Assignment #4 Project - Design specification

Design documentation is a written-down summary of the design work done. The program design is a complete and clear description of how the program is organized. It is language-independent and includes a description of the overall structure of your program. The more time spent designing, the less time spent changing, coding, debugging.

Class diagrams pictorially describe the classes from which a system is built with their static relationships. Descriptions of classes include details about their data and member functions. Often high-level pseudocode is included for the algorithms used by methods which show the flow of the program. All other functions clearly comment the task they perform (a function performs one task). Diagrams include descriptions of

-- Generalization/Inheritance (parent/child classes),
-- Aggregation/Composition (construction of complex objects from simpler ones)

Besides class diagrams and class descriptions (see below), include data structures, actual memory used. Give each piece context (how it fits) so that it makes sense. Additionally, describe interaction of objects via pseudocode for essential use cases. Describe instantiations, actual objects and the methods called to accomplish the task. Imagine that someone other than you have to implement, and then extend your program; or you have to implement someone else's design. This must be a good object-oriented design. This means everything except very basic data is an object. Do NOT store data as long concatenated strings, violating good OO design. Strings are not parameters in any methods. Use strings for only one item such as one name. Do NOT concatenate strings anywhere.

Design beyond the specifications (in what you turn in). Design so that the answer to all these questions is *yes* (maintaining good design principles). Could you easily add more hard copies to your design? Can you easily add other categories of books? Could you easily add other kinds of formats, for example, digital? Could you easily add new media to your design, for example, music? Could you expand to check out other kinds of items, for example, digital players? Could you easily add new operations to your design, for example, buy? Could you incorporate time, for example, a due date for checked out items? Could you easily add an additional library, or handle a library system?

You must make your design, **show in what you turn in**, more extensive than the assignment (the specs), as extensive as you would like. Include extensions, actual classes, in the UML of the design turned in. You do not have to implement all the extensions of your design, only assignment specifications. Incorporating as much as you can think of in the design phase helps you design a better, more extensible application.

UML symbols: open arrow for inheritance (is-a); line with diamond for composition (has-a or part-of); Composition (has-a or part-of) is just a special kind of association. Some things in the UML of the design are best shown as classes. For example, a List class may be implemented as an STL list or even as an array, but in the design, it is critical the reader knows it is a collection, so show a class for it. For typical attributes, say the amount of something, you would not show a class. Do not show in the UML something used for implementation, such as Node for BinTree. And while you might want to use the STL or templates often, do not. The focus is on inheritance. Run any template use by me.

Content of the Design Spec

Since you are totally designing this program, clear documentation is essential. Turn in (order they are viewed):

-- System design including (Use software for creating diagrams, e.g., Visio, draw.io.)

1. UML class diagram (only class names must be included in diagram, although more is fine)
2. Picture of data structures (visual of memory organization, the actual memory of your primary data structures – the boxes and arrows) (Note that in an actual design spec, this is not likely included.) (Hand-drawn is best.)
3. Show how classes interact via pseudocode (objects/methods invoked for the *Use Cases* of insert and check out.)
   E.g., Insert an item into your database (from book data file): `Y Seuss Dr., Yertle the Turtle, 1950`
   E.g., Check out a item (from commands data file): `C 1234 F H Walker Alice, The Color Purple,`

(On a high level, check out would entail: finding the patron; finding the book; associating patron with book so the book becomes a part of the patron's history; associating book with patron so it is known that the book is checked out by the patron and that the book has been checked out.) Give objects with methods called, i.e., high-level pseudocode, so the functionality can be easily implemented later. Note that conciseness is as important as clarity, i.e., do not burden the reader with excessive wordiness or low-level coding details.

-- Then main: What objects do you have? It is usually very short as all the actual work is done in class functions.

-- The .h files, with clear/complete descriptions of all base/derived classes. (Since this is a design, you might not include all, or any, parameters in methods. Or they might change. That's okay, but overall, the design is to be sound. You do not need to include .h files of classes you will not implement, of the extensions beyond the assignment specifications, but you must include a description of those classes if their name does not relay what it is.) Note – if you were turning in a hardcopy, you would order the files as a hierarchy, i.e., classes central to your design first; parent classes before children classes. This is the order a reader expects to find information.