CSE 20212 Fundamentals of Computing II
Spring 2015

Lab handout for Week of February 2

Objectives

1. Practice with inheritance (Chapter 12) and polymorphism (Chapter 13)
2. Inheritance vs. composition example
3. Have fun!

Part 1 (in-lab activities)

1. (1 point) Report to lab **on time**. Attendance will be taken at the scheduled lab time.

2. For this lab, you can work in a team of 2; however, your teammate must share the same lab section.

3. Blackbeard has determined that pillaging is no longer for him and he desires to create a full-featured bank where his mates (and some enemies too, maybe) can keep checking and savings accounts in addition to borrowing his plunder (at a reasonable pirate rate, of course). Design an inheritance hierarchy based on a BankAccount. Add at least three variables to your base class.

4. Make at least two new derived classes from BankAccount. Examples include but are not limited to: checking account, savings account, boat loan, credit card, etc. Please add at least two variables to the derived class that are unique. For reference, the boat loan class can look like the following:

   ```
   class BoatLoan : public BankAccount {
   ```

   and two fun variables are: maxLootStorage, numOfCannons. Remember, we want to model real-life (or quasi-imaginary things) with object-oriented code so have fun and be as creative as possible in this framework!

5. Add fun print functions to your two derived classes. For example, here is one for a boat loan (probably more important to pirates than to the rest of us):

   ```
   void BoatLoan::print() {

           cout << "You only have " << numberofMonths << "until your galleon is
           your own!" << endl;
           cout << "Please fill up your " << maxLootStorage << "sqft responsibily."
     }
   ```

6. Check out with a TA when you have completed your in-lab portion. If you have time, you can proceed to Part II.

Part 2: What can you do for Amazon?

1. Online vendors offer varied shipping options, each with specific costs to them even though it may be "free" for the customer. Here, we will develop an inheritance hierarchy for packages to help track total costs for the company.

2. Develop a base class called "Package" that contains at least the following private data members for both the sender and recipient of packages: name, address, zip code. In addition, store the weight of the package and the cost per ounce to ship the package, and both should be non-negative. This will be important to determine the cost for the company even though the customer may not pay this cost directly (i.e., Amazon Prime).

3. Provide a public base class member function called CalculateCost() that returns a double indicating the cost to ship a package, which is simply the result of multiplying the cost per ounce for shipping by the weight of the item(s).

4. Develop a derived class called TwoDayPackage that inherits from the base class Package. UPS – for this example – would charge Amazon a flat fee for expedited delivery that may or may not be passed on to the customer. Represent this fixed additional cost as a variable initialized via TwoDayPackage's constructor. Refine TwoDayPackage's calculateCost function by adding this fixed additional cost to the result of the base class' calculateCost function. Please refer to the text or ask us how to do this if you are unsure.

5. In this lab, suppose UPS charges an extra fee per ounce for overnight delivery. Develop another derived class called OvernightPackage with an additional data member for extraCostPerOunce. As above, redefine the calculateCost function in OvernightPackage: add this additional cost to the cost per ounce to calculate total shipping cost for the company. You ideally should use the base class' calculateCost function, but it is also acceptable to calculate this cost directly in the new derived class function.

6. Provide a small driver program that will display the contents of a derived TwoDayPackage and an Overnight package. Make the output slightly different so the TAs can tell there is a difference during grading.

Part III: Finishing up

1. Many programs that use inheritance could be written with composition (and visa versa). Rewrite BankAccount to use composition (i.e., include a BankAccount object in the CreditCard class, similar to lab #1 with cols in a board. After you complete this exercise, write a few additional paragraphs in your report on which approach (composition vs. inheritance) is more natural and why? For grading please include a simple driver program that creates and tests at least one object that contains a bankAccount for this question.

2. Use the Package inheritance hierarchy created in Part 2 to create a program that displays address information and calculates shipping costs for several packages using base class pointers/polymorphism. The program can use either a static array or a vector of pointers (your call) but must loop through at least three TwoDayPackages and three OvernightPackages and compute the total cost of sending these packages by calling virtual functions. For example, this can be the cost of processing/shipping a day's orders.

3. Discuss with your partner what would have to be done to make your base class abstract and what would be needed to make the derived classes concrete. You should mention both the technical difference (how) and the big picture difference (why) in your answer. Please summarize this discussion in your individual reports in addition to the three normal questions. Students should submit their own report.txt for this lab. **This is often a significant midterm question.**

Coding challenge! Lightning round (optional)

This week's topic is preparation….

Like any great entrepreneur, the quality of your software often is directly related to the quality of your preparation.

As an intro of sorts to the software engineering we are beginning to cover, note that the world of shapes is very rich and may be an ideal case study for inheritance. In an optional text file submitted with the answers to the required questions, list all of the shapes you can think of – both two and three-dimensional – and form them into a complete Shape hierarchy with as many levels possible.

At a minimum your hierarchy should have Shape as the base class and two immediate derived classes: Two-dimensional and Three-dimensional. These two immediate derived classes can have as many additional levels as you see fit (e.g., Quadrilateral).

Up to $10 coder dollars will be awarded for solutions to this exercise. Note that you do not need to write any code for this problem.