LAB ASSIGNMENT #3

Due Date: **Second Class, Week 6,** **Marks/Weightage: 30/10%**

Purpose: The purpose of this Lab Assignment is to:

- Practice the use of instance data members, constructors, methods in classes and objects

References: Read the course's text book **chapter 10 – Inheritance and Polymorphism** and the lecture notes/ppts. This material provides the necessary information that you need to complete the exercises.

Instructions: Be sure to read the following general instructions carefully:

This lab should be completed individually by all the students. You will have to demonstrate your solution in a scheduled lab session and submitting the project **through drop box link on e-Centennial**.

You must name your solution according to the following rule:
FirstName-LastName_SectionNumber_COMP123_Labnumber
For Example: **Joh-Smith_Sec002_COMP123_Lab01**

Each exercise should be placed in a separate namespace named firstname-last-name_*exercise1*, firstname-last-name_*exercise2* etc.

Submit your assignment in a **zip file** that is named according to the following rule:
FirstName-LastName_SectionNumber_COMP123_Labnumber.zip
Example: **Joh-Smith_Sec002_COMP123_Lab01.zip** *(if your section is 001..)*

Apply the naming conventions for variables, methods, classes, and packages:
- *variable names* start with a *lowercase* character for the first word and uppercase for every other word
- *classes* start with an *uppercase* character of every word
- namespaces use only *lowercase* characters
- *methods* start with a *uppercase* character for the first word and uppercase for every other word

Exercise #1: *[15 marks]*

Write a C# application that implements the following class(es) as per business requirements:
Let **Package** (Package.cs) be an abstract base class and **TwoDayPackage** (**TwoDayPackage**.cs) and **OverNightPackage** (**OverNightPackage**.cs) are its two derived classes.
The **Package** class defines following:
- Instance data members **package id**, **sender's name** and **sender's address**.
- Implement properties and toString() method.
- Instance members to represent the **weight of package** in grams and **rate per gram**. Add validations and ensure that these should never be negative
- Define an abstract method – double **CalculatePackageCost().**

Implement the **properties** and toString() method for the above instance variables of class **Package.**

The **TwoDayPackage (TwoDayPackage.cs)** should include the following:
i) Instance variable – **admin charges** which is fixed amount, added to the cost of mailing the package. It can not be negative.
ii) Define constructor which initializes all the instance variables by using the base class constructor.
iii) Define properties and override toString() method from the base class.
iv) Override/Implements method - **double CalculatePackageCost(). ()** which calculates total cost of mailing the package (use formula: weight * rate per gram + admin charges)

The **OverNightPackage(OverNightPackage.cs)** should include the following:
i) Instance variable – express charges which is a fixed amount, added to the cost of mailing the package. It cannot be negative.
ii) And rate per gram is 10 cents more than the standard rate.
iii) Define constructor which initializes all the instance variables by using the base class constructor.
iv) Define properties and override toString() method from the base class.
v) Override/Implements method - **double CalculatePackageCost(). ()** which calculates total cost of mailing the package, but here you also need to take into account extra charge which is 10 cents more than standard rate per gram.
vi) Define properties and toString() method

Create a test class – PackageTest (PackageTest.cs) which tests above class by at least creating two objects of the **TwoDayPackage** and **OverNightPackage** class. And you are required to process them in standard way as well as polymorphically i.e assigning their references to **Package** base class and then processing them.

Exercise #2: *[15 marks]*

Write a C# application that implements the following class(es) as per business requirements mentioned below:

Write a program named **SalespersonDemo** that instantiates objects using classes named `RealEstateSalesperson` and `GirlScout`. Demonstrate that each object can use a `SalesSpeech()` method appropriately. Also, use a `MakeSale()` method two or three times with each object and display the final contents of each object's data fields. First, create an abstract class named `Salesperson`. Fields include first and last names; the `Salesperson` constructor requires both these values. Include properties for the fields. Include a method that returns a string that holds the `Salesperson`'s full name—the first and last names separated by a space. Then perform the following tasks:

- Create two child classes of `Salesperson`: `RealEstateSalesperson` and `GirlScout`. The `RealEstateSalesperson` class contains fields for total value sold in dollars and total commission earned (both of which are initialized to 0), and a commission rate field required by the class constructor. The `GirlScout` class includes a field to hold the number of boxes of cookies sold, which is initialized to 0. Include properties for every field.
- Create an interface named `ISellable` that contains two methods: `SalesSpeech()` and `MakeSale()`. In each `RealEstateSalesperson` and `GirlScout` class, implement `SalesSpeech()` to display an appropriate one- or two-sentence sales speech that the objects of the class could use.

In the RealEstateSalesperson class, implement the MakeSale() method to accept an integer dollar value for a house, add the value to the RealEstateperson total value sold, and compute the total commission earned.

In the GirlScout class, implement the MakeSale() method to accept an integer representing the number of boxes of the cookies sold and add it to the total field.

Evaluation:

Functionality	
Correct implementation of classes (instance variable declarations, constructors, properties and methods etc.)	70%
Correct implementation of test classes (declaring and creating objects, calling their methods, interacting with user, displaying results)	20%
Comments, correct naming of variables, methods, classes, exception handling etc.	5%
Friendly input/output	5%
Total	100%