Program 0

Worth: 50 points

Due: May 21 (by 11:59 PM)

Purpose: This assignment explores the use of composition (HAS-A) and inheritance as relationships between classes

In this assignment you will use what you have learned through Chapter 12 to design and implement a simple class hierarchy for package-delivery services, such as UPS or FedEx. The hierarchy will consist of several abstract and concrete classes. At the top of the hierarchy is the abstract base class **Parcel**. It will contain the To and From addresses for the parcel. This will be done via composition with the **Address** class. Each address consists of a name, two address lines, city, state, and zip code. Each parcel will also support a *CalcCost* method that will return the cost of shipping the item. Derived from **Parcel** are two classes, **Letter** and **Package**. Letters will be delivered for a fixed price. The **Package** class and others will be developed in the next assignment. Follow the naming, return types,and parameter lists specified below carefully. You may include **private/protected** helper methods/properties but no additional **public** methods or properties may be added without agreement from your instructor.

Address

- Constructor that specifies the name (a string), address line 1 (a string), address line 2 (a string), city (a string),state (a string), and zip code (a non-negative integer, <= 99999).
- Get and set property for each of the data fields (*Name*, *Address1*, *Address2*, *City*, *State*, and *Zip*).
- *ToString* method that will return all the data fields as a formatted **String** consisting of four lines. The name and two address lines are on separate lines with city, state and zip code on the final line. [Hint: How do you get zip codes to print leading zeros if the integer is less than 10000? Try D5 format.]

Parcel (abstract)

- Constructor that specifies the origin address and the destination address.
- Get and set property for each of the data fields (*OriginAddress* and *DestinationAddress*).
- Abstract *CalcCost* method (no parameters) that returns the given cost of the parcel as a **decimal**.
- *ToString* method that will return the all the values of the data fields as a formatted **String**. All costs and fees should be formatted to display using currency formatting.

Letter - derived from **Parcel**

- Constructor that specifies the origin address, the destination address and the fixed cost (non-negative decimal), in dollars. The cost may vary from store to store due to local competition, so it will be specified in the constructor.
- Override the *CalcCost* method to return the fixed cost specified in constructor.
- Override the *ToString* method as needed to return the all the values of the data fields as a formatted **String**. All costs and fees should be formatted to display using currency formatting.

In addition to the classes described above, you will need to create a simple console application to test your letters and addresses. In your application file (where the **Main** method is located), add at least 3 **Letter** objects (you may hard code your test data) using as least 4 **Address** objects. Add your **Letter** objects to a **List** of **Parcel** objects (**List<Parcel>**) using the **List** class introduced in Chapter 9. Print all the letters and associated cost to the console. Include other statements as necessary to test the properties included in the classes.

Be sure to add appropriate comments in your code for each file, including your name, 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.