Program 1A

Worth: 75 points

Due: Sunday, May 31 (by 11:59 PM)

Purpose: This program explores the creation of a simple class hierarchy including (limited) use of polymorphism.

In this assignment you will build on your instructor's solution to Program 0 (find under Course Documents). Adding to the **Parcel** hierarchy is first. Derived from **Parcel** are two classes, **Letter** and **Package**. The **Letter** class was already built in Program 0. The **Package** class will remain abstract but will track the dimensions (length, width, and height) in inches and weight in pounds of each package. Derived from **Package** are two classes, **GroundPackage** and **AirPackage**. Ground packages will calculate cost based on the package dimensions and weight and the distance between the origin and destination addresses (the Zone Distance), as described below. The **AirPackage** class will remain abstract but will add methods to determine if the package is large or heavy. Finally, derived from **AirPackage** are two classes, **NextDayAirPackage** and **TwoDayAirPackage**, both of which are concrete. Next-day air packages are charged an extra Express Fee. Two-day air packages may be delivered in the morning or the afternoon. If morning delivery is requested the package is an Early. If afternoon delivery is requested, the package is a *Saver* and a discount will apply. The detailed **public** specifications for the classes in this assignment appear below. 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.

The detailed **public** requirements for the classes in this assignment appear below.

Package (abstract) - derived from **Parcel**

- Constructor that specifies the origin address, destination address, length in inches (positive double), width in inches (positive double), height in inches (positive double), and the weight in pounds (positive double).
- Get and set property for each of the data fields (*Length*, *Width*, *Height*, and *Weight*).
- *ToString* method that will return all the data fields as a formatted **String**. All costs and fees should be formatted to display using currency formatting.

GroundPackage - derived from **Package**

- Constructor that specifies the origin address, destination address, length in inches (positive double), width in inches (positive double), height in inches (positive double), and the weight in pounds (positive double).
- int property *ZoneDistance* with a get (but no set). The zone distance is the positive difference between the first digit of the origin zip code and the first digit of the destination zip code. For example, the zone distance between zip codes 50000 and 10000 is 4 (5-1).
- Override method *CalcCost* to calculate the shipping cost as
  Cost (in dollars) = .20*(Length + Width + Height) + .05*(ZoneDist + 1)*(Weight)
- *ToString* method that will return the all the values of the data fields (and zone distance) as a formatted **String**. All costs and fees should be formatted to display using currency formatting.

AirPackage (abstract) - derived from **Package**

- Constructor that specifies the origin address, destination address, length in inches (positive double), width in inches (positive double), height in inches (positive double), and the weight in pounds (positive double).
- Method *IsHeavy* accepts no parameters and returns a boolean. An air package is considered heavy if it weighs 75 pounds or more.
- Method *IsLarge* accepts no parameters and returns a boolean. An air package is considered large if the total of its dimensions (Length + Width + Height) is 100 inches or more.
- *ToString* method as needed to return the all the values of the data fields (and heavy/large status) as a formatted **String**. All costs and fees should be formatted to display using currency formatting.

NextDayAirPackage - derived from **AirPackage**

- Constructor that specifies the origin address, destination address, length in inches (positive double), width in inches (positive double), height in inches (positive double), and the weight in pounds (positive double), and the express fee (non-negative decimal), in dollars. The express fee may vary from store to store due to local competition, so it will be specified in the constructor.
- Get property (no set) for *ExpressFee*.
- Override method *CalcCost* to calculate the shipping cost as follows.
  Base Cost (in dollars) = .40*(Length + Width + Height) + .30*(Weight) + ExpressFee
  If the package is heavy, add a weight charge of
  Weight Charge (in dollars) = .25*(Weight)
  If the package is large, add a size charge of
  Size Charge (in dollars) = .25*(Length + Width + Height)
- *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.

TwoDayAirPackage - derived from **AirPackage**

- Define *Delivery* enum with values *Early* and *Saver*.
- Constructor that specifies the origin address, destination address, length in inches (positive double), width in inches (positive double), height in inches (positive double), and the weight in pounds (positive double), and delivery type (a *Delivery* enum value, for *Early* or *Saver*).
- Get and set property for *DeliveryType* (using the *Delivery* enum).
- Override method CalcCost to calculate the shipping cost as follows.
- Base Cost (in dollars) = .25*(Length + Width + Height) + .25*(Weight)
  If the package is *Saver* delivery type, the final cost will be reduced by 10%.
- *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.

Please note the following restrictions to be enforced throughout your classes. Any attempt to set a property to or provide a constructor with an illegal value should throw an **ArgumentOutOfRangeException**.

This part of the assignment will only focus on the hierarchy. While you will need to write a test program of some sort to verify that your classes work, it will not be graded. Only the **Parcel** hierarchy classes will be evaluated.

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 1A" 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 fi

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 1A" 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 "Prog1A.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.