Object Design

Return to: ____

Members Present: ____

As always choose one person to lead the discussion and keep the group on track…choose a second person who will record the final results for the team.

Task 1: Lab review

In your groups, share your worksheets and solutions to lab 0204 (which extends to 0206). These are rhetorical questions and don't need to be answered on paper but you should share your code and discuss these answers in the group.

1. How did each team member solve the GPA calculation problem with the arrays and with the enum type?
2. How did each team member test their solution? What were the test examples and what should the result have been?
3. How did each team member solve the problem with adding a graded option to the Grade(s) enum type? How did this affect the calculation?
4. Discuss the following in relation to classes and class design:
    a. Consider the role of the enumerated class. In what situations would an enumerated class be preferable over a regular class to describe participants in an application (such as Rainfall for PA1). In other words, describe the properties of the class that would make it a candidate for an enumerated class rather than a regular class.
    b. In what situations would you NOT want to create an enumerated type to solve a particular problem (again thinking of low level classes like Rainfall, Box from last semester or your grades classes)?
    c. In what situations would you build behaviors into your enumerated class rather than simply use a list of values?
    d. In what situations would a simple list of values be helpful to solving a problem?

Task 2: Designing objects – beginning the design process

Read the attached description of objects and the problem statement. You should also refer to the Gaddis book chapter 6.7 where it talks about classes and responsibilities.

1. Individually, read the problem description at least twice slowly. While you are reading it, pay particular attention to the important noun phrases and important verb phrases that describe the major players or entities in the problem.
2. Individually, read a third time, this time underlining the major nouns, those that you think should become possible objects in a problem solution.
3. Individually, read again, this time underlining with a different color those nouns that may be instance variables or important to describing the state of the particular one of the nouns defined in step 2.

4. Individually, read again, and circle the important actions or verb phrases that define what behaviors the program must exhibit.

5. Now in your group, share your results. The cards represent the UML diagram for each of the classes that you are designing and the final design should be placed on the card. The sticky notes represent members of the class that you are not quite sure about either where they go or their necessity. They can be moved around until you are happy with your class design.

6. From your collective work, decide on which are your major classes and write them on your "UML" diagram (index cards), one card per each class. Are all of these entities classes or are some of them enumerations? If enumeration write that at the top like our UML diagram did last time.

7. From your collective work, decide on which descriptors belong with which of the major entities. Write these on the appropriate index card or alternately if you are not sure, put them on a sticky note and attach to the card.

8. From your collective work, decide what the relationship among the different entities is. Do we use-a class or have-a class or is the class an end point and does not have or use any other classes?

9. Based on the description in the problem statement, which of the verb phrases are key actions of any of the classes? Write those as methods in the method portion of the card, or alternately on a sticky note until you are sure. Do not concern yourself with parameters or return types unless that is obvious from the description.

Task 3: Refinement phase

At this point, you have a beginning of each of your classes. Now stop and step back a minute.

1. How will each of the entities interact with each other? Do a brief walkthrough of the process using the entities, at this point disregarding "housekeeping details" such as creating objects, performing the calculations. You should simply take each task in the process of serving a customer and make sure that you have a "player" involved in that process and a method for handling that process.

2. At this point, you should be able to write in the methods and attributes on your card. Do so if you have not already done so. Be sure to put visibility modifiers on each of the members of the class.

3. Look at each of the classes (cards). You will need to create a constructor for each of your classes. What should the constructor take in as parameters (if anything)? What instance variables will your constructor need to set?

4. Look at each of your methods. What parameters do you need (if any)? What is the return type of the methods?

5. Look at each of your classes. What accessor methods or mutator methods are important to each class? Be sure to define the parameters and return types of those methods as well.

6. Did you define a toString method? Each class should have a toString defined (except your driver).

7. This particular problem talks about the issue of error checking. How have you built that into your structure? In which class(es) will it occur and what methods will be helpful to the validation process?

8. Look at each method and on scratch paper or sticky notes, write a very brief pseudo code description of what that method may do. Does it seem that there are a lot of steps? Are there any steps that should be their own method? Add additional methods as helper (or private) methods in your class again thinking about what parameters and returns you might need.

9. Finally, go through the whole design again and make sure that you understand how you would code this solution. (Don’t code yet, but make sure that you understand how you would go about coding it.)

When finished, turn in your card deck (and return pens, unused cards and sticky notes to the pen box).

Case: Designing a class from a problem statement

Background (Refer also to Chapter 6 in the Gaddis books):

In OOP (Object Oriented Programming) your focus is not on the series of steps needed to carry out the action, but in assigning roles and responsibilities for those actions. The "who" is far more important that just looking at the "what" and by manipulating or using the players we can get the task done, although it is important to keep track of the big picture.

Think about a play. In a play, each actor is simulating some character. Each actor has behaviors and interacts with the other characters in clearly delineated ways. The script tells each actor what role he or she should play in the production. It tells them how they should talk, how they should behave, and with whom they interact. Sometimes actors are in groups…they all carry out the same actions (like crowds or passengers on a plane); in others, individuals will have very specific behaviors. A director of sorts controls when and how each of the participants "behaves" in the production.

In object oriented programming then, while we have a task to do, we want to look at it from the point of view of the players involved in that task. What players are important to the task and how will they interact? What behaviors will they each need to carry out? How can we describe those players (what describes their current state)? We also know that users of each of the players will choose which of the behaviors they will use at what time…no sequence is necessarily imposed.

So how do you find which players are important?...In the project description, the players and their attributes (instance variables) will usually be the nouns and noun phrases. Actions will be verbs. So just like we name variables with nouns and methods with verbs, we can do the reverse to find the actors, their attributes and their behaviors.

Problem Statement: A small concession stand that operates during non-varsity athletic events on campus has had an increasing problem with cash shortages and they have determined that the problem seems to be the poor math skills of some of their employees. To remedy the problem, they want to install a cash register program that will do all of the calculations for the cashiers to eliminate math as a source of their difficulty. The concession stand offers for sale a very limited number of items, no more than 20 different items, and these have a fixed price per item. For each sale, the cashier should be able to enter a code for the item and the quantity and when they are finished entering items, the program should echo the list of items, their price, the quantity, and the total for each item. In addition, it should print the total, tax (at 11% Harrisonburg meals rate) and the total purchase. When the customer presents their cash to the cashier, the cashier should enter it and the register should show how much change to return in total and by denomination (no higher than $20.00.) We must also error check that the quantity is valid, the code is valid and the money entered is valid.

For example, a typical customer might by one bag of popcorn, 2 candy bars, and 3 sodas. Popcorn and candy bars sell for $1.00 each and soda is $.75. The cashier would enter:

```
PC 1
CB 2
SD 2
```

and the register would display:

```
Item           QuantityTotal
Popcorn        1            $1.00
Candy bar      2            $2.00
Soda           3            $2.25
                            -------
Total                       $5.25
Tax                         $0.58
Total                       $5.83
Cash tendered               $20.00
Change                      $14.17
tens           1
ones           4
dime           1     nickel 1     penny  2
```