COP 3331: Object Oriented Design
Project 3
Due July 17th at 11:59 P.M.

You are not allowed to use the Internet. You may only consult approved references*.
This is an individual project.
This policy is strictly enforced.

Project Objectives

To learn more about

- Inheritance
- Polymorphism
- File I/O

Submission Guidelines

- A zipped Visual Studio solution submitted through Canvas. We will be using VS 2013 for this course.
- **Points will be deducted for not submitting a proper VS solution.**
- **No late submissions will be accepted.**

For full credit, the code that is submitted must:

- Have a comment block including your name.
- Be implemented in a file using the specified file name, if applicable.
- Be readable and easy to understand. You should include comments to explain when needed, but you should not include excessive comments that makes the code difficult to read.
  - Every class definition should have an accompanying comment that describes what is for and how it should be used.
  - Every function should have declarative comments which describe the purpose, preconditions, and postconditions for the function.
  - In your implementation, you should have comments in tricky, non-obvious, interesting, or important parts of your code.
  - Pay attention to punctuation, spelling, and grammar.
- Have no memory leaks.
- Use no magic numbers. Use constants in place of hard-coded numbers. Names of constants must be descriptive.

*You may use all class material and the C++ reference sites posted at the beginning of the semester. Any additional resources must be pre-approved. You must cite all references used.

Some additional coding guidelines which will help you to create efficient, organized code (optional)

- Using #define header guards to prevent multiple file inclusion. The typical form of the symbol name is <FILENAME>_H_
- Limit the amount of code you copy and paste. If you need to reuse a section of code, you should create a function to place it in instead.
- Define functions inline only when they are simple and small, say, 5 lines or less
- Function names, variable names, and filenames must be descriptive. Avoid confusing abbreviations.
- It is best to use spaces instead of tabs. Most IDEs and text editors will give you the option to insert spaces in place of tabs (typically 4).

Project Details

For this project, you will be implementing inheritance and polymorphism in your program. You are to create a new class ***Item*** that will serve as the base class for your ***EventItem*** and ***ConcessionItem***. The ***Item*** class should

1. Be an abstract base class (So you won't be able to construct an ***Item*** object)
2. Contain all member variables shared by the ***EventItem*** and ***ConcessionItem*** classes
3. Contain any Getter/Setter functions needed to access and modify these shared variables. You should not have any Getter/Setter functions for these shared variables in either of the derived classes.

In addition to this, you need to use polymorphism at least once somewhere in your program. By using polymorphism, you will be allowed to access ***EventItem*** and ***ConcessionItem*** objects through an ***Item*** pointer. In combination with the ***virtual*** keyword, your program will be able to group all types of ***Items*** together, but still be able to differentiate between them on the fly at runtime.

Hint: The cart class is a good place to start thinking about polymorphism!

Extra Credit [10pts]

For extra credit, add the ability to maintain your kiosk's inventory between runs. Consider that you start out with an inventory of 2 movies and 2 concession items:

- "The Avengers" – 100 tickets
- "Star Trek" – 100 tickets
- "Large Farva" – 150 available
- "A Liter-a Cola" – 150 available

At the close of business your kiosk has sold 25 of each item

- "The Avengers" – 75 tickets
- "Star Trek" – 75 tickets
- "Large Farva" – 125 available
- "A Liter-a Cola" – 125 available

the kiosk is then powered down for the night, saving its inventory list to non-volatile memory. When powered back up the next day, this data is read from the non-volatile memory back into your program to reflect the current stock available. That is, your inventory at the start of business day 2 is

- "The Avengers" – 75 tickets
- "Star Trek" – 75 tickets
- "Large Farva" – 125 available
- "A Liter-a Cola" – 125 available

In order to accomplish this task, you will need to do a few things

1. Shut down your program gracefully (meaning don't crash it just to get it to stop)
2. Write to a text file all pertinent information about your inventory (items, stock, etc)
3. Be able to read all of this data in order to recreate your inventory from the previous run of your program

This is identical in nature to the inventory initialization through file I/O from project 2; the flow of data simply moves in the opposite direction. In fact, if you implemented the file I/O initialization in project 2, you should already have number three done.