LAB DEMO Student class

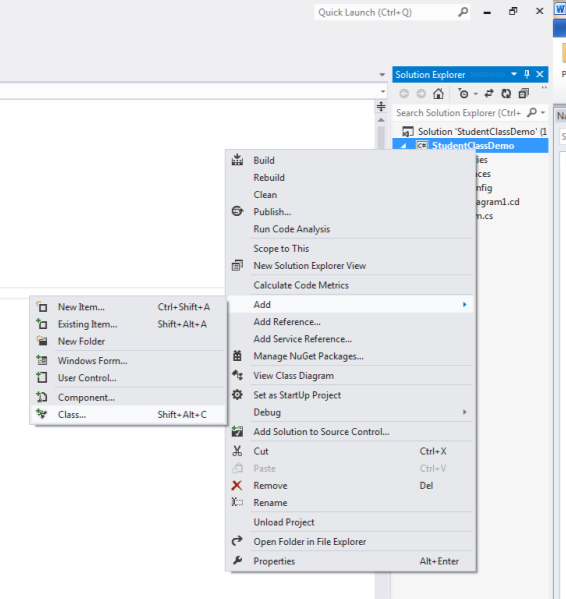
Defining Classes

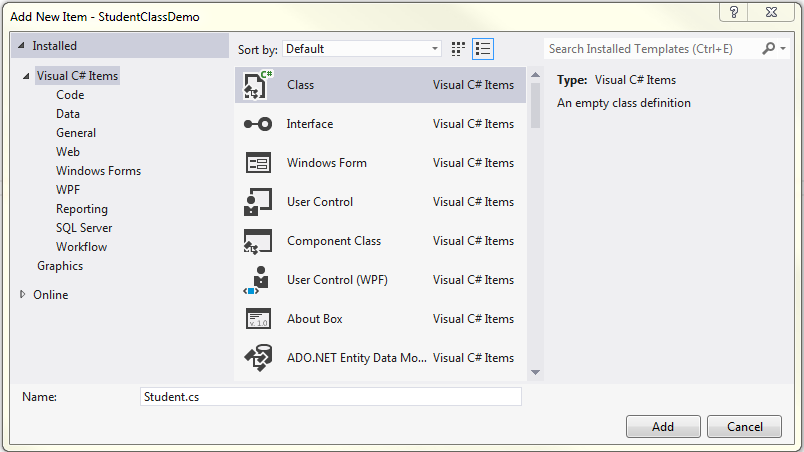
# Learning Objectives

* 1. Explain and demonstrate how to define data members in classes.
  2. Explain and demonstrate how to write class methods.
  3. Explain and demonstrate how to use a calculate method

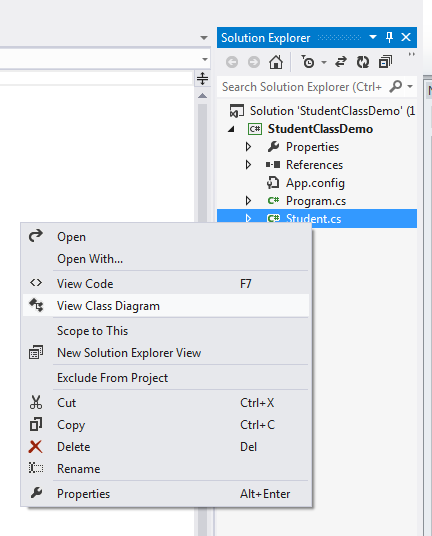
# Create the project

* 1. Create a new project. Call it “StudentClassDemo”.
  2. Right click your project and add a class call it “Student”



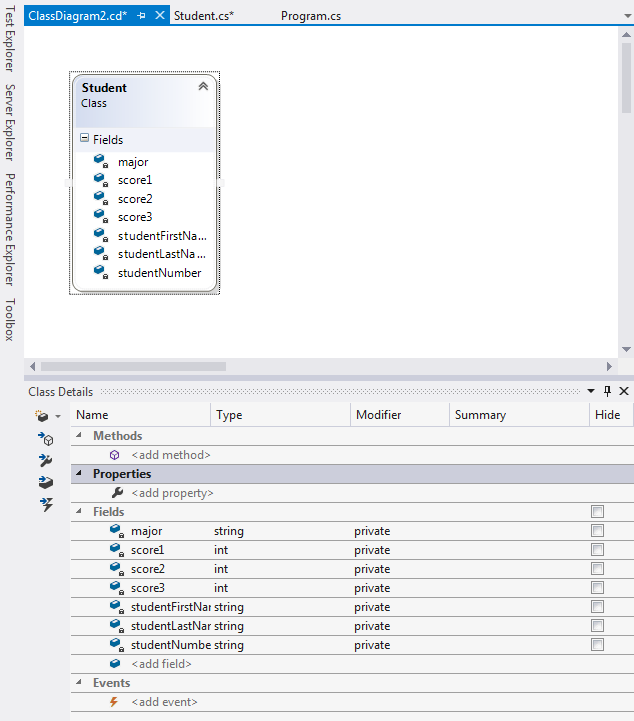


* 1. Right click the Student class and “View Class Diagram”

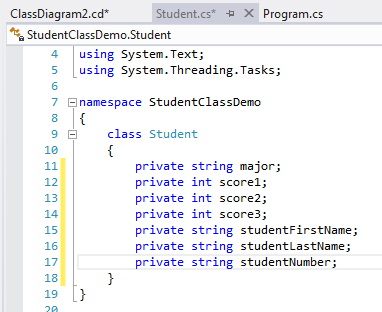


# Add Data Members

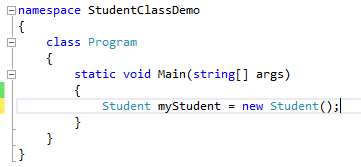
* 1. In the Class Details section of the diagram enter fields for your student class:



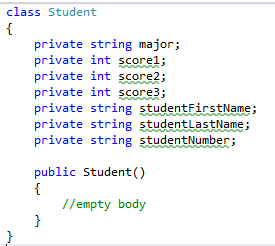
* 1. View the student class. Notice that visual studio has written some code for you:



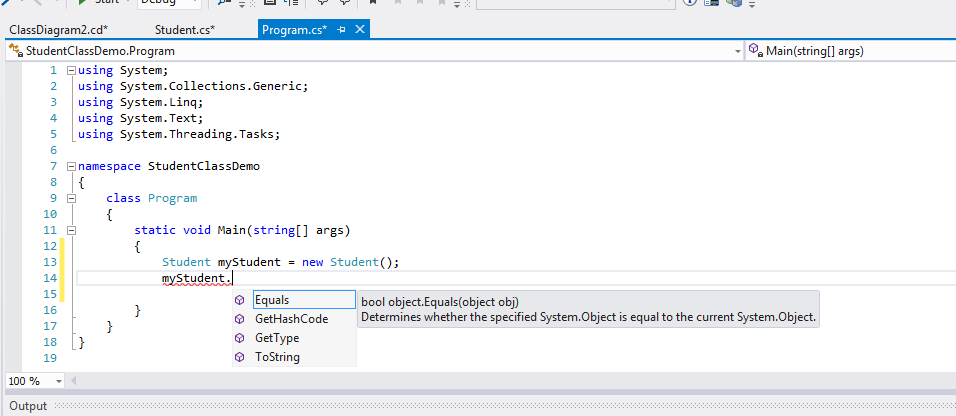
* 1. Note that you don’t have to use the class diagram. You can write the code directly into the class. It’s simply a convenience provided by visual studio.
  2. In Program.cs, declare a new variable using your newly defined class.



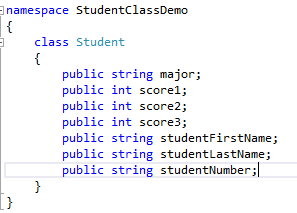
* 1. What does this syntax mean?
  2. “Student myStudent” is identical in syntax to a variable declaration like “string myName”.
  3. Instead of string, however, we are declaring a variable who’s type is the class we just wrote. We can now essentially write our own types! We call the classes however.
  4. The “new” keyword allows us to “instantiate” a new instance of that type. When we use the new keyword the computer will allocate memory and associate that memory space with the type we select.
  5. In addition we have to follow the “new” keyword with a special method called a “constructor”. In this case “Student()” which consists of the class name and “()” just like any other method. In this case since we used () we are specifying the “default” constructor which C# creates automatically for us (which is why it isn’t in our code).
  6. Let’s add a default constructor we actually define:



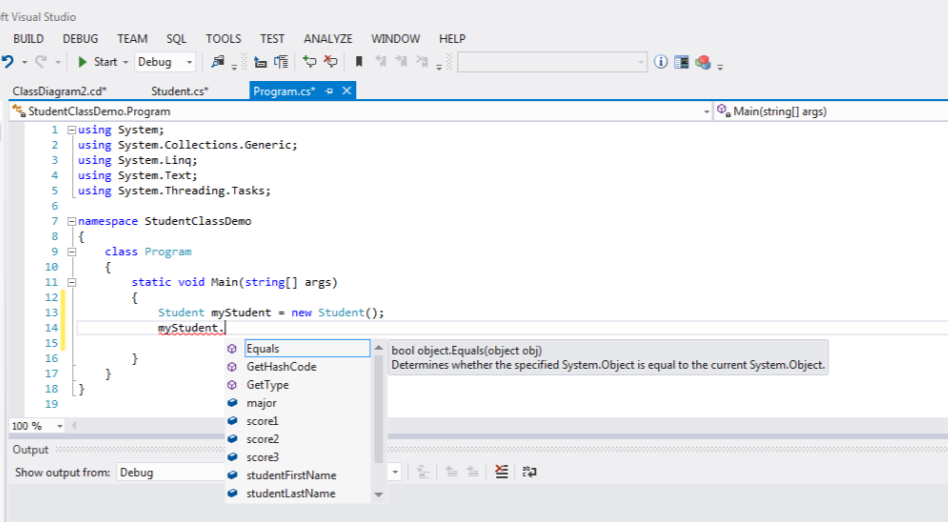
* 1. This constructor is identical to the default one C# implicitly used for us up till now. It doesn’t do anything other than create a new instance of our class with empty fields.
  2. Let’s look at myStudent, the instantiated object that was created with new Student().
  3. Type the name of your new variable, type a “.” and look at the intellisense.



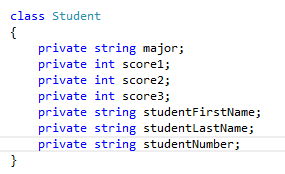
* 1. Notice that none of the field names appear.
  2. Go back to your class and change the fields from private to public.



* 1. Now go back to main and look at Intellisence again (you may need to retype the “.”):



* 1. Notice that your fields now appear because they were made public!
  2. Note that we rarely make fields public. It allows too much access outside the class and we rather use another approach that allows us to control how users of the class can access our data. So make your fields private again.



# class methods

* 1. Let’s add a method called an Accessor:

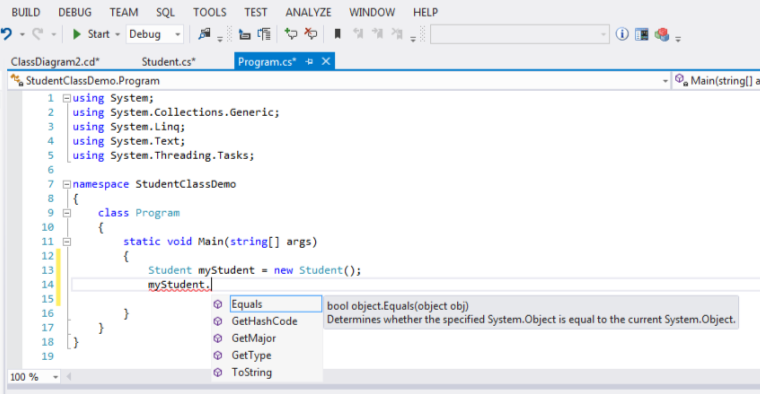
public string GetMajor()

{

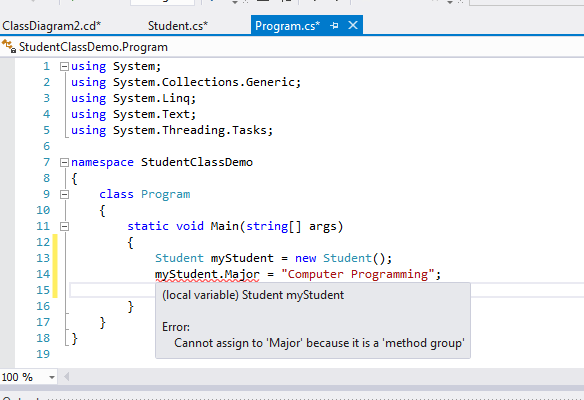
return major;

}

* 1. Note that by convention the accessor is made public has a return type (string in this case) and is always capitalized while the field it provides access to starts with a lower case letter.
  2. Go back to main and look at intellisense:



* 1. Notice that our accessor is now visible!
  2. Let’s do something with it. Let’s try to set a value into myStudent.GetMajor:



* 1. We get an error! GetMajor only allows us to get a value. It doesn’t allow us to set one.
  2. Let’s write a Mutator function:

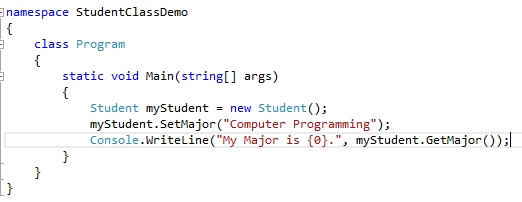
public void SetMajor(string newMajor)

{

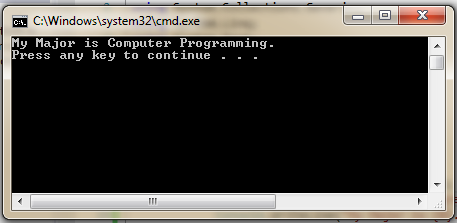
major = newMajor;

}

* 1. Note that by convention mutator functions are public, start with a capital, always returns nothing (void) and has a parameter to set into the field.
  2. Lets go back to main and this time lets use the mutator function to set the value of the field and the accessor to get access to the field and output it to the console:



* 1. Run the program:



* 1. Congratulations you have written a C# class with fields, an accessor method and a mutator method.
  2. Write Accessors and Mutators for all your fields:

class Student

{

private string major;

public string GetMajor()

{

return major;

}

public void SetMajor(string major)

{

this.major = major;

}

private int score1;

public int GetScore1()

{

return score1;

}

public void SetScore1(int score1)

{

this.score1 = score1;

Calc();

}

private int score2;

public int GetScore2()

{

return score2;

}

public void SetScore2(int score2)

{

this.score2 = score2;

Calc();

}

private int score3;

public int GetScore3()

{

return score3;

}

public void SetScore3(int score3)

{

this.score3 = score3;

Calc();

}

private string firstName;

public string GetFirstName()

{

return firstName;

}

public void SetFirstName(string firstName)

{

this.firstName = firstName;

}

private string lastName;

public string GetLastName()

{

return lastName;

}

public void SetLastName(string lastName)

{

this.lastName = lastName;

}

private string studentNumber;

public string GetStudentNum()

{

return studentNumber;

}

public void SetStudentNumber(string studentNumber)

{

this.studentNumber = studentNumber;

}

//Read only attributes

private float average;

public float GetAverage()

{

return average;

}

public string GetSummary()

{

return firstName + " " + lastName + " " + studentNumber

+" Average: "+average;

}

//Private methods

private void Calc()

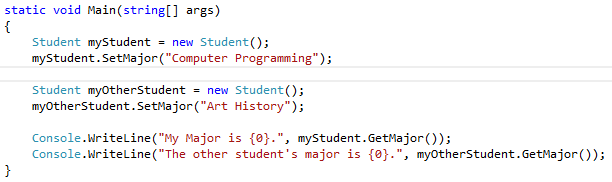
{

average = (score1 + score2 + score3)/3;

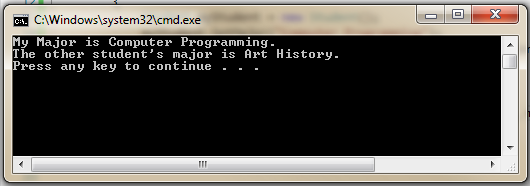
}

}

* 1. Let’s see if we can instantiate more than one student:



* 1. Run the updated program:



* 1. Notice that we did not have to write another class. Instead we say that we “instantiated” a new instance of the class, in this case called myOtherStudent. The new object “myOtherStudent” has available the same fields and methods as “myStudent”. The fields, however are in a separate part of memory and hence can carry different values.

# Adding a Do Another Loop:

* 1. Change the code in program.cs to:

static void Main(string[] args)

{

string doAnother;

do

{

Student st = new Student();

string firstName = GetInput("First Name");

st.SetFirstName(firstName);

string lastName = GetInput("Last Name");

st.SetLastName(lastName);

string major = GetInput("Major");

st.SetMajor(major);

string studentNum = GetInput("Student Number");

st.SetStudentNumber(studentNum);

int score1 = int.Parse(GetInput("Score 1"));

st.SetScore1(score1);

int score2 = int.Parse(GetInput("Score 2"));

st.SetScore1(score1);

int score3 = int.Parse(GetInput("Score 3"));

st.SetScore1(score1);

Console.WriteLine(st.GetSummary());

doAnother = GetInput("\nDo another (y/n)");

} while (doAnother == "y");

}

* 1. Also add the following helper function after the main function inside the program class:

private static string GetInput(string prompt)

{

    Console.Write(prompt);

    return Console.ReadLine();

}

* 1. Try it in Debug mode. Notice how execution flow goes to the class to create each student object.