menu, select *Preferences/Configured Assignment*.
  2. Select your project.
  3. From the Project menu, select *Submit Assignment*.
  4. Under *Select the assignment to submit*, select *CS 2334/Lab 04 (2334): Inheritance*.
  5. Click *Change Username or Password....* Enter your Web-Cat username and password. Click *OK*. You should only need to do this step once per session.
  6. Click *Finish*.
  7. Your browser should automatically open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.
- Method 2: Submit directly to the Web-Cat server
  1. From the File menu, select *Export*.
  2. Select *Java/JAR File*. Click *Next*.
  3. Select and expand your project folder.
  4. Select your *src* and *doc* folders. **Do not include csv files.**
  5. Select *Export Java source files and resources*.
  6. Select an export destination location (e.g., your *Documents* folder/directory). This file should end in *.jar*
  7. Select *Add directory entries*.
  8. Click *Finish*.
  9. In your web browser, login to the Web-Cat server.
  10. Click the *Submit* button.
  11. Browse to your jar file.

12. Click the *Upload Submission* button.
13. The next page will give you a list of all files that you are uploading. If you selected the correct jar file, then click the *Confirm* button.
14. Your browser will then open a Web-Cat page that shows your submission being graded. After a short wait, the page will show a report of your submission. See the main class web page for a link that describes the Web-Cat output.

Rubric

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

Correctness/Testing: 45 points

The Web-Cat server will grade this automatically upon submission. Your code will be compiled against a set of tests (called *Unit Tests*). These unit tests will not be visible to you, but the Web-Cat server will inform you as to which tests your code passed/failed. This grade component is proportional to the fraction of tests that your code passes (so 22.5 points means that your code passed half of the tests)

Style/Coding: 20 points

The Web-Cat server will grade this automatically upon submission. Every violation of the *Program Formatting* standard described in Lab 1 will result in a subtraction of a small number of points (usually two points). Looking at your submission report on the Web-Cat server, you will be able to see a notation for each violation that describes the nature of the problem and the number of subtracted points.

Design/Readability: 35 points

This element will be assessed by a grader (typically sometime after the lab deadline). Any *errors* in your program will be noted in the code stored on the Web-Cat server, and two points will be deducted for each. Possible errors include:

- Non-descriptive or inappropriate project- or method-level documentation
- Missing or inappropriate inline documentation
- Inappropriate choice of variable or method names

- Inefficient implementation of an algorithm
- Incorrect implementation of an algorithm
- Incomplete coverage of your Unit Tests. We expect that your unit tests will test all lines of your code

If you do not submit compiled Javadoc for your lab, 5 points will be deducted from this part of your score.

Note that the grader may also give *warnings* or other feedback. Although no points will be deducted, the issues should be addressed in future submissions(where points may be deducted).

Bonus: up to 5 points

You will earn one bonus point for every two hours that your assignment is submitted early.

Penalties: up to 100 points

You will lose ten points for every minute that your assignment is submitted late.