NANDINI BHATTACHARYA

Understandments and Conclusions

MODULE – 1

1. Visual Studio 2019 IDE Overview

1.1 Different types of windows (solution exp.,properties etc.)

- The IDE has two basic window types: tool windows and document windows.
- **Document windows:** Document windows contain source code files, arbitrary text files, config files, and so on.
- Document windows can be dragged by their tab. Right-click on the tab or title bar to set other options on the window.

- **Tool windows:** The Toolbox window contains all the controls you can use to build your application's interface.
- This window is usually retracted, and you must move the pointer over it to view the Toolbox.

- **Solution Explorer window:** The Solution Explorer window contains a list of the items in the current solution. A solution can contain multiple projects, and each project can contain multiple items.
- The Solution Explorer displays a hierarchical list of all the components, organized by project.
- You can right-click any component of the project and choose Properties in the context menu to see the selected component's properties in the Properties window.
- If the solution contains multiple projects, you can right-click the project you want to become the startup form and select Set As StartUp Project.
- You can also add items to a project with the Add Item command of the context menu, or remove a component from the project with the Exclude From Project command.
- The Delete command removes the selected component from the project and also deletes the component's file from the disk.

- **Properties window:** This window (also known as the Properties Browser) displays all the properties of the selected component and its settings.
- Every time you place a control on a form, you switch to this window to adjust the appearance of the control.

- **Server Explorer window:** The Server Explorer is a shortcut to accessing servers, either installed on the system, or connected to the system.
- These servers are usually database servers such as SQL Server.
- By accessing the server, you access all the databases on the specific server, and then you can build the connections needed inside your program, for your program.

1.2 Solution,Project

- **Project:** A project contains all files that are compiled into an executable, library, or website.
- Those files can include source code, icons, images, data files, and so on.
- A project also contains compiler settings and other configuration files that might be needed by various services or components that your program communicates with.

- Visual Studio uses MSBuild to build each project in a solution, and each project contains an MSBuild project file.
- The file extension reflects the type of project, for example, a C# project (.csproj) or a database project (.dbproj).
- The project file is an XML document that contains all the information and instructions that MSBuild needs in order to build your project, including the content, platform requirements, versioning information, web server or database server settings, and the tasks to perform.

- **Solution:** A project is contained within a solution.
- It's simply a container for one or more related projects, along with build information, Visual Studio window settings, and any miscellaneous files that aren't associated with a particular project.

1.3 Code editor features

- **Improved Search:** Search results appear dynamically as you type. And, search results can often include keyboard shortcuts for commands.
- **Refactoring:** To make it easier to organise your code. They show up as suggestions in the light bulb.
- **IntelliCode:** Enhances your software development efforts by using AI.
- **Code cleanup:** this new command to identify and then fix both warnings and suggestions with a single action.
- **Live Share:** Developer service that allows you to share a codebase and its context with a teammate and get instant bidirectional collaboration directly from within Visual Studio.
- **Search while debugging**
- **Git-first workflow:** The start window presents you with several options to get you to code quickly. We've placed the option to clone or check out code from a repo, first.
- **Structure Visualizer:** Dotted lines connect matching braces in code files, making it easier to see opening and closing brace pairs.

1.4 Shortcuts

- Add a new class to project – ctrl + shift + A
- Go to definition of any project – F12
- Move lines up and down in the editor – alt + up/down arrows
- Open quick action and refactoring – ctrl + .
- Quickly navigate to anything – ctrl + , and start writing
- Search solution explorer – ctrl + ;
- Run an automated test – ctrl + R,T
- Search – ctrl + Q
- Start debugging – F5
- Formatting – ctrl +K, ctrl + D
- Commenting – ctrl + K + C
- Remove comment – ctrl + K + U
- Rename – F2

2. Project Types

2.1 Windows App, Class Library

- **Windows App:** A UI framework that creates rich desktop client apps for Windows.
- The Windows development platform supports a broad set of app development features, including controls, graphics, data binding, and user input.
- Windows features a drag-and-drop visual designer in Visual Studio to easily create Windows Forms apps.
- With Windows, you develop graphically rich apps that are easy to deploy, update, and work while offline or while connected to the internet.
- Windows apps can access the local hardware and file system of the computer where the app is running.
- Many apps must display data from a database, XML or JSON file, web service, or other data source. Windows Forms provides a flexible control that is named the DataGridView control for displaying such tabular data in a traditional row and column format, so that every piece of data occupies its own cell.
- When you use DataGridView, you can customize the appearance of individual cells, lock arbitrary rows and columns in place, and display complex controls inside cells, among other features.
- Most apps must retain some information about their run-time state, such as the last-known size of forms, and retain user preference data, such as default locations for saved files. The Application Settings feature addresses these requirements by providing an easy way to store both types of settings on the client computer.
- After you define these settings by using either Visual Studio or a code editor, the settings are persisted as XML and automatically read back into memory at run time.

- **Class library:** A class library defines types and methods that are called by an application.
- A class library is a collection of class definitions contained in a .dll or .exe file.
- In order to use class library, add a reference to the library.
- Once the library has been referenced, the classes in the library can be instantiated and exercised.
- In order to instantiate the class, you must either fully qualify the class name or bring it into scope.

2.2 Web Application

- The Web Application Project was created as an add-in.
- It will compile the application into a single DLL file at build time. In order to update the project, it must be recompiled and the DLL file published for changes to occur.
- Another nice feature of the Web Application project is it's much easier to exclude files from the project view.
- In the Web Application Project, the project just keeps track of which files to include/exclude from the project view without renaming them, making things much tidier.
- Has a project file (structure based on project files).
- Supports both IIS and the built-in ASP.NET Development Server.
- Can build a Web application using multiple Web projects.

3. Create First C# Program "Hello World"

3.1 What is namespace?

- The C# language provide a keyword namespace to create a user defined name space. The general form of declaring a namespace is as follows.

```
namespace <namespace_name>
{
// Classes and/or structs and/or enums etc.
}
```

- Namespaces are used in C# to organize and provide a level of separation of codes. They can be considered as a container which consists of other namespaces, classes, etc.
- A namespace can have following types as its members:

1. Namespaces (Nested Namespace)
2. Classes
3. Interfaces
4. Structures

- Where namespace is the required keyword. The name of namespace can be any valid C# identifier or combinations of identifiers separated by commas.
- It is not possible to use any access specifiers like private, public etc with a namespace declarations. The namespaces in C# are implicitly have public access and this is not modifiable.
- The namespace elements can't be explicitly declared as private or protected. The namespace allows only public and internal elements as it members. The default is internal.
- The following code doesn't compile in C#, since the class inside the namespace is declared as private.

```
namespace Csharp.Codes
{
private class MyClass
{
}
}
```

- The namespace members can be accessed by using a fully qualified name, which including the namespace name and member name separated by dot(.) from outside the namespace. We can use using keyword so that we don't have to use complete name all the time.
- In C#, namespaces can be nested with each other.
- With the help of the keyword using, it is possible to create an alias name for a namespace or type.

3.2 What is class?

- A class is a group of related methods and variables.
- A class describes these things, and in most cases, you create an instance of this class, now referred to as an object.
- On this object, you use the defined methods and variables. Of course, you can create as many instances of your class as you want to.
- A class can be defined by using the class keyword.
- Access modifiers (Public, private, protected, and internal) are applied to the declaration of the class, method, properties, fields, and other members. They define the accessibility of the class and its members.
- The constructor will be called when you create an instance of a class. Constructors can be defined by using an access modifier and class name:

<access modifiers> <class name>(){ }

- A method can be defined using the following template:

  {access modifier} {return type} MethodName({parameterType parameterName})

- Property encapsulates a private field. It provides getters (get{}) to retrieve the value of the underlying field and setters (set{}) to set the value of the underlying field.

3.3 Variable & Method Declaration

- **Variable declaration:** In C#, a variable is declared like this:
  <data type> <name>;
- You can assign one at a later point or at the same time as declaring it, like this:
  <data type> <name> = <value>;
- If this variable is not local to the method you're currently working in (e.g. a class member variable), you might want to assign a visibility to the variable:
  <visibility> <data type> <name> = <value>;
  And a complete example:
  private string name = "Nandini";
- There are different types of variables (defined with different keywords), for example: int, char, double, string and bool.
- A variable must be assigned a value before using it, otherwise, C# will give a compile-time error.
- The value of a variable can be changed anytime after initializing it.

```
int num = 100;
num = 200;
Console.WriteLine(num); //output: 200
```

- **Method declaration:** In C# method declaration, you can declare method by following way:

```
<Access specifier> <Return type> <Method name> (Parameter list)
{
        //body
}
```

- Here, **specifiers** are public, private, protected, internal defining the scope of use of variables in classes.
- **Return type** – it is the type of the value method will be returning.
- **Parameters** – arguments which are passed during call of the function are received as parameters during defining it.

4. Understanding C# Program

4.1 Program Flow

- A C# program executes in order of the following parts −

1. Namespace declaration
2. A class
3. Class methods

4. Class attributes
5. A Main method
6. Statements and Expressions
7. Comments

- C# programs are case sensitive, which means “string” is different from “String”.
- All the statements written in the program must be concluded with a semicolon i.e. “;”. A semicolon tells the program that the current line of the statement has ended.
- The execution of the C# program starts from the Main method, hence the program should have the Main method as its starting point.

4.2 Understanding Syntax

- Example:-

```
using System;
namespace Hello {
  class HelloWorld {
    static void Main(string[] args) {
      /* my first program in C# */
      Console.WriteLine("Hello World");
    }
  }
}
```

- The first line of the program **using System;** - the **using** keyword is used to include the **System** namespace in the program. A program generally has multiple **using** statements.
- The next line has the **namespace** declaration. A **namespace** is a collection of classes. The Hello namespace contains the class HelloWorld.
- The next line has a **class** declaration, the class HelloWorld contains the data and method definitions that your program uses.
- Classes generally contain multiple methods. Methods define the behavior of the class. However, the HelloWorld class has only one method **Main.**
- The next line defines the **Main** method, which is the entry point for all C# programs. The **Main** method states what the class does when executed.
- The next line /*...*/ is ignored by the compiler and it is put to add **comments** in the program.
- The Main method specifies its behavior with the statement **Console.WriteLine("Hello World");**
- WriteLine is a method of the Console class defined in the System namespace. This statement causes the message "Hello, World!" to be displayed on the screen.

- Few more things to keep in mind:-

1. **Single-line Comments in C#**

A single-line comments starts with //. That is, all the words in the same line after the // are referred as single-line comments.

```
// this is single line comments
```

2. **Multi-line Comments in C#**

Multi-line comments starts with /* and ends with */. No matter, how many lines you are using.

```
/* hello,
 * i am multi-line
 * comment */
```

3. **Identifiers in C#**

An identifier is a name simply used to identify a variable, class, function, or any other user-defined items.

- An identifier name must begin with a letter, followed by sequence of letters, digits (0 to 9) or underscore.
- An identifier name must not be a C# keyword

4. **Keywords in C#**

Keywords in C#, are the reserved words, having special meaning to the C# compiler.

5. Working with code files, projects & solutions

5.1 Understanding structure of solution

- A solution contains a collection of projects, along with information on dependencies between those projects.
- The projects themselves contain files.
- You can have as many projects as you like in a solution, but there can be only one solution open at a time in a particular instance of VS.NET. (You can, of course, open multiple solutions by running multiple instances of VS.NET.)
- Solutions contain only projects—you cannot nest one solution inside another.
- However, projects can belong to multiple solutions, which gives you great flexibility for organizing your builds.
- Two files are typically created for each solution. The *.sln* file contains a complete description of the contents of the solution. The *.suo* file just contains information such as editor window positions and breakpoint settings. The *.suo* file is essentially dispensable since it is not required in order to build the projects in the solution; unlike *.sln* files, *.suo* files are not normally checked into source control.

5.2 Understanding structure of project (Win app, web app,web api, class library)

1. **Structure of Windows forms app**

- File name is same as app name and **[appname].csproj** file contains references of other projects and of packages used and also version of project etc.
- It has got **dependencies** which has all the server side NuGet Packages and other third party packages required in the project
- It has a .cs file which contains code of windows form that we create.
- It also contains **Program.cs** file which has Main method as required to run app as console application.
- The configuration for app is in **CreateHostBuilder** Method called in Main method.

2. **Structure of Web app**

- File name is same as app name and **[appname].csproj** file contains references of other projects and of packages used and also version of project etc.
- It contains **Connected Services** folder which is used to connect project to services like Azure and is basically used during deployment.
- It has got **dependencies** which has all the server side NuGet Packages and other third party packages required in the project
- Then there is properties folder which contains launchSettings.json which contains debug settings
- Then it contains **Pages** folder which contains some demo interface pages.
- **wwwroot** folder contains static files like images, css, JavaScript, etc. These are the only files which are served over http request.
- It also contains **Program.cs** file which has Main method as required to run app as console application.
- It also contains **Startup.cs** file which runs always first when the project is executed and it contains configurations of services used in project
- **appsettings.json** is an application configuration file and contains configurations like database settings, any global variables for whole application.

3. **Structure of Class Library**

- File name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc.
- It has got **dependencies** which has all the server side NuGet Packages and other third party code files required in the project
- It has a .cs file which contains code of a particular class that we define in it.

4. **Structure of Web Api Project**

- File name is same as app name and [appname].csproj file contains references of other projects and of packages used and also version of project etc.
- It contains **Connected Services** folder which is used to connect project to services like Azure and is basically used during deployment.
- It has got **dependencies** which has all the server side NuGet Packages and other third party packages required in the project
- Then there is properties folder which contains launchSettings.json which contains debug settings
- Then it contains **Controller** folder which contains controllers which has code of controlling or manipulating an api.
- **wwwroot** folder contains static files like images, css, JavaScript, etc. These are the only files which are served over http request.
- It also contains **Program.cs** file which has Main method as required to run app as console application.
- It also contains **Startup.cs** file which runs always first when the project is executed and it contains configurations of services used in project
- **appsettings.json** is an application configuration file and contains configurations like database settings, any global variables for whole application.

5.3 Familier with different type of file extentions

- **.sln** – It contains information of the projects contained in the solution as it is the solution file.
- **.csproj** – It contains the references of other projects and of packages used and also version of project etc.
- **.cs** – class file of C#.
- **.json** – JavaScript Object Notation file which stores simple objects and data structures.

6. Understanding data types & variables with conversion

6.1 Base data type

There are 3 types of data types in C# language:-

- **Value Data type:** Assigns a value directly in both signed/unsigned form and the system allocates memory to store the value. They are derived from the class System.ValueType.

  The value types directly contain data which stores numbers, alphabets, and floating point numbers etc.

  There are 2 types of value data type in C# language:-

  1. **Predefined data type:** These are the already defined datatypes in C#
     Eg, Integer – int, Decimal – decimal, Float – float, Character – char, Double – double, Boolean-bool
  2. **User Defined data type:** It is a Datatype which is defined and used by the users.
     Eg, Structure – struct, Enumerations – enum

- **Reference Data type:** The reference data types do not contain the actual data stored in a variable, but they contain a reference to the variables.

  In other words, they refer to a memory location.

  If the data in the memory location is changed by one of the variables, the other variable automatically reflects this change in value.

  1. **Predefined data type:** such as Objects, String.
  2. **User Defined data type:** such as Classes, Interface.

- **Pointer Data type:** The pointer is a variable that points to an address of a value.

6.2 Data type Conversion

- **Implicit Type conversion:** In this, there is not any need for the special syntax. This conversion takes place automatically when we apply value of one data type to other.
- In this conversion, there is not any loss of the data.
- Implicit conversions include the conversion of the small type to large integral types and from the derived class to the base class conversion.
- Two data types should be compatible, Eg, – both should be numerical if one is numerical and other is Boolean it doesn't work.

- **Explicit Type conversion:** Explicit conversion will be done with the cast operator ().
- These conversions are done explicitly by users using the pre-defined functions.
- Casting is done when there is the situation of the data loss.
- This conversion is useful when data types aren't compatible.

- **Built-in Type conversion methods:** ToBoolean, ToChar, ToDateTime, ToDouble, ToDouble, ToInt16, ToInt32, ToString etc.

6.3 Boxing/Unboxing

- **Boxing:** Boxing is the process of converting a value type to the object type or any interface type implemented by this value type. Boxing is implicit.

  Eg,

```
int i = 10;
object o = i;      //performs boxing
```

- **Unboxing: I**t is the process of converting a reference type to value type. Unboxing extract the value from the reference type and assign it to a value type.
- Unboxing is explicit. It means we have to cast explicitly.

  Eg,

```
object o = 10;
int i = (int)o;      //performs unboxing
```

7. Understanding Decision making & statements

7.1 if else, switch

- **If Statement:** The **if statement** contains a boolean condition followed by a single or multi-line code block to be executed. At runtime, if a boolean condition evaluates to true, then the code block will be executed, otherwise not.

```
if(condition)
{
   // code block to be executed when if condition evaluates to true
}
```

- **else if Statement:** Multiple **else if** statements can be used after an **if statement**. It will only be executed when the **if** condition evaluates to false. So, either **if** or one of the **else if** statements can be executed, but not both.

```
if(condition1)
{
   // code block to be executed when if condition1 evaluates to true
}
else if(condition2)
{
   // code block to be executed when
   //     condition1 evaluates to flase
   //     condition2 evaluates to true
}
```

- **else Statement:** The **else statement** can come only after **if** or **else if** statement and can be used only once in the if-else statements.
- The **else statement** cannot contain any condition and will be executed when all of the previous if and else if conditions evaluate to false.

- **switch Statement:** The **switch** statement can be used instead of **if else** statement when you want to test a variable against multiple conditions.
- The **switch** statement starts with the **switch** keyword that contains a match expression or a variable in the bracket **switch(match expression).**
- The result of this match expression or a variable will be tested against conditions specified as cases, inside the curly braces { }. Each case includes one or more statements to be executed.
- The case will be executed if a constant value and the value of a match expression/variable are equal.
- The switch statement also contains a **default label**. The default label will be executed if no cases executed.
- The **break** keyword is used to exit the program control from a switch case.

```
switch(match expression/variable)
{
   case constant-value:
      statement(s) to be executed;
      break;
```

```
    default:
        statement(s) to be executed;
        break;
}
```