CS632p: Python Project 2 – My Buddy's Business Software
OPTIONAL PROJECT WORTH 10% OF YOUR GRADE
Joseph Everett jeverett@pace.edu

Due date: 12/18/2019 11:59 PM
LATE SUBMISSIONS WILL NOT BE ACCEPTED

READ THESE INSTRUCTIONS VERY CAREFULLY

Overview
Your friend runs a business where he or she buys and sells a particular type of commodity (e.g. bikes, wine, clothes). It is up to you to decide what it is that your friend buys and sells. Be creative and only copy my idea if you are really stuck. For example, John's friend Andy buys and sells used computer games and so I will use that as a running example in the project specification below. Substitute your idea everywhere you see [game] below. You are to come up with something different, but maybe similar. As you will see, you will start to get ideas as to how your software could be adapted to buy and sell many different types of things. But do not try to do that in this project!! Stick to one commodity.

Your task is to develop some object-oriented software to allow your friend to manage his/her inventory and to work out if he/she is making a profit or not. This software is to offer a text based interface so that it will run on any slow machine anywhere in the world with Python 3 installed. The software will allow your friend to : 1) enter details into a database of [game] which they have bought from customers; 2) see what [game] are available for sale; and 3) when a [game] is sold, record its sale and remove it from the inventory so it cannot be sold twice. **Do not use an actual database for this project**, please use a file instead. You will want to update this file after you have made purchases and sales when each interactive session closes so as to keep the stock/inventory up to date. This will provide for a much more useful application; without files, all the data will be lost as the end of each session. I suggest you leave this part of the project until the main functionality is working.

Deliverables
You are expected to email a zip file with your working program and source code, compiled executables, with documentation and instructions on how to run the program. Marks will be lost if either of these elements is missing. Documentation should include at a minimum a UML class diagram / flow chart explaining the design and proof of testing of the program, showing all its functionality in action. You should also describe what you should do to improve your program if you were to design and develop a second (i.e. new / improved) version. Make sure you have a cover page which clearly gives the course details and your name. Remember, I am your customer and presentation is important.

THERE ARE NO EXTENSIONS FOR THIS OPTIONAL PROJECT

Marking

Students are encouraged to help each other if they wish, but DO NOT SHARE YOUR CODE OR DOCUMENTATION. If you have sought assistance, you **must** indicate this in your documentation. If you have had help with parts of your code, you **must** indicate this with comments in your program at applicable places. You will not be penalized for seeking help – this is how you learn (myself and the CSIS tutors are here to help you). Please make sure to remind yourself of the CSIS policy on plagiarism (included with the course syllabus). This policy will be upheld, so you will be expected to explain your code to me.

Marks will be allocated as follows:

- Fully functioning application as specified (commented appropriately) 30%
- Quality of design material and class structure 15%
- Good use of methods (and naming of methods and variables) 15%
- Evidence of testing and error handling in code (hint – run scenarios) 15%
- Quality of documentation and overall deliverables 10%
- Sales assistant enhancement 10%
- Thoughtfulness of suggestions for improving your project 5%

Bonus marks will be given for any useful functionality that you implement.

[Games] Exchange Program – Details

Each [Game] object should store (at a minimum) the following information:

- Title
- Genre
- Publisher
- Platform
- Condition
- Purchase Price
- For Sale Price
- Sold Flag

NOTE: Whatever it is you are buying and selling, it MUST have a set of attributes like this that helps to make each item being sold slightly different.

The [game/inventory] object will be a collection type datastructure (e.g. list, dictionary, or one of the advanced structures we learned about)

Your program should be capable of providing the following user experience (please elaborate on this and make it unique to your software):

```
Welcome to Andy's Computer Games Exchange:
   1) Purchase game
   2) Games for sale
   3) Sell a game

   Enter Option:  1

   PURCHASE GAME

   Title: Call of Duty: Black Ops IV
   Genre:  Military First person shooter
   Publisher: Activision:
   Platform: Xbox
   Condition: Good
   Purchase Price: 4.50
   Sale price: 9.99

   1) Purchase Game
   2) Games for Sale
   3) Sell a Game

   Enter option: 2

   GAMES FOR SALE

   Number   Title      Genre                   Condition        Price
   1        Halo3      SciFi Fist Person Shooter  Good             9.99
   2        Pacman     Classic                    Average          7.50

   1) Purchase Game
   2) Games for Sale
   3) Sell a Game

   SELL A GAME
   Which Game:  1
   What Price:  9.00

   Sold Halo3

   You just made $4.50 profit
```

You will need to create at least 2 classes, one called [game] this is the actual [game] object) and one called [GamesInventory] (this is an array of [games] for sale – hint, rather than remove [games] from the array when they are sold, use a flag to mark them as sold). You will also need to create an additional driver class for the program. This will contain the

components for running the program (e.g. main program) which will create objects and call their methods. Don't allow someone to sell a [game] and not to make a profit on it (that's just not good business sense!).

Sales Assistant Enhancement (get everything else working before you tackle this)
Your friend is looking to hire some help – his/her business is really growing. You need to extend the above program so that it remembers (a) how many [game] sales have been made by each assistant and (b) how much money each sales assistant has made. Add a menu item as follows:

```
4.  Display Sales Assistance

Enter option:  4

SALES ASSISTANTS
PERSON        number of games sold       Total profit made
Joe           30                         223
Fiona         8                          50
John          62                         534
Eric          0                          0
TOTAL         100                        807
```

For this to work, you will need to create a new class called SalesAssistant that will store each name and a running total of the profit they have made. You will also need to create a way to add/remove sales assistants. You can predefine the above salespeople in the program but you will also need to be able to create them manually. Now modify the sales method in the [Game/Inventory] class so that the user of the program is asked for sales assistant's name each time a sale is made. The user experience will/can now look like this:

```
Enter option:  3

SELL A GAME
Which game:  1
What price:     9.00
Which sales person:  Joe
Sold Halo 3
Joe, you made Andy a profit of $4.50
```

You can also create a login/logout experience for a salesperson, but this is more advanced.

TESTING QUESTIONS / SCENARIOS

These are the kind of scenarios that I might create to test my game exchange software for Andy, my buddy. So think of what you could try out to test your software, recording what is expected (what you want to happen) and then what actually happens. Write these questions BEFORE coding, and try them with your design. This is the start of TEST-FIRST (aka test driven design). . Try it.

- Can you buy and sell a game and make a loss (try to buy a game for $10 and try to sell it for $5) you shouldn't be able to do this (without an override perhaps?)
- Does the application tell your friend how much profit he has made to date? Try to buy and then sell 2 games for profit to check
- What if you try to buy the same game? Can you buy multiple copies? Should you? Maybe there should be a demand?
- What happens when you enter a string in the price field when entering game details?
- What happens when you enter digits in the title field of the game
- When a game is sold, is it totally removed from the collection or just marked as sold? Do you accept returns? How does that work?
- Does the display accurately update to reflect games that are purchased and sold
- Does the application display all available games?
- Does the application display all sold games?
- Try to sell the same game twice.
- Try to sell the first game in the list? Can you?
- Try to sell the last game in the list
- Try to sell a non existing game. Do you get an error?
- Can you check how many games have been sold to date
- Can you keep buying and selling forever?
- Does the application allow you to export state information about the current stock?
- Does the application allow you to import state information about the current stock?
- Are the salespeople fixed or can you add new ones?
- Select one salesperson to make all the sales and make sure that the application calculates their profit correctly.
- Can you see who the salesperson of the day is (who made the most profit today, this week, month, etc)?
- Etc, etc, etc.

YOUR PROGRAM SHOULD NOT CRASH. TEST EVERYTHING. HANDLE YOUR EXCEPTIONS.

Have fun with this!