W. P. CAREY
SCHOOL *of* BUSINESS
ARIZONA STATE UNIVERSITY

CIS 345 – Business Information Systems Development II – Fall 2014

Assignment 5

Due on Blackboard on Monday, October 20, at 10 PM

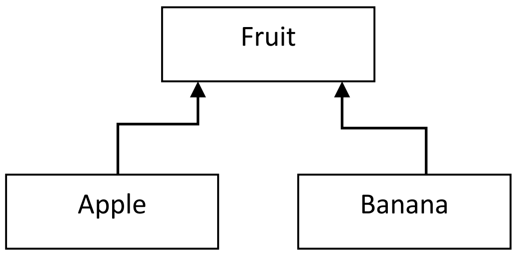
Create a class Fruit that abstracts the notion of a fruit. Create classes Apple and Banana which are types of fruits (use inheritance). Then create a FruitBasket class that has an apple and a banana in it. This entire application is then run inside a FruitGarden class, which will have two FruitBasket objects in it. *The Fruit Garden contains two FruitBaskets. Each basket contains an apple and a banana!*

The FruitBasket class will be where a lot of the work of this application happens. Its EatFruits() method will tell the user that there is an apple and banana available within the basket. It will ask the user how much of each should be eaten and tell the user how much of each fruit remains after it runs the Eat() method.

The various variables and properties you will implement in this project are all instance variables and properties. Thus, you are able to create multiple instances of each. You will maintain two fruit baskets and within each is one apple and one banana – a total of 4 fruits. You will see that you will be able to maintain how much of each fruit is left independently – eating one apple doesn't affect another apple in another basket (as it shouldn't!). Eating a banana doesn't affect the apple in the same basket. All of this will be run on the basis of only one property in the class Fruit (PercentLeft) but will work seamlessly across all objects because it is an instance property.

Consequently, your major task in this assignment is to understand this implementation of instance properties as well as being able to comprehend and work with objects inside objects (inside another object!)

Skills developed: Creating classes, writing accessors, implementing inheritance, creating object instances, calling instance methods and properties.

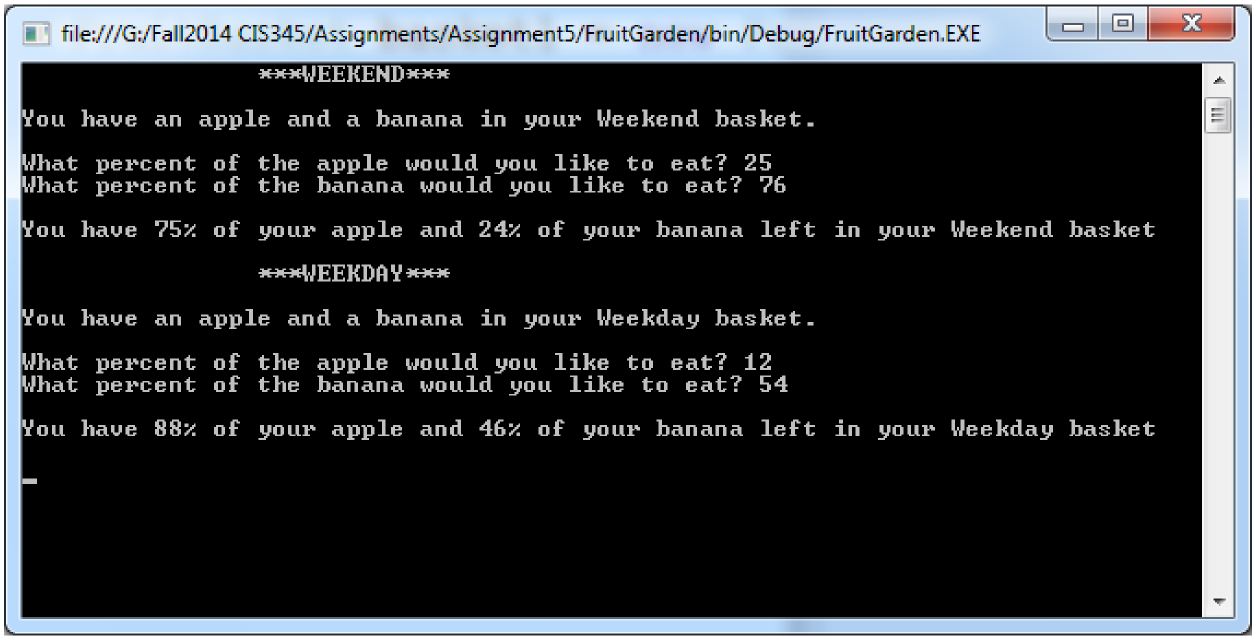


Project Details and Requirements

A sample executable file is uploaded and available on Blackboard for you to test program behavior.

Project Name: FruitGarden
Classes: FruitGarden, FruitBasket, Fruit, Apple, Banana. (Name all your classes and filenames appropriately).

Sample Output


file:///G:/Fall2014 CIS345/Assignments/Assignment5/FruitGarden/bin/Debug/FruitGarden.EXE

```
                ***WEEKEND***

You have an apple and a banana in your Weekend basket.

What percent of the apple would you like to eat? 25
What percent of the banana would you like to eat? 76

You have 75% of your apple and 24% of your banana left in your Weekend basket

                ***WEEKDAY***

You have an apple and a banana in your Weekday basket.

What percent of the apple would you like to eat? 12
What percent of the banana would you like to eat? 54

You have 88% of your apple and 46% of your banana left in your Weekday basket

_
```

Implement each of the following classes using the UML diagram for details on the class structure and the following instructions for details on the logic to be implemented.

Make sure you read the entire document before starting. Use the sequence provided here i.e. Make Fruit class first, Apple class next, and so on. The Main method should be last thing you implement.

<u>FRUIT CLASS</u>

Fruit
-fruitPercentLeft: double + <<property>> PercentLeft : double {read-only} + <<property>> PeelThickness : int
+ <<constructor>> Fruit() + Eat (eatenAmount : double): void

Purpose: The Fruit class abstracts the notion of a fruit. It tracks how much of a fruit is left and the peel thickness using instance variables and also has a method, Eat(), that allows users of the class to "eat" the food.

Properties: There are two properties. PercentLeft should provide read-only access to fruitPercentLeft, the private instance variable. PeelThickness can be an auto-implemented property.

Constructors: It should implement a parameterless constructor that accepts no arguments, Fruit(). Within this constructor, the private variable fruitPercentLeft should be initialized to 100.0, to indicate that when a fruit is created, 100% of it is left.

Eat() method: The Eat() method should have one parameter, *eatenAmount*, of type double. The parameter reflects how much of a fruit should be eaten. To implement this functionality, subtract the parameter variable *eatenAmount* from the instance variable that stores how much fruit is left.

APPLE CLASS

Apple
+ <<property>> Radius : double
+ <<constructor>> Apple (radius : double)

Purpose: An Apple class abstracts the notion of an apple. An apple is also a fruit. Therefore, the Apple class should inherit from the Fruit class. Use ":" notation in the class declaration

i.e. "class Apple : Fruit"

Properties: The Apple class should have a public property called radius (double, get/set).

Constructors: The Apple class should provide one overloaded constructor that accepts the radius as a parameter. The constructor should initialize the Radius property using the parameter variable.

BANANA CLASS

Banana
+ <<property>> Length : double
+ <<constructor>> Banana (length : double)

Purpose: A Banana class abstracts the notion of an apple. A banana is also a fruit. Therefore, the Banana class should inherit from the Fruit class. Use ":" notation in the class declaration

i.e. "class Banana : Fruit"

Properties: The Banana class should have a public property called Length (double, get/set).

Constructors: The Banana class should provide one overloaded constructor that accepts the length as a parameter. The constructor should initialize the Length property using the parameter variable.

FRUITBASKET CLASS

FruitBasket
- apple : Apple - banana : Banana + <<property>> BasketName : string
+ MakeFruits() : void + EatFruits() : void

Purpose: A fruit basket has fruits in it. Accordingly, the FruitBasket class will abstract the notion of a real fruit basket and will have instances of Apple and Banana. It will have only one of each.

Properties: The FruitBasket class should have a public property called BasketName (string, get/set). The name of the basket identifies which basket it is.

Instance Variables: It should have two instance variables - apple of type Apple and banana, of type Banana.

Methods:

MakeFruits() method:

Purpose: The purpose of the MakeFruits method is to create instances of apple and banana.

General Method Logic:

- Initialize the Apple and Banana objects by creating instances of those respective classes. The instance variables apple and banana should already be declared in the class (do not re-declare them here). Use the Apple and Banana constructors to supply the radius and length as parameters.
    - The apple should be of radius 1.5 inches
    - The banana should be of length 3.5 inches.
- Set the apple’s peel thickness to 1 and the banana’s peel thickness to 4.

EatFruits() method:

Purpose: The EatFruits() method informs the users that there is an apple and banana available. It will ask the users how much of each they want eaten. It will read input and call the Eat() method in reference to the apple and banana objects. Finally, it will tell the users how much of each fruit is left.

General Method Logic:

- EatFruits should declare a local variable of type double called amountToEat.
- Print the name of the Basket at the top. Use the ToUpper() method (access it by typing your string variable name dot ToUpper) to convert the name to all upper case.
- Tell the user that she has an apple and a banana in her fruit basket. Include the name of the basket within the text.
- Ask user how many percent of the apple she would like to eat.
- Read the percent and store it in amountToEat as a double.
- Call the Eat() method for the apple and pass it the appropriate argument.
- Ask the user how much of the banana she would like to eat
    - Process this in the same manner as the apple object i.e. prompting user, reading user input, calling Eat() method, etc.
    - Reuse the existing amountToEat variable to store user input.
- Tell the user how much of the apple and the banana she has left. Include the name of the basket within the text.

FruitGarden
- MakeFruitBaskets() : void + Main (args: string[]) : void

Purpose: FruitGarden is the class that will act as your main program, where the program will start. Accordingly, it will have the Main() method.

Methods:

MakeFruitBaskets() method:

- Declare two local variables, both of type FruitBasket. Call them basket1 and basket2. Instantiate them by calling the default constructor.
- Using the property available within the objects, set the name of the baskets to "Weekend" and "Weekday"
- For both the FruitBasket objects, call the MakeFruits() and EatFruits() methods.

Main() method:

Purpose: Main starts the program by declaring a local variable of type FruitGarden. It will instantiate FruitGarden and call its MakeFruitBaskets() method.

General Method Logic:

- Declare and instantiate an instance of the FruitGarden class. Call it myFruitGarden.
- Call the MakeFruitBaskets method in reference to myFruitGarden.
- Insert a ReadLine() here.

Optional Learning Challenge

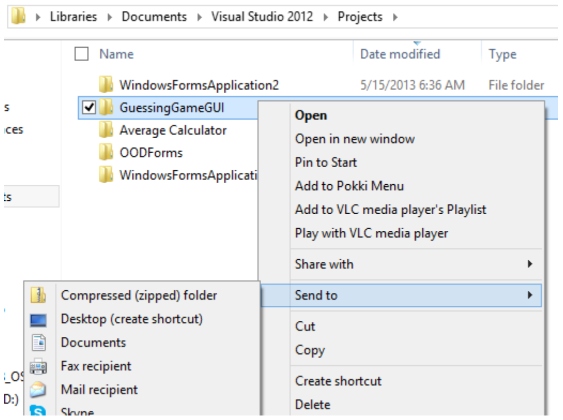
Set up your FruitBasket class to have an array of apples and bananas! Create, perhaps, 5 apples and 3 banana objects inside the respective arrays. Now, access the fruit basket, then the array inside it, then the fruit objects inside that, and then the properties inside those. Manage the level eaten of all of these apples and bananas from the OUTERMOST class – FruitGarden!

Experiment with changing the access modifiers for the fruits inside FruitBasket and of the properties inside Fruit/Apple/Banana.

<u>Submission Instructions</u>

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to ***zip the entire project folder*** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \ProjectName\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
  - e.g. " // Assignment 5, Jane C. Smith, CIS 345, Tuesday 9:00 AM"
- Class files are called Apple, Banana, Fruit, FruitBasket, and FruitGarden
- Your Visual Studio project is called FruitGarden.
- Zip filename is: FirstNameLastName_Assignment5.zip
- Your code is commented and you are using all prescribed programming conventions.
- Your code utilizes PascalCase and camelCase as appropriate.

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present. Test your files in Visual Studio after uncompressing.
- Make sure you have <u>submitted</u> your file and not just saved a draft on Blackboard. A blue clock indicates a submission in progress, i.e. a draft, not a submission.

This takes an extra couple of minutes. Please do it if your grade is important to you. If you do this, you will not end up submitting a bad file. If you submit an empty file, or one containing only a shortcut, or a bad zip file, or a bad project file, you will receive a score of zero and your only recourse will be to do the makeup assignment at the end of the semester.