Program 0

Worth: 50 points

Due: Monday, January 29 by 11:59 PM

Purpose: This assignment explores the use of composition (HAS-A) as a relationship between classes.

This assignment builds on the **LibraryBook** class of CIS 199 Program 4 using a modified, Console application version of your instructor's solution (attached - `Prog0-Start.zip`) as a starting point. In our new library application, books are checked out by library patrons and our solution needs to reflect this. I've created a new **public** class named **LibraryPatron** to represent users of the library. Each patron will be represented by their name (a **string**) and a patron ID (a **string**). A more complete representation would include contact information, such an address or phone number but we will not include these for the sake of simplicity. In addition to the data already being stored by the **LibraryBook** class, you will now create a HAS-A relationship between the two classes. Specifically, when a book gets checked out, the book will keep track of which patron checked it out. Class **LibraryBook** HAS-A **LibraryPatron** in this relationship.

Modify the public elements of the **LibraryPatron** class as follows:

- Modify the set accessor of property *PatronName*to validate the value. It must not be null or empty even after the value has been trimmed to remove leading and trailing whitespace. **String** methods *IsNullOrEmpty*, *IsNullOrWhiteSpace* and *Trim*will be helpful. If the value received is invalid, throw an **ArgumentOutOfRangeException** with appropriate error message as shown in the text.
- Modify the set accessor of property *PatronID* to validate the value. It must not be null or empty even after the value has been trimmed to remove leading and trailing whitespace. **String** methods *IsNullOrEmpty*, *IsNullOrWhiteSpace* and *Trim*will be helpful. If the value received is invalid, throw an **ArgumentOutOfRangeException** with appropriate error message as shown in the text.

Modify the public elements of the **LibraryBook** class as follows:

- Modify the set accessor of property*Title* to validate the value. It must not be null or empty even after the value has been trimmed to remove leading and trailing whitespace. **String** methods *IsNullOrEmpty*, *IsNullOrWhiteSpace* and *Trim*will be helpful. If the value received is invalid, throw an **ArgumentOutOfRangeException** with appropriate error message as shown in the text.
- Modify the set accessor of property *CallNumber*to validate the value. It must not be null or empty even after the value has been trimmed to remove leading and trailing whitespace. **String** methods *IsNullOrEmpty*, *IsNullOrWhiteSpace* and *Trim*will be helpful. If the value received is invalid, throw an **ArgumentOutOfRangeException** with appropriate error message as shown in the text.
- Modify the set accessor of property *CopyrightYear* to throw an **ArgumentOutOfRangeException** with appropriate error message when the value received is invalid (basically, replace the **else**).
- When a book is constructed, we know that is not checked out and so does not have a patron relationship yet. The special value **null** may be stored instead of an actual patron object reference to indicate this.
- Modify the method named *CheckOut* so that it accepts a single parameter, a reference to a **LibraryPatron** object. This method will change the book's checked out status to reflect that the book has been checked out by the specified patron.
- Modify the method named *ReturnToShelf* so that a patron is no longer associated with the book (in addition to changing the book's checked out status as before). The special value **null** may be stored instead of an actual patron object reference to indicate this.
- Add a read-only property named *Patron* that returns a **LibraryPatron** reference and accepts no parameters. This property's **public** get accessor has a significant precondition requirement, namely that the book is currently checked out. When satisfied, return the reference to the **LibraryPatron** that has the book checked out. Otherwise, a **null** value may be returned.
- Modify the method named *ToString* that returns a **String** and accepts no parameters. Remember, you must also use keyword **override** when defining a *ToString* method. Change the formatting of the book's checked out status to read "Checked Out By:" followed by the patron's information (name and ID number) or "Not Checked Out" (requiring a conditional). You may use string concatenation, composite formatting with the **String.Format** method, string interpolation or some combination to create the formatted text that the method will return.

In addition to the changes to the **LibraryBook** class, you will need to modify the simple console application to test your books. In your application source file (where the **Main** method is located), add at least 3 **LibraryPatron**objects (you may hard code your test data). When the existing book objects from the test program are checked out, pass one of the **LibraryPatron** objects to each book to represent the person checking out the book. Print out each book's information as before. In addition, replace the use of the array of **LibraryBook** objects with the use of a **List** of **LibraryBook** objects (**List<LibraryBook>**) using the **List** class as we learned about in CIS 199. Include other statements as necessary to test the changes you made to the classes.

Be sure to add appropriate comments in your code for each file, including your **Grading ID** (not name nor student ID), program number, due date, course section, and description of each file's class. Each variable used in your program needs a comment describing its purpose. These requirements are expected for every program and are listed in the syllabus. Preconditions and postconditions are now required, as well. So, for each constructor, method, get property, and set property a pair of comments describing the precondition and postcondition must be provided. Please review the PowerPoint presentation (under Course Documents) for further details about preconditions and postconditions.

As with our labs, I'm asking you to upload a compressed ZIP archive of the entire project. The steps for doing this will vary somewhat based on the ZIP utility being used. Before you upload this .ZIP file, it's a good idea to make sure that everything was properly zipped. Make sure your code is present and you can run your file.

Once you have verified everything, return to the *Assignments* area of Blackboard. Click on "Program 0" and the *Upload Assignment* page will appear. Add any comments you like in *Comments* field. Click *Browse* next to *File to Attach* to browse the system for your file. Browse to the location of your .ZIP file and select it. Note, multiple files may be attached using the *Add Another File*option. For this assignment, we just need the "Prog0.zip" file. Make sure everything is correct in the form and then click *Submit* to complete the assignment and upload your file to be graded.