# Lab Exercise 4: Inheritance and Polymorphism
# CS 2334

September 14, 2017

## 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 *Assignments* has a name, a date, a score, a total point count and a weight. We will use class inheritance in this lab to capture the fact that different types of Assignments have some properties in common, but differ in other properties. For example, a Lab might have a specification document, and a Project might have both a specification and a data file. In addition, we will use class aggregation to represent a list of Assignments.

## Learning Objectives

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

1. Read and understand UML diagrams involving class inheritance
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. Download:
   `http://www.cs.ou.edu/~fagg/classes/cs2334/labs/lab4/lab4.zip`

2. Import into your Eclipse Workspace:

   (a) *File* menu: *Import*

   (b) Select *General/Existing Projects into Workspace* and then click *Next*

   (c) *Select archive file*: browse to the lab4.zip file

   (d) Click *Finish*

3. Once you create the new project, it may not initially know where to find the standard Java libraries (it varies depending on your configuration).

   (a) *Project* menu: Select *properties*

   (b) Select *Java Build Path / Libraries*

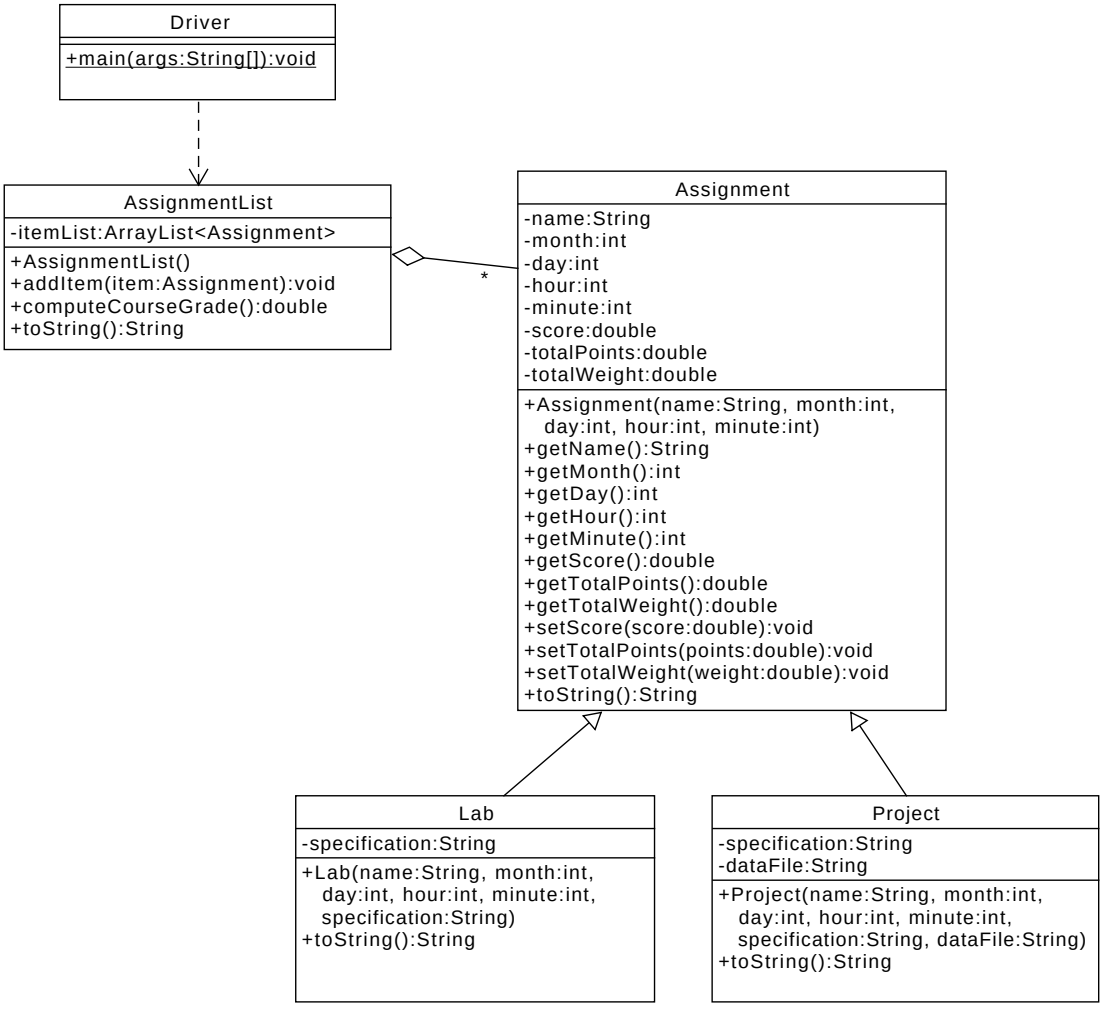
   (c) If the *JRE System Library* is not listed, then select *Add library: JRE System Library* and click *Next*. Select *Workspace default*. Click *OK*

   (d) If the *JUnit Library* is not listed, then select *Add library: JUnit* and click *Next*. Select *JUnit 4*. Click *Finish*

   (e) Click *Apply* and then *OK*

# Representing Assignments

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 Lab to Assignment indicates that a Lab *is-a* Assignment. You should use inheritance to capture this relationship. Likewise, for the Project and Assignment classes.

The line from AssignmentList to Assignment indicates that AssignmentList *has-a* Assignment. Observe that there is an itemList attribute (instance variable) in the middle section of the AssignmentList 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 AssignmentList can have any number of Assignment objects. This fact is captured in the AssignmentList through the use of the ArrayList of Assignment (this is the meaning of the ArrayList<Assignment> notation in the type definition).

How much a particular Assignment contributes to your course grade is determined by three properties:

1. **score** is the number of points that you have received on the assignment,
2. **totalPoints** is the number of points that are possible for the assignment, and
3. **totalWeight** is the weight of this Assignment relative to other assignments.

AssignmentList.computeCourseGrade() returns your score for the class based on the list of assignments. In particular:

$$return \quad = \quad \frac{\sum_{i=0}^{n-1}(totalWeight_i \times score_i / totalPoints_i)}{\sum_{i=0}^{n-1} totalWeight_i}$$

where $n$ is the total number of Assignments in the list, and $totalWeight_i$ refers to the $totalWeight$ of the $i^{th}$ Assignment. If $n = 0$, then the returned course grade must be zero.

# Lab 4: Specific Instructions

1. We have provided a complete implementation of the *AssignmentTest* class, and partial implementations of the *AssignmentList* and *Driver* classes.
2. Create a new Java class for each of the *Assignment*, *Lab* and *Project* classes, as 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
3. 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!

4. Assignment.toString(): for a Assignment item by the name of "Unit Testing", a month of 12, day of 1, hour of 23, minutes of zero, score of 0.78, totalPoints of 0.99 and totalWeight of 0.30, toString() must return a String in the following format:

```
Unit Testing (date: 12-01 at 23:00): score = 0.78; totalPoints = 0.99; totalWeight = 0.30
```

   Note that each of month, day, hour and minute must be represented using exactly two digits, and that the score, totalPoints and totalWeight must have exactly two digits following the decimal point. The location and number of spaces must also match this format.

5. Lab.toString(): for the same Unit Testing object as above and a specification of "UnitTesting.pdf", toString() must return a String in the following format:

```
Unit Testing (date: 12-01 at 23:00): score = 0.78; totalPoints = 0.99; totalWeight = ↩
    0.30; specification = UnitTesting.pdf
```

   Note that in the box above, the *left arrow* character and the following newline are not part of the returned String (they were added so that the String would fit in this document).

   This toString() method must make use of the Assignment.toString() method.

6. Project.toString(): for the same Unit Testing object as above, and a specification of "UnitTesting.pdf" and dataFile of "Snacks.csv", toString() must return a String in the following format:

```
Unit Testing (date: 12-01 at 23:00): score = 0.78; totalPoints = 0.99; totalWeight = ↩
    0.30; specification = UnitTesting.pdf; data file = Snacks.csv
```

   This toString() method must make use of the Assignment.toString() method.

7. Examine our AssignmentList.toString() implementation. It returns 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 is the same order in which they were added to the AssignmentList.

8. Create JUnit classes to thoroughly test all of your code (called *ProjectTest*, *LabTest* and *AssignmentListTest*)

   - You need to convince yourself that all of your primary classes are covered and that they are working properly
   - 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 **lab4** project is selected, as are all of your Java files
   - Select your *doc* directory
   - Select *Private* visibility
   - Use the default destination directory
   - 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 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 7pm on Saturday, September 16)

  **Submission must be done through the Web-Cat server.**

- Use the same submission process as you used in lab 1. You must submit your implementation to the *Lab 4: Inheritance* area on the Web-Cat server.

## Hints

- See the API for *ArrayList*
- See the method *String.format()*

# 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 a product of the fraction of **our tests** that your code passes and the fraction of **your code** that is covered by *your tests*. In other words, your submission must perform well on both metrics in order to receive a reasonable grade.

**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 (up to 10 points)
- Missing or inappropriate inline documentation (2 points per violation; up to 10 points)
- Inappropriate choice of variable or method names (2 points per violation; up to 10 points)
- Inefficient implementation of an algorithm (minor errors: 2 points each; up to 10 points)
- Incorrect implementation of an algorithm (minor errors: 2 points each; up to 10 points)

If you do not submit compiled Javadoc for your project, 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. For a submission to be considered *on time*, it must arrive at the server by the designated minute (and zero seconds). For a deadline of 9:00, a submission that arrives at 9:00:01 is considered late (in this context, it is one minute late).

After 15 submissions to Web-Cat, you will be penalized one point for every additional submission.

For labs, the server will continue to accept submissions for three days after the deadline. In these cases, you will still have the benefit of the automatic feedback. However, beyond ten minutes late, you will receive a score of zero.

The grader will make their best effort to select the submission that yields the highest score.