CSCI 204 – Data Structures & Algorithms

PROJECT 1: E-COMMERCE INVENTORY

PHASE ONE: Design and Implement the Inventory Class

This phase due dates

Check Point: ~~1/27/2023~~,1/30/2023 11:55 pm on replit

Completed Phase: 2/6/2023, 11:55 pm on replit

Read this document in its entirety before beginning the project - there are pieces throughout it that will help you begin! This document:

- First, outlines the entire set of tasks for phase 1, which is due on 2/6
- Second, specifies what is due for the Checkpoint on 1/27
- Finally, gives some advice about getting started, and how to approach the problem

Phase 1 Overall Goals

Your Goal: At a high level, your goal of this phase is to create an **`Inventory`** class that reads in data about various products and then puts them into additional classes and subclasses that you design. This will involve several specific subtasks:

- Creating an **Item** class and several subclasses, following OOP principles, that will store an inventory of various types of items (books, cds, electronics, etc).
- Creating an **Inventory** class that reads in data from several spreadsheets, building a database of items
- Creating several methods for your **Inventory** class that allow the user to interact with it (for example, computing the value of all items in the inventory)

Understanding the Project Files

You are given a set of test programs and data files in the **Project 1 Phase 1** replit folder. Your program will read data from CSV files and create the appropriate objects to build its database. These files will give rise to a total of four general categories of items in the database: Books, CD/Vinyl, Collectibles, and eBay items. The eBay items will be divided into three sub-categories, Electronics, Fashion, and Home/Garden items.

Table 1: Description of the CSV files

FILE NAME	CONTENT
`book.csv`	A collection of 10 books with title, publishing date, publisher, author, unit price, ISBN, and quantity or count in the database.

`cd_vinyl.csv`	A collection of 10 CDs or vinyls with title, artist(s), label, ASIN, date, unit price, and quantity.
`collectible.csv`	A list of seven collectibles with title, unit price, date, owner, and quantity.
`electronics.csv`	A collection of nine items with name, unit price, date, manufacturer, and quantity.
`fashion.csv`	Same type of information as in the '**`electronics.csv`**.'
`home_garden.csv`	Same type of information as in the '**`electronics.csv`**.'

Note that the amount of data is very small. But your programs should be able to work correctly with data of any size: the logic is the same.

Examine the program **`test_inventory.py`**. From the **`test_inventory.py`** program you can tell that your **`Inventory`** class needs to have the following public interface that consists of the methods that a public program can use.

Table 2: Methods of the Inventory class that are accessible to public calls

METHOD	DESCRIPTION
`Inventory()`	Constructor, i.e., the __init__() method
`check_type()`	Return the type of object, using `isinstance()`
`compute_inventory()`	Compute the total value of all items in the inventory
`print_inventory()`	Print the information of the inventory
`print_category()`	Print information by categories of the items
`search_item()`	Search items whose name contain the given pattern

The text file **`output.txt`** is the direct result of executing **`test_inventory.py`** (once your code is fully working!) Your programs, when completed, should generate a similar, if not identical result. **Pay careful attention to this file and the output, as this will guide how you create the methods above.**

Design your Inventory class, Item class, and subclasses of the Item class

Now that you have a sense how a user may use the class through programs such as **`test_inventory.py`**, you are asked to design and implement the **`Inventory`** class.

First, **draw a diagram of classes indicating the relations among the classes and subclasses (you don't have to submit this; it will just help you conceptualize your approach).** Remember the key feature in this design is to minimize the redundancy and to facilitate the future expansion of the class. Draw the diagram either using any graphics tool (you could even do this in Google Slides). As an example, this is the **`Counter`** class hierarchy we saw in our lab.

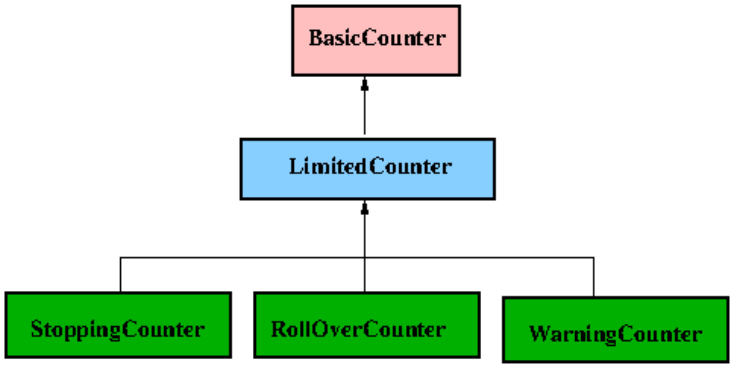

Figure: The Counter class hierarchy used in the lab

While the Inventory itself might not need any subclasses, consider the items that are in an inventory. Read the CSV files and their descriptions in Table 1. Identify what are the common features in these items that should be included in a base Item class. Specific features in items such as Book, Collectible and others indicate that they may belong to a subclass of the Item class.

IMPLEMENT YOUR CLASSES IN `inventory.py`

Implement the **`Inventory`** class, an **`Item`** class, and all subclasses of the **`Item`** class in **`inventory.py`**.

You may have certain degrees of freedom in how to design and implement these classes. You do have to pay attention to the following notes.

`Inventory`

1. **Attributes and methods:** The **`Inventory`** class should have one Python list storing all items. The **`test_inventory.py`** file will help you determine which other attributes to include in **`Inventory`** class, and the sample output will guide some of the methods and their output.
    a. *Hint: take a look at the* **test_inventory.py** *file before you get started. It creates an* **Inventory** *object on line 27. Lines 31-38 reveal two attributes you'll want that class to have, as well as an attribute your* **Item** *class should have. Working with these attribute names from the get-go will save you some debugging.*
2. **Constructor:** The **`__init__`** function should read in information from the 6 CSV files listed in Table 1 (possibly using other methods of the Inventory class).

a. To read and parse CSV files, use the existing Python package named **`csv`** (yes, the exact same name as the type of the CSV files.) Look online for template code for reading in CSVs (for example). You should write a method for your Inventory class to read all the CSV data files.

   To get started, try running the following code in your `main.py` file. What does data store? What's `data[0]`? How do you access the title of the first book?

```
import csv

with open('data/book.csv', newline='') as f:
   reader = csv.reader(f)
   data = list(reader)

print(data)
```

b. **The `__init__` method for `Inventory` should not be too lengthy.** You may want to add other methods to facilitate a concise **`__init__`**: perhaps a method that reads in files (e.g. a read_items method that takes in a file name and possibly a type), and method(s) that add individual types of items.

`Item` and subclasses

1. The csv files and **`output.txt`** should guide the attributes and **`__init__`** method for your **`Item`** class and its subclasses.
   a. We realize that these directions are more ambiguous than most projects you've had in the past - the real world is ambiguous! A lot of the work in software engineering is figuring out how to design something... you're doing the same here.

Design Guidelines

1. When implementing methods, make sure they are **cohesive** and **loosely-coupled**. ***Cohesive*** means only relevant and closely related actions should be put in one method; ***loosely-coupled*** means reduce or eliminate inter-dependency between methods.

2. **Don't make any methods too long.** If an action needs many steps to complete, consider splitting it into multiple methods. You can assume everything is public in the interface for this project. For example, you do not need to begin attribute names with an underscore, or have separate getter/setter methods.

‼ TEST YOUR CODE OFTEN ‼
You should test at each step after completing a method or an object. You can comment out sections of code in **`test_inventory.py`** to test one component at a time. For example, comment everything out except the constructor:

```
invent = Inventory()
```

that allows you to test if your constructor works. Of course, when completed, you must run the original given test program **test_inventory.py** to show that everything works fine.

Think also about how to test your code as you go by creating your own tests. For example, if you create an **Item** class with an **__init__** and **__str__** method, create an Item in the console (`myItem = Item(...)`). Then type `print(myItem)` and make sure that it prints reasonably.

By the Phase One deadline, all of the above features should be working. In particular, your test_inventory.py output should match output.txt. Don't forget about your readme, as described in the project overview.

Phase 1 Checkpoint: due 1/27

While a complete phase 1 submission is due on 2/6, you have a checkpoint on 1/27. This checkpoint is worth 5 points of your grade. To get these 5 points, you need your program that demonstrates the core functionality for your phase 1 project to be successful. The following is a *minimal* set of requirements -- being further along by the checkpoint will make the second week of this project substantially more pleasant.

- Have an **Item** class and a **Book** subclass, both of which have working **__init__** and **__str__** methods and take advantage of inheritance. Your **__str__** methods should lead to printing that resembles what you see in **output.txt**.
- Have an **Inventory** class with the following working methods:
    - An **__init__** method that successfully reads in the book.csv and initializes an **Inventory** object (which should store a list of objects, at this point all of which may be **Book** objects)
    - A **print_inventory** method that prints the entire inventory (eventually this method will be able to print pieces of the inventory, but for the checkpoint it's OK if it just prints all of the inventory)
    - A **compute_inventory** method that computes the value of the inventory (it should only compute the value of the items in it, which at this point might just be the book items)
- Have a **initial_test.py** file that follows the structure of the **test_inventory.py** file, robustly testing the above methods. It might, e.g.
    - Manually create and print an **Item** (e.g. **myItem=Item(...); print(myItem)**)
    - Manually create and print a **Book**

- Create an **Inventory()** object that reads in the **books.csv** file
- Uses **print_inventory** to print the entire inventory (of books)
- Uses **compute_inventory** to get the value of the entire inventory (of books)

If you're further along, great! We'll be running the **initial_test.py** file and want to see at least the above functionality. If your program reads in all objects, or has a more complicated **print_inventory** method, great! If not, as long as it does at least the above, great! Similarly, you don't need to worry about modular code quite yet. Eventually you'll probably want helper methods to read from all the csv files, e.g., but we won't be expecting that yet.

Finally, note that you should be testing each of the above steps as you go. You should not wait to create the **initial_test.py** file, e.g., to test that you can load in and print an Item!

Getting Started, Broadly

Think about how you want to approach building the **`Inventory`**, **`Item`**, and **`Item`** subclass structure. Try to break the task into small, easily testable pieces. You might, for example, start by creating the **`Item`** class and subclasses of **`Item`**, similar to what you did in lab, creating **`__init__`** and **`__str__`** methods. Test that they work: that you can create and print items of different types.

Then think about how to read in data from one of the spreadsheets. How can you read it in, parse it, and create the appropriate objects? Do you need method(s) that help you read the file? Once the **`__init__`** of the **`Inventory`** class is working on a single type of item, which methods can you write? Can you, e.g., compute or print the inventory? Do those methods work correctly on the single sheet you are processing? Then what's left to do?

Note that the Phase 1 Checkpoint, in some sense, helps you start getting the minimal pieces necessary for real functionality. It gives specific guidance for getting started, and you'll want to apply that thinking -- and constant testing!! -- as you go.

Finally, a few hints on testing:

- In repl.it, you can run specific files through the shell (like typing **python inventory.py** into the shell). This will run everything in that file, so you could have tests at the end. E.g. after defining your Item **__init__** and **__str__** methods, you could add statements like

  ```
  # Fill in ... based on defined Item arguments
  myItem=Item("Sams favorite item", ...)
  print(myItem)
  ```

  Then, each time you type **python inventory.py** into the shell, these lines will also run

(assuming your code doesn't crash beforehand or run into errors!), and you should see something print like the below, but with lots of ...'s filled in

```
----------
ID: ...
Name: ...
...
```

- Alternatively, you can test by adding

  ```
  from inventory import *
  ```

  to your **main.py** file. Then, if you click "Run" on repl.it, this will run everything in your **inventory.py** file. Tests (like the one in the previous bullet) will still run, and you can also work directly in the console. E.g. you could type the same two test lines, one at a time, into the console
- Finally, you can -- and eventually will -- build another file that exists purely to run tests. This is how **test_inventory.py** works, and you can run these tests (once you've finished the full stage) by typing **python test_inventory.py** into the Shell. For the first checkpoint, you'll make an **initial_test.py** file that works in this fashion -- but probably will want to start by iteratively testing-as-you-go using one of the above two methods.

FINAL PHASE 1 NOTES

- **18** of the points on this project (between the two phases) are for good Python style. Please read the course Style Guide handout for details about what we're looking for.

- **9** of the points (between the two phases) are for program quality and features that stand out. Think about what features you can add that will help your work stand out. You might think about adding additional features to your inventory, or later, the design and features of your web interface. To go from an A (a 92) to an A+, you'll want to think about adding useful features that make your work stand out. Be creative!

Don't forget to closely look at the Project Overview doc for further grading, collaboration, and submission instructions!

Finally, don't forget to click submit when your phase 1 submission is complete! The submission time is what we will use to calculate late days, so it's important to click submit when you're finished, and to not make further changes to your submission.