

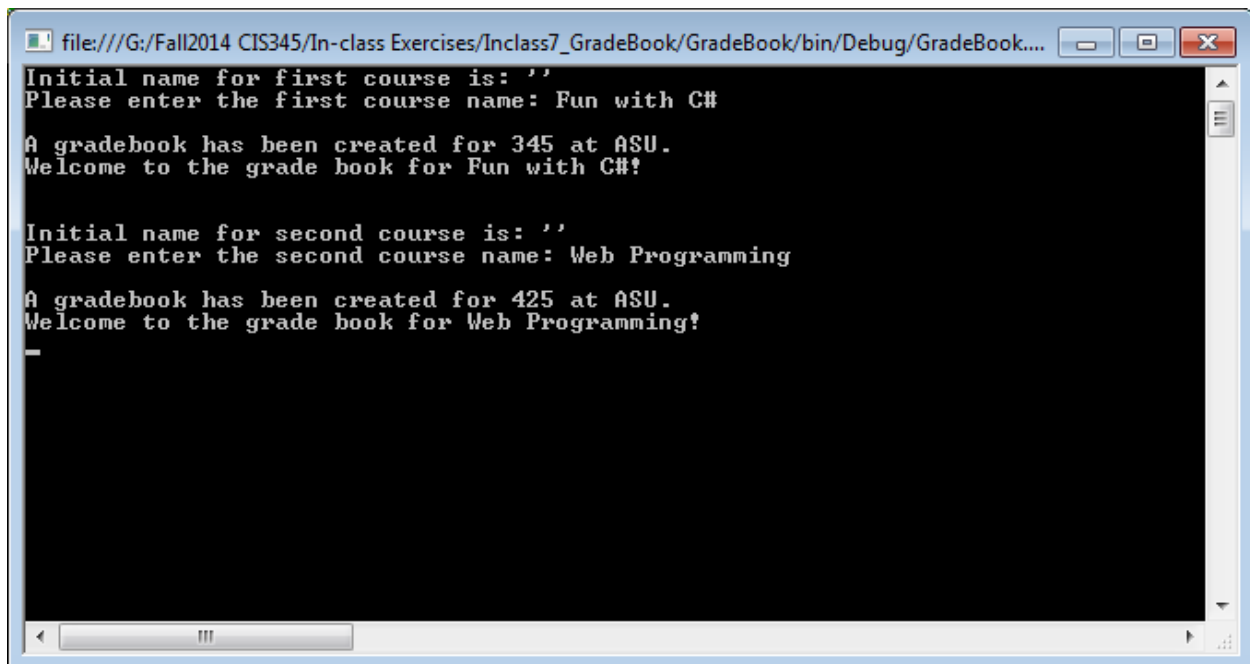
CIS 345 – Business Information Systems Development II – Fall 2014

In-class Exercise 6: Implementing Objects and Properties

Due on Blackboard: Today, Monday, October 6 by 10:00 PM

In this exercise, you will create objects, experiment with different ways of implementing properties in Visual Studio 2012/13 as well as test behavior when properties are static vs. instance.

Create the GradeBook and GradeBookTest classes from Fig 4.7 and 4.8 and add new properties to the existing structure of the GradeBook class. Utilize *accessors* to *get* and *set* the properties of GradeBook from within the GradeBookTest class.

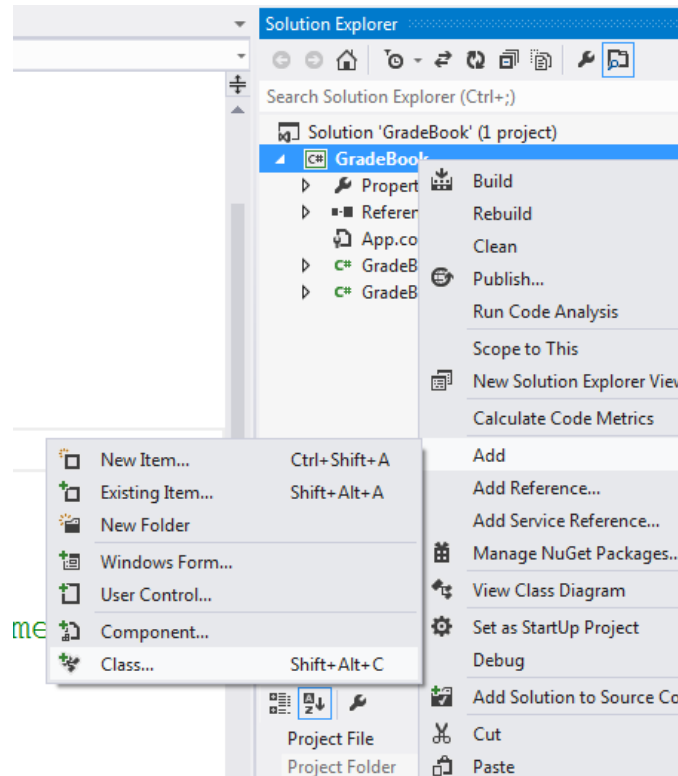


```
file:///G:/Fall2014 CIS345/In-class Exercises/Inclass7_GradeBook/GradeBook/bin/Debug/GradeBook....
Initial name for first course is: ''
Please enter the first course name: Fun with C#
A gradebook has been created for 345 at ASU.
Welcome to the grade book for Fun with C#!

Initial name for second course is: ''
Please enter the second course name: Web Programming
A gradebook has been created for 425 at ASU.
Welcome to the grade book for Web Programming?
-
```

Unless otherwise stated, **do not use *static*** either for methods or for class variables (fields).

1. Create a new Project called GradeBook
2. Rename your Program.cs file to GradeBookTest.
3. Add another class to the Project. Call it GradeBook.cs.



In the GradeBook.cs class file (Work on GradeBook before you work on GradeBookTest)

4. Work on the GradeBook class:
 - Create the properties for the GradeBook as seen in the book. The property is implemented as a combination of the instance variable declaration as well as the get and set accessors. *Note that these are instance properties.*
 - Create the instance method DisplayMessage() as well.

After creating the properties in the book example, do the following in addition:

5. Create a public property, CourseNumber of type integer:
 - Declare a *private instance* variable of type integer called courseNumber. courseNumber will be non-static.
 - Create a get accessor
 - The get accessor returns courseNumber
 - Create a set accessor
 - The set accessor assigns “value” to the private variable (courseNumber). Make sure you do not use quotes.

6. Create a public property, CollegeName of type string:
 - Declare a *private instance* variable of type string called collegeName (non-static).
 - Create the *get* accessor for it.
The get accessor returns collegeName
 - Create the *set* accessor for it.
The set accessor assigns “value” to the private variable (collegeName). Make sure you do not use quotes.

7. Create a property ClassAverage of type double, with only a get accessor. Having no set accessor will make this property read-only. It will be public.
 - Declare a *private instance* variable of type double called classAverage (non-static). Initialize it with the value 0.0.
 - Create the *get* accessor for it. The get accessor returns classAverage.

In the GradeBookTest class, do the following. The GradeBookTest class in the book provides you with the syntax to declare and instantiate objects. It uses the object reference myGradeBook. You should model your syntax after it.

8. Within the Main method,
 - Declare two objects references, gradeBook345 and gradeBook425 of type GradeBook. These should be done in a manner similar to myGradeBook from the book example.

Do the following separately for ***each*** of your two GradeBook objects.

- Instantiate a new GradeBook object by calling the default constructor. Store the resulting reference in your GradeBook object reference.
- Set the CourseNumber using the *set accessor*.
- Set the CollegeName using the *set accessor*.
- Similarly, try setting the ClassAverage to 85. Does it let you? Why / Why not?
- Use the *get accessor* to read the properties. Print the CourseNumber and the CollegeName to the Console in the format:

“A new grade book was created for 425 at ASU”

The way to use the *get accessor* is shown below. Type in the name of the object reference and access the property after placing a dot, just like how you call methods except that there will be no parentheses.

ObjectReference.Property
e.g. WriteLine(myCar.Color);

- Call the DisplayMessage() method for the objects.
9. Investigate:
 - What happens if you set one of the variables like courseNumber or courseName to static?
 - What happens if you set one of the properties to private?

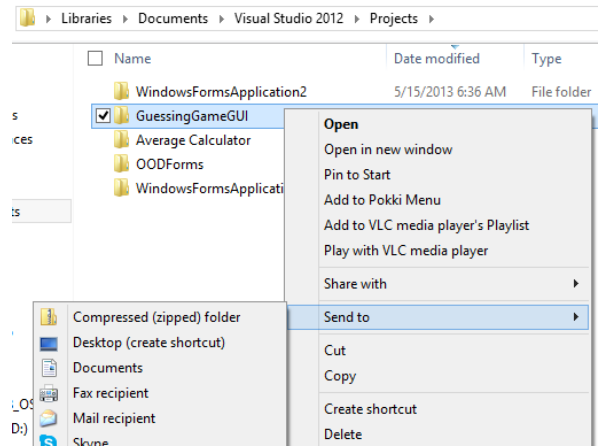
Go back and enhance your GradeBook class:

10. Since all gradebooks are located within the same college, CollegeName does not need to be an instance variable. Modify it so that it is static.
 - In GradeBook.cs, modify the instance variable collegeName so that it is static.
 - In GradeBookTest.cs, comment out the line that sets the CollegeName for the 425 object. Does your program still continue to work when you run it? What value does the 425 object have for CollegeName (bearing in mind that nothing is assigned to it)?
 - Experiment with removing the static keyword from collegeName. Instead, prefix it to the Property CollegeName. Does your program continue to work?
 - In order to use static properties, you will need to call them in reference to the class, rather than object name. i.e. ClassName.PropertyName
11. Enhance the CourseNumber property's *set* accessor. This GradeBook is meant for undergraduate courses. Therefore, in the *set* accessor, assign "value" to the private variable **ONLY** if value is greater than 100 and less than 500. *You can see this being done in figure 4.15 on page 129 with the Balance property.*
12. Utilize an alternate structure for implementing properties. Use what are known as auto-implemented properties. Auto-implemented properties do not require that a private instance variable be declared separately and automatically implements the accessors. *See section 4.8 on page 123.*
 - Create ClassSize as an auto-implemented property of GradeBook.
 - Utilize the ClassSize property within the Main method – use its get/set accessors to read and write values. Note that it is storing the class size but there is no private variable that you know of where the value is being stored. *It's implemented automatically!*

Submission Instructions

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. “ // Inclass 6, Jane C. Smith, CIS 345, Tuesday 9:00 AM”
- Class file is called GradeBook.cs and GradeBookTest.cs (rename from Program.cs).
- Your Visual Studio project is called GradeBook.
- Zip filename is: FirstNameLastName_Inclass6.zip

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.
- Make sure you have submitted 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.