CSCI 1301 – Lab 05

1 Variable Types: From String to Integer

1. Create a new project.

2. Write two statements, one that declares a variable of type int named intVar and one that declares a variable of type string named stringVar.

3. Assign the value 3 to intVar and "4" to stringVar.

4. Display the values of intVar and stringVar.

5. Write a statement that assigns the value of stringVar to intVar. Why is the compiler complaining? Comment out the statement you just added (that is, add // in front of it, so that the compiler won’t try to execute it).

6. Copy the following statement to “convert” the string value of stringVar into an integer value and assign it to intVar:

   ```
   intVar = int.Parse(stringVar);
   ```


8. Change the value of stringVar to be "Train" and assign it to intVar using int.Parse as previously shown. What happened?

2 Reading Other Numeric Datatypes

1. Create a new project.

2. Declare three variables: one of type int, one of type float, and one of type double.

3. We can use the following to read an int from the user (assuming your int variable is named intVar):

   ```
   intVar = int.Parse(Console.ReadLine());
   ```

   Similarly, we can read float using float.Parse(Console.ReadLine()) and double using double.Parse(Console.ReadLine()).

4. Write statements that ask the user to enter an int, a float, and a double, store those values in the appropriate variables, and then display them on the screen.

3 Using a Pre-Defined Class

This lab will guide you in your first manipulation of a programmer-defined class. We will use the example shown in class. The last part is challenging; therefore, we will give a possible solution to it in class, but make sure you try to solve it by yourselves beforehand.
3.1 Manipulating Two .cs Files at a Time

1. Download the Rectangle1 project, extract it, and open it with VS. Note that in the “Solution Explorer”, there are two cs files listed: Program.cs and Rectangle.cs.
2. In the Solution Explorer, double-click on Rectangle.cs and note how close it is from what was presented during the lecture.
3. In the Solution Explorer, double-click on Program.cs and observe it.
4. Compile and execute the code.
5. Now, do the following:
   - Introduce a syntactical error in Program.cs (e.g., remove a ;), and try to build the solution: what do you observe? Restore the program to its previous state, using CTRL + z to “undo” your operation.
   - Introduce a syntactical error in Rectangle.cs (e.g., remove a ;), and try to build the solution: what do you observe? Undo the modification using CTRL + z.
   - Add length = 12; in the main method of Program.cs and try to build the solution: what do you observe? Undo the modification using CTRL + z.

3.2 Enriching Program.cs

Edit the Main method of Program.cs by adding at its end statements that perform the following:

1. Create a second Rectangle object and set its length and width to 3.
2. Create a third Rectangle object and ask the user to specify its length and width. Display the area of this rectangle on the screen.
3. Create a fourth Rectangle object, do not specify its length or width, and display them on the screen. What do you observe?

In the last part, you may notice that the length and the width of the newly created object were assigned default values. To know more about this, refer to https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/default-values-table.

3.3 Editing Rectangle.cs

Edit Rectangle.cs:

1. Rename every instance of lengthParameter to lengthP in the SetLength method (that is, replace the two occurrences). You can use the symbol refactoring2 of C# to do so. Compile and run your program. What do you observe?
2. Some people use the convention of prefixing instance variables with _ (the underscore character), m (for “member”), or even _m_. You will always find someone furiously advocating for one particular convention, the truth is that if you’re not forced to use one, you should pick whichever suits you best. Still, just to use it at least once, rename every instance of width into m_width and see how it feels. Compile and run your program. What do you observe? Either undo this modification or rename length into m_length (you have to be consistent!).
3. Change the name of one of the accessor methods in Rectangle.cs without changing it in Program.cs. Compile and run your program. What do you observe? Undo your modification.

---

1 Rectangle.zip
3.4 Enriching Rectangle.cs

By taking inspiration from the ComputeArea() method, write three new methods:

1. A method that returns the perimeter of the calling object.
2. A method that doubles the length and the width of the calling object.
3. A method that swaps the length and the width of the calling object.

For each method: pick a (valid) name, think about the return type and the parameters, and write the body of the method carefully. After compilation succeeds, call that method in Program.cs and see if it has the expected behavior.

You can compare your answer with this proposition: Rectangle_Sol.zip.