Program 4: Object-oriented design

Submit Assignment

Due Wednesday by 11:59pm **Points** 100 **Submitting** a file upload **File Types** zip

Collectibles store simulation

A local collectibles store wishes to automate their inventory tracking system. Three types of items are to be tracked:

- Coins
- Comic books
- Sports cards

Purchases and trades of used items by customers are also to be tracked. Five types of actions are desired in the system:

- Sell: removes an item from the inventory
- Buy: adds an item to the inventory
- Display: outputs the entire inventory of the store, including the number of each item in inventory
- Customer: outputs all of the transactions for a customer, including the item (in chronological order)
- History: outputs the history for every customer, with the customers in alphabetical order.

You will design and implement a program that will initialize the contents of the inventory from a file, the customer list from another file, and process an arbitrary sequence of commands from a third file.

Details

All coins have type, year, and grade (an integer). All comic books have a title, publisher, year, and grade (a string). All sports cards have a player, a manufacturer, a year, and a grade (a string). The items in the inventory are sorted as follows:

- Coins are sorted first by type, then by year, then by grade

- Comics are sorted first by publisher, then by title, then by year, then by grade
- Sports cards are sorted by player, then by year, then by manufacturer, then by grade

You can assume that each item is uniquely identified by the complete set of sorting criteria. To facilitate processing of the files, Coins are marked ‘M’, comics are marked ‘C’, and sports cards are marked ‘S’.

The data file for initialization of the inventory lists each item on a separate line with a character denoting the type of item, number in inventory, year, grade, type/title/player, publisher/manufacture (if necessary). Fields are separated by commas as follows:

```
M, 3, 2001, 65, Lincoln Cent
M, 10, 1913, 70, Liberty Nickel
C, 1, 1938, Mint, Superman, DC
C, 2, 2010, Excellent, X-Men, Marvel
Z, 4, 1986, Raging, Metallica, Master of Puppets
S, 9, 1989, Near Mint, Ken Griffey Jr., Upper Deck
S, 1, 1952, Very Good, Mickey Mantle, Topps
```

You may assume correct formatting, but the data could be invalid. In this data, the ‘Z’ code is an invalid entry.

Customer information is stored in a similar file. Customers have a 3-digit ID number that uniquely identifies them:

```
001, Serena Williams
456, Keyser Soze
999, Pele
```

You can assume that this data is formatted correctly, but not that there will be exactly two names. (Go ahead and sort by first name, if that is first in the file.)

Your code will be tested using a third file containing the commands (S for Sell, B for Buy, D for Display, C for Customer, and H for History). Except for the Display and History commands, the second field is customer ID. For buy and sell commands, the third field is the item type and the remaining fields are the item.

```
S, 001, S, 1989, Near Mint, Ken Griffey Jr., Upper Deck
B, 456, M, 1913, 70, Liberty Nickel
C, 999
D
X, 999, Z, 1986, Raging, Metallica, Master of Puppets
```

```
S, 000, Q, 2112, Gnarly, Windows, Microsoft
H
```

Once again, the data will be formatted correctly, but may include errors. You should check to make sure that the number of each item in the inventory doesn't go below zero. Do not print output for successful buy or sell commands, but do print error messages for incorrect data. Output for your Display, Customer, and History commands should be neatly formatted with *one line per item/transaction*. The inventory should output all coins, then all comics, then all sports cards.

Design

Your design should document the work that needs to be done to complete the assignment. It should be a complete and clear description of how the program is organized. The more time you spend on your design, the less you will spend coding, debugging, and modifying.

Your design should include (at least) the following components **in this order**:

- Overview: This is a short description of the design and how the pieces fit together (the interaction between the classes). Include a description of main. (List the objects that you have. Main should be short.)
- Class diagram: This is a UML diagram showing class relationships, including inheritance and composition. You can use Visio (or another tool) or **neatly** draw this by hand.
- Memory diagram: This is a diagram showing the use of memory (with respect to data structures) in the program. Show, for example, any linked lists, trees, graphs, and/or hash tables and any relationships between the memory used for them. (Do they share data? Is there a pointer from one to another?) An simple example would be diagram for the adjacency list in **Lecture notes: Graphs and search algorithms.**
- Class descriptions: For each class in the design, describe the data and methods as part of a documented C++ header file (with pre/post-conditions). The task that each function performs and the purpose of each data member should be clearly described. High-level pseudo-code should be included for the most important methods (for example, those that control the flow of the program). Not all parameters need to be included for methods. (For these, put the most important classes first, including manager classes and top-level base classes before derived classes.)

The design for this program is weighted as much as the implementation for other programs. You should put considerable effort into the design (comparable to a program implementation), or you should not expect to receive a good score. Note that you should not use

templates in the assignment to avoid inheritance. Any collection classes (for example, SearchTree) should store a pointer to a base class object. Pointers to descendant classes can also be stored in the collection.

A useful approach is to imagine that you are writing a design that someone else will need to implement and extend. Document your design as you would want someone else to document for you. Some questions you might want to ask yourself:

- Can your design be extended beyond the specifications given here?
- Could you easily add new types of collectibles or books?
- Could you easily add new types of items to the store (including very different items, such as clothing)?
- Could you handle age restrictions on purchasing certain items?
- Could you easily add other operations, such as returns or regrading an item?
- Could you easily add a way for customers/employees to review items? Items could have several reviews.
- Could you handle another branch of the store or a chain of stores?

Your design can go beyond the scope of these specifications (and you won't need to implement extensions). Thinking of possible extensions in advance often improves the design.

Hash tables

You are required to use at least one hash table in your program (we'll cover these soon). There are actually several places where they could be used. If you want to get started designing with them, the important aspect of hash tables is that they are used for fast lookup of items. For example, if each item can be mapped into a unique number, you can use an array to store the items according to their unique number and look them up in O(1) time. Hash tables usually waste some memory, since not all of the array will be filled. However, the waste is not too bad, if you store pointers to items, rather than the items themselves. It is usually better to use a hash table than a switch statement (or if-then-else-if) when there are several cases that must be tested to determine which function to call.

Design review

After the design due date, each student will be assigned two designs to review. The score for performing the review is part of the score for the design.

Relevant module goals

- Create object-oriented program designs
- Implement object-oriented design principles