**Introduction**

C# is fast, capable, and productive. I shall be using Visual Studio 2017 Community edition for this project. The .NET framework is a software framework that can provide a standard way of building applications and deploying applications. This also includes arrays of libraries and tools useful for software developers.

**Building strings with StringBuilder**

StringBuilder uses idea is that I instantiate a string with some seed text and then build that string up by writing many lines of code after that. C# sharp is a very rich string formatter. In C#, I can use *string.Format* , this method allows me to perform different formatting operations. It is a static method which means that I can call the method without creating an instance of that class. In C#, we have a method *TryParse*. The *TryParse* has two parts to its return, and it works in a unique way.

**Namespaces in C#**

C# uses the idea of namespaces. The namespaces (in C#) and packages (in Java) helps us to organise our code and avoid namespace collision. Auto-properties are useful when we want to add encapsulation to our properties by adding curly brackets and word get/set along with a semicolon in the end.

Now I create a property that requires logic. That property will be a twitter address with a symbol ‘@’ in it. An access modifier is useful to limit the visibility of properties, methods, fields in my class in such a way that nobody can accidentally/unintentionally change something in the class.

Constructor is a special function, in OOP languages constructor runs immediately whenever an object is instantiated. Inside C# classes, properties and fields are used to store the class information. On the other hand, methods also known as functions are used to manipulate the class data. Static methods are defined as methods which I can call without instantiating the class.

Another useful method to do in my class is to override the *ToString* method. In my school class, the *ToString* method will output the name of the school, the address, the phone number, and the Twitter account. Once I built up the class with all its components. I present the class to be consumed by another application (Windows Classic Desktop -> Windows Forms App). Then I shall find the project that I am working on that depends on the School library and add it as a reference for the Windows Forms App. It is important to make school forms app as a start up project. Next, I shall create a small user interface that allows me to manipulate the class that I created previously.

**Creating a user interface**

I shall be creating a user interface by using the Toolbox inside my Windows Forms app. This interface has labels, text boxes and a button according to the class fields. I can add some functionality to the button, I double click the button to get to the button handler (code that runs whenever I click this button). This is to test that all my getters and setters are working properly. Instead of using hard coded values in the user interface, inside the button handler for my button, I write code to set the values equal to whatever’s inside the form. It is a good idea to use try-catch statement around twitter address. I shall be showing a simple message box, here I do *ex.Message*, this allows my class to define the error message.

Before I finish my application, I want to use my *ToString* override. I shall call *ToString* to get a nicely formatted string version regarding the details of school. I can use the OOP concepts such as extending a class to create a new class (inheritance). I shall add a base class called Person. I am going to add a special keyword abstract in front of Person class. This will prevent developers from directly instantiating *person*. At any point of time, I can go back to my form app then look on ways to consume my subclasses.

Once we know about how the abstract class works. I shall look at abstract methods. I go back to my abstract class *person* and add an abstract method *ComputeGradeAverage* to it. As this method is in base class, meaning it must be implemented in all the subclasses. Now, let us look at my test program, by switching to my form and adding two new buttons (Test Teacher and Test Student). All right, now to open the code for these, I double click on these buttons and add code in the event handlers. Later, I click Start and see if all goes well once my app launches.

After studying about abstract method/class. In C# programming language, there are going to be times where I just need to write something in the base class, but I have the option of overriding it later. These are known as virtual methods. As I mentioned earlier, virtual method is optional. In the case of teachers, I send the message as it is to the teachers. But in case of students, I shall add a disclaimer at the bottom of all messages.

The concept of interfaces becomes relevant when I am dealing with two things that apparently seem to go together, but they don’t have much in common in terms of properties. With the help of interfaces, I can create a set of behaviours that are attached to different student’s assignments. So that they behave in such a way that is somewhat standard/common for all. In other words, an advantage of interface is that objects can implement multiple interfaces, whereas they inherit from only one class. After defining interfaces, I can create a utility class (ScoreUtility) inside the school library for computing scores of assignments.

Another concept in C# is Extension Methods. For example, in our class English paper, we had a minimum word count. It is cool idea to somehow be able to get a word count out of paper text and permanently attach it to the String method in C#.

There are lot more concepts to cover in C# such as arrays and collections, flow control, loops, list processing with LINQ, C# Test Driven Development, unit testing and exceptions.