

C#

PROGRAMMING BEGINNERS



LEARN IN 1 DAY

DOWNLOAD PDF   CODING BUGS  NOTES GALLERY

C# for Beginners: Learn in 1 Day

By Krishna Rungta

Copyright 2016 - All Rights Reserved – Krishna Rungta

ALL RIGHTS RESERVED. No part of this publication may be reproduced or transmitted in any form whatsoever, electronic, or mechanical, including photocopying, recording, or by any informational storage or retrieval system without express written, dated and signed permission from the author.

Table of Contents

Chapter 1: What is .NET Framework

1. [What is the .NET Framework](#)
2. [Different versions of the .Net framework](#)

Chapter 2: Download and Install Visual Studio

Chapter 3: Data Types

1. [Building the first console application](#)
2. [Basic Data Types](#)
3. [Enumeration](#)
4. [Variables and Operators](#)
5. [Flow Control and conditional statements](#)
6. [Arrays](#)

Chapter 4: Class and Object

1. [What are classes and objects](#)
2. [Fields and methods](#)
3. [Access Modifiers](#)
4. [Constructors](#)
5. [Inheritance](#)
6. [Polymorphism](#)
7. [Abstract classes](#)
8. [Interfaces](#)

Chapter 5: Collections

1. [ArrayList](#)
2. [Stack](#)
3. [Queues](#)
4. [Hashtable](#)

Chapter 6: Windows Forms Application

1. [Windows Forms Basics](#)
2. [Hello World in Windows Forms](#)

3. [Adding Controls to a form](#)
4. [Event Handling for Controls](#)
5. [Other Controls](#)

Chapter 7: Database Access

1. [Fundamentals of Database connectivity](#)
2. [Connections in .Net](#)
3. [Accessing data](#)
4. [Inserting Records](#)
5. [Updating Records](#)
6. [Deleting Records](#)
7. [Connecting Controls to Data](#)
8. [Using DataGrids](#)

Chapter 8: File Operations

1. [Basic File I/O Commands](#)
2. [Streams – Reading and Writing to files](#)
3. [Serialization](#)

Chapter 1: What is .NET Framework

The .Net framework is a software development platform developed by Microsoft. The framework was meant to create applications, which would run on the Windows Platform. The first version of the .Net framework was released in the year 2000.

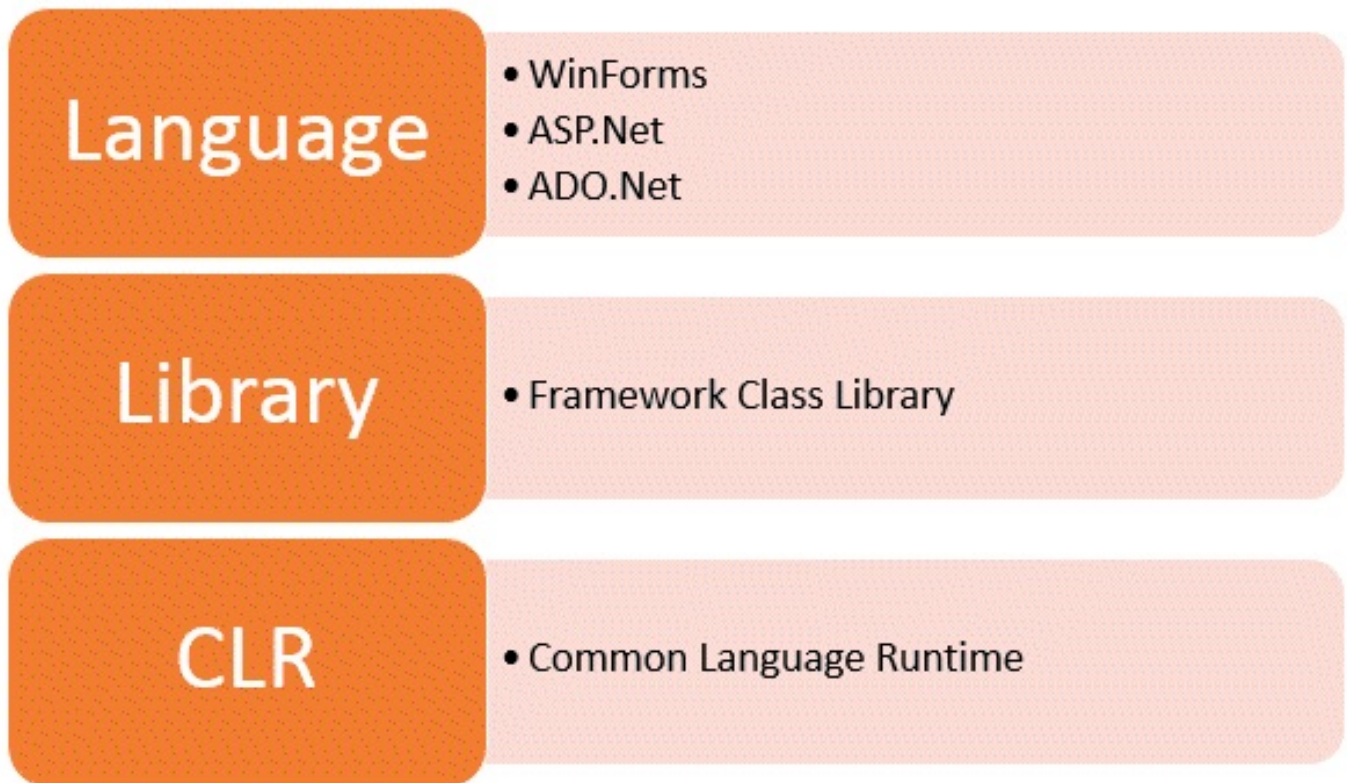
The version was called .Net framework 1.0. The .Net framework has come a long way since then, and the current version is 4.6.1.

The .Net framework can be used to create both - **Form based** and **Web based** applications. Web services can also be developed using the .Net framework.

The framework also supports various programming languages such as Visual Basic and C#. So developers can choose and select the language to develop the required application. In this chapter, you will learn some basics of the .Net framework.

.Net Framework Architecture

The basic architecture of the .Net framework is as shown below.



The architecture of the .Net framework is based on the following key components;

1. **Common Language Runtime** - The “Common Language Infrastructure” or CLI is a platform on which the .Net programs are executed.

The CLI has the following key features:

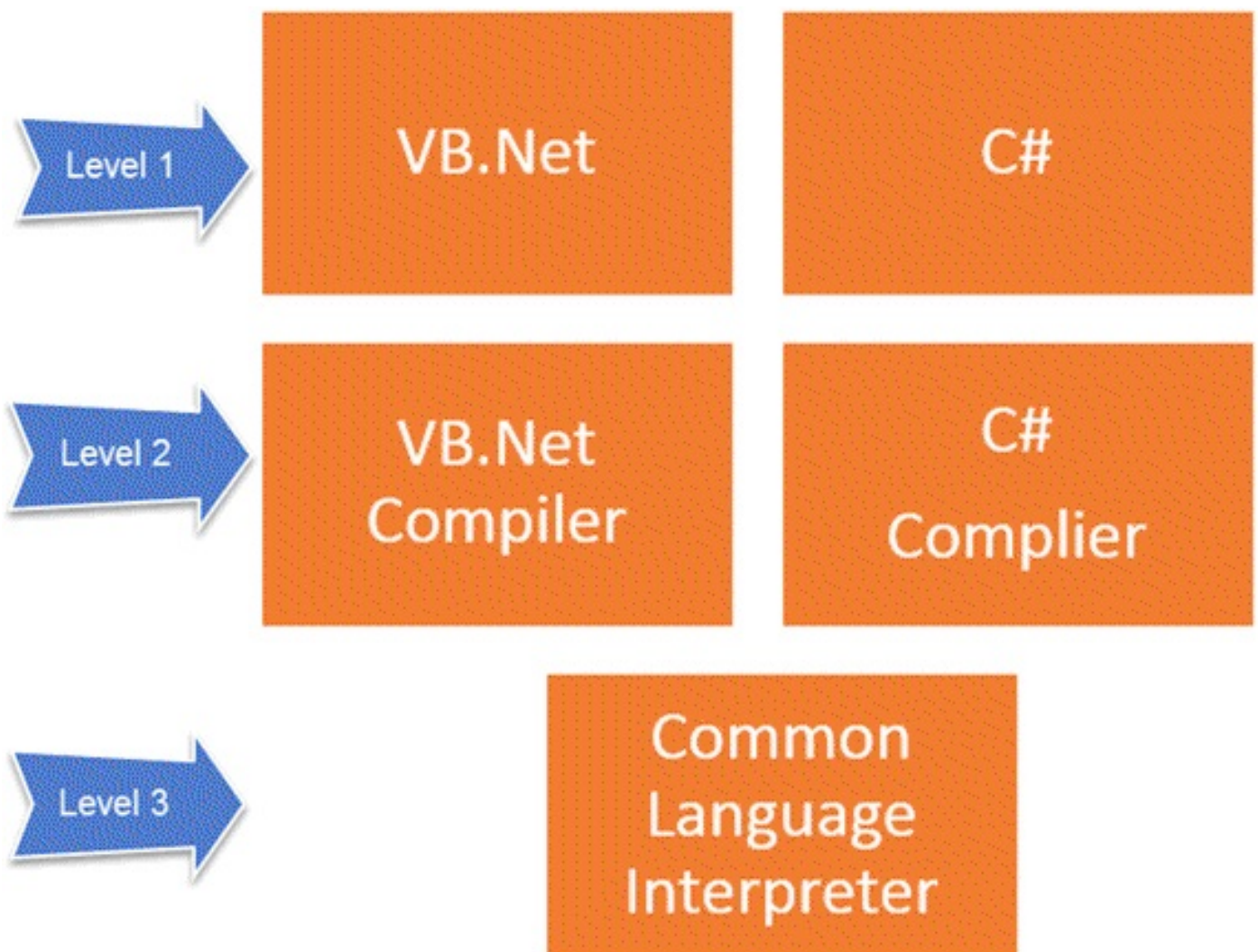
- **Exception Handling** - Exceptions are errors which occur when the application is executed.
Examples of exceptions are:
 - If an application tries to open a file on the local machine, but the file is not present.
 - If the application tries to fetch some records from a database, but the connection to the database is not valid.
- **Garbage Collection** - Garbage collection is the process of removing unwanted resources when they are no longer required.
Examples of garbage collection are
 - A File handle which is no longer required. If the application has finished all operations on a file, then the file handle may no longer be required.
 - The database connection is no longer required. If the application has finished all operations on a database, then the database connection may no

longer be required.

- **Working with Various programming languages –**

As noted in an earlier section, a developer can develop an application in a variety of .Net programming languages.

1. **Language** - The first level is the programming language itself, the most common ones are VB.Net and C#.
2. **Compiler** – There is a compiler which will be separate for each programming language. So underlying the VB.Net language, there will be a separate VB.Net compiler. Similarly for C#, you will have another compiler.
3. **Common Language Interpreter** – This is the final layer in .Net which would be used to run a .net program developed in any programming language. So the subsequent compiler will send the program to the CLI layer to run the .Net application.



2. **Class Library** - The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be

used for the core purpose.

For example, there is a class library with methods to handle all file level operations. So there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the System.* or Microsoft.* namespaces. (The asterisk * just means a reference to all of the methods that fall under the System or Microsoft namespace)

A namespace is a logical separation of methods. We will learn these namespaces more in detail in the subsequent chapters.

3. **Languages** - The types of applications that can be built in the .Net framework are classified broadly into the following categories.

- **WinForms** – This is used for developing Forms-based applications, which would run on an end user machine. Notepad is an example of a client-based application.
- **ASP.Net** – This is used for developing web based applications, which are made to run on any browser such as Internet Explorer, Chrome or Firefox.
 - The Web application would be processed on a server, which would have Internet Information Services Installed.
 - Internet Information Services or IIS is a Microsoft component which is used to execute an ASP.net application.
 - The result of the execution is then sent to the client machines, and the output is shown in the browser.
- **ADO.Net** – This technology is used to develop applications to interact with Databases such as Oracle or Microsoft SQL Server.

Different versions of the .Net framework

Below is the table of .Net framework versions, which have been released with their release dates. Every version has relevant changes to the framework.

For example, in framework 3.5 and onwards a key framework called the **Entity framework** was released. This framework is used to change the approach in which the applications are developed while working with databases.

Version number	CLR version	Release date
1.0	1.0	2002-02-13
1.1	1.1	2003-04-24
2.0	2.0	2005-11-07
3.0	2.0	2006-11-06
3.5	2.0	2007-11-19
4.0	4	2010-04-12
4.5	4	2012-08-15
4.5.1	4	2013-10-17
4.5.2	4	2014-05-05
4.6	4	2015-07-20
4.6.1	4	2015-11-17

The biggest advantage of the .Net framework is that it supports Windows platform. Almost everyone works with Windows machines.

Microsoft always ensures that .Net frameworks are in compliance with all the supported Windows operating systems.

The following design principles of the .Net framework is what makes it very relevant to create .Net based applications.

1. **Interoperability** - The .Net framework provides a lot of backward support. Suppose if you had an application built on an older version of the .Net framework, say 2.0. And if you tried to run the same application on a machine which had the higher version of the .Net framework, say 3.5. The application would still work. This is because with every release, Microsoft ensures that older framework versions gel well with the latest version.
2. **Portability**- Applications built on the .Net framework can be made to work on any Windows platform. And now in recent times, Microsoft is also envisioning to make Microsoft products work on other platforms, such as iOS and Linux.

3. **Security** - The .NET Framework has a good security mechanism. The in-built security mechanism helps in both validation and verification of applications. Every application can explicitly define their security mechanism. Each security mechanism is used to grant the user access to the code or to the running program.
4. **Memory management** - The Common Language runtime does all the work or memory management. The .Net framework has all the capability to see those resources, which are not used by a running program. It would then release those resources accordingly. This is done via a program called the “Garbage Collector” which runs as part of the .Net framework. The garbage collector runs at regular intervals and keeps on checking which system resources are not utilized, and frees them accordingly.
5. **Simplified deployment** - The .Net framework also have tools, which can be used to package applications built on the .Net framework. These packages can then be distributed to client machines. The packages would then automatically install the application.

Summary

- .Net is a programming language developed by Microsoft. It was designed to build applications which could run on the Windows platform.
- The .Net programming language can be used to develop Forms based applications, Web based applications and Web services.
- Developers can choose from a variety of programming languages available on the .Net platform. The most common one's are VB.Net and C#.

Chapter 2: Download and Install Visual Studio

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows. Visual Studio is one stop shop for all applications built on the .Net platform. One can develop, debug and run applications using Visual Studio.

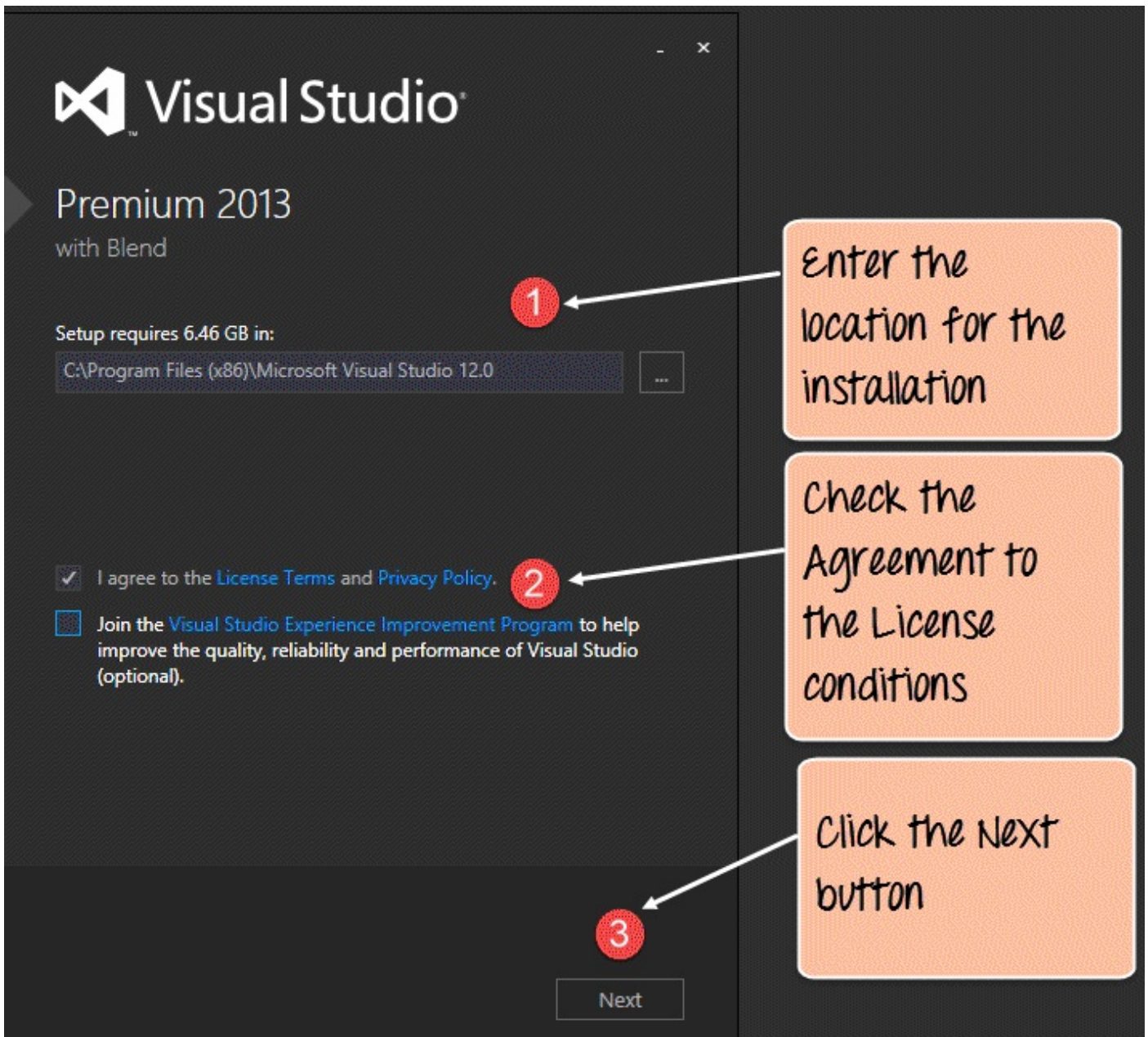
Both Forms based and web based applications can be designed and developed using this IDE. The Visual Studio has the below-mentioned features

1. **Creation of an application in any .Net language** – The Visual Studio IDE can be used to create an application in any .Net language. Hence, a developer can use C#, VB.Net or even F# to develop an application.
2. **Creation of any application type** – The Visual Studio IDE can be used to create an application of any type. (Web-based application or Windows Forms based application).
3. **Debug Applications on the fly** – Applications can be tested as they are being built. The IDE allows one to run the program at any point of time during the development process. Hence, a developer can check for any errors during the development phase itself.
4. **Extensions** – The IDE has the facility to install third-party extensions. An example can be Subversion, which is used for source code repository management. Subversion is used to upload code to a central repository. This is done so that a copy of the code will always exist. Visual Studio has the facility to integrate with such software from the IDE itself. Hence, a developer can work with code repositories from the IDE itself.

Let's look at the installation of this IDE, so that we can work with creating programs in the subsequent chapters. Visual Studio can be downloaded from the following link - <https://www.visualstudio.com/en-us/downloads/download-visual-studio-vs.aspx>

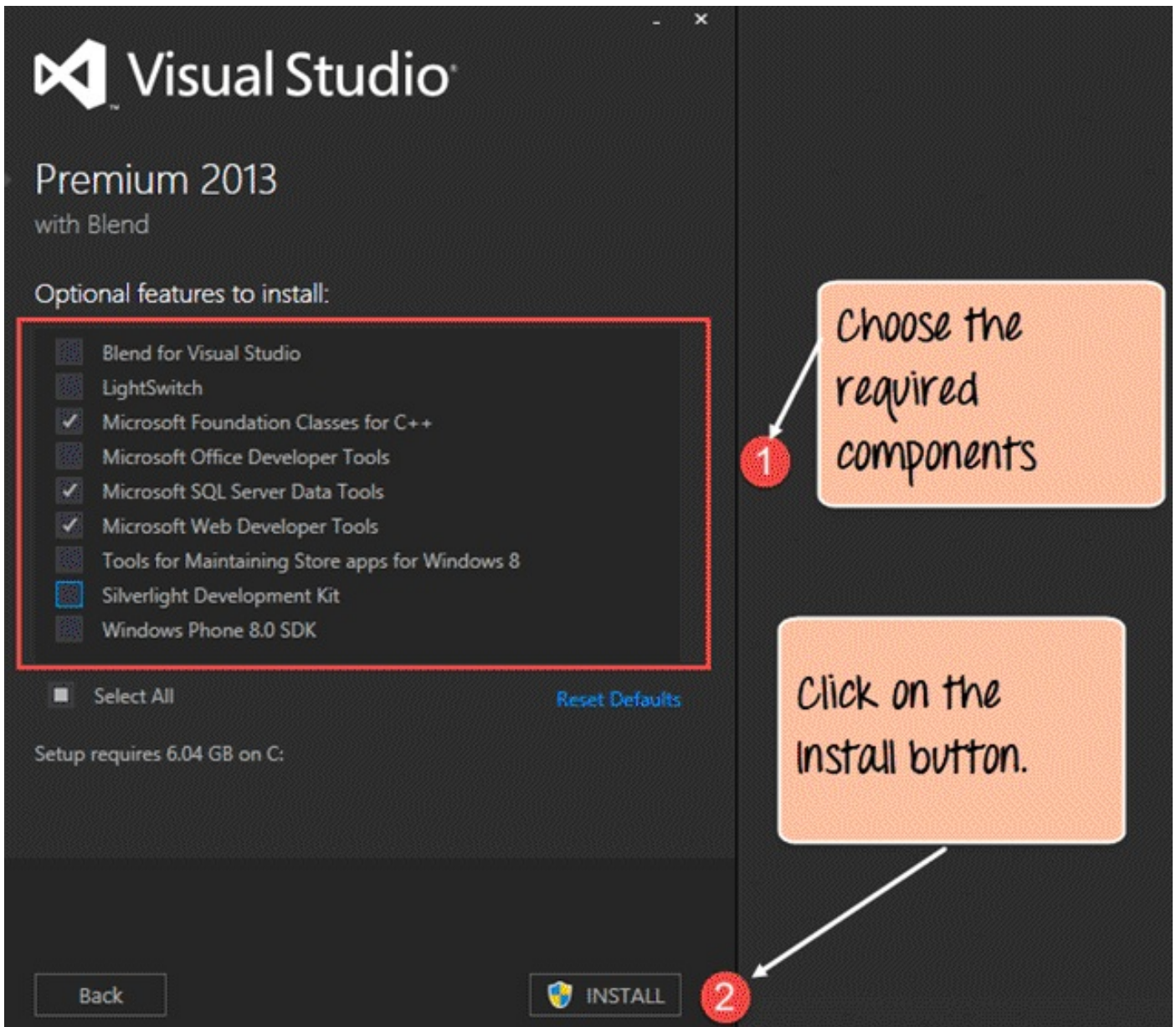
Once you have the Visual Studio Installer, you can install it using the below mentioned steps

Step 1) The first steps involves choosing the location for the installation. In this step one also has to accept the License Terms and conditions.



1. Choose the location where Visual Studio should be installed. Ensure you have enough hard disk space on your machine for the setup.
2. Agree to the License Terms and Privacy policy.
3. Click the Next button to Proceed.

Step 2) The next steps involves choosing the require components as part of the installation.



1. Choose only the required components. This can save on the amount of space required for the Installation. Below are the important components which should be installed

- Microsoft Foundation Classes for C++
- Microsoft SQL Server Data Tools
- Microsoft Web Developer Tools

2. After that, click on the Install button to start the installation.

Once the installation is complete, you will get the below screen showing the success of the installation



Visual Studio[®]

Premium 2013

with Blend

Setup Successful!

All specified components have been installed successfully.

LAUNCH

Chapter 3: Data Types

C# is one of the languages provided by Microsoft to work with .Net. This language encompasses a rich set of features, which allows developing different types of applications.

C# is an object-oriented programming language and resembles several aspects of the C++ Language. In this tutorial, we see how to develop our first application.

This will be a basic console application, we will then explore different data types available in the C# language as well as the control flow statements.

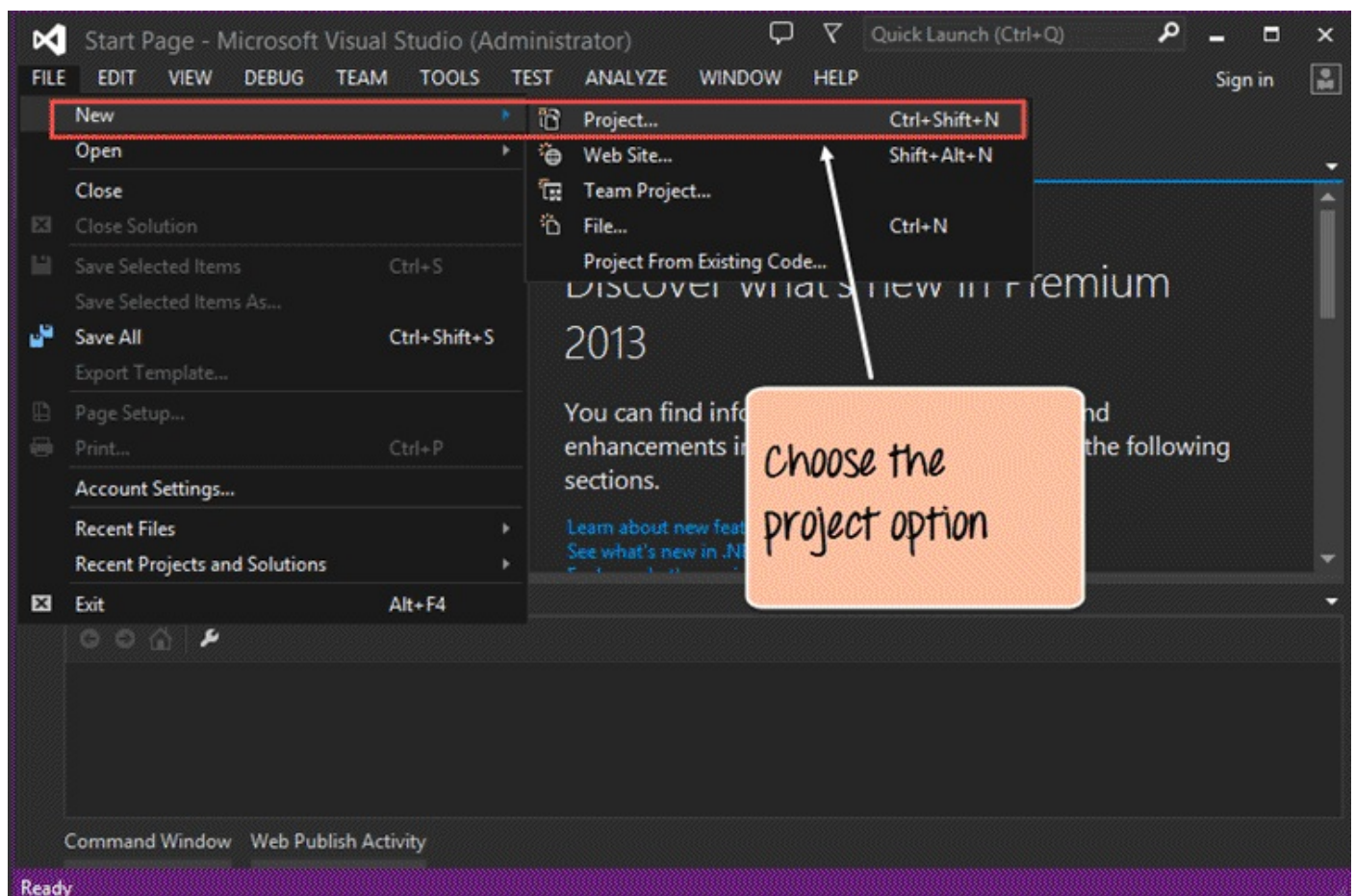
Building the first console application

A console application is an application that can be run in the command prompt in Windows. For any beginner on .Net, building a console application is ideally the first step to begin with.

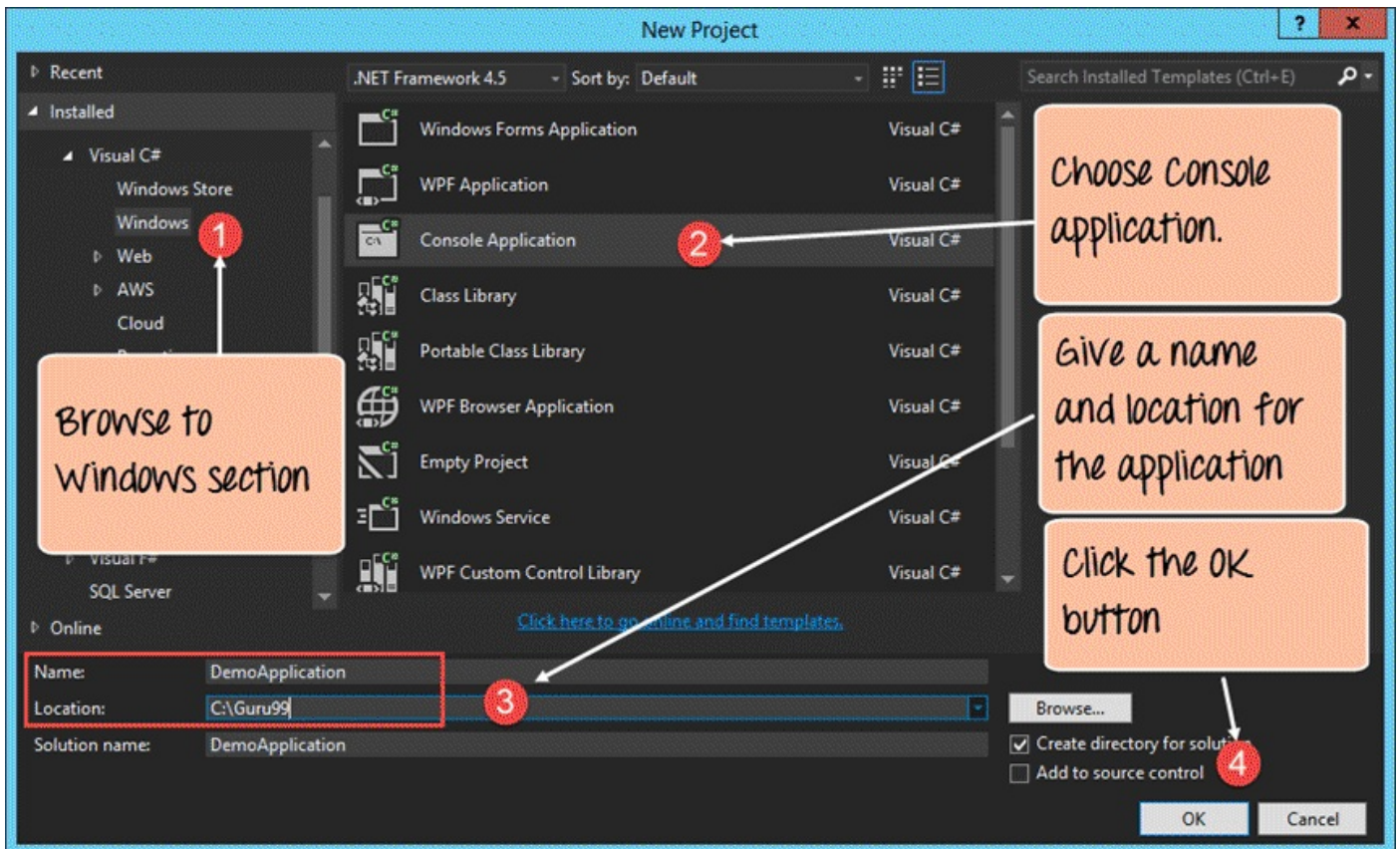
In our example, we are going to use Visual Studio to create a console type project. Next, we are going to use the console application to display a message “Welcome to .Net”. We will then see how to build and run the console application.

Let’s follow the below-mentioned steps to get this example in place.

Step 1) The first step involves the creation of a new project in Visual Studio. For that, once the Visual Studio is launched, you need to choose the menu option New->Project.



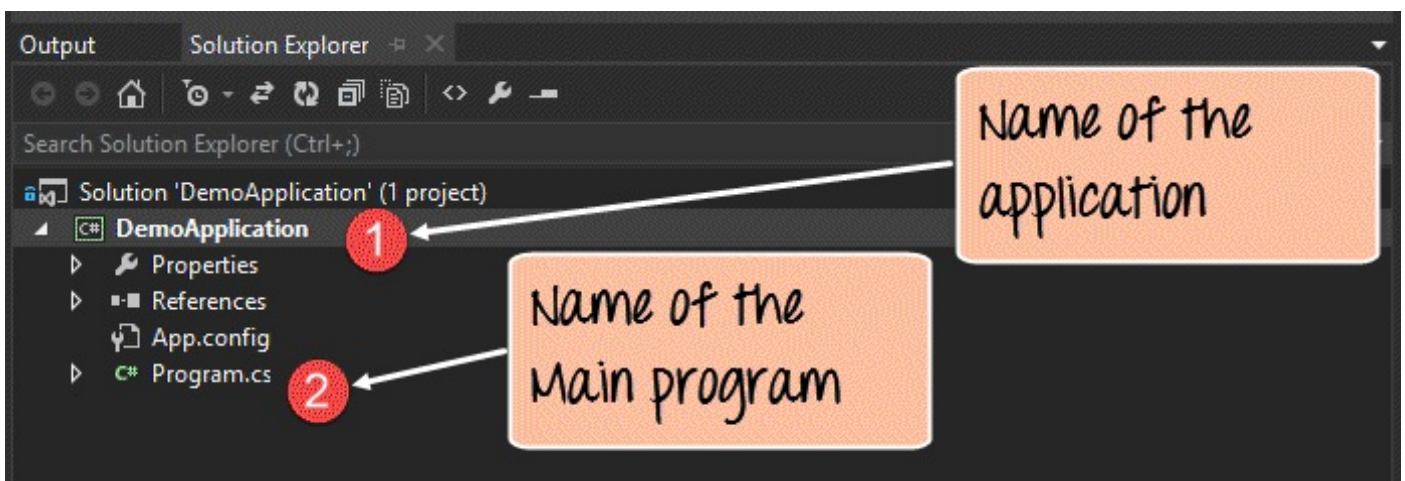
Step 2) The next step is to choose the project type as a Console application. Here, we also need to mention the name and location of our project.



1. In the project dialog box, we can see various options for creating different types of projects in Visual Studio. Click the Windows option on the left-hand side.
2. When we click the Windows options in the previous step, we will be able to see an option for Console Application. Click this option.
3. We then give a name for the application which in our case is DemoApplication. We also need to provide a location to store our application.
4. Finally, we click the 'OK' button to let Visual Studio to create our project.

If the above steps are followed, you will get the below output in Visual Studio.

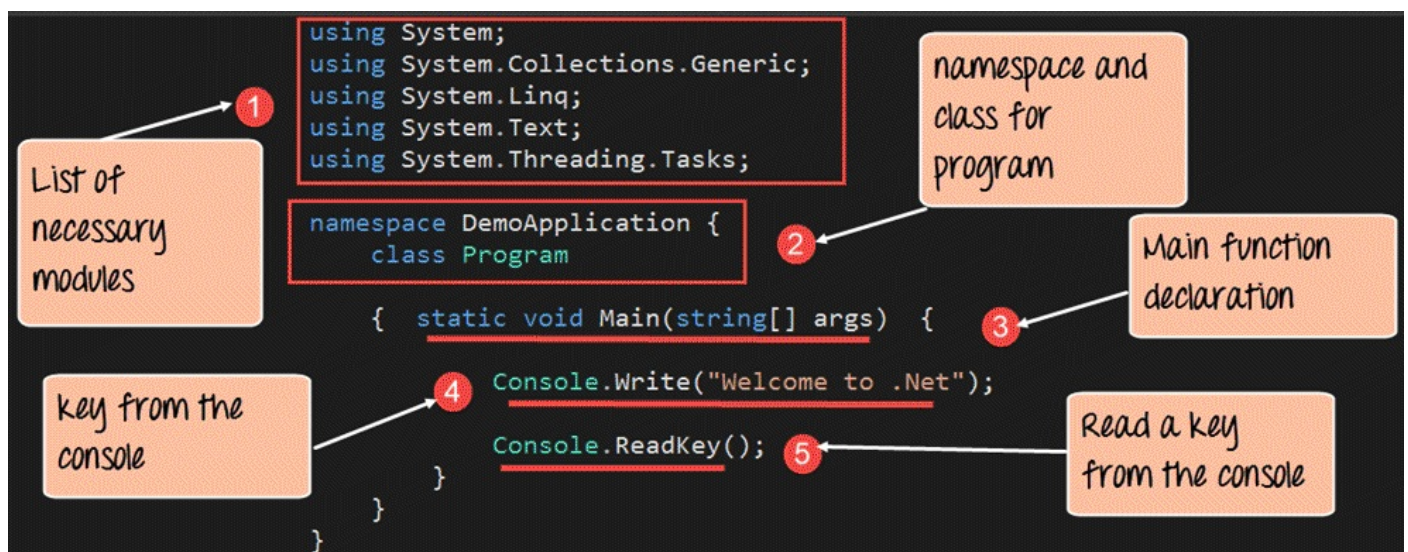
Output:-



1. A project called 'DemoApplication' will be created in Visual Studio. This project will contain all the necessary artifacts required to run the Console application.
2. The Main program called Program.cs is default code file which is created when a new application is created in Visual Studio. This code will contain the necessary code for our console application.

Step 3) Now let's write our code which will be used to display the string "Welcome to .Net" in the console application.

All the below code needs to be entered in the Program.cs file. The code will be used to write "Welcome to .Net" when the console application runs.

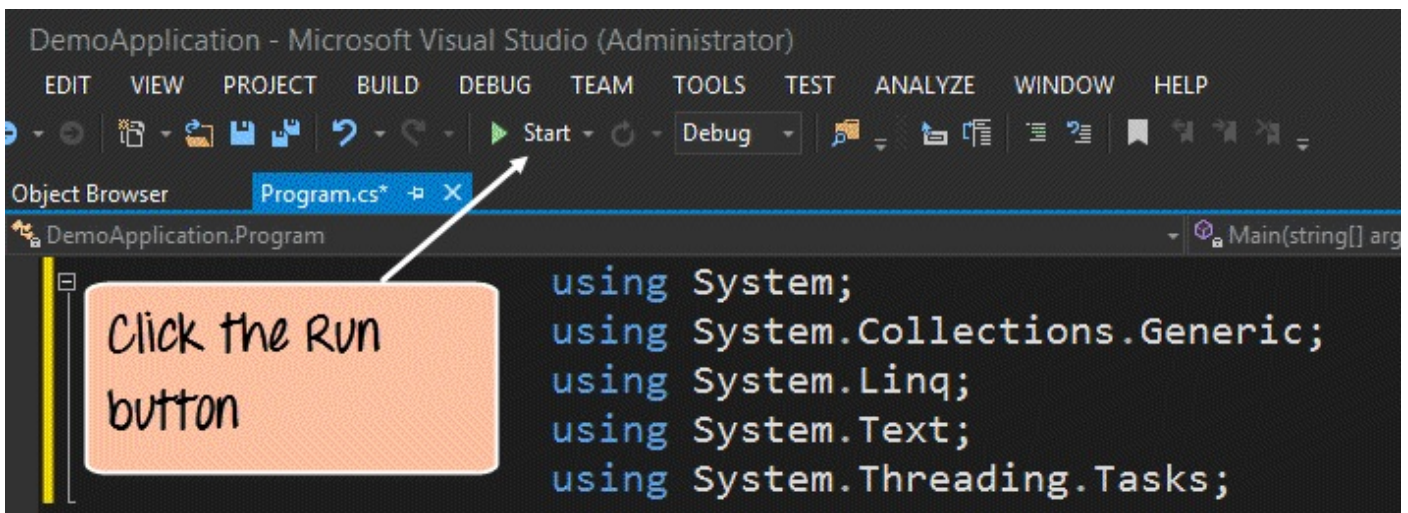


Code Explanation:-

1. The first lines of code are default lines entered by Visual Studio. The 'using' statement is used to import existing .Net modules in our console application. These modules are required for any .Net application to run properly. They contain the bare minimum code to make a code work on a Windows machine.
2. Every application belongs to a class. C# is an object-oriented language, and hence, all code needs to be defined in a self-sustaining module called a 'Class.' In turn, every class belongs to a namespace. A namespace is just a logically grouping of classes.
3. The Main function is a special function which is automatically called when a console application runs. Here you need to ensure to enter the code required to display the required string in the console application.
4. The Console class is available in .Net which allows one to work with console applications. Here we are using an inbuilt method called 'Write' to write the string "Welcome to .Net" in the console.

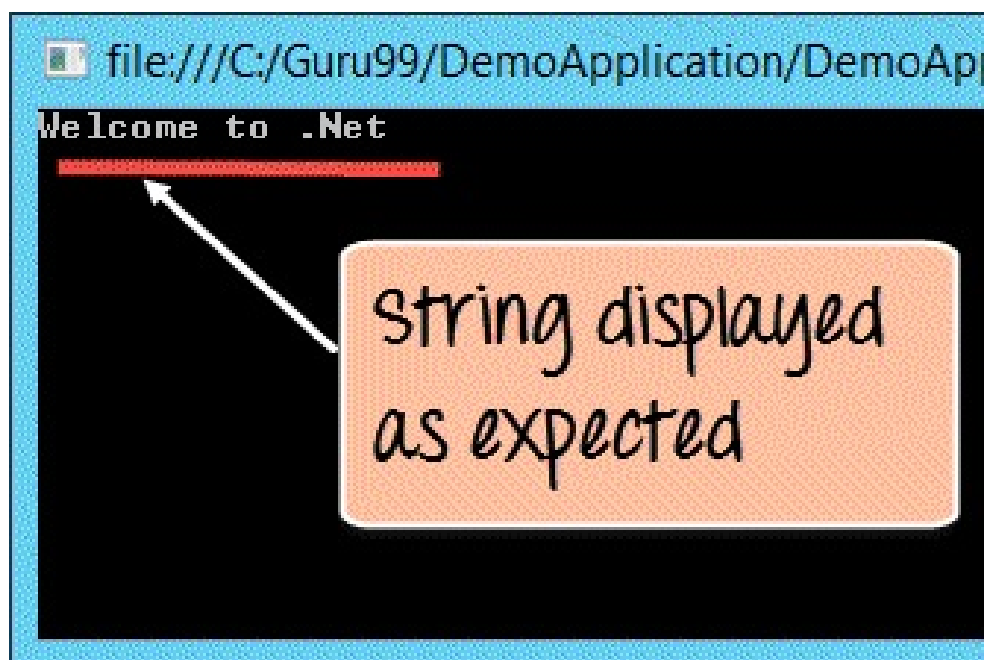
5. We then use the `Console.ReadKey()` method to read any key from the console. By entering this line of code, the program will wait and not exit immediately. The program will wait for the user to enter any key before finally exiting. If you don't include this statement in code, the program will exit as soon as it is run.

Step 4) Run your .Net program. To run any program, you need to click the Start button in Visual Studio.



If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:

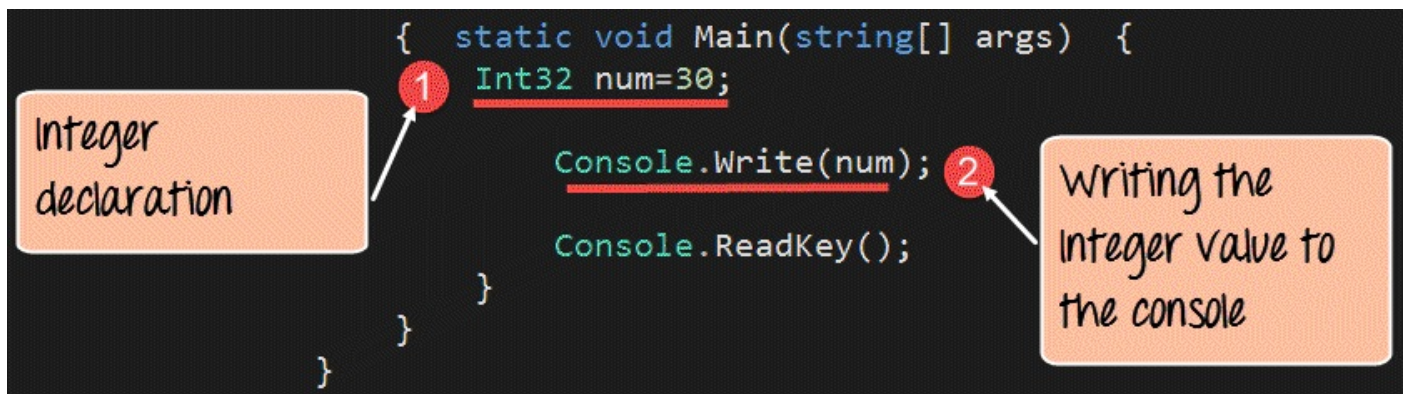


From the output, you can clearly see that the string "Welcome to .Net" is displayed properly. This is because the `Console.write` statement causes this string to be sent to the console.

C# Data Types

The C# language comes with a set of Basic data types. These data types are used to build values which are used within an application. Let's explore the basic data types available in C#. For each example, we will modify just the main function in our Program.cs file.

1. **Integer** – An Integer data types is used to work with numbers. In this case, the numbers are whole numbers like 10, 20 or 30. In C#, the datatype is denoted by the **Int32 keyword**. Below is an example of how this datatype can be used. In our example, we will define an Int32 variable called num. We will then assign an Integer value to the variable and then display it accordingly.



```
{ static void Main(string[] args) {  
    Int32 num=30;  
  
    Console.Write(num);  
    Console.ReadKey();  
}
```

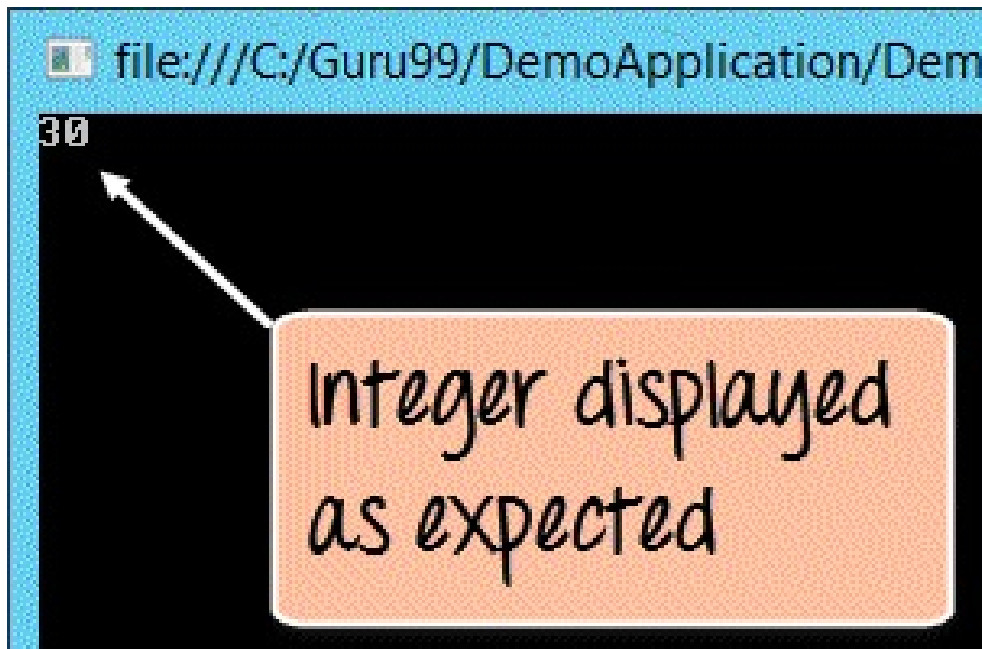
The image shows a code snippet with two callouts. Callout 1 points to the line `Int32 num=30;` and is labeled "Integer declaration". Callout 2 points to the line `Console.Write(num);` and is labeled "Writing the Integer value to the console".

Code Explanation:-

1. The Int32 data type is specified to declare an Integer variable called num. The variable is then assigned a value of 30.
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

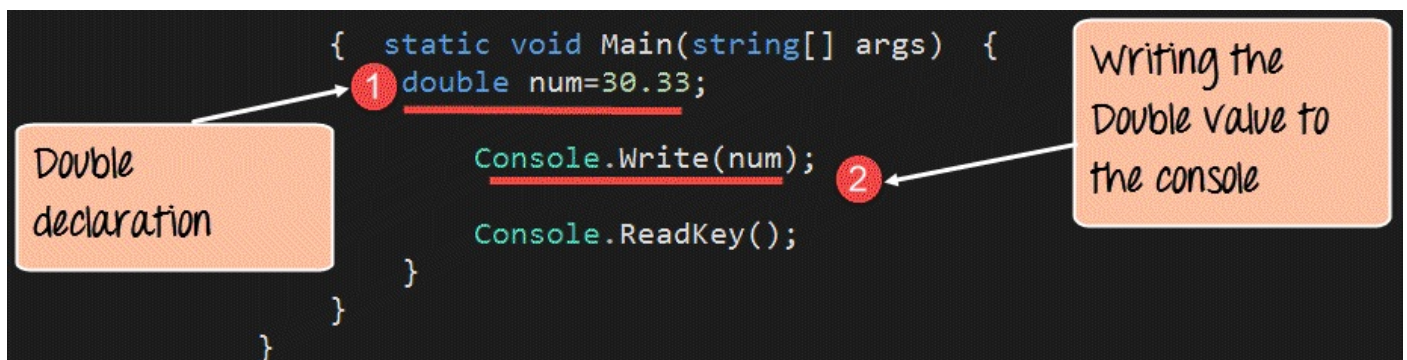
Output:



From the output, you can clearly see that the Integer variable called num was displayed in the console

2. **Double** - A double data type is used to work with decimals. In this case, the numbers are whole numbers like 10.11, 20.22 or 30.33. In C#, the datatype is denoted by the keyword “**Double**“. Below is an example of this datatype .

In our example, we will define a double variable called num. We will then assign a Double value to the variable and then display it accordingly.

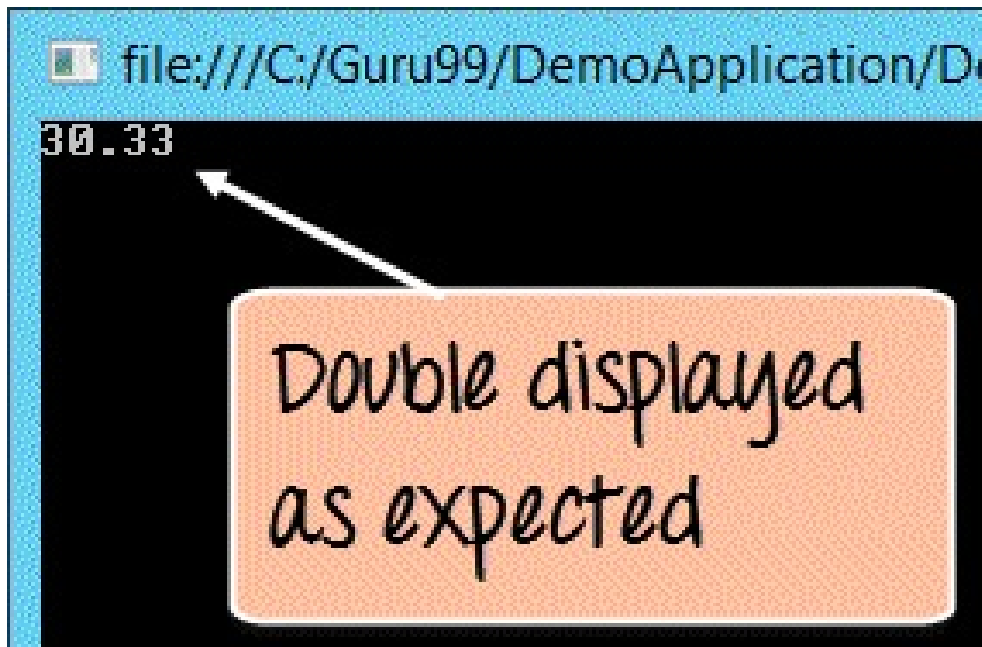


Code Explanation:-

1. The double data type is specified to declare a double type variable called num. The variable is then assigned a value of 30.33.
2. Finally the console.write function is used to display the number to the console.

If the above code is entered properly and the program is executed successfully, following output will be displayed.

Output:



From the output, you can clearly see that the double variable called num was displayed in the console

3. **Boolean** - A boolean data type is used to work with Boolean values of **true** and **false**. In C#, the datatype is denoted by the Boolean keyword. Below is an example of this datatype can be used.

In our example, we will define a Boolean variable called 'status.' We will then assign a boolean value to the variable and then display it accordingly.

```
class Program
{
    static void Main(string[] args) {
        Boolean status=true;
        Console.Write(status);
        Console.ReadKey();
    }
}
```

Boolean declaration

1

2

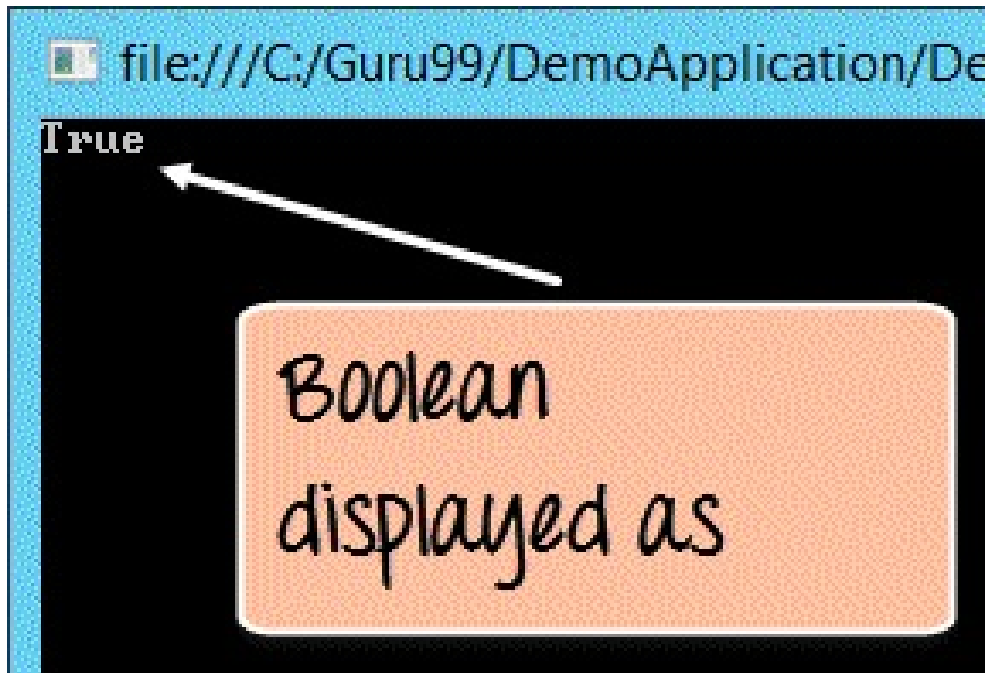
Writing the boolean value to the console

Code Explanation:-

1. The boolean data type is specified to declare a Boolean variable called 'status.' The variable is then assigned a value of true/false.
2. Finally the console.write function is used to display the Boolean value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

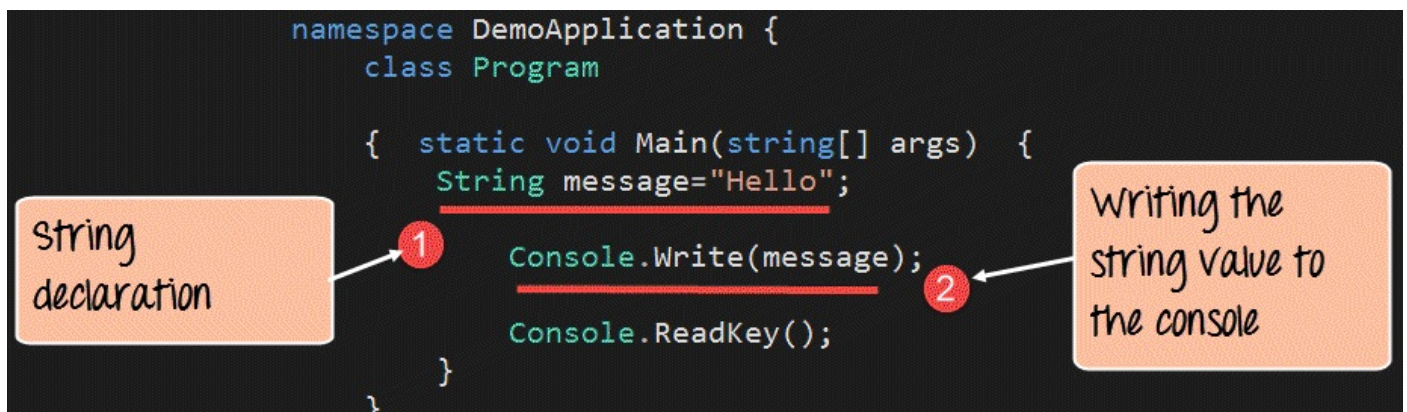
Output:



From the output, you can clearly see that the Boolean variable which equals true was displayed in the console

4. **String** - A String data type is used to work with String values. In C#, the datatype is denoted by the keyword 'String'. Below is an example of this datatype.

In our example, we will define a String variable called 'message.' We will then assign a String value to the variable and then display it accordingly.

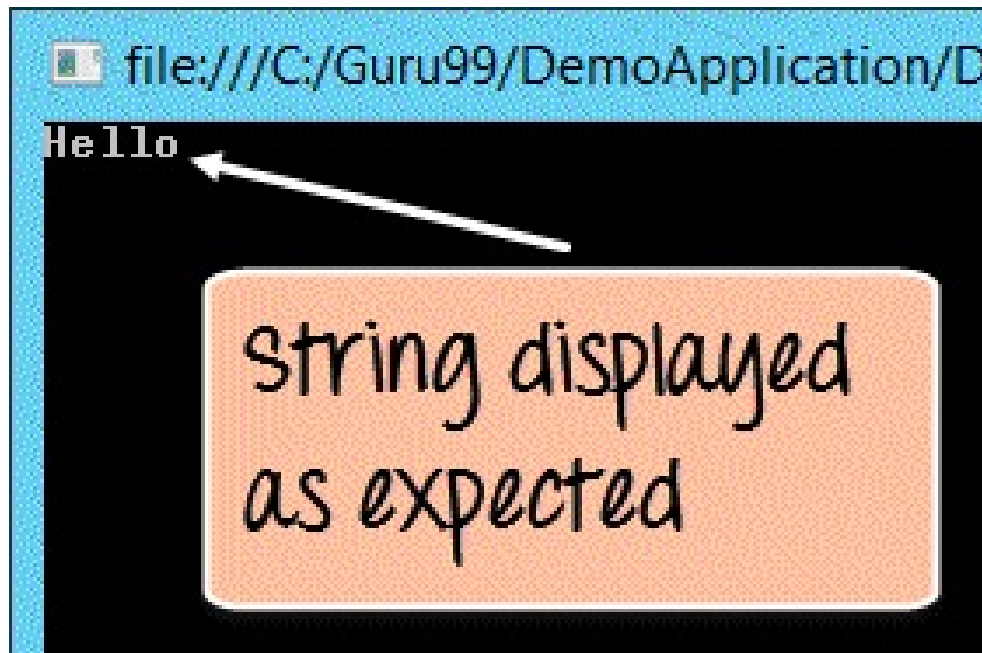


Code Explanation:-

1. The String data type is specified to declare a string variable called message. The variable is then assigned a value of "Hello".
2. Finally, the console.write function is used to display the string value to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:



From the output, you can clearly see that the String variable called message was displayed in the console

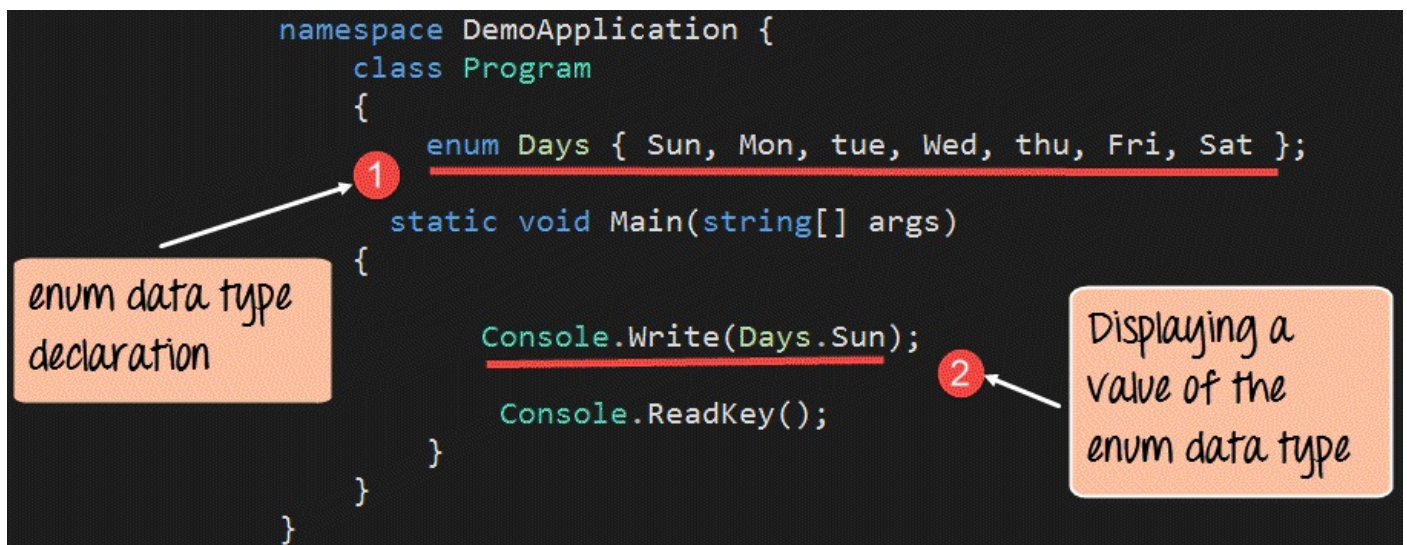
C# Enumeration

An enumeration is used in any programming language to define a constant set of values. For example, the days of the week can be defined as an enumeration and used anywhere in the program. In C#, the enumeration is defined with the help of the keyword 'enum'.

Let's see an example of how we can use the 'enum' keyword.

In our example, we will define an enumeration called days, which will be used to store the days of the week. For each example, we will modify just the main function in our Program.cs file.

```
namespace DemoApplication {
    class Program
    {
        1 enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
        static void Main(string[] args)
        {
            Console.Write(Days.Sun);
            Console.ReadKey();
        }
    }
}
```

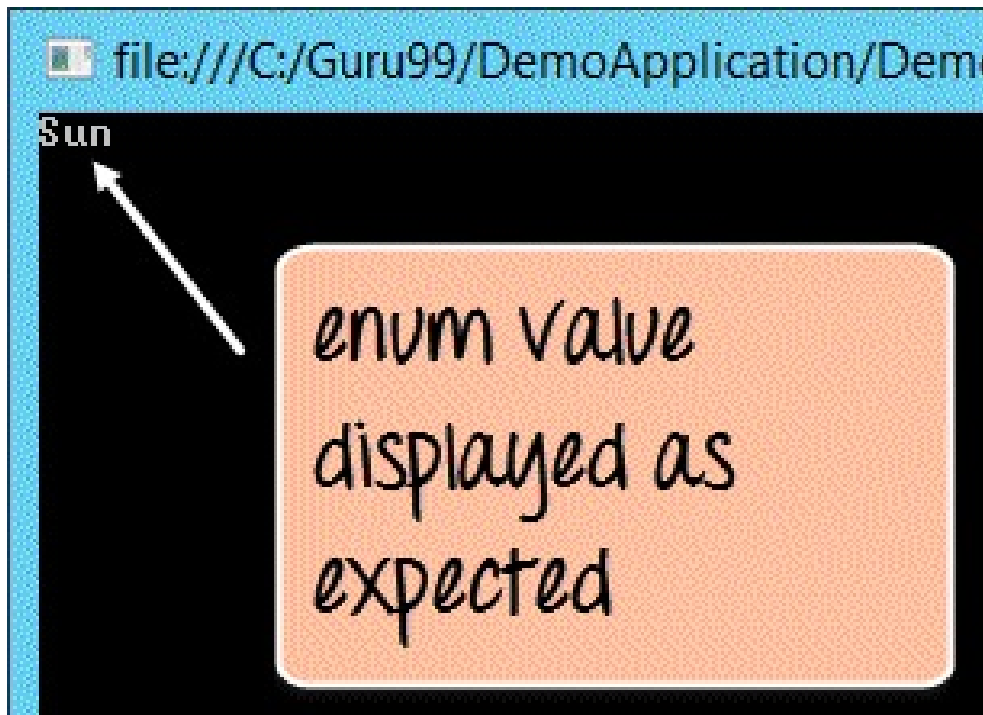


Code Explanation:-

1. The 'enum' data type is specified to declare an enumeration. The name of the enumeration is Days. All the days of the week are specified as values of the enumeration.
2. Finally the console.write function is used to display one of the values of the enumeration.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



From the output, you can clearly see that the 'Sun' value of the enumeration is displayed in the console.

C# Operators and Variables

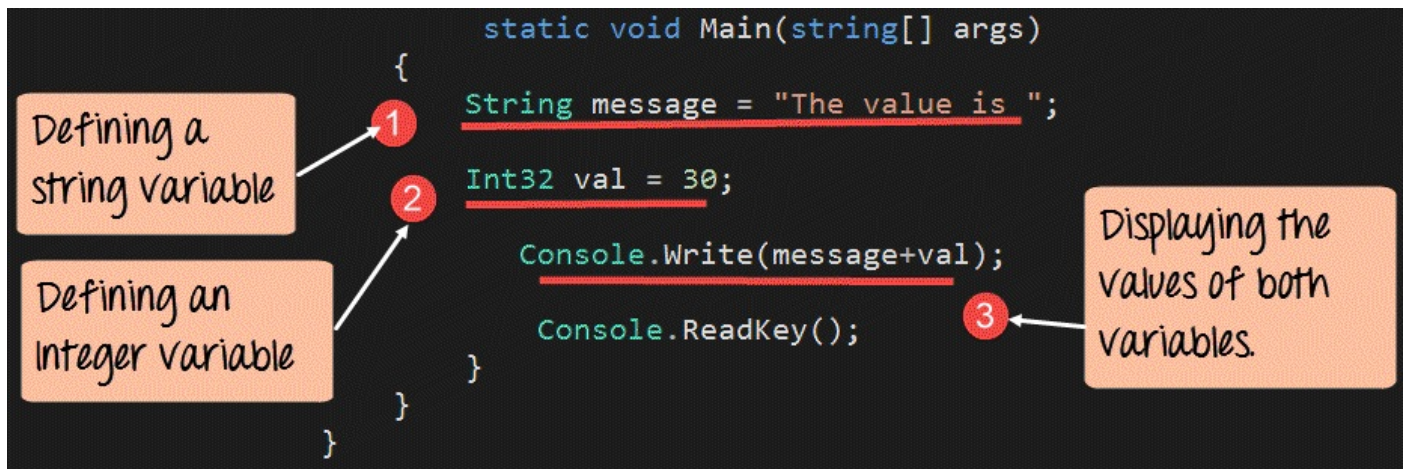
A variable is a name given to a storage area that is used to store values of various data types. Each variable in C# needs to have a specific type, which determines the size and layout of the variable's memory.

For example, a variable can be of the type String, which means that it will be used to store a string value. Based on the data type, specific operations can be carried out on the variable.

For instance, if we had an Integer variable, then operations such as addition and subtraction can be carried out on the variable. One can declare multiple variables in a program.

Let's look at a quick example on the declaration of multiple variables of different data types.

In our example, we will define 2 variables, one of the type 'string' and the other of the type 'Integer'. We will then display the values of these variables to the console. For each example, we will modify just the main function in our Program.cs file.



```
static void Main(string[] args)
{
    String message = "The value is ";
    Int32 val = 30;
    Console.Write(message+val);
    Console.ReadKey();
}
```

The image shows a code editor with three annotations in orange boxes:

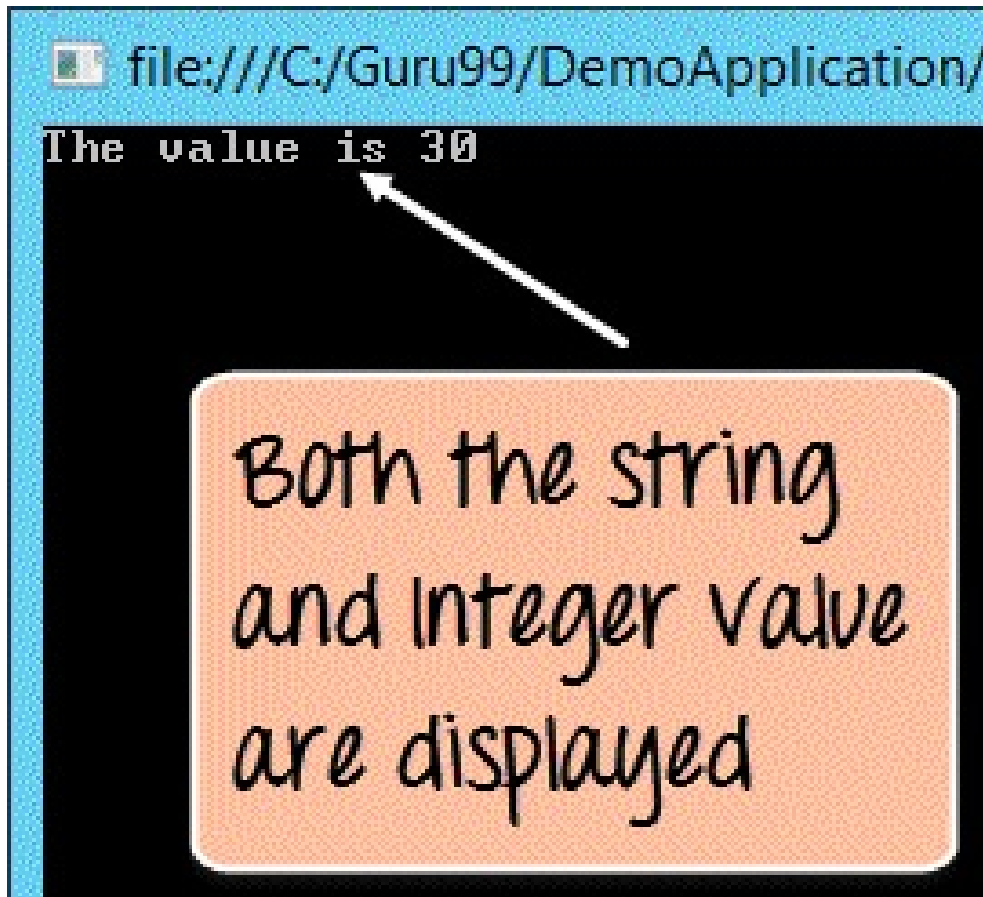
- Box 1: "Defining a string variable" with an arrow pointing to the line `String message = "The value is ";`
- Box 2: "Defining an Integer variable" with an arrow pointing to the line `Int32 val = 30;`
- Box 3: "Displaying the values of both variables." with an arrow pointing to the line `Console.Write(message+val);`

Code Explanation:-

1. A variable of the data type String is declared. The name of the variable is 'message'. The value of the variable is "The value is ".
2. A variable of the data type Integer (Int32) is declared. The name of the variable is 'val'. The value of the variable is 30.
3. Finally the Console.write statement is used to output both the value of the String and Integer variable.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/  
The value is 30
```

Both the string
and integer value
are displayed

From the output, you can clearly see that the values of both the string and integer variable are displayed to the console.

Operators are used to perform operations on values of various data types. For example, to perform the addition of 2 numbers, the + operator is used.

Let's see the table of operators available for the various data types

1. **Arithmetic Operators** – These are operators used for performing mathematic operations on numbers. Below is the list of operators available in C#.

Operator	Description
+	Adds two operands
-	Subtracts second operand from the first
*	Multiplies both operands
/	Divides numerator by de-numerator
%	Modulus Operator and remainder of after an integer division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

2. **Relational Operators** – These are operators used for performing Relational operations on numbers. Below is the list of relational operators available in C#.

Operator	Description
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.

3. **Logical Operators** – These are operators used for performing Logical operations on values. Below is the list of operators available in C#.

Operator	Description
&&	This is the Logical AND operator. If both the operands are true, then condition becomes true.
	This is the Logical OR operator. If any of the operands are true, then condition becomes true.
!	This is the Logical NOT operator.

Let's look at a quick example of how the operators can be used in .Net.

In our example, we will define 2 Integer variables and one Boolean variable. We will then perform the following operations

```

namespace DemoApplication {
    class Program
    {
        static void Main(string[] args)
        {
            Int32 val1=10, val2=20;
            bool status = true;

            Console.WriteLine(val1+val2);
            Console.WriteLine(val1 < val2);
            Console.WriteLine(!(status));
            Console.ReadKey();
        }
    }
}

```

The code is annotated with four callouts:

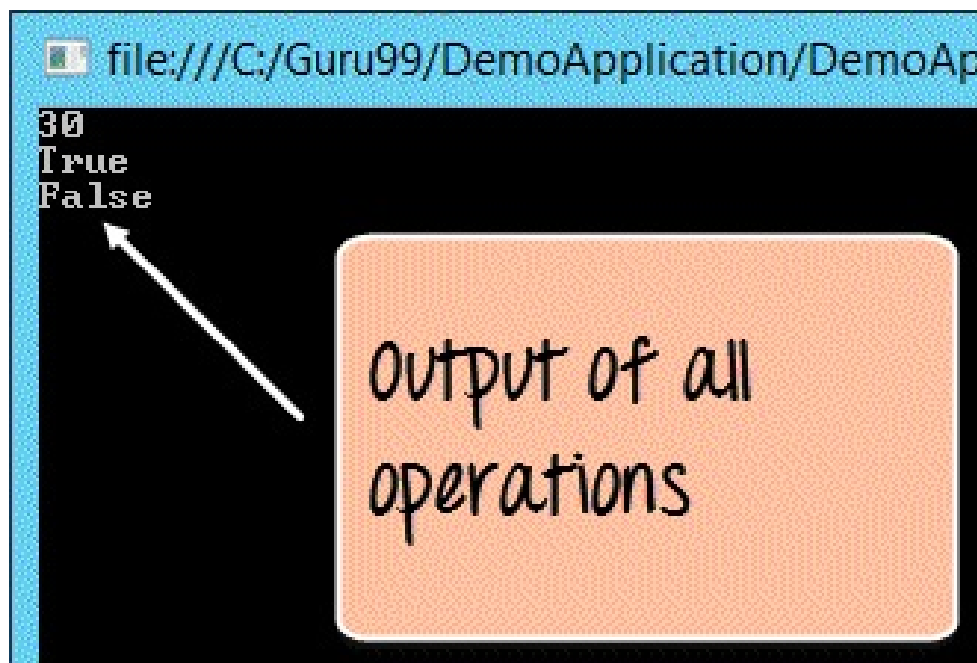
- 1**: Defining variables (points to the variable declarations).
- 2**: using arithmetic operators (points to `val1+val2`).
- 3**: using Relational operators (points to `val1 < val2`).
- 4**: using logical operators (points to `!(status)`).

Code Explanation:-

1. Two Integer variables are defined, one being val1 and the other being val2. These will be used to showcase relational and arithmetic operations. A Boolean variable is defined to showcase logical operations.
2. An example of the arithmetic operation is shown wherein the addition operator is carried out on val1 and val2. The result is written to the console.
3. An example of the relational operation is shown wherein the less than operator is carried out on val1 and val2. The result is written to the console.
4. An example of the logical operation is shown, wherein the logical operator (!) is applied to the status variable. The logical NOT operator reverses the current value of any Boolean value. So if a Boolean value is 'true', the logical NOT will return the value 'false' and vice versa. In our case since the value of the status variable is 'true', the result will show 'false'. The result is written to the console.

If the above code is entered properly and the program is executed successfully, the output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoAp
30
True
False
```

OUTPUT of all operations

Flow Control and conditional statements

Flow control and conditional statements are available in any programming language to alter the flow of a program.

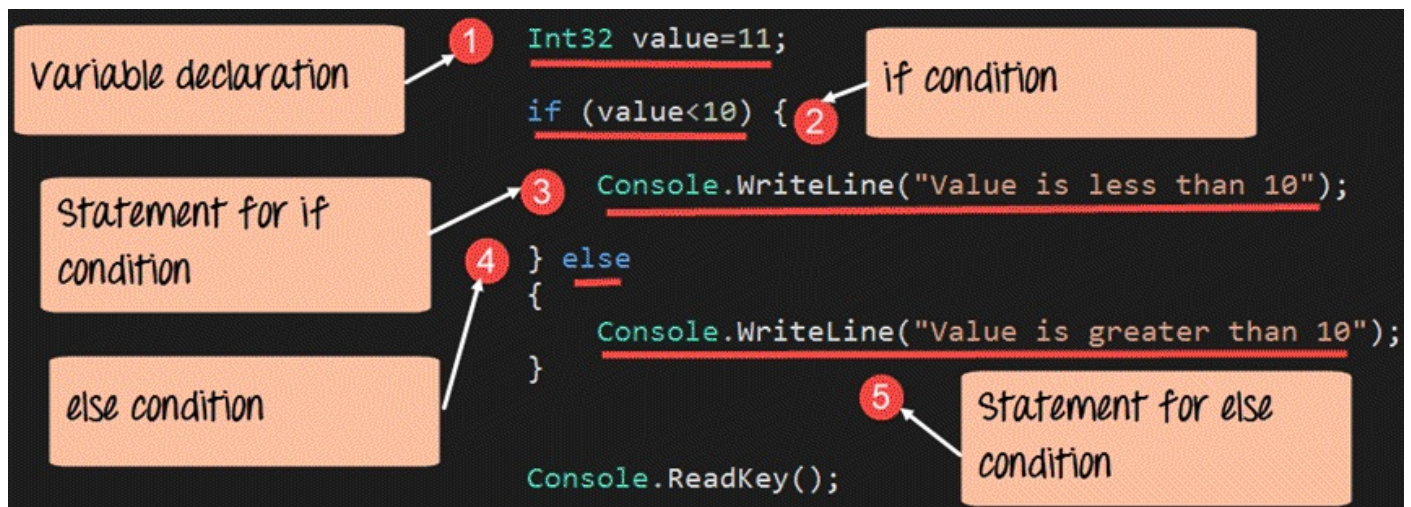
For example, if someone want to execute only a particular set of statements based on some certain logic, then Flow control and conditional statements will be useful.

You will get a better understanding as we go through the various statements which are available in C#.

Please note that all the code below is made to the Program.cs file.

1. **If statement** – The if statement is used to evaluate a boolean expression before executing a set of statements. If an expression evaluates to true, then it will run one set of statements else it will run another set of statements.

In our example below, a comparison is made for a variable called value. If the value of the variable is less than 10 ,then it will run one statement , or else it will run on another statement.

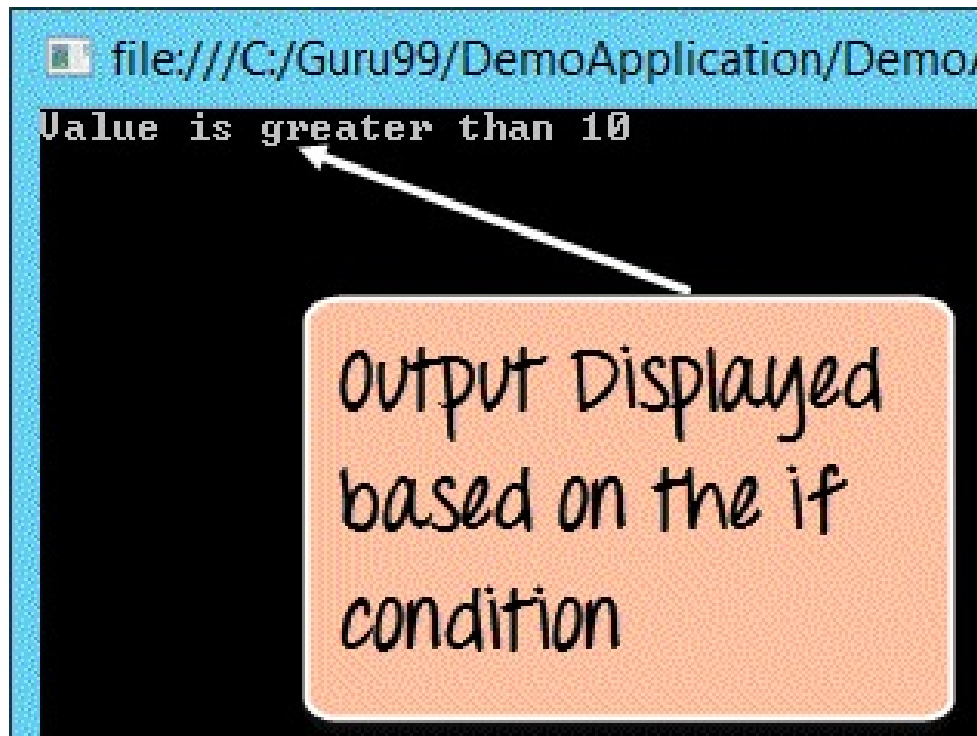


Code Explanation:-

1. We first define a variable called value and set it to the value of 11.
2. We then use the 'if' statement to check if the value is less than 10 of the variable. The result will either be true or false.
3. If the if condition evaluates to true, we then send the message "Value is less than 10" to the console.
4. If the if condition evaluates to false, we then send the message "Value is greater than 10" to the console.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:

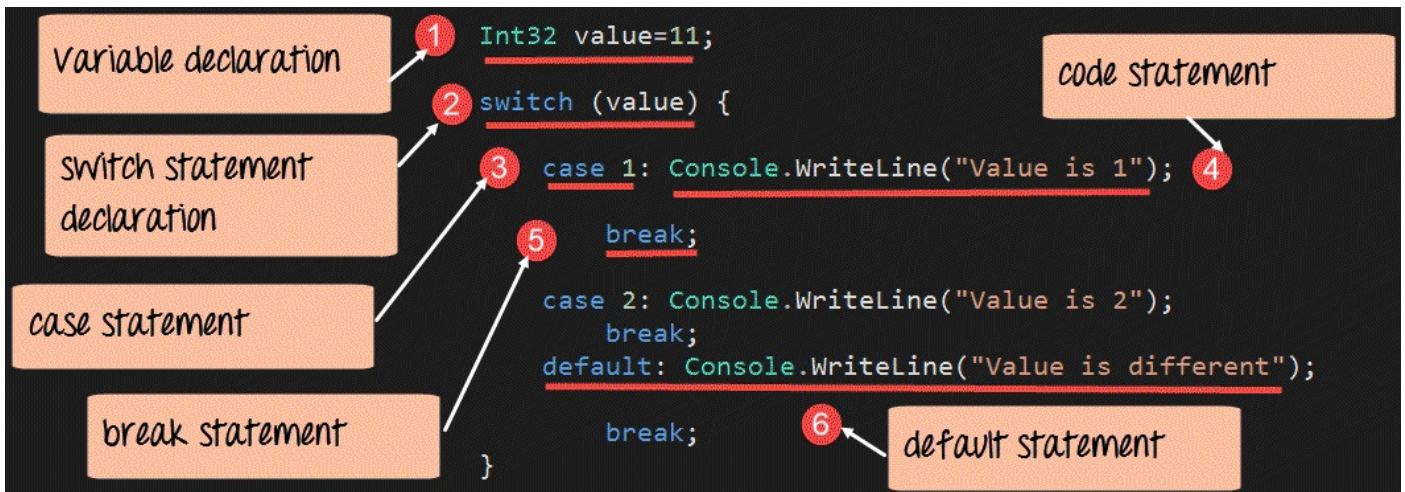


We can clearly see that the ‘if’ statement was evaluated to false. Hence the message “Value is greater than 10” was sent to the console.

- 2. **Switch statement** – The switch statement is an enhancement to the ‘if’ statement. If you have multiple expressions that need to be evaluated in one shot, then writing multiple ‘if’ statements becomes an issue.

The switch statement is used to evaluate an expression and run different statements based on the result of the expression. If one condition does not evaluate to true, the switch statement will then move to the next condition and so forth.

Let’s see, how this works with the below example. Here, we are again comparing the value of a variable called ‘value.’ We then check if the value is equal to 1, or 2, or something totally different.

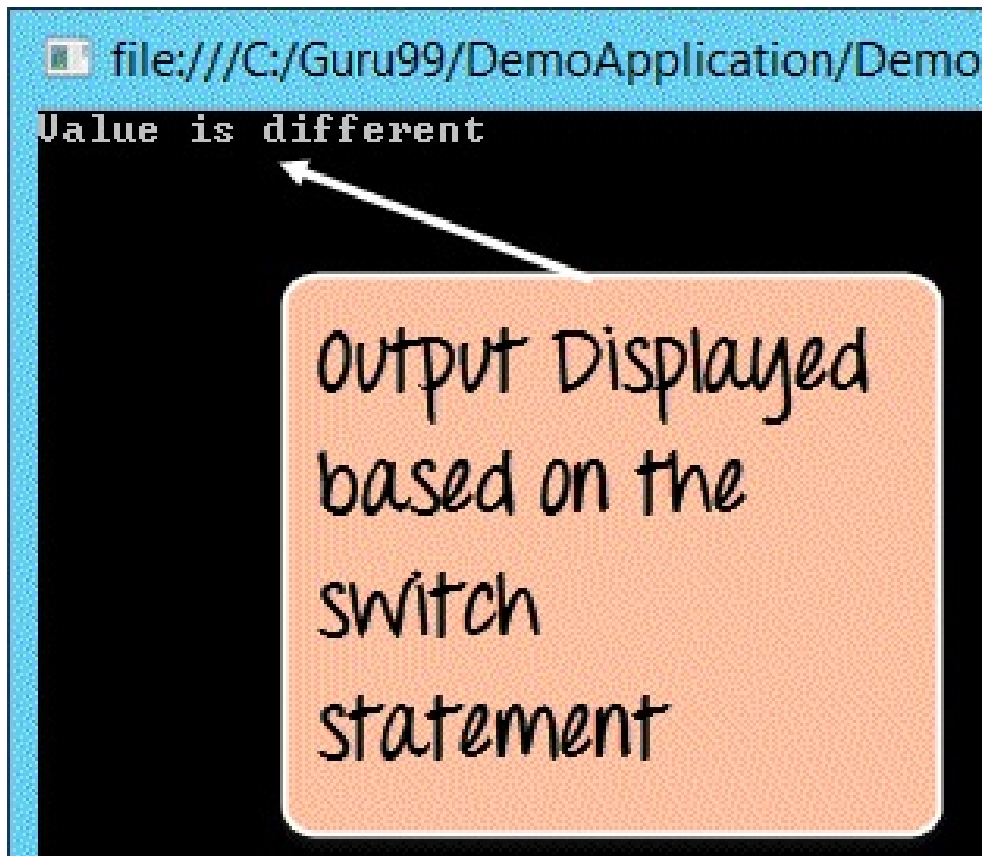


Code Explanation:-

1. We first define a variable called 'value' and set it to the value of 11.
2. We then use the 'switch' statement to check the value of the variable 'value.'
3. Case statements are used to set different conditions. Based on the conditions, a set of statements can be executed. A switch statement can have multiple case conditions. The first case statement checks to see if the value of the variable is equal to 1.
4. If the first case statement is true, then the message "Value is 1" is written to the console.
5. The break statement is used to break from the entire switch statement, once a condition is true.
6. The default condition is a special condition. This just means that if no case expression evaluates to true, then run the set of statements for the default condition.

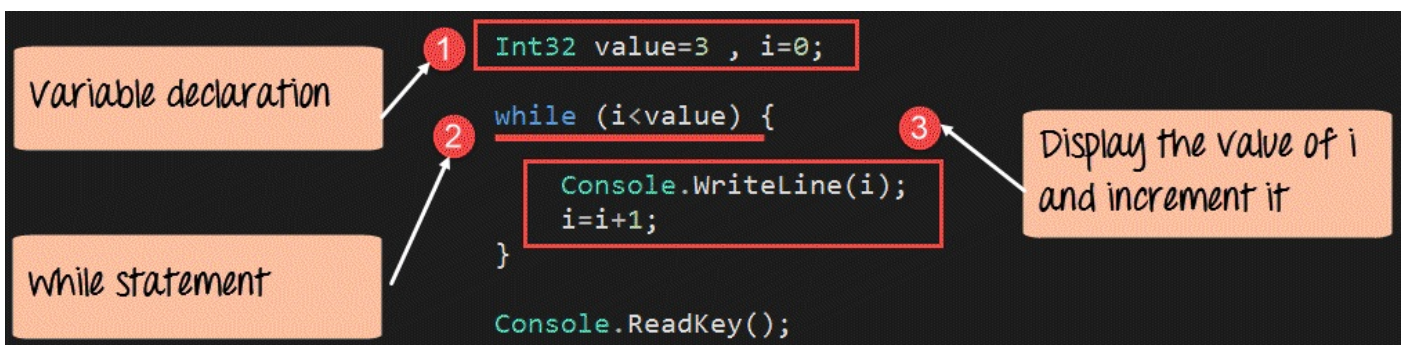
If the above code is entered properly and the program is executed successfully, the following output will be displayed. The output prints the default value "Value is different", since no condition is satisfied.

Output:



3. **While loop** – The while loop is used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then while loop is used.

In our example below, we use the while statement to display the value of a variable 'i'. The while statement is used to display the value 3 times.

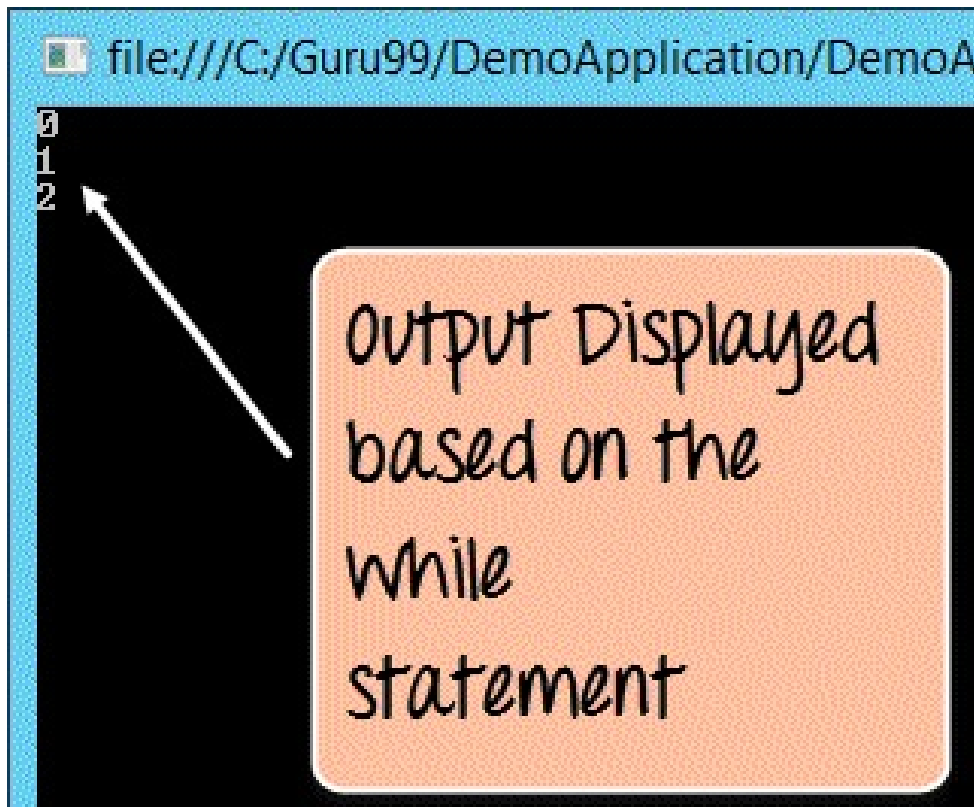


Code Explanation:-

1. Two Integer variables are defined, one being value and the other being 'i'. The value variable is used as the upper limit to which we should iterate our while statement. And 'i' is the variable which will be processed during the iteration.
2. In the while statement, the value of 'i' is constantly checked against the upper limit.
3. Here we display the value of 'i' to the console. We also increment the value of 'i'.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

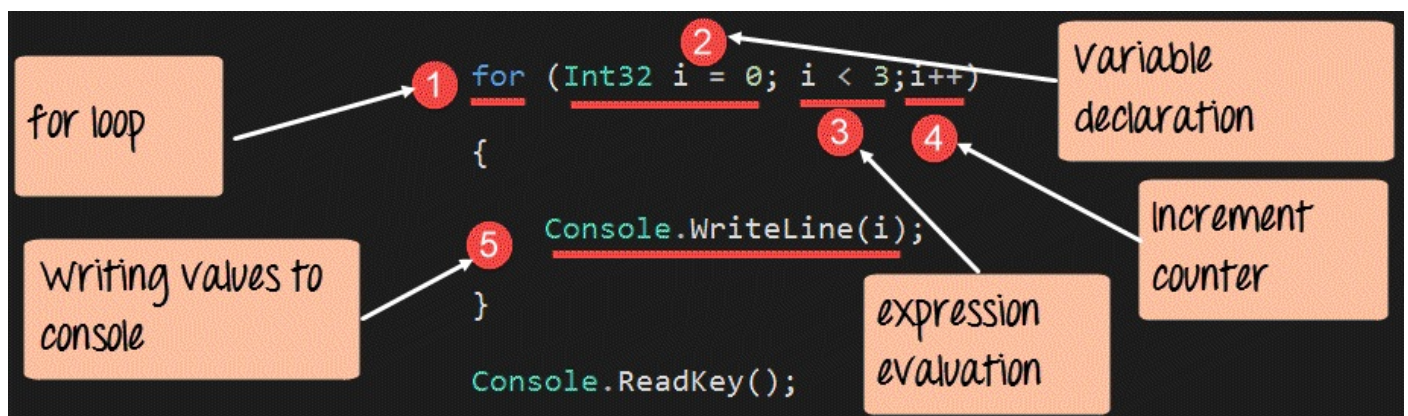
Output:



Here you can see that the while statement is executed 3 times and incremented at the same time. And each time, it displayed the current value of the variable 'i'.

- 4. **For loop** - The 'for' loop is also used for iterative purposes. Suppose if you want to repeat a certain set of statements for a particular number of times, then for loop is used.

In our example below, we use the 'for' statement to display the value of a variable 'i'. The 'for' statement is used to display the value 3 times.



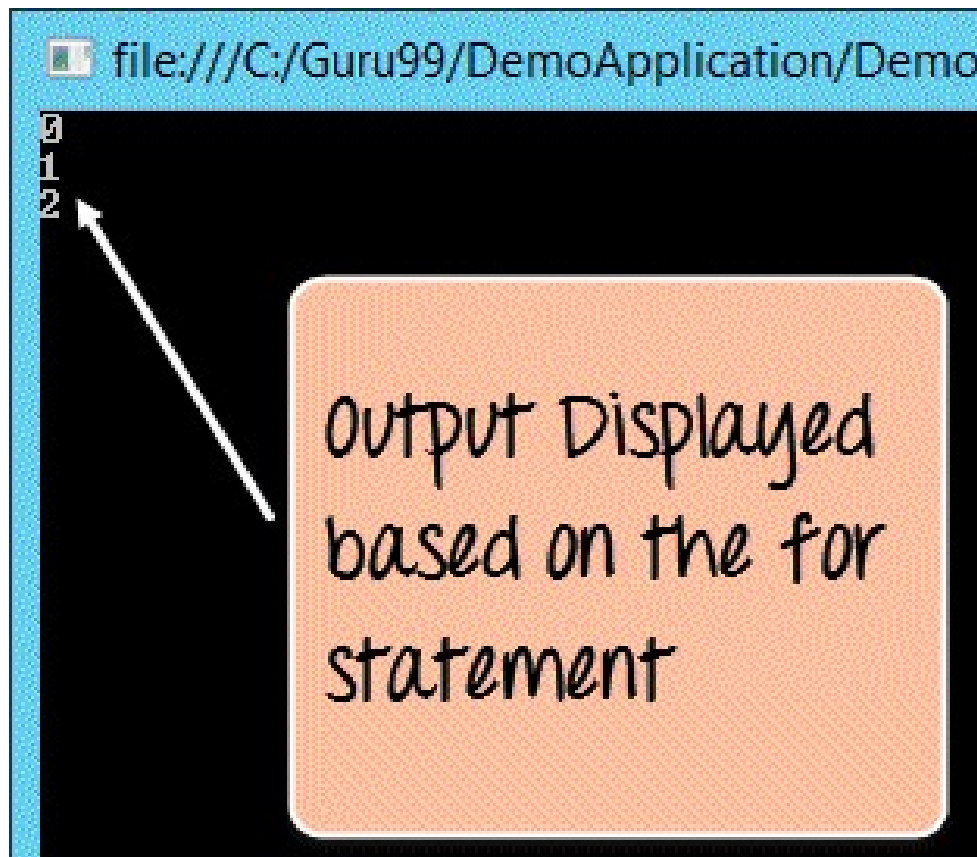
Code Explanation:-

- 1. The 'for' keyword is used to start off the 'for loop' statement.

2. In the 'for loop', we define 3 things. The first is to initialize the value of a variable, which will be used in the 'for loop'.
3. The second is to compare the value of the 'i' against an upper limit. In our case, the upper limit is the value of 3 ($i < 3$).
4. Finally, we increment the value of 'i' accordingly.
5. Here we display the value of 'i' to the console.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/Demo...
1
1
2
```

Output Displayed
based on the for
statement

Here you can see that the 'for' statement is executed 3 times. And each time, it displayed the current value of the variable 'i'.

C# Arrays

An array is used to store a collection or series of elements. These elements will be of the same type.

So for example, if you had an array of Integer values, the array could be a collection of values such as [1, 2, 3, 4]. Here the number of elements in the array is 4.

Arrays are useful when you want to store a collection of values of the same type. So instead of declaring a variable for each and every element, you can just declare one variable.

This variable will point to an array or list of elements, which will be responsible for storing the elements of the array.

Let's look at how we can work with arrays in C#. In our example, we will declare an array of Integers and work with them accordingly.

Note that all of the below code is being made to the Program.cs file.

Step 1) Declaring an array – The first step is to declare an array. Let's see how we can achieve this by the below code example.

```
namespace DemoApplication {
    class Program
    {
        static void Main(string[] args)
        {
            Int32 [  ] values;
        }
    }
}
```

The diagram illustrates the components of the array declaration `Int32[] values;`. Three red circles with numbers 1, 2, and 3 are placed below the code. Three orange boxes with arrows point to these circles: 'Data type' points to circle 1, 'Shows this is an array' points to circle 2, and 'Variable name' points to circle 3.

Code Explanation:-

1. The first part is the datatype. It specifies the type of elements used in the array. So in our case, we are creating an array of Integers.
2. The second part [], which specifies the rank of the array. (The rank is a placeholder which specifies the number of elements the array will contain)

3. Next is the Name of the array which in our case is 'values'. Note you are seeing a green squiggly underline, don't worry about that. That is just .Net saying that you have declared an array, but not using it anywhere.

Step 2) The next step is to initialize the array. Here we are going to specify the number of values the array will hold. We are also going to assign values to each element of the array.

```
static void Main(string[] args)
{
    Int32[] values;
    values = new Int32[3];
    values[0] = 1;
    values[1] = 2;
    values[2] = 3;
}
```

The diagram shows the code with two callouts. Callout 1 points to the number 3 in the array declaration, stating 'Array is going to hold 3 elements'. Callout 2 points to the three lines of element assignment, stating 'Assigning values to the elements of the array'.

Code Explanation:-

1. First, we are setting the number of elements the array will hold to 3. So in the square brackets, we are saying that the array will hold 3 elements.
2. Then we are assigning values to each individual element of the array. We can do this by specifying the variable name + the index position in the array.

So values[0] means that we are storing a value in the first position of the array. Similarly to access the second position, we use the notation of values[1] and so on and so forth.

Note: - In Arrays, the index position starts from .

Step 3) Let's now display the individual elements of the array in the Console. Let's add the below code to achieve this.

```
values = new Int32[3];
```

```
values[0] = 1;
```

```
values[1] = 2;
```

```
values[2] = 3;
```

```
Console.WriteLine(values[0]);
```

```
Console.WriteLine(values[1]);
```

```
Console.WriteLine(values[2]);
```

```
Console.ReadKey();
```

Using
Console.WriteLine
to send each
element value to
the console

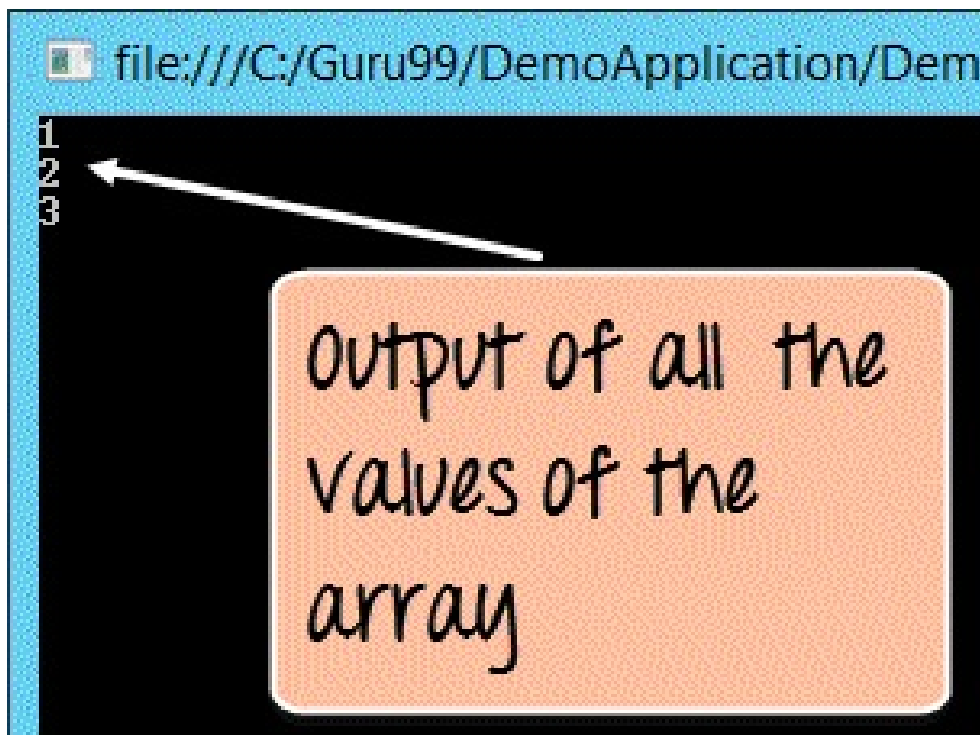
Code Explanation:-

This is the simple part wherein we just use the Console.Write method to send each value of the element to the console.

Note that again , we are accessing each element with the help of the array variable name along with the index position.

If the above code is entered properly and the program is executed, the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/Dem  
1  
2  
3
```

Output of all the
values of the
array

From the output, you can see all the values of the array being displayed in the Console.

Summary

- A Console application is one that can be made to run in the command prompt on a windows machine.
- The Console.write method can be used to write content to the console.
- The basic data types available in C# are Integer, Double, Boolean, and String.
- Enumerations are used to declare a set of Constant values. In C# enumerations are declared with the use of the enum keyword.
- The various operators available in C# are broadly classified into the categories of Arithmetic, Relational, and Logical operators.
- Variables are used to point to memory locations which contain values of a particular data type.
- Arrays are used to store elements of the same type. Individual elements of the array can be assigned values.

Chapter 4: Class and Object

C# is based on the C++ programming language. Hence, the C# programming language has in-built support for classes and objects. Class is nothing but an encapsulation of properties and methods that are used to represent a real-time entity.

For an example, if you want to work with employee's data in a particular application.

The properties of the employee would be the ID and name of the employee. The methods would include the entry and modification of employee data.

All of these operations can be represented as a class in C#. In this chapter, we will look at how we can work with classes and objects in C# in more detail.

What are classes and objects

Let's first begin with classes.

As we discussed earlier classes are an encapsulation of **data properties** and **data methods**.

- The properties are used to describe the data the class will be holding.
- The methods tells what are the operations that can be performed on the data.

To get a better understanding of class and objects, let's look at an example below of how a class would look like.

The name of the class is "Tutorial". The class has the following properties

1. **Tutorial ID** – This will be used to store a unique number which would represent the Tutorial.
2. **Tutorial Name** – This will be used to store the name of the tutorial as a string.

A class also comprises of methods. Our class has the following methods,

1. **SetTutorial** – This method would be used to set the ID and name of the Tutorial. So for example, if we wanted to create a tutorial for .Net, we might create an object for this. The object would have an ID of let's say 1. Secondly, we would assign a name of ".Net" as the name of the Tutorial. The ID value of 1 and the name of ".Net" would be stored as a property of the object.
2. **GetTutorial** - This method would be used to get the details of a specific tutorial. So if we wanted to get the name of the Tutorial , this method would return the string ".Net".

Tutorial Class

- Properties
 - TutorialID
 - TutorialName
- Methods
 - SetTutorial
 - GetTutorial

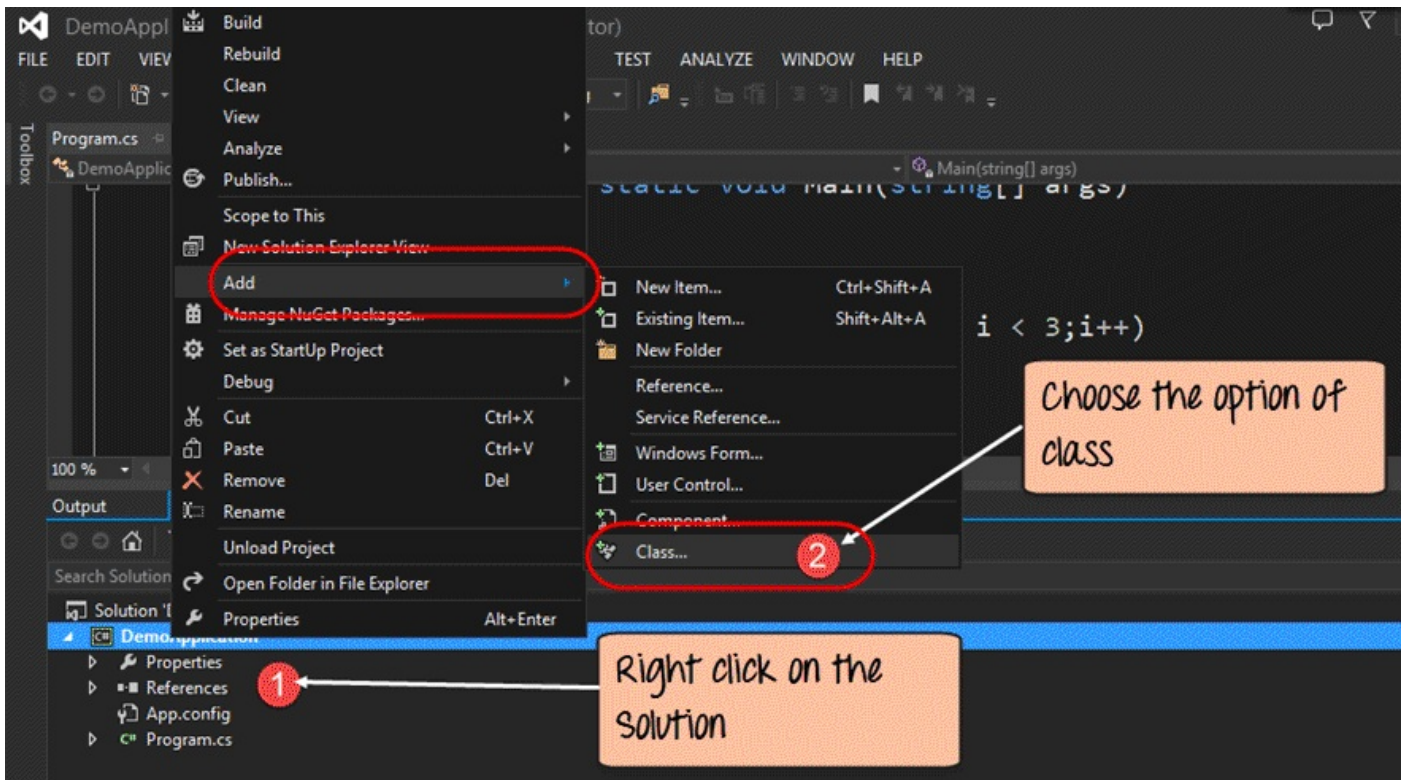
Below is a snapshot of how an object might look like for our Tutorial class. We have 3 objects, each with their own respective TutorialID and TutorialName.

TutorialObject1	TutorialObject2	TutorialObject3
<ul style="list-style-type: none">• TutorialID =1• TutorialName = C#	<ul style="list-style-type: none">• TutorialID =2• TutorialName = ASP.Net	<ul style="list-style-type: none">• TutorialID =3• TutorialName = VB.Net

Let's now dive into Visual Studio to create our class. We are going to build upon our existing console application which was created in our earlier chapter. We will create a class in Visual Studio for our current application.

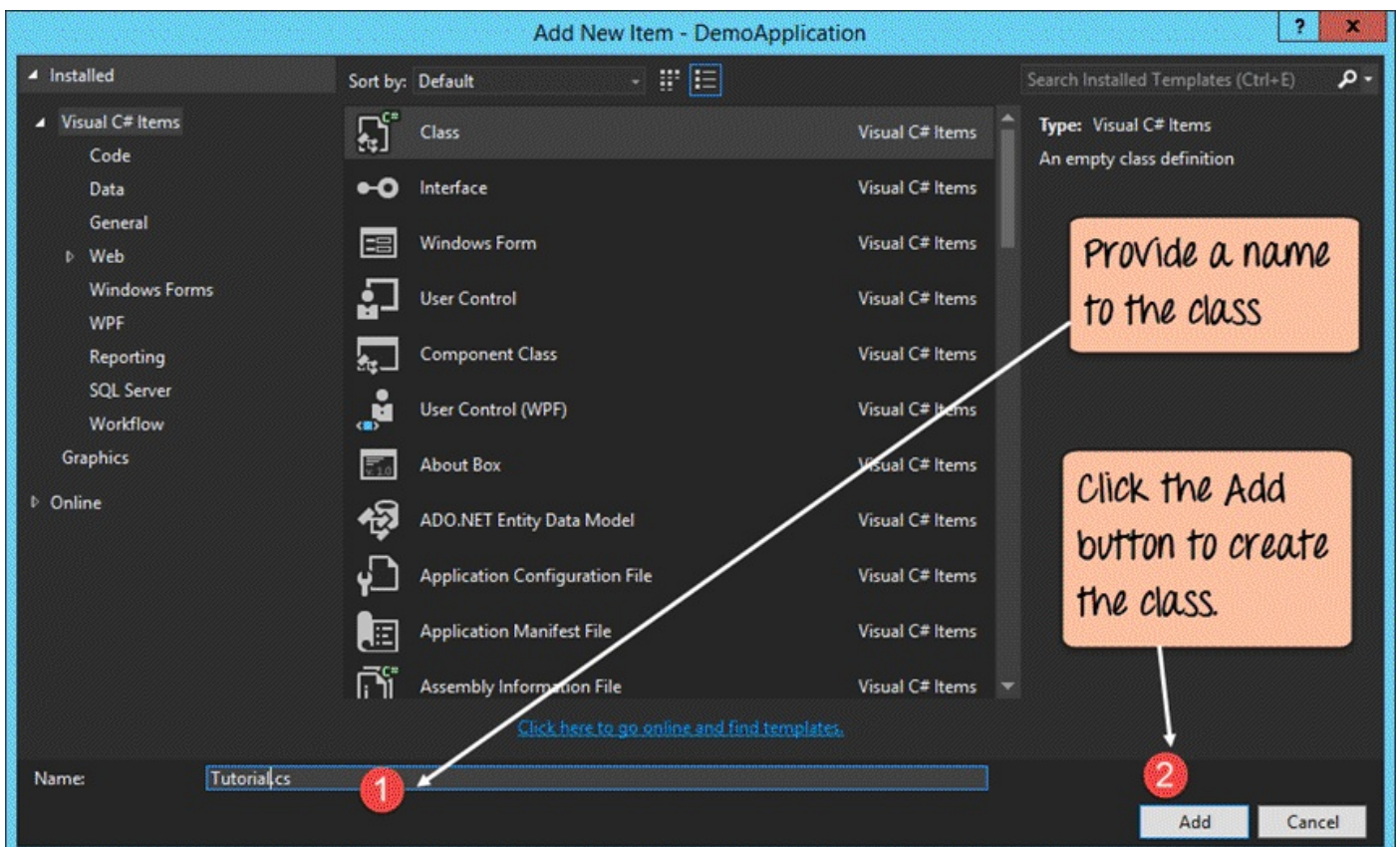
Let's follow the below-mentioned steps to get this example in place.

Step 1) The first step involves the creation of a new class within our existing application. This is done with the help of Visual Studio.



1. The first step is to right click on the solution, which in our case is 'DemoApplication'. This will bring up a context menu with a list of options.
2. From the context menu choose the option Add->Class. This will provide the option to add a class to the existing project.

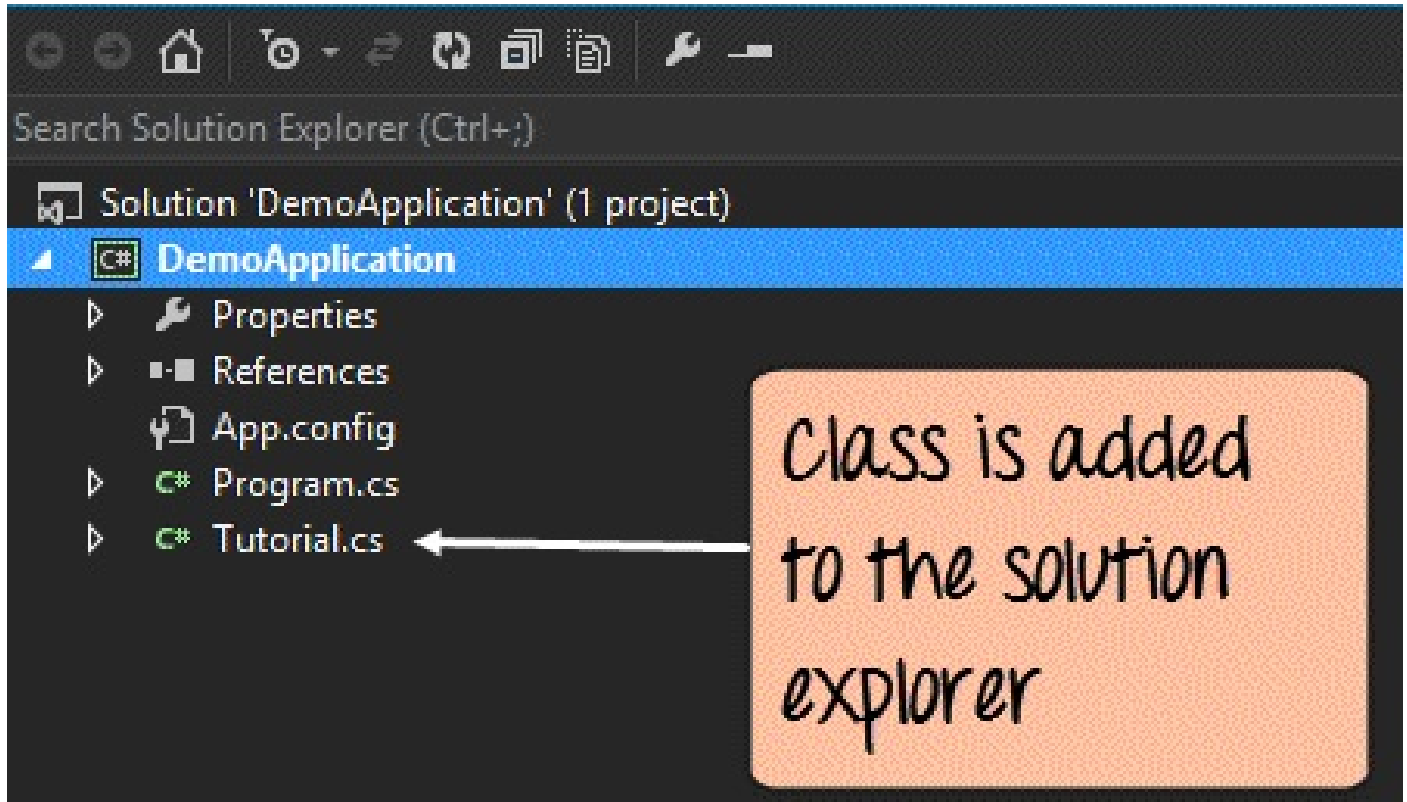
Step 2) The next step is to provide a name for the class and add it to our solution.



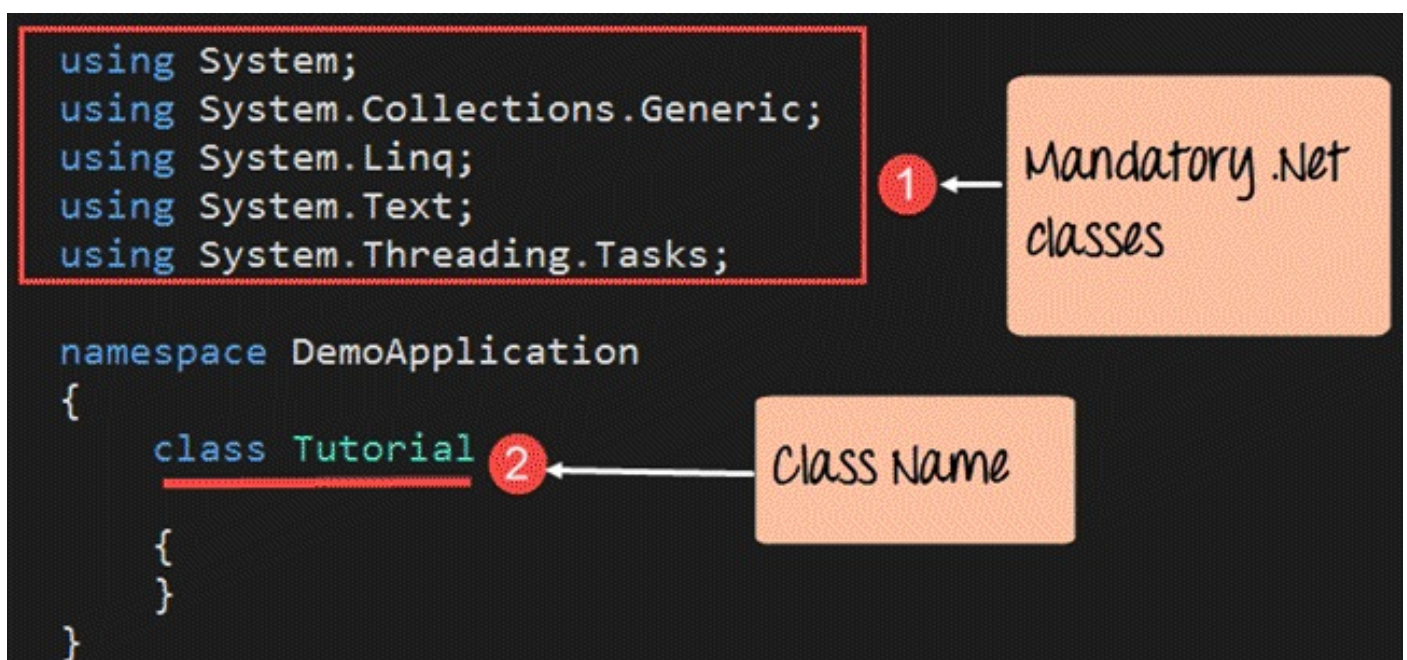
1. In the project dialog box, we first need to provide a name for our class. Let's provide a name of Tutorial.cs for our class. Note that the file name should end with .cs to ensure it is treated as a proper class file.
2. When we click the Add button, the class will be added to our solution.

If the above steps are followed, you will get the below output in Visual Studio.

Output:-



A class named Tutorial.cs will be added to the solution. If you open the file, you will find the below code added to the class file.



Code Explanation:-

1. The first part contains the mandatory modules which Visual Studio adds to any .Net file. These modules are always required to ensure any .Net program runs on a Windows environment.
2. The second part is the class which is added to the file. The class name is 'Tutorial' in our case. This is the name which was specified with the class was added to the solution.

For the moment, our class file does not do anything. In the following topics, we will look into more details on how to work with the class.

Fields and methods

We have already seen how fields and methods are defined in classes in the earlier topic.

For our Tutorial class, we can have the following properties.

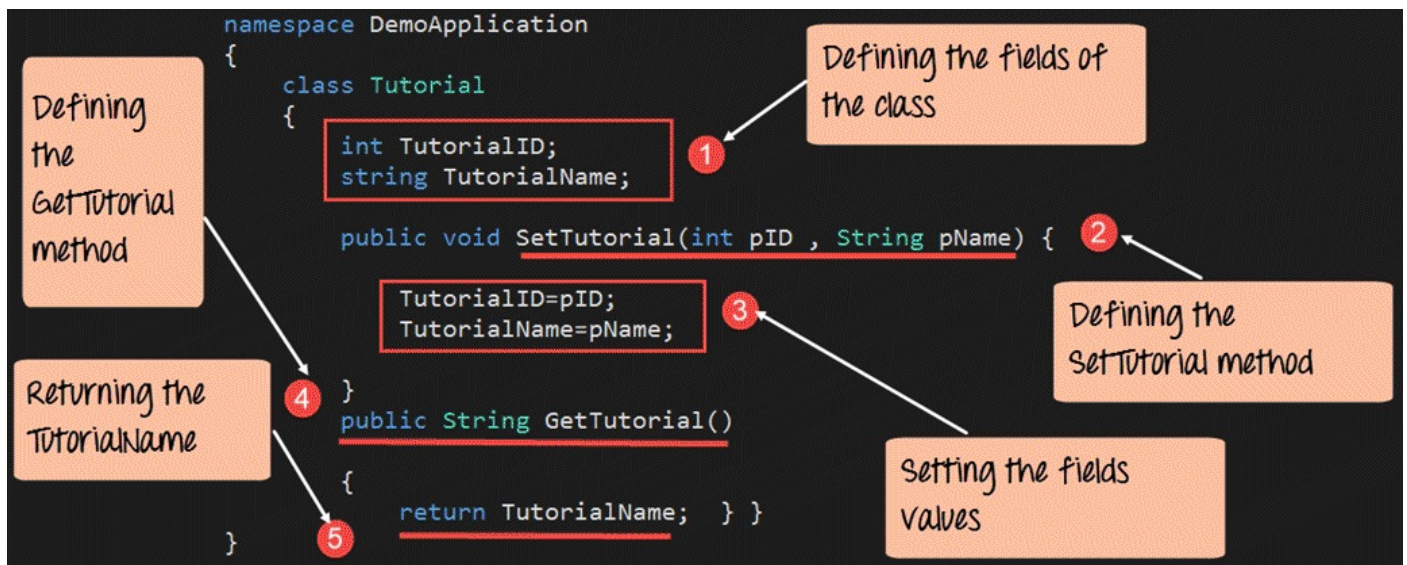
1. Tutorial ID – This will be used to store a unique number which would represent the Tutorial.
2. Tutorial Name – This will be used to store the name of the tutorial as a string.

Our Tutorial class can also have the below-mentioned methods.

1. SetTutorial – This method would be used to set the ID and name of the Tutorial.
2. GetTutorial - This method would be used to get the details of a specific tutorial.

Let's now see how we can incorporate fields and methods in our code.

Step 1) The first step is to ensure the Tutorial class has the right fields and methods defined. In this step, we add the below code to the Tutorial.cs file.



Code Explanation:-

1. The first step is to add the fields of TutorialID and TutorialName to the class file. Since the TutorialID field will be a number, we define it as an Integer, while TutorialName will be defined as string.
2. Next, we define the SetTutorial method. This method accepts 2 parameters. So if Program.cs calls the SetTutorial method, it would need to provide the values to these parameters. These values will be used to set the fields of the

Tutorial object.

Note:-Let's take an example and assume our Program.cs file calls the SetTutorial with the parameters "1" and ".Net". The below steps would be executed as a result of this,

- a. The value of pID would become 1
 - b. The value of pName would be .Net.
 - c. In the SetTutorial method, these values would then be passed to TutorialID and TutorialName.
 - d. So now TutorialID would have a value of 1 and TutorialName would have a value of ".Net".
3. Here we set the fields of the Tutorial class to the parameters accordingly. So we set TutorialID to pID and TutorialName to Pname.
 4. We then define the GetTutorial method to return value of the type "String". This method will be used to return the TutorialName to the calling program. Likewise, you can also get the tutorial id with method Int GetTutorial
 5. Here we return the value of the TutorialName field to the calling program.

Step 2) Now let's add code to our Program.cs, which is our Console application. The Console application will be used to create an object of the "Tutorial class" and call the SetTutorial and GetTutorial methods accordingly.

(**Note:-** An object is an instance of a class at any given time. The difference between a class and an object is that the object actually contains values for the properties.)

```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial();
    pTutor.SetTutorial(1, ".Net");
    Console.WriteLine(pTutor.GetTutorial());
    Console.ReadKey();
}
```

The diagram illustrates the execution of the provided C# code. Three callout boxes with arrows point to specific lines of code:

- Callout 1: "Creating a new Tutorial object" points to the line `Tutorial pTutor = new Tutorial();`.
- Callout 2: "Calling the SetTutorial method" points to the line `pTutor.SetTutorial(1, ".Net");`.
- Callout 3: "Calling the GetTutorial method" points to the line `Console.WriteLine(pTutor.GetTutorial());`.

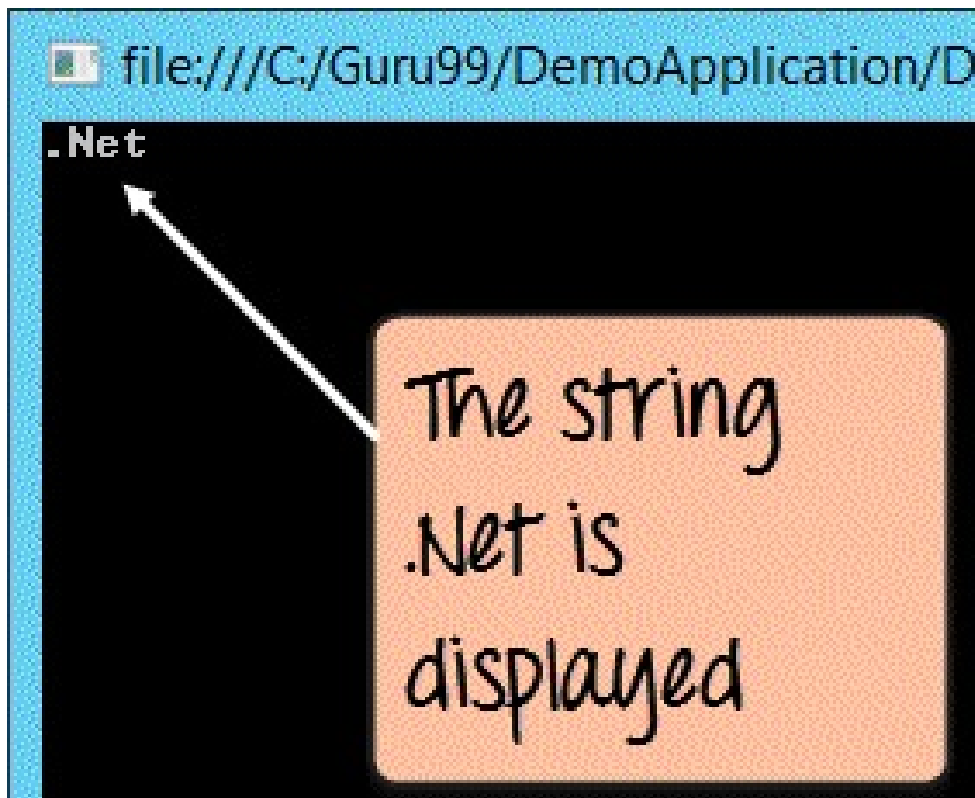
Code Explanation:-

1. The first step is to create an object for the Tutorial class. Mark here that this is done by using the keyword 'new'. The 'new' keyword is used to create an object from a class in C#. The object is then assigned to the pTutor variable.

2. The method SetTutorial is then called. The parameters of 1 and “.Net” are passed to the SetTutorial method. These will then be used to set the “TutorialID” and “TutorialName” fields of the class accordingly.
3. We then use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, you can clearly see that the string “.Net” was returned by the GetTutorial method.

Access Modifiers

Access Modifiers are used to define the visibility of a class property or method. There are times when you may not want other programs to see the properties or the methods of class. In such cases, C# gives the ability to put modifiers on class properties and methods. The class modifiers have the ability to restrict access so that other programs cannot see the properties or methods of a class.

There are generally 3 types of access modifiers. They are explained below.

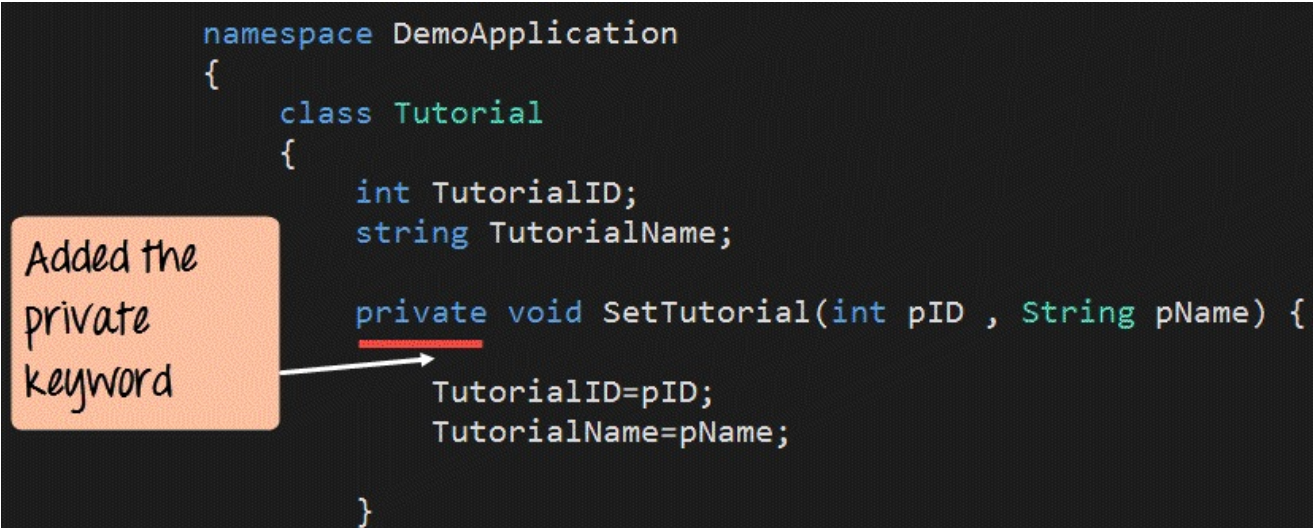
1. **Private** – When this access modifier is attached to either a property or a method, it means that those members cannot be accessed from any external program.

Let's take an example and see what happens when we use the private access modifier.

Let's modify the current code in our Tutorial.cs file. In the SetTutorial method, let's change the public keyword to private.

```
namespace DemoApplication
{
    class Tutorial
    {
        int TutorialID;
        string TutorialName;

        private void SetTutorial(int pID , String pName) {
            TutorialID=pID;
            TutorialName=pName;
        }
    }
}
```

A screenshot of a code editor showing C# code. The code defines a namespace 'DemoApplication' containing a class 'Tutorial'. Inside the class, there are two integer and string properties, and a 'SetTutorial' method. The 'private' keyword before the method signature is underlined with a red squiggly line. An orange callout box on the left contains the handwritten text 'Added the private keyword' with an arrow pointing to the underlined 'private' keyword.

Now let's switchover to our Program.cs file. You will notice that there is a red squiggly line under the SetTutorial method.

Since we have now declared the SetTutorial method as private in our Tutorial class, Visual Studio has detected this. It has told the user by highlighting it that now this method will not work from the Program.cs file.

```
{
    static void Main(string[] args)
    {
        Tutorial pTutor = new Tutorial();
        pTutor.SetTutorial(1, ".Net");
        Console.WriteLine(pTutor.GetTutorial());
        Console.ReadKey();
    }
}
```

Error in the SetTutorial method

2. **Public** – When this access modifier is attached to either a property or a method, it means that those members can be accessed from any external program. We have already seen this in our earlier examples.

```
int TutorialID;
string TutorialName;

public void SetTutorial(int pID , String pName) {
    TutorialID=pID;
    TutorialName=pName;
}
public String GetTutorial()
{
```

Public access modifier

Since we have defined our methods as public in the Tutorial class, they have the ability to be accessed from the Program.cs file.

3. **Protected** - When this access modifier is attached to either a property or a method, it means that those members can be accessed only by classes inherited from the current class. This will be explained in more detail in the Inheritance class.

C# Constructor

Constructors are used to initialize the values of class fields when their corresponding objects are created. A constructor is a method which has the same name as that of the class. If a constructor is defined in class, then it will be the first method which is called when an object is created. Suppose if we had a class called Employee. The constructor method would also be named as Employee().

The following key things need to be noted about constructor methods

1. The access modifier for the constructor needs to be made as public.
2. There should be no return type for the constructor method.

Let's now see how we can incorporate the use of constructors in our code. We will use constructors to initialize the TutorialID and TutorialName fields to some default values when the object is created.

Step 1) The first step is to create the constructor for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.

```
namespace DemoApplication
{
    public class Tutorial
    {
        int TutorialID;
        string TutorialName;
        public Tutorial() {
            TutorialID = 0;
            TutorialName = "Default";
        }
        public void SetTutorial(int pID , String pName) {
            TutorialID=pID;
            TutorialName=pName;
        }
    }
}
```

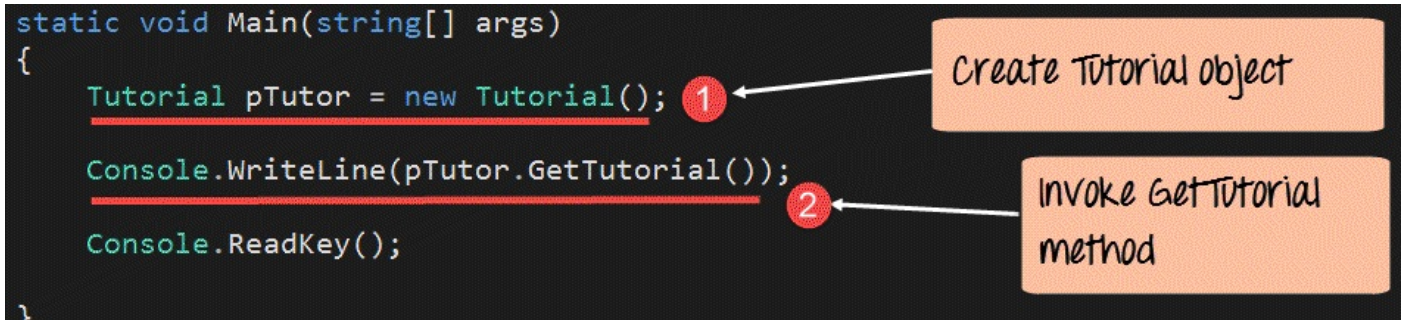
Code Explanation:-

1. We first add a new method which has the same name as that of the class. Because it is the same name as the class, C# treats this as a constructor method. So now whenever the calling method creates an object of this class, this method will be called by default.

2. In the Tutorial constructor, we are setting the value of TutorialID to 0 and TutorialName to "Default". So whenever an object is created, these fields will always have these default values.

Now let's switchover to our Program.cs file and just remove the line, which calls the SetTutorial method. This is because we want to just see how the constructor works.

```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial(); 1
    Console.WriteLine(pTutor.GetTutorial()); 2
    Console.ReadKey();
}
```

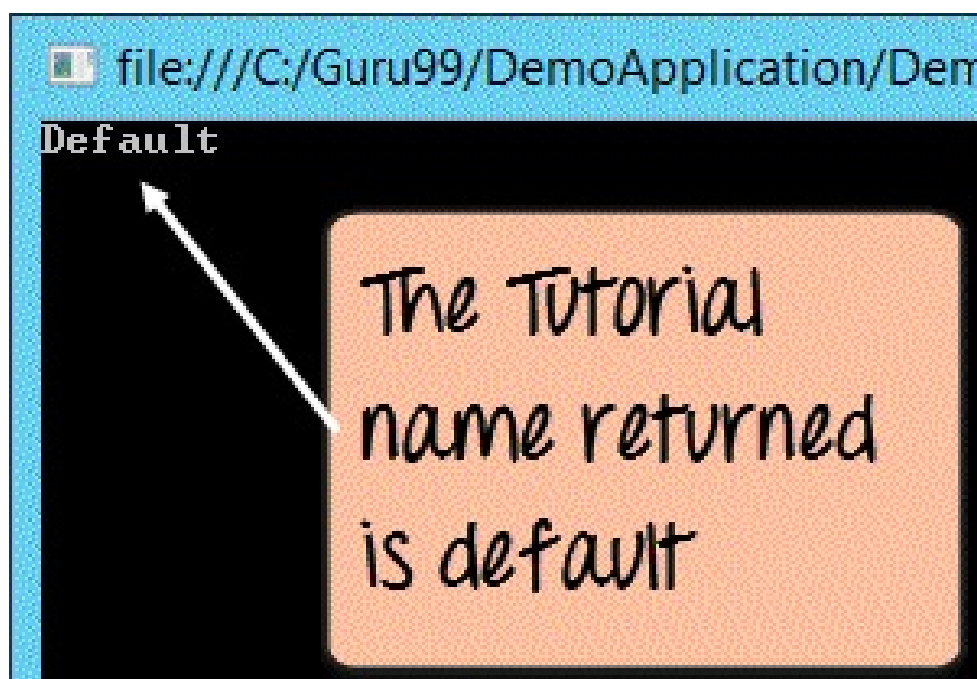


Code Explanation:-

1. The first step is to create an object for the Tutorial class. This is done via the 'new' keyword.
2. We use the GetTutorial method of the Tutorial class to get the TutorialName. This is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is executed, the following output will be displayed.

Output:



From the output, we can see that the constructor was indeed called, and that the value of the TutorialName was set to “Default”.

Note: Here the value “default” is fetched from the constructor.

C# Inheritance

Inheritance is an important concept in C#. Inheritance is a concept in which you define parent classes and child classes.

The child classes inherits methods and properties of the parent class, but at the same time, they can also modify the behavior of the methods if required. The child class can also define methods of its own if required.

You will get a better understanding if we see this action.

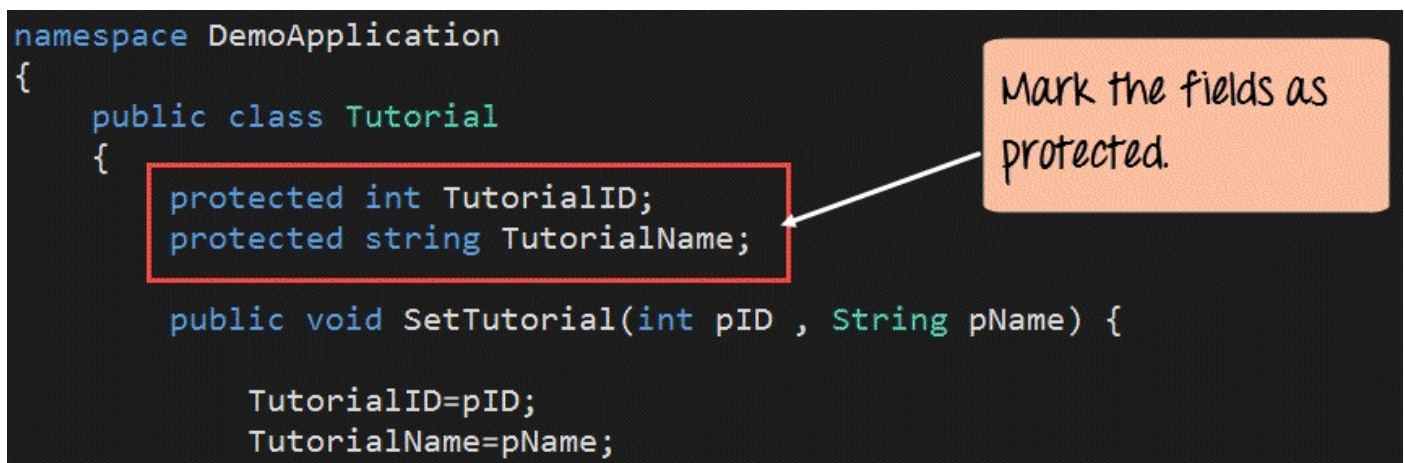
Let's now see how we can incorporate the concept of inheritance in our code.

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.

```
namespace DemoApplication
{
    public class Tutorial
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID , String pName) {

            TutorialID=pID;
            TutorialName=pName;
        }
    }
}
```

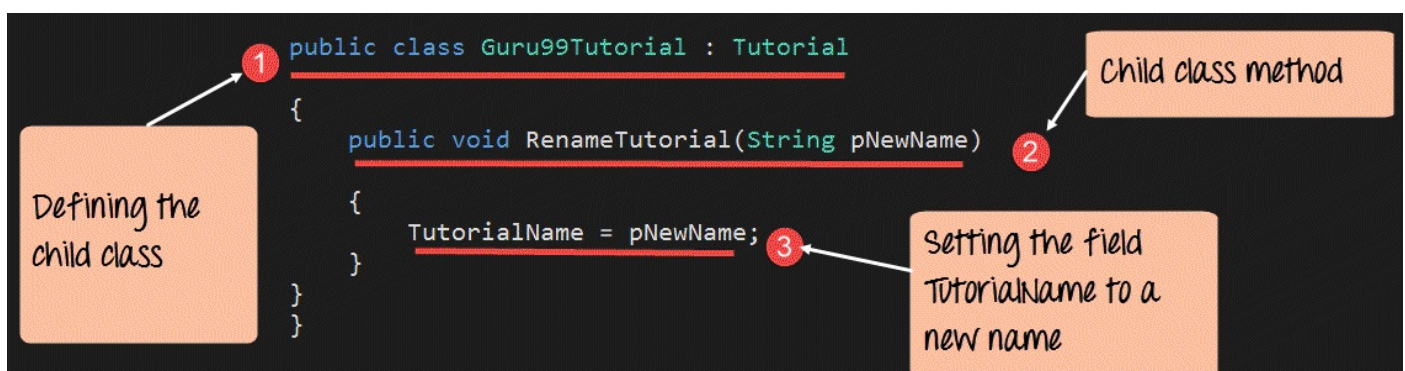


Note that we need to now add the access modifier of 'protected' to both the TutorialID and TutorialName field.

Remember we had mentioned this access modifier in the Access Modifier chapter. Well here you can see the purpose of having this. Only when you have this access modifier (protected), the child class be able to use the fields of the parent class.

Step 2) The second step is to add our new child class. The name of this class will be "Guru99Tutorial". In this step, we add the below code to the Tutorial.cs file. The code should be placed after the Tutorial class definition.

```
public class Guru99Tutorial : Tutorial
{
    public void RenameTutorial(String pNewName)
    {
        TutorialName = pNewName;
    }
}
```



Code Explanation:-

1. The first step is to create the Guru99Tutorial child class. We also need to mention that this class is going to be a child class of the Tutorial class. This is done by the ':' keyword.
2. Next, we are defining a method called RenameTutorial. It will be used to rename the TutorialName field. This method accepts a string variable which contains the new name of the Tutorial.
3. We then assigned the parameter pNewName to the TutorialName field.

Note: - Even though we have not defined the TutorialName field in the "Guru99Tutorial" class, we are still able to access this field. This is because of the fact that "Guru99Tutorial" is a child class of Tutorial class. And because we made the fields of the Tutorial class as protected, they can be accessed by this class.

Step 3) The last step is to modify our main Program.cs file. In our console application, we are going to make an object of the Guru99Tutorial class. With this object, we are going to call the RenameTutorial method. We are then going to display the TutorialName field with the help of the GetTutorial method.

```
class Program
{
    static void Main(string[] args)
    {
        Guru99Tutorial pTutor = new Guru99Tutorial();
        pTutor.RenameTutorial(".Net by Guru99");
        Console.WriteLine(pTutor.GetTutorial(););
        Console.ReadKey();
    }
}
```

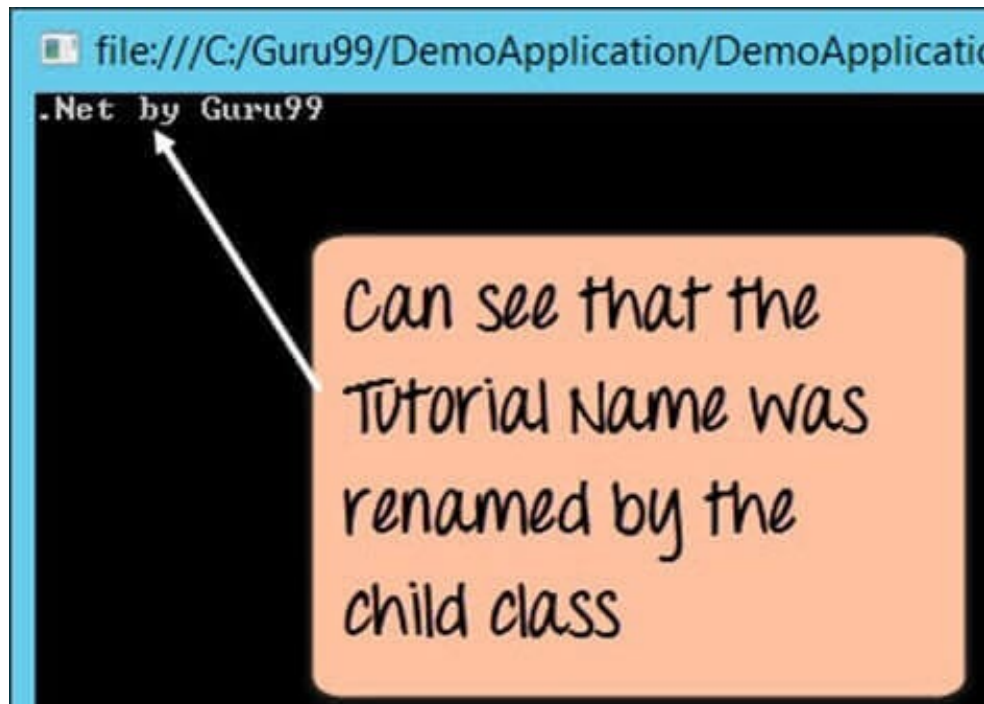
The diagram shows three callouts pointing to specific lines of code in the Program class. Callout 1 points to the line `Guru99Tutorial pTutor = new Guru99Tutorial();`. Callout 2 points to the line `pTutor.RenameTutorial(".Net by Guru99");`. Callout 3 points to the line `Console.WriteLine(pTutor.GetTutorial());`. The code is displayed on a dark background with red underlines under the three lines mentioned in the callouts.

Code Explanation:-

1. The first step is to create an object for the Guru99Tutorial class. This is done via the 'new' keyword. Note that this time we are not creating an object of the Tutorial class.
2. We use the RenameTutorial method of the Guru99Tutorial class to change the TutorialName field. We pass the string ".Net by Guru99" to the RenameTutorial method.
3. We then call the GetTutorial method. Note that even though this method is not defined in the Guru99Tutorial class, we are still able to access this method. The output of the GetTutorial method is then displayed to the console via the Console.WriteLine method.

If the above code is entered properly and the program is executed successfully, the following output will be displayed.

Output:



From the output, we can clearly see that the TutorialName field was renamed to “.Net by Guru99”. This was made possible of the RenameTutorial method called by the child class.

C# Polymorphism

Polymorphism is a concept wherein a method can be defined more than one time. But each time, the function would have a different set of parameters passed on to it.

You will get a better understanding if we see this action.

Let's now see, how we can incorporate the concept of Polymorphism in our code.

Step 1) The first step is to change the code for our Tutorial class. In this step, we add the below code to the Tutorial.cs file.

```
public void SetTutorial(int pID, String pName)
{
    TutorialID = pID;
    TutorialName = pName;
}

public void SetTutorial(String pName)
{
    TutorialName = pName;
}
```

The image shows a code editor with two methods. The first method, `SetTutorial(int pID, String pName)`, is annotated with a red circle '1' pointing to its signature and a red circle '3' pointing to the assignment lines. A callout box next to it says "Setting both the TutorialID and TutorialName". The second method, `SetTutorial(String pName)`, is annotated with a red circle '2' pointing to its signature and a red circle '4' pointing to the assignment line. A callout box next to it says "Setting just the TutorialName". A larger callout box on the right says "Both methods have the same name. They only have different parameters passed to them."

Code Explanation:-

1 & 2) The first step is the same as in our earlier examples. We are keeping the definition of the SetTutorial method as it is.

3) This method sets the TutorialID and the TutorialName based on the parameters pID and pName.

4) This is where we make a change to our class wherein we add a new method with the same name of SetTutorial. Only this time we are only passing one parameter which is the pName. In this method, we are just setting the field of TutorialName to pName.

Step 2) The last step is to modify our main Program.cs file. In our console application, we are going to make an object of the Guru99Tutorial class.

```
static void Main(string[] args)
{
    Tutorial pTutor = new Tutorial();

    pTutor.SetTutorial(1, "First Tutorial");
    Console.WriteLine(pTutor.GetTutorial());

    pTutor.SetTutorial("Second Tutorial");
    Console.WriteLine(pTutor.GetTutorial());

    Console.ReadKey();
}
```

Call the first method

Call the second method

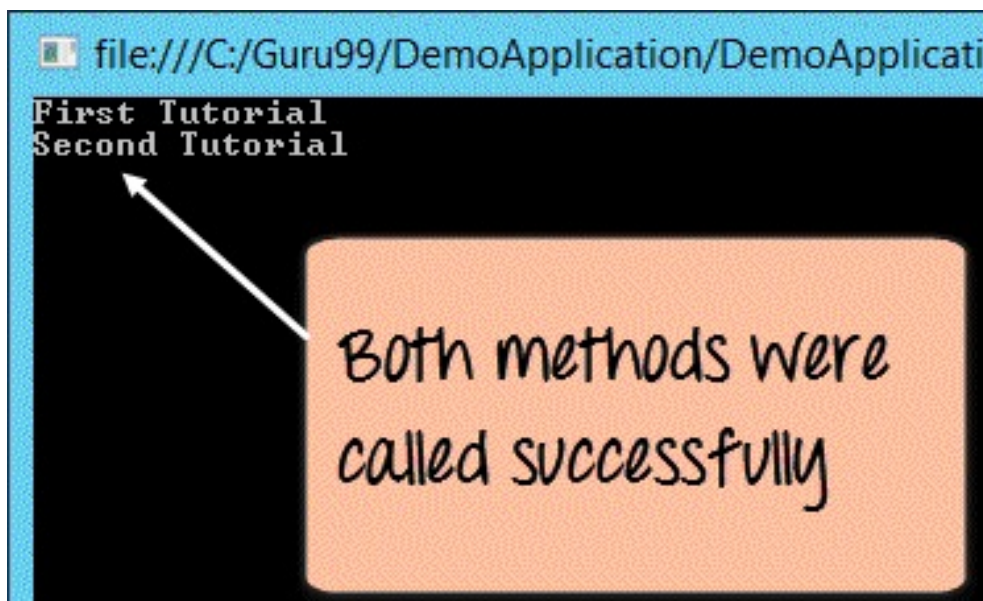
Code Explanation:-

1. In the first step, we are using the SetTutorial method with 2 parameters. Where we are passing both the TutorialID and TutorialName to this method.
2. In the second step, we are now calling the SetTutorial method with just one parameter. We are just passing the TutorialName to this method.

If the above code is entered properly and the program is run the following output will be displayed. If in case you wanted to also fetch the Tutorial ID along with the Tutorial Name , you should follow the below step

1. Create a separate method called public int GetTutorialID
2. In that method write the code line "return TutorialID." This can be used to return the TutorialID to the calling program.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplicati
First Tutorial
Second Tutorial
```

Both methods were called successfully

From the output, we can clearly see that both methods were called successfully. Because of this, the strings “First Tutorial” and “Second Tutorial” were sent to the console.

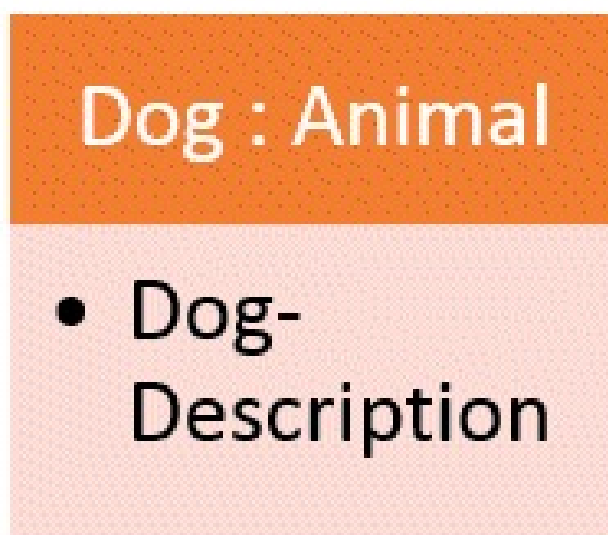
C# Abstract classes

An abstract class is used to define what is known as a base class. A base class is a class which has the most basic definition of a particular requirement.

A typical example of an abstract class is given below. Below is the definition of a class called 'Animal.' When the 'Animal' class is defined, there is nothing known about the animal, whether it is a dog or a cat. The method called description is just a generic method defined for the class.



Now when it is known what exactly the Animal is going to be, we create another class which inherits the base class. If we know that the animal is in fact a Dog, we create Dog class which inherits the main base class. The key difference here is that the Dog class cannot change the definition of the Description method of the Animal class. It has to define its own method called Dog-Description. This is the basic concept of abstract classes.

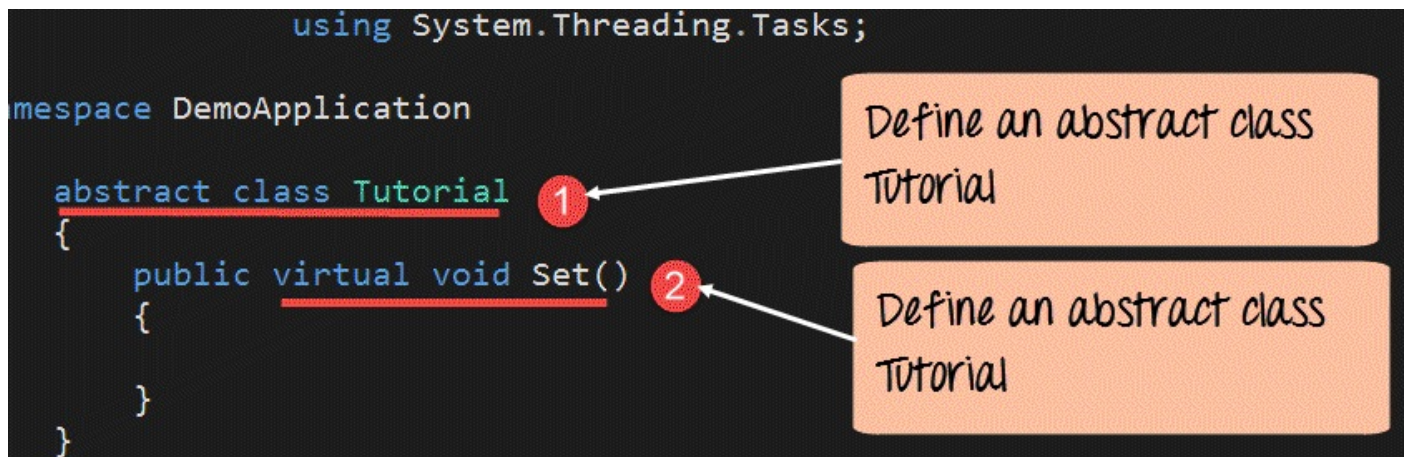


Let's see how we can change our code to include an abstract class. Note that we will not be running the code, because there is nothing that can be run using an abstract class.

Step 1) As a first step, let's create an abstract class. The class will be called Tutorial and will just have one method. All the code needs to be written in the Program.cs file.

```
using System.Threading.Tasks;

namespace DemoApplication
{
    abstract class Tutorial 1
    {
        public virtual void Set() 2
        {
        }
    }
}
```



Code Explanation:-

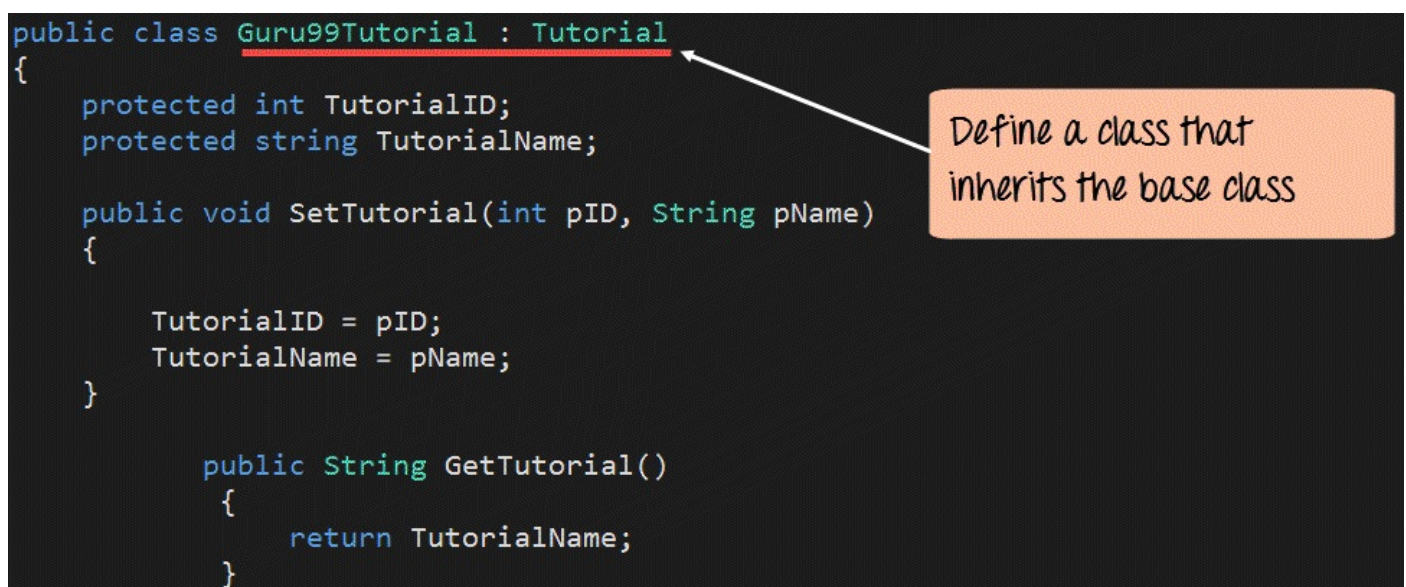
1. We first define the abstract class. Note the use of the abstract keyword. This is used to denote that the class is an abstract class.
2. Next, we are defining our method which does nothing. The method must have the keyword called virtual. This means that the method cannot be changed by the child class. This is a basic requirement for any abstract class.

Step 2) Now let's add our child class. This code is added to the Program.cs file.

```
public class Guru99Tutorial : Tutorial
{
    protected int TutorialID;
    protected string TutorialName;

    public void SetTutorial(int pID, String pName)
    {
        TutorialID = pID;
        TutorialName = pName;
    }

    public String GetTutorial()
    {
        return TutorialName;
    }
}
```



There is nothing exceptional about this code. We just define a class called 'Guru99Tutorial' which inherits the abstract Tutorial class. We then define the

same methods as we have been using from before.

Note: Here we cannot change the definition of the Set method which was defined in the Tutorial class. In the Tutorial class, we had defined a method called 'Set' (public virtual void Set()). Since the method was part of the abstract class, we are not allowed to define the Set method again in the Guru99Tutorial class.

C# Interface

Interfaces are used along with classes to define what is known as a contract. A contract is an agreement on what the class will provide to an application.

An interface declares the properties and methods. It is up to the class to define exactly what the method will do.

Let's look at an example of an interface by changing the classes in our Console application. Note that we will not be running the code, because there is nothing that can be run using an interface.

Let's create an interface class. The class will be called "Guru99Interface." Our main class will then extend the defined interface. All the code needs to be written in the Program.cs file.

```
namespace DemoApplication
{
    public interface Guru99Interface {
        public void SetTutorial(int pID, String pName);
        public String GetTutorial();
    }
    public class Guru99Tutorial : Guru99Interface
    {
        protected int TutorialID;
        protected string TutorialName;

        public void SetTutorial(int pID, String pName)
        {
            TutorialID = pID;
            TutorialName = pName;
        }
    }
}
```

The diagram illustrates the C# code for defining an interface and a class that extends it. It includes three callouts:

- Define the public interface:** Points to the `public interface Guru99Interface` declaration.
- Define the methods:** Points to the method declarations `public void SetTutorial(int pID, String pName);` and `public String GetTutorial();` inside the interface.
- Class extends the interface:** Points to the `public class Guru99Tutorial : Guru99Interface` declaration.

Code Explanation:-

1. We first define an interface called "Guru99Interface." Note that the keyword "interface" is used to define an interface.
2. Next, we are defining the methods that will be used by our interface. In this case, we are defining the same methods which are used in all of earlier examples. Note that an interface just declares the methods. It does not define the code in them.
3. We then make our Guru99Tutorial class extend the interface. Here is where we actually write the code that defines the various methods declared in the interface. This sort of coding achieves the following

- It ensures that the class, Guru99Tutorial, only adds the code which is necessary for the methods of “SetTutorial” and “GetTutorial” and nothing else.
- It also ensures that the interface behaves like a contract. The class has to abide by the contract. So if the contract says that it should have 2 methods called “SetTutorial” and “GetTutorial,” then that is how it should be.

Summary

- The class is an encapsulation of data properties and methods. The properties are used to define the type of data in the class. The methods define the operations which can be performed on the data.
- A constructor is used to initialize the fields of a class whenever an object is created.
- The constructor is a method which has the same names as the class itself.
- Inheritance is where a child class inherits the fields and methods of the parent class. The child class can then also define its own methods.
- Polymorphism is the concept wherein one method can be defined multiple times. The only difference is the number of parameters which are passed to the method.
- An abstract class is a base class which has the very basic requirements of what a class should look like. It is not possible for the child class to inherit the methods of the base class.
- An interface defines a contract which the class will comply by. The interface defines what are the operations that the class can perform.

Chapter 5: C# Collections

In our previous chapter, we have learned about how we can use arrays in C#. Let's have a quick overview of it, Arrays in programming are used to group a set of related objects. So one could create an array or a set of Integers, which could be accessed via one variable name.

Collections are similar to Arrays, it provide a more flexible way of working with a group of objects.

In arrays, you would have noticed that you need to define the number of elements in an array beforehand. This had to be done when the array was declared.

But in a collection, you don't need to define the size of the collection beforehand. You can add elements or even remove elements from the collection at any point of time. This chapter will focus on how we can work with the different collections available in C#.

C# ArrayList

The ArrayList collection is similar to the Arrays data type in C#. The biggest difference is the dynamic nature of the array list collection.

For arrays, you need to define the number of elements that the array can hold at the time of array declaration. But in the case of the Array List collection, this does not need to be done beforehand. Elements can actually be added or removed from the Array List collection at any point of time. Let's look at the operations available for the array list collection in more detail.

1. **Declaration of an Array List** – The declaration of an ArrayList is provided below. An array list is created with the help of the ArrayList Data type. The “new” keyword is used to create an object of an Arraylist. The object is then assigned to the variable a1. So now the variable a1 will be used to access the different elements of the array list.

ArrayList a1 = new ArrayList()

2. **Adding elements to an array** – The add method is used to add an element to the ArrayList. The add method can be used to add any sort of data type element to the array list. So you can add an Integer, or a string, or even a Boolean value to the array list. The general syntax of the addition method is given below

ArrayList.add(element)

Below are some examples of how the “add” method can be used. The add method can be used to add various data types to the Array List collection.

Below you can see examples of how we can add Integer's Strings and even Boolean values to the Array List collection.

- a1.add(1) – This will add an Integer value to the collection
- a1.add(“Example”) – This will add a String value to the collection
- a1.add(true) – This will add a Boolean value to the collection

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file.

In the program below, we will write the code to create a new array list. We will also show to add elements and to display the elements of the Array list.

```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
    a1.Add(1);
    a1.Add("Example");
    a1.Add(true);

    Console.WriteLine(a1[0]);
    Console.WriteLine(a1[1]);
    Console.WriteLine(a1[2]);

    Console.ReadKey();
}
```

Adding elements to the array list

Defining an array list

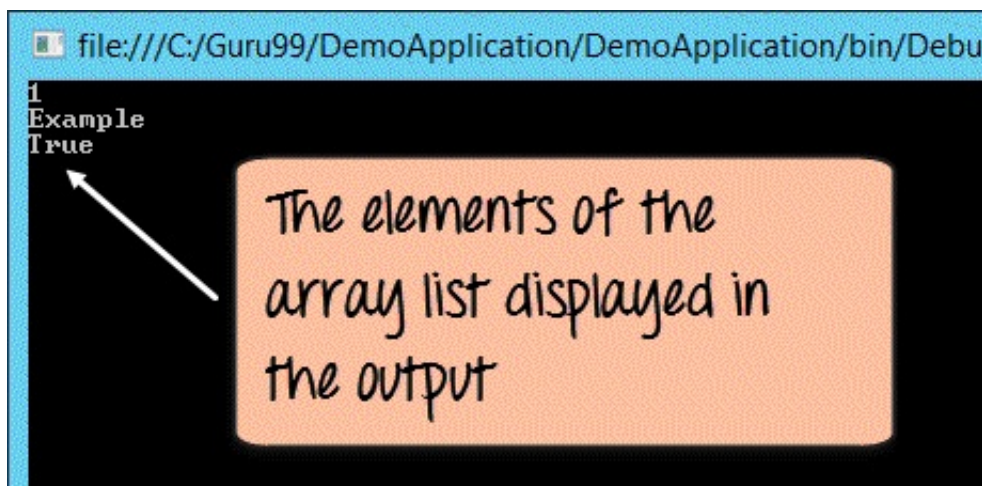
Displaying the elements of the array list

Code Explanation:-

1. The first step is used to declare our Array List. Here we are declaring a1 as a variable to hold the elements of our array list.
2. We then use the add keyword to add the number 1 , the String “Example” and the Boolean value ‘true’ to the array list.
3. We then use the Console.WriteLine method to display the value of each array lists element to the console. You will notice that just like arrays, we can access the elements via their index positions. So to access the first position of the Array List, we use the [0] index position. And so on and so forth.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, you can clearly see that all of the elements from the array list are sent to the console.

Let’s look at some more methods which are available as part of the ArrayList.

- **Count** – This method is used to get the number of items in the ArrayList collection. Below is the general syntax of this statement. `ArrayList.Count()` –

This method will return the number of elements that the array list contains.

- **Contains** - This method is used to see if an element is present in the ArrayList collection. Below is the general syntax of this statement

`ArrayList.Contains(element)` – This method will return true if the element is present in the list , else it will return false.

- **RemoveAt** - This method is used to remove an element at a specific position in the ArrayList collection. Below is the general syntax of this statement

`ArrayList.RemoveAt(index)` – This method will remove an element from a specific position of the Array List.

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file.

In the below program, we will write the code to see how we can use the above-mentioned methods.

```
static void Main(string[] args)
{
    ArrayList a1 = new ArrayList();
    a1.Add(1);
    a1.Add("Example");
    a1.Add(true);

    Console.WriteLine(a1.Count);
    Console.WriteLine(a1.Contains(2));

    Console.WriteLine(a1[1]);
    a1.RemoveAt(1);
    Console.WriteLine(a1[1]);
}
```

The diagram shows three callout boxes with arrows pointing to specific lines of code. Callout 1 points to `Console.WriteLine(a1.Count);`. Callout 2 points to `Console.WriteLine(a1.Contains(2));`. Callout 3 points to the block of code containing `Console.WriteLine(a1[1]);`, `a1.RemoveAt(1);`, and `Console.WriteLine(a1[1]);`.

Code Explanation:-

1. So the first property we are seeing is the Count property. We are getting the Count property of the array list a1 and then writing it to the Console.
2. In the second part, we are using the Contains method to see if the arraylist a1 contains the element 2. We then write the result to the Console via the Writeline command.
3. Finally to showcase the Remove element method , we are performing the below steps,
 - a. First, we write the value of the element at Index position 1 of the array list to the console.
 - b. Then we remove the element at Index position 1 of the array list.

- c. Finally, we again write the value of the element at Index position 1 of the array list to the console. This set of steps will give a fair idea whether the remove method will work as it should be.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



```
file:///C:/Guru99/DemoApplication/DemoApplication/bin/Debug/Dev
3
False
Example
True
```

Output of the various
Array List functions

Why the last value is true?

If you see the sequence of events , the element Example is removed from the array because this is at position 1. Position 1 of the array then gets replaced by what was in position 2 earlier which the value 'true'

C# Stack

The stack is a special case collection which represents a last in first out (LIFO) concept. To first understand LIFO, let's take an example. Imagine a stack of books with each book kept on top of each other.

The concept of last in first out in the case of books means that only the top most book can be removed from the stack of books. It is not possible to remove a book from between, because then that would disturb the setting of the stack.

Hence in C#, the stack also works in the same way. Elements are added to the stack, one on the top of each other. The process of adding an element to the stack is called a push operation. To remove an element from a stack, you can also remove the top most element of the stack. This operation is known as pop.

Let's look at the operations available for the Stack collection in more detail.

- **Declaration of the stack** – A stack is created with the help of the Stack Data type. The keyword “new” is used to create an object of a Stack. The object is then assigned to the

variable st. Stack st = new Stack()

- **Adding elements to the stack** – The push method is used to add an element onto the stack. The general syntax of the statement is given below.

Stack.push(element)

- **Removing elements from the stack** – The pop method is used to remove an element from the stack. The pop operation will return the topmost element of the stack. The general syntax of the statement is given below

Stack.pop()

- **Count** – This property is used to get the number of items in the Stack. Below is the general syntax of this statement.

Stack.Count

- **Contains** - This method is used to see if an element is present in the Stack. Below is the general syntax of this statement. The statement will return true if the element exists, else it will return the value false.

Stack.Contains(element)

Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file.

In the below program, we will write the code to see how we can use the above-mentioned methods.

In this example, we will see

- How a stack gets created.
- How to display the elements of the stack, and use the Count and Contains methods.

```
static void Main(string[] args)
{
    Stack st = new Stack();
    st.Push(1);
    st.Push(2);
    st.Push(3);
    foreach (Object obj in st){
        Console.WriteLine(obj);
    }
    Console.WriteLine(); Console.WriteLine();
    Console.WriteLine("The number of elements in the stack=" + st.Count);
    Console.WriteLine("Does the stack contain the element 3=" + st.Contains(3));
}
```

The diagram shows five callout boxes with arrows pointing to specific lines of code in the Program.cs file. Callout 1 points to the Stack declaration. Callout 2 points to the Push method calls. Callout 3 points to the foreach loop. Callout 4 points to the st.Count property access. Callout 5 points to the st.Contains method call.

Code Explanation:-

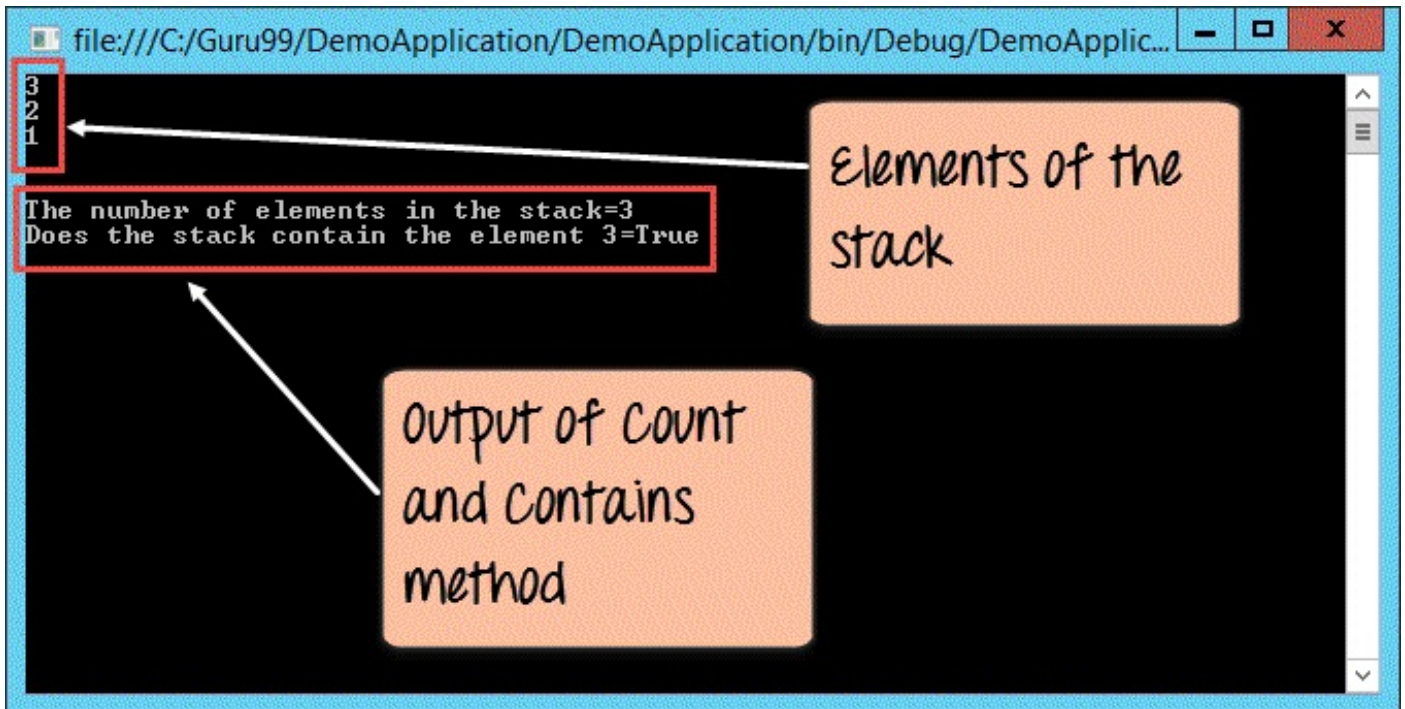
1. The first step is used to declare the Stack. Here we are declaring “st” as a variable to hold the elements of our stack.
2. Next, we add 3 elements to our stack. Each element is added via the Push method.
3. Now since the stack elements cannot be accessed via the index position like the array list, we need to use a different approach to display the elements of the stack. The Object (obj) is a temporary variable, which is declared for holding each element of the stack. We then use the foreach statement to go through each element of the stack. For each stack element, the value is assigned to the obj variable. We then use the Console.Writeline command to display the value to the console.
4. We are using the Count property (**st.count**) to get the number of items in the stack. This property will return a number. We then display this value to the

console.

5. We then use the Contains method to see if the value of 3 is present in our stack. This will return either a true or false value. We then display this return value to the console.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, we can clearly see that the elements of the stack are displayed. Also, the value of True is displayed to say that the value of 3 is defined on the stack.

Note: You have noticed that the last element pushed onto the stack is displayed first. This is the topmost element of the stack. The count of stack elements is also shown in the output.

Now let's look at the "remove" functionality. We will see the code required to remove the topmost element from the stack.

```
static void Main(string[] args)
{
    Stack st = new Stack();

    st.Push(1);
    st.Push(2);
    st.Push(3);

    st.Pop(); 1
}

foreach (Object obj in st){
    Console.WriteLine(obj);
}
```

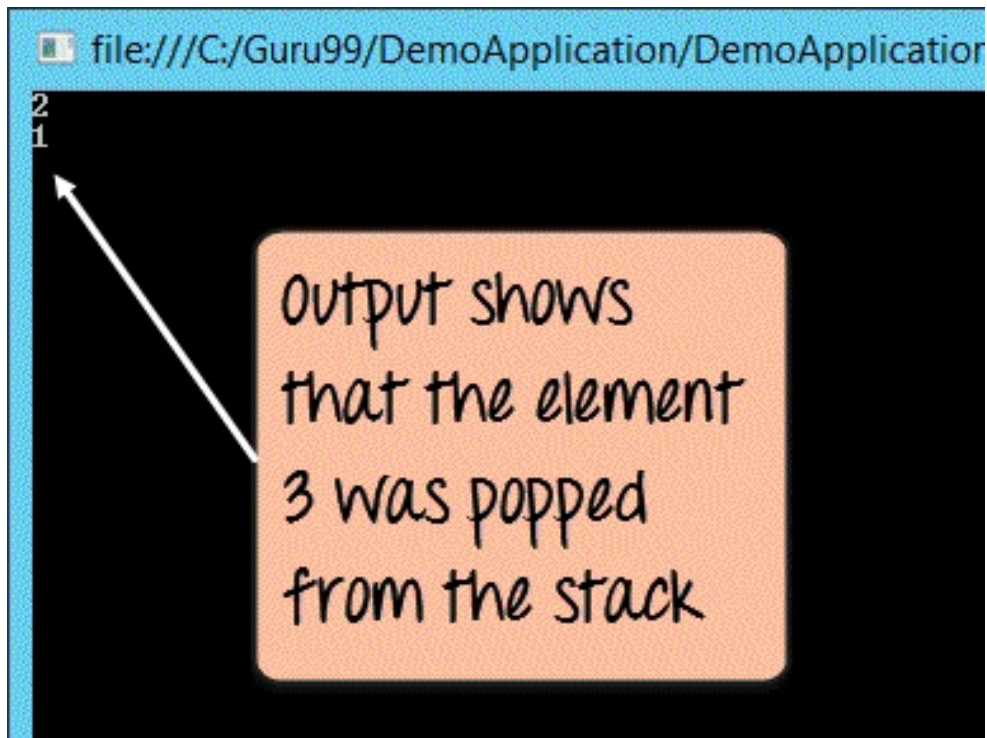
Popping an
element from
the stack

Code Explanation:-

1. Here we just issue the pop method which is used to remove an element from the stack.

If the above code is entered properly and the program is run, the following output will be displayed.

Output:



We can see that the element 3 was removed from the stack.

C# Queue

The Queue is a special case collection which represents a first in first out concept. Imagine a queue of people waiting for the bus. Normally, the first person who enters the queue will be the first person to enter the bus. Similarly, the last person to enter the queue will be the last person to enter into the bus. Elements are added to the stack, one on the top of each other.

The process of adding an element to the queue is the enqueue operation. To remove an element from a queue, you can use the dequeue operation. The operation in queues are similar to stack we saw previously.

Let's look at the operations available for the Queue collection in more detail.

- **Declaration of the Queue** – The declaration of a Queue is provided below. A Queue is created with the help of the Queue Data type. The “new” keyword is used to create an object of a Queue. The object is then assigned to the variable qt.

Queue qt = new Queue()

- **Adding elements to the Queue** – The enqueue method is used to add an element onto the queue. The general syntax of the statement is given below.

Queue.enqueue(element)

- **Removing elements from the queue** – The dequeue method is used to remove an element from the queue. The dequeue operation will return the last element of the queue. The general syntax of the statement is given below

Queue.pop()

- **Count** – This property is used to get the number of items in the queue. Below is the general syntax of this statement.

Queue.Count

- **Contains** - This method is used to see if an element is present in the Queue. Below is the general syntax of this statement. The statement will return true if the element exists, else it will return the value false.

Queue.Contains(element)

Now, let's see this working at a code level. All of the below-mentioned code will be written to our Console application.

The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the above-mentioned methods.

In this example, we will see how a queue gets created. Next, we will see how to display the elements of the queue, and use the Count and Contains methods.

```
static void Main(string[] args)
{
    Queue qt = new Queue();
    qt.Enqueue(1);
    qt.Enqueue(2);
    qt.Enqueue(3);
    foreach (Object obj in qt){
        Console.WriteLine(obj);
    }
    Console.WriteLine(); Console.WriteLine();
    Console.WriteLine("The number of elements in the queue" + qt.Count);
    Console.WriteLine("Does the queue contain" + qt.Contains(3));
}
```

The diagram shows five callout boxes with arrows pointing to specific lines of code in the program above. Callout 1 points to the line `Queue qt = new Queue();`. Callout 2 points to the three `qt.Enqueue` lines. Callout 3 points to the `foreach` loop. Callout 4 points to the `qt.Count` property access. Callout 5 points to the `qt.Contains(3)` property access.

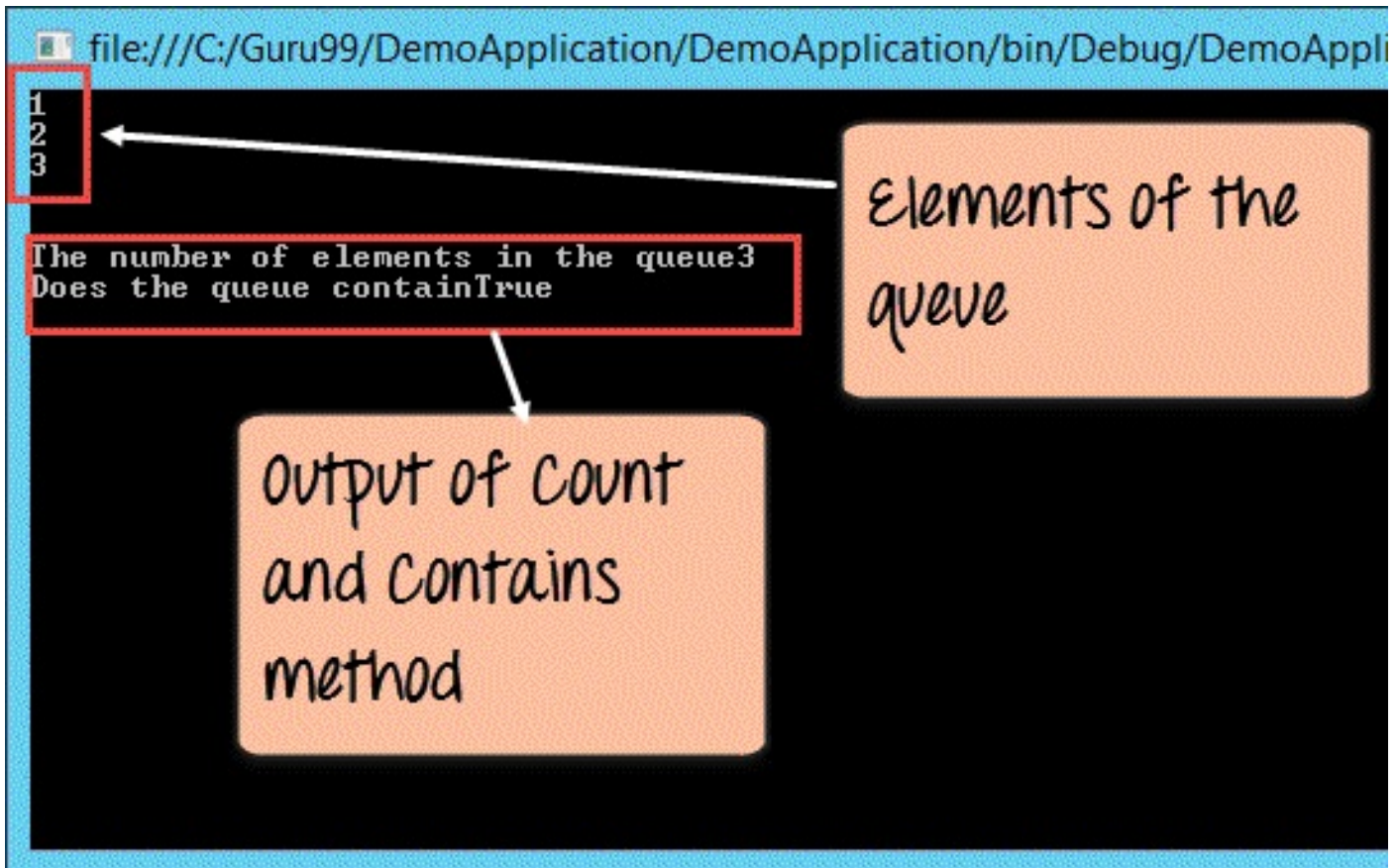
Code Explanation:-

1. The first step is used to declare the Queue. Here we are declaring qt as a variable to hold the elements of our Queue.
2. Next, we add 3 elements to our Queue. Each element is added via the “enqueue” method.
3. Now one thing that needs to be noted about Queues is that the elements cannot be accessed via the index position like the array list. We need to use a different approach to display the elements of the Queue. So here’s how we go about displaying the elements of a queue.
 - We first declare a temporary variable called obj. This will be used to hold each element of the Queue.
 - We then use the foreach statement to go through each element of the Queue.
 - For each Queue element, the value is assigned to the obj variable.
 - We then use the Console.Writeline command to display the value to the console.
4. We are using the “Count” property to get the number of items in the Queue. This property will return a number. We then display this value to the console.

5. We then use the “Contains” method to see if the value of 3 is present in our Queue. This will return either a true or false value. We then display this return value to the console.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, we can clearly see that the elements of the Queue are displayed. Note that, unlike “stack” in “queue” the first element pushed on to the queue is displayed first. The count of queue elements is also shown in the output. Also, the value of True is displayed to say that the value of 3 is defined on the queue.

Now let’s look at the remove functionality. We will see the code required to remove the last element from the queue.

```
static void Main(string[] args)
{
    Queue qt = new Queue();

    qt.Enqueue(1);
    qt.Enqueue(2);
    qt.Enqueue(3);

    qt.Dequeue(); 1
    foreach (Object obj in qt){

        Console.WriteLine(obj);
    }
}
```

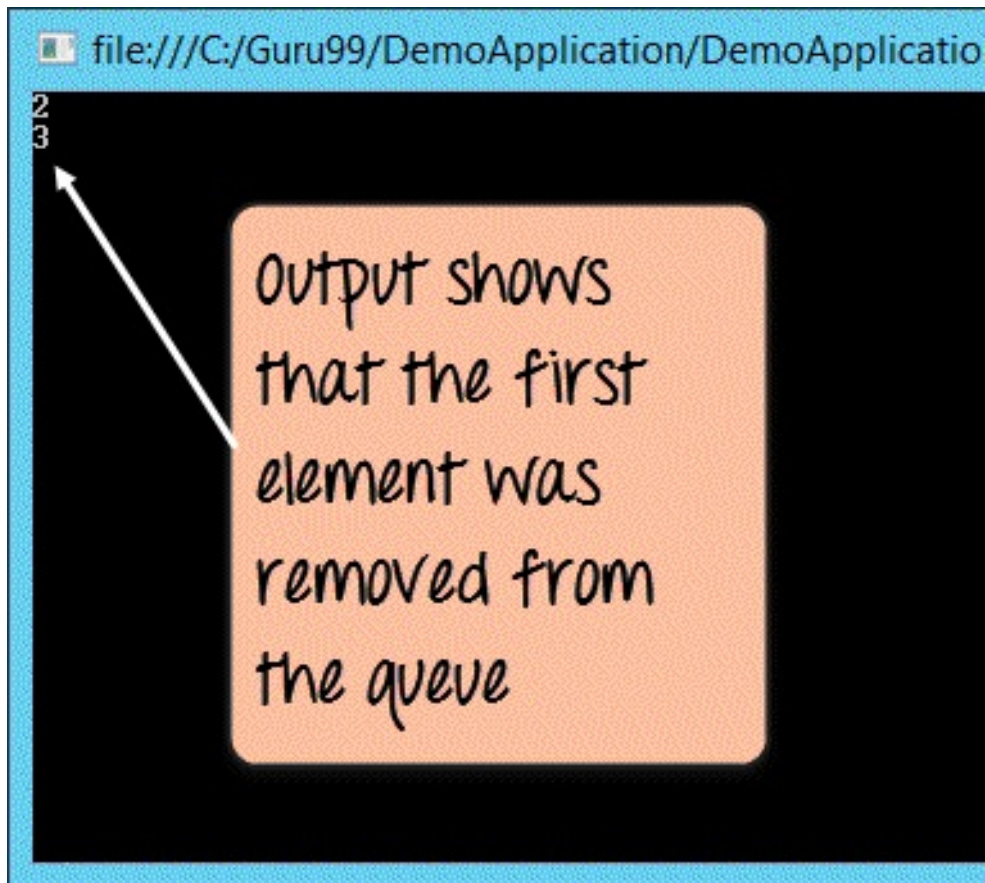
Removing an
element from
the queue

Code Explanation:-

1. Here we just issue the “dequeue” method, which is used to remove an element from the queue. This method will remove the first element of the queue.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, we can see that the first element which was added to the queue, which was the element 1, was removed from the queue.

C# Hashtable

A hash table is a special collection that is used to store key-value items. So instead of storing just one value like the stack, array list and queue, the hash table stores 2 values. These 2 values form an element of the hash table.

Below are some example of how values of a hash table might look like.

```
{ "001" , ".Net" }
```

```
{ "002" , ".C#" }
```

```
{ "003" , "ASP.Net" }
```

Above we have 3 key value pairs. The keys of each element are 001, 002 and 003 respectively. The values of each key value pair are ".Net", "C#" and "ASP.Net" respectively.

Let's look at the operations available for the Hashtable collection in more detail.

- **Declaration of the Hashtable** – The declaration of a Hashtable is shown below. A Hashtable is created with the help of the Hashtable Data type. The "new" keyword is used to create an object of a Hashtable. The object is then assigned to the variable ht.

```
Hashtable ht = new Hashtable()
```

- **Adding elements to the Hashtable** – The Add method is used to add an element on to the queue. The general syntax of the statement is given below

```
HashTable.add("key","value")
```

Remember that each element of the hash table comprises of 2 values, one is the key, and the other is the value.

Now, let's see this working at a code level. All of the below-mentioned code will be written to our Console application.

The code will be written to our Program.cs file. In the below program, we will write the code to see how we can use the above-mentioned methods.

For now in our example, we will just look at how we can create a hashtable , add elements to the hashtable and display them accordingly.

```
static void Main(string[] args)
{
    Hashtable ht = new Hashtable();
    ht.Add("001", ".Net");
    ht.Add("002", "C#");
    ht.Add("003", "ASP.Net");
    ICollection keys = ht.Keys;
    foreach (String k in keys){
        Console.WriteLine(ht[k]);
    }
    Console.ReadKey();
}
```

The diagram shows a code snippet with four numbered callouts in orange boxes:

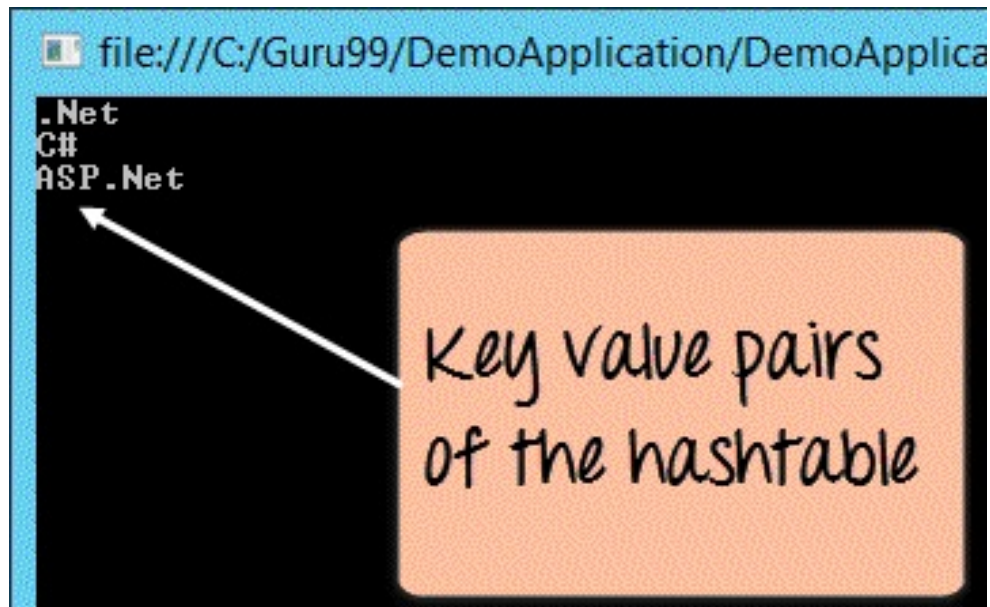
- 1. **Creating a hashtable variable**: Points to the line `Hashtable ht = new Hashtable();`
- 2. **Adding elements to the hashtable**: Points to the three `ht.Add` lines.
- 3. **Getting the keys collection**: Points to the line `ICollection keys = ht.Keys;`
- 4. **Displaying the value for each key**: Points to the `Console.WriteLine` line inside the `foreach` loop.

Code Explanation:-

1. First, we declare the hashtable variable using the Hashtable data type by using keyword “New.” The name of the variable defines is ‘ht’.
2. We then add elements to the hash table using the Add method. Remember that we need to add both a key and value element when adding something to the hashtable.
3. There is no direct way to display the elements of a hash table.
 - In order to display the hashtable , we first need to get the list of keys (001, 002 and 003) from the hash table.
 - This is done via the ICollection interface. This is a special data type which can be used to store the keys of a hashtable collections. We then assign the keys of the hashtable collection to the variable ‘keys’.
4. Next for each key value, we get the associated value in the hashtable by using the statement `ht[k]`.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



Let's look at some more methods available for hash tables.

- **ContainsKey** - This method is used to see if a key is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the key exists, else it will return the value false.

Hashtable.Containskey(key)

- **ContainsValue** - This method is used to see if a Value is present in the Hashtable. Below is the general syntax of this statement. The statement will return true if the Value exists, else it will return the value false.

Hashtable.ContainsValue(key)

Let's change the code in our Console application to showcase how we can use the "Containskey" and "ContainsValue" method.

```
static void Main(string[] args)
{
    Hashtable ht = new Hashtable();

    ht.Add("001", ".Net");
    ht.Add("002", "C#");
    ht.Add("003", "ASP.Net");

    Console.WriteLine(ht.ContainsKey("001"));
    Console.WriteLine(ht.ContainsValue("C#"));
    Console.ReadKey();
}
```

Searching for a key

1

Searching for a value

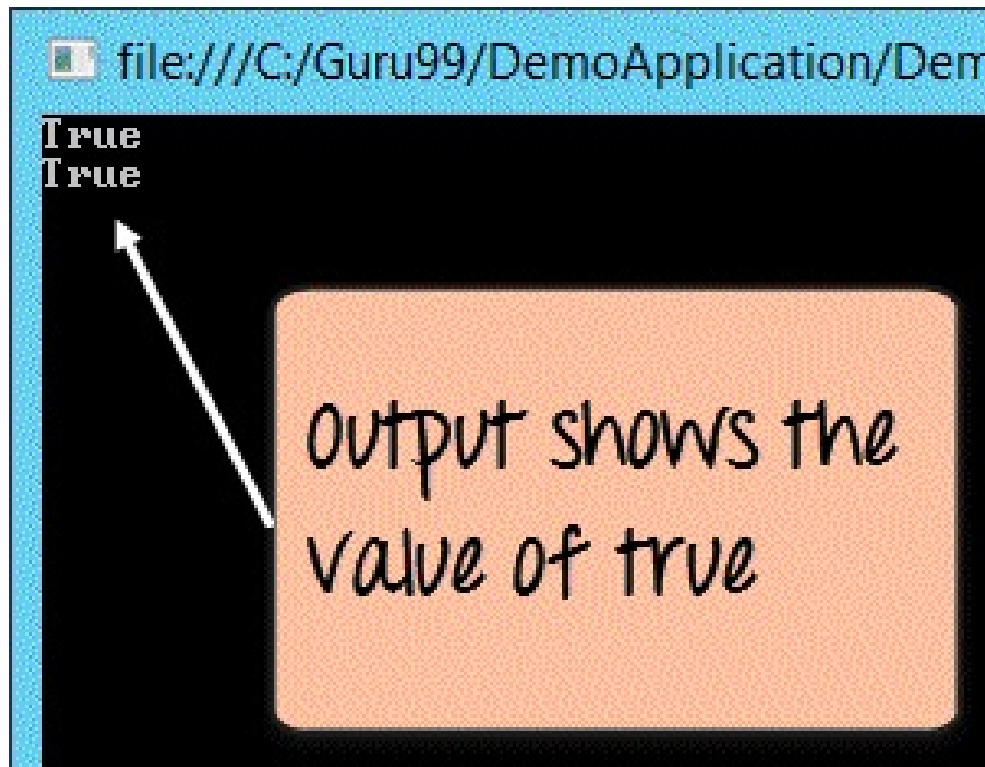
2

Code Explanation:-

1. First, we use the ContainsKey method to see if the key is present in the hashtable. This method will return true if the key is present in the hashtable. This method should return true since the key does exist in the hashtable.
2. We then use the ContainsValue method to see if the value is present in the hashtable. This method will return 'true' since the Value does exist in the hashtable.

If the above code is entered properly and the program is run the following output will be displayed.

Output:



From the output, you can clearly see that both the key and value being searched are present in the hash table.

Summary

- The Array List collection is used to store a group of elements. The advantage of the Array list collection is that it is dynamic in nature. You can add and remove elements on the fly to the array list collection.
- A Stack is based on the last in first out concept. The operation of adding an element to the stack is called the push operation. The operation of removing an element to the stack is called the pop operation.
- A Queue is based on the first in first out concept. The operation of adding an element to the queue is called the enqueue operation. The operation of removing an element to the queue is called the dequeue operation.
- A Hashtable is used to store elements which comprises of key values pairs. To access the value of an element , you need to know the key of the element.

Chapter 6: Windows Forms Application

So far we have seen how to work with C# to create console based applications. But in a real life scenario teams normally use Visual Studio and C# to create either Windows Forms or Web-based applications.

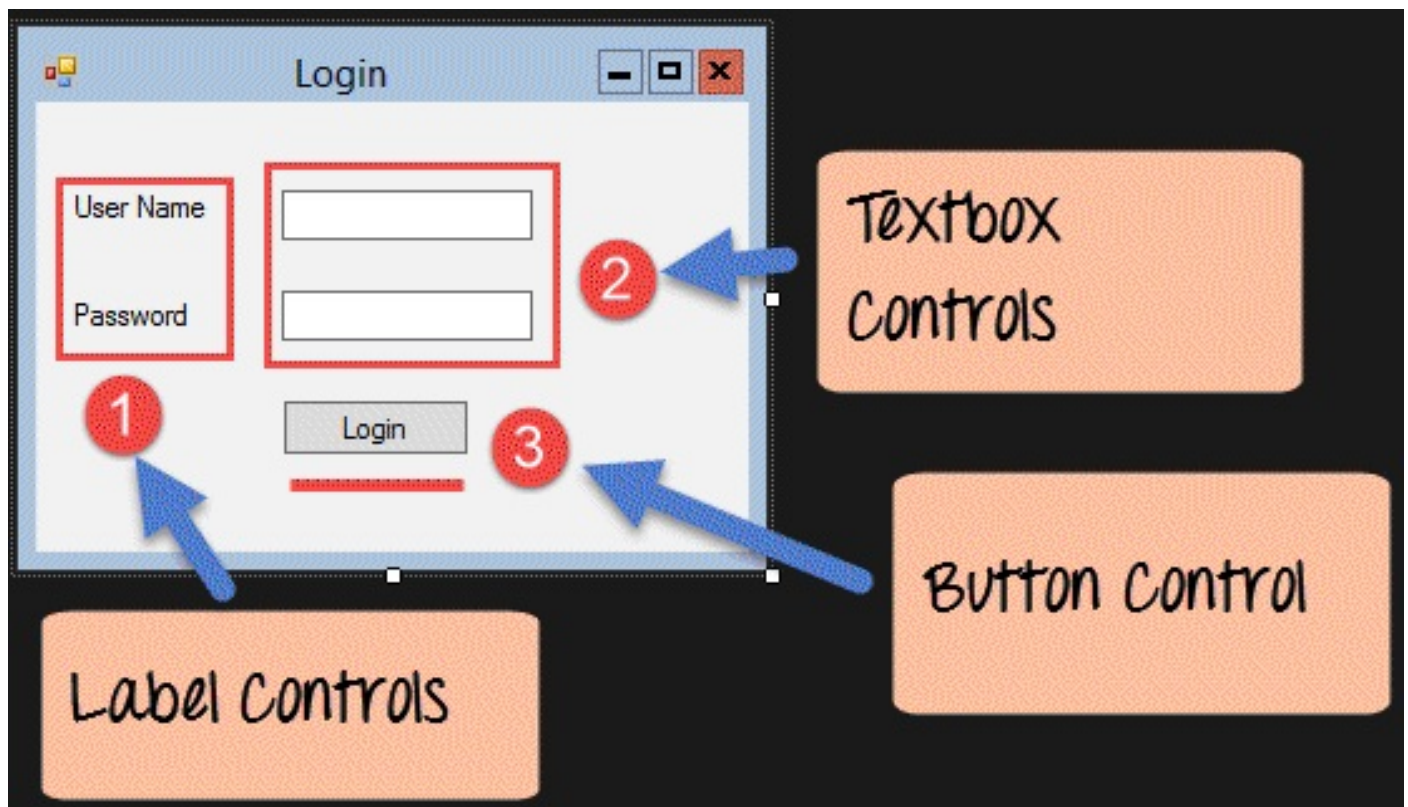
A windows form application is any application, which is designed to run on a computer. It will not run on web browser because then it becomes a web application.

This chapter will focus on how we can create Windows-based applications. We will also learn some basics on how to work with the various elements of Windows applications.

Windows Forms Basics

A Windows forms application is one that runs on the desktop computer. A Windows forms application will normally have a collection of controls such as labels, textboxes, list boxes, etc.

Below is an example of a simple Windows form application. It shows a simple Login screen, which is accessible by the user. The user will enter the required credentials and then will click the Login button to proceed.



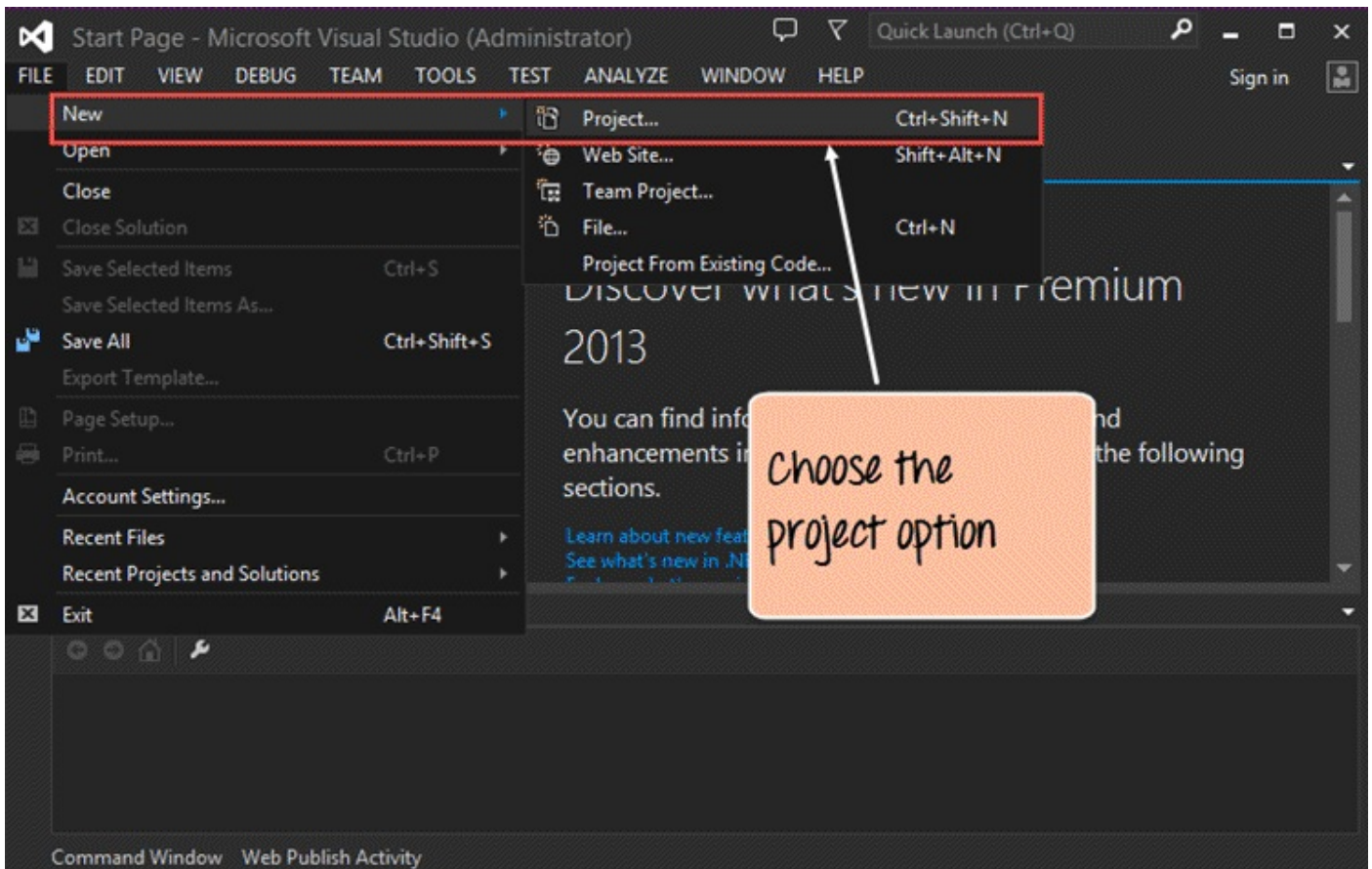
So an example of the controls available in the above application

1. This is a collection of label controls which are normally used to describe adjacent controls. So in our case, we have 2 textboxes, and the labels are used to tell the user that one textbox is for entering the user name and the other for the password.
2. The 2 textboxes are used to hold the username and password which will be entered by the user.
3. Finally, we have the button control. The button control will normally have some code attached to perform a certain set of actions. So for example in the above case, we could have the button perform an action of validating the user name and password which is entered by the user.

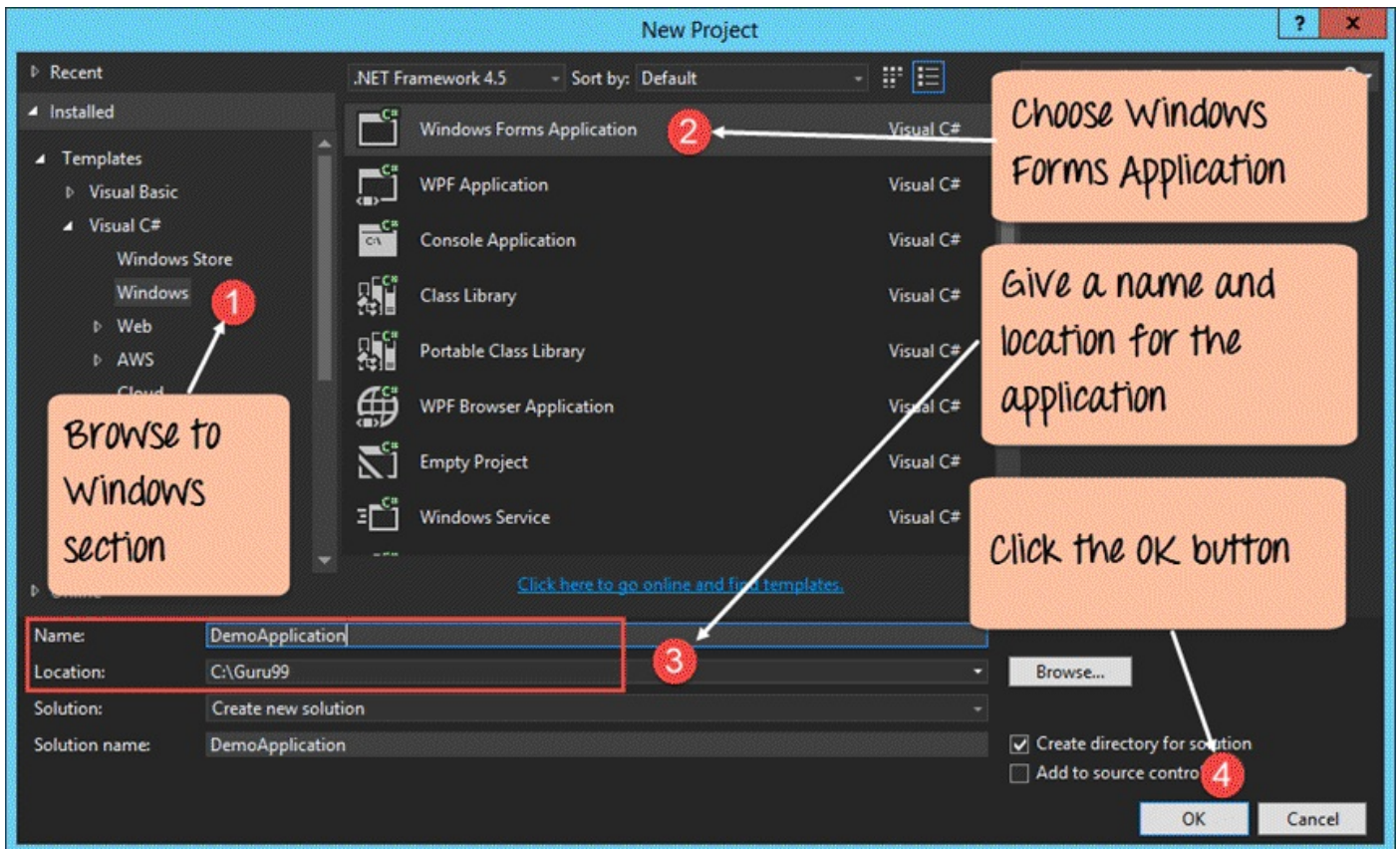
C# Hello World

Now let's look at an example of how we can implement a simple 'hello world' application in Visual Studio. For this, we would need to implement the below-mentioned steps

Step 1) The first step involves the creation of a new project in Visual Studio. After launching Visual Studio, you need to choose the menu option New->Project.



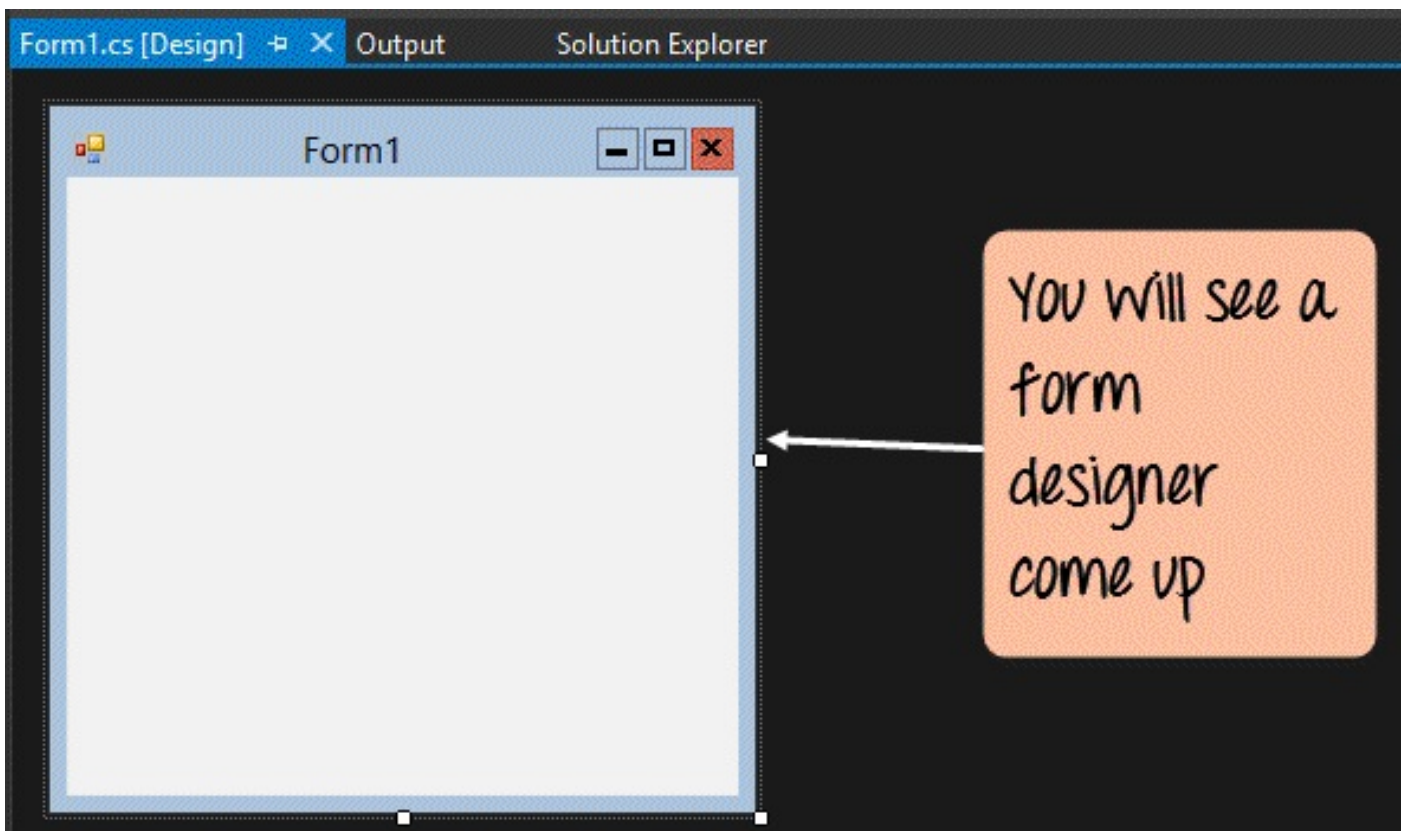
Step 2) The next step is to choose the project type as a Windows Forms application. Here we also need to mention the name and location of our project.



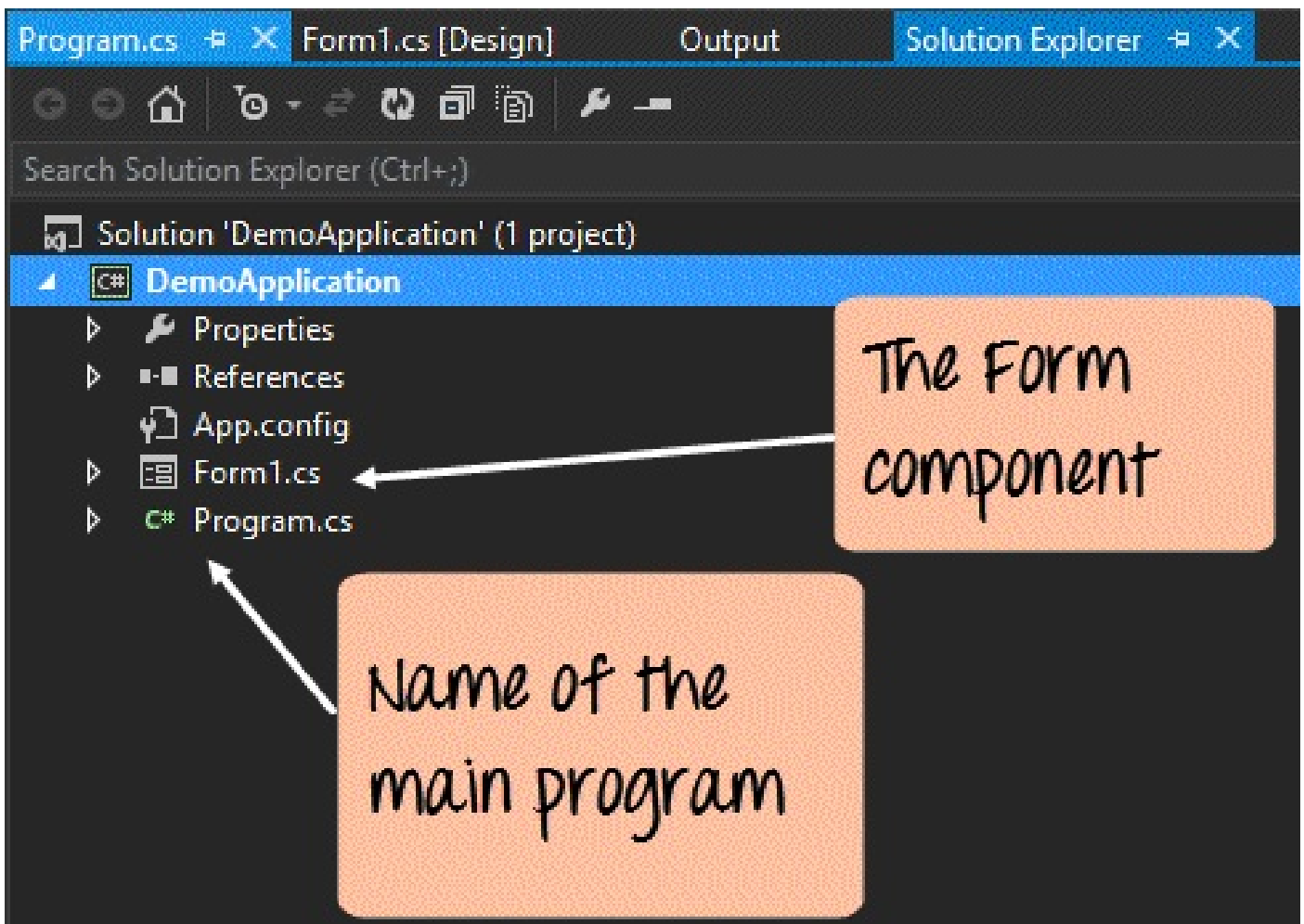
1. In the project dialog box, we can see various options for creating different types of projects in Visual Studio. Click the Windows option on the left-hand side.
2. When we click the Windows options in the previous step, we will be able to see an option for Windows Forms Application. Click this option.
3. We then give a name for the application which in our case is DemoApplication. We also need to provide a location to store our application.
4. Finally, we click the 'OK' button to let Visual Studio to create our project.

If the above steps are followed, you will get the below output in Visual Studio.

Output:-



You will actually see a Form Designer displayed in Visual Studio. It's on this Form Designer that you will start building your Windows Forms application.

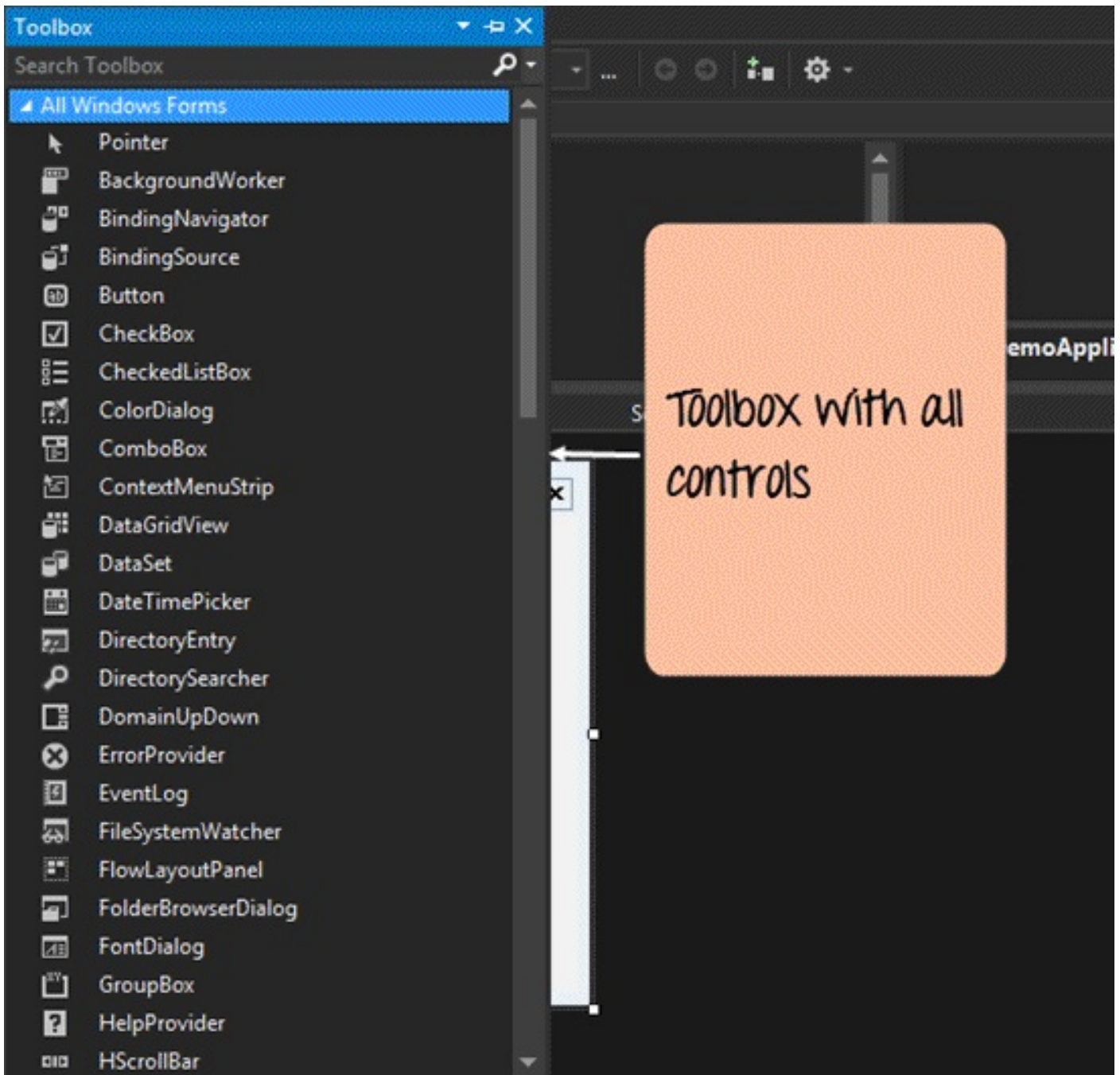


In the Solution explorer, you will also be able to see the DemoApplication Solution. This solution will contain the below 2 project files

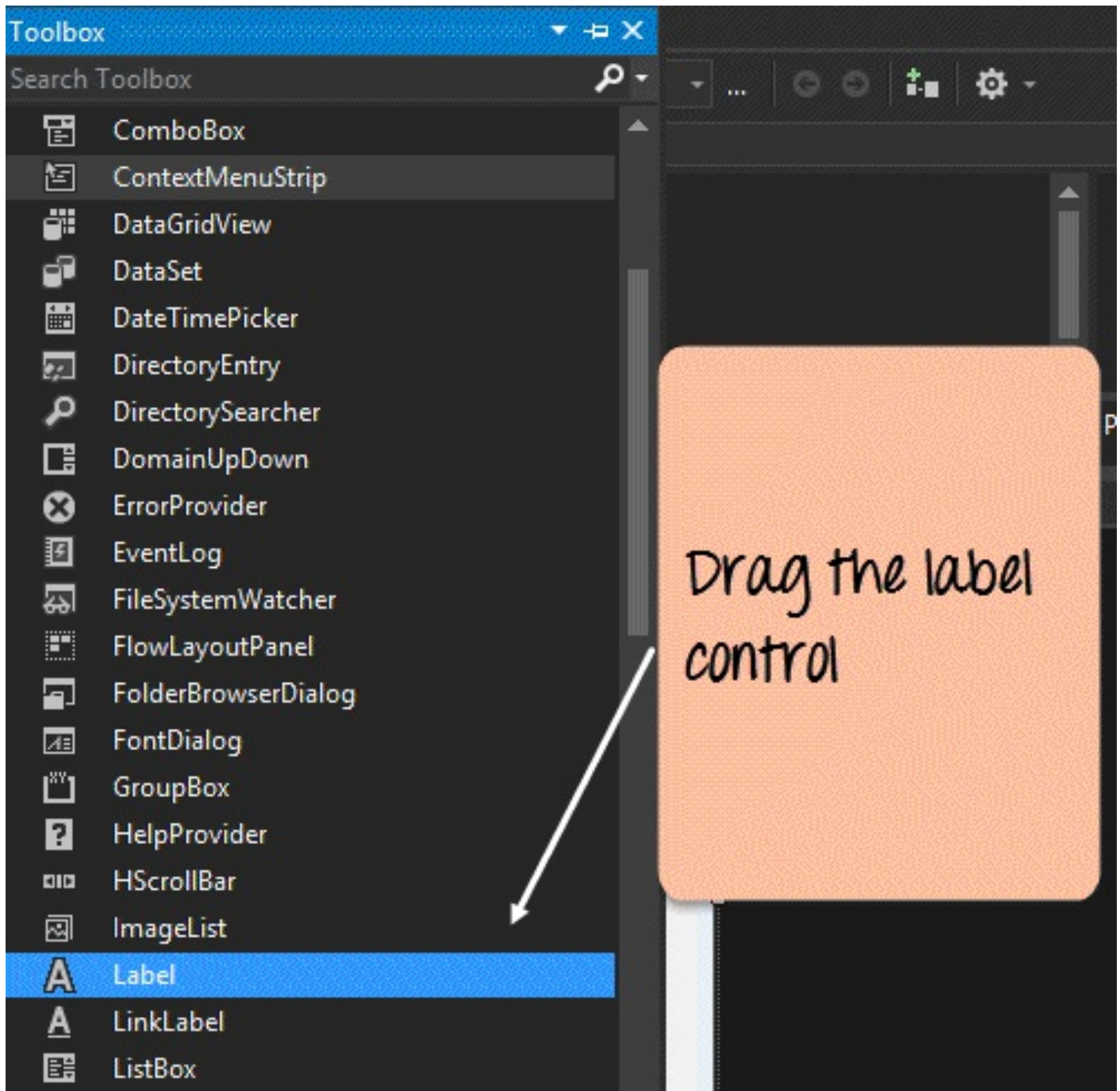
1. A Form application called Forms1.cs. This file will contain all of the code for the Windows Form application.
2. The Main program called Program.cs is default code file which is created when a new application is created in Visual Studio. This code will contain the startup code for the application as a whole.

On the right-hand side of Visual Studio, you will also see a ToolBox. The toolbox contains all the controls which can be added to a Windows Forms. Controls like a text box or a label are just some of the controls which can be added to a Windows Forms.

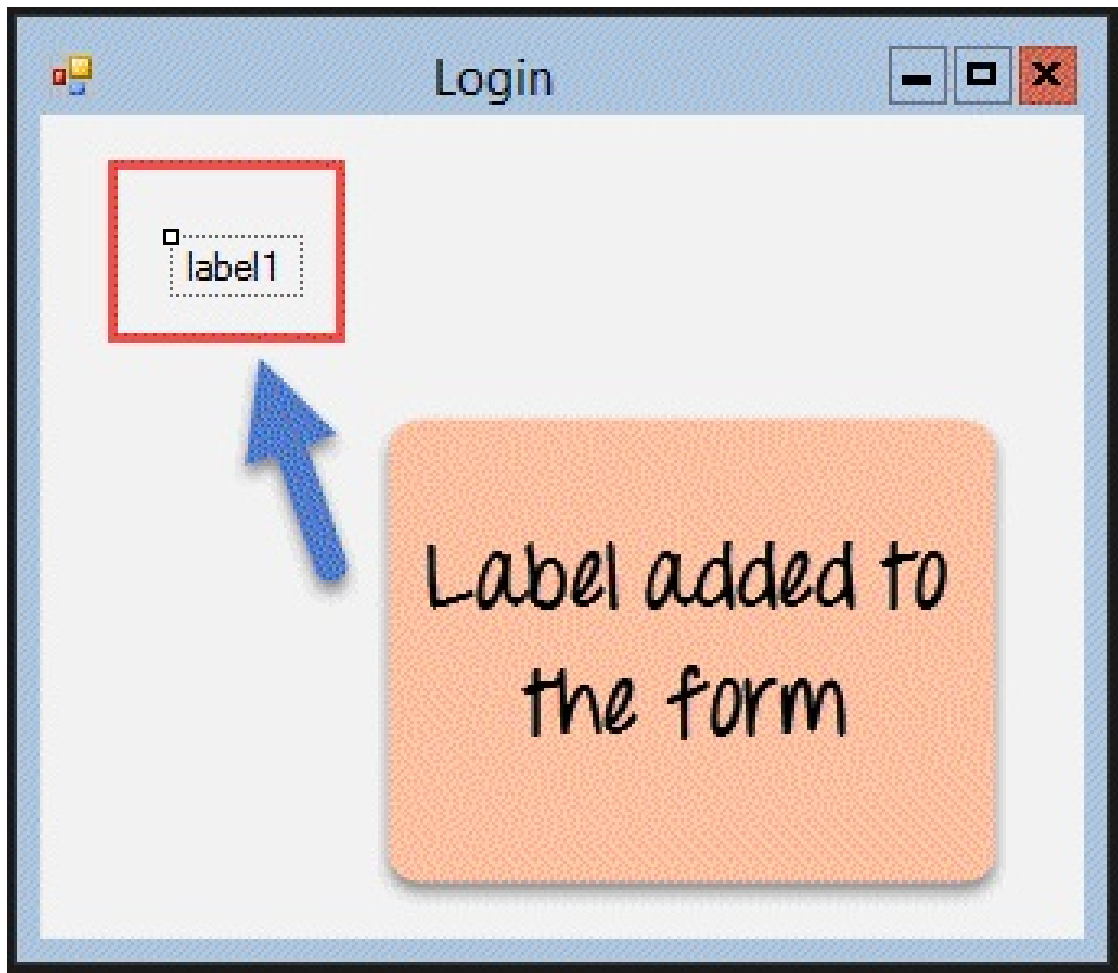
Below is a screenshot of how the Toolbox looks like.



Step 3) In this step, we will now add a label to the Form which will display “Hello World.” From the toolbox , you will need to choose the Label control and simply drag it onto the Form.

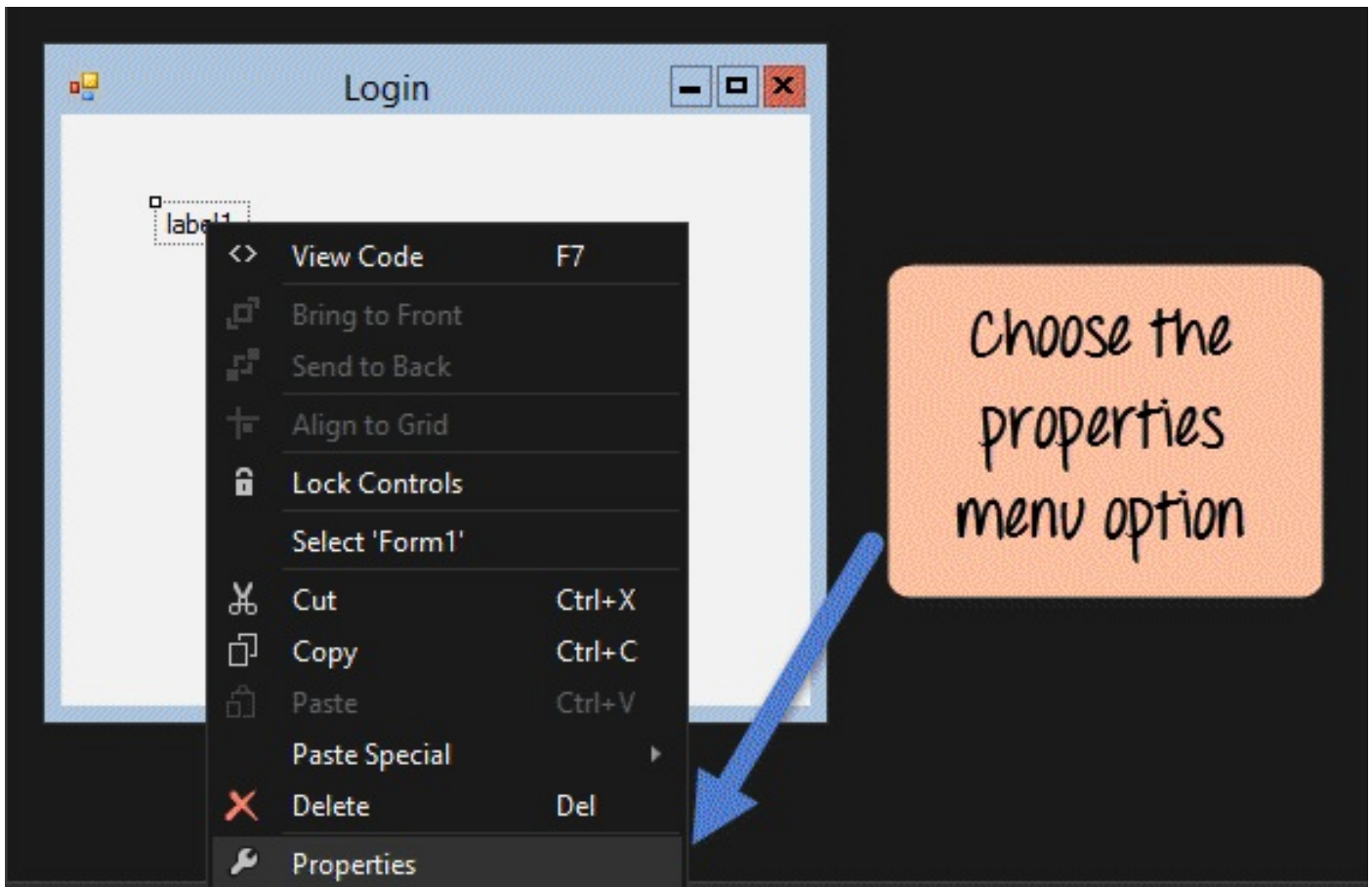


Once you drag the label to the form, you can actually see the label embedded on the form as shown below.

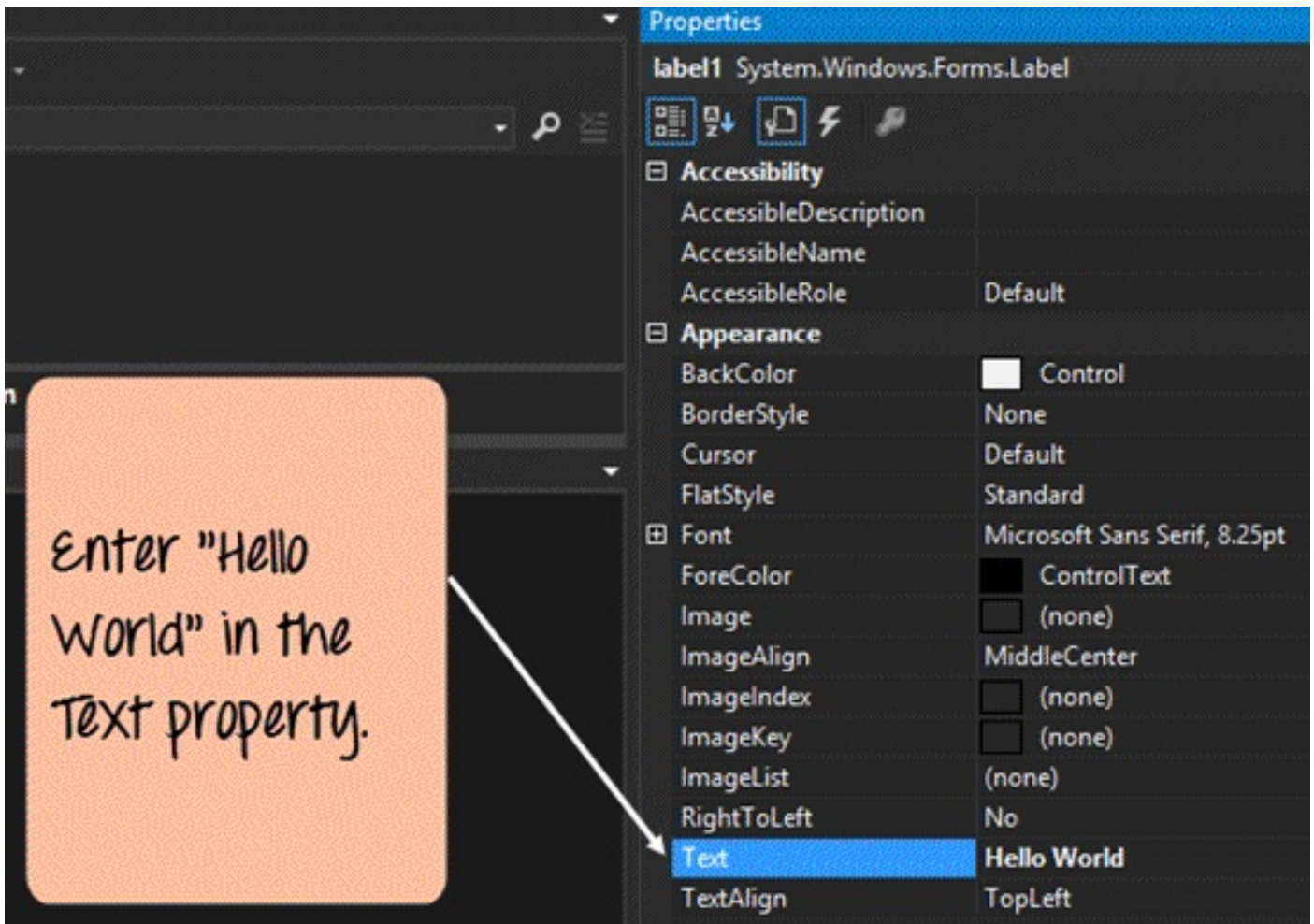


Step 4) The next step is to actually go to the properties of the control and Change the text to 'Hello World'.

To go to the properties of control, you need to right-click the control and choose the Properties menu option

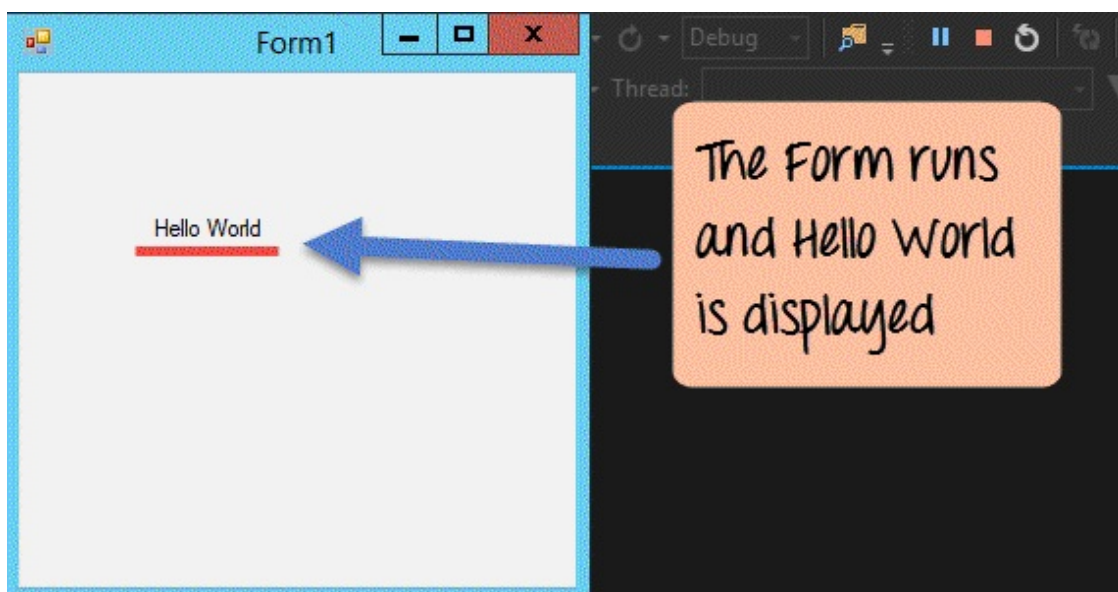


- The properties panel also shows up in Visual Studio. So for the label control, in the properties control, go to the Text section and enter “Hello World”.
- Each Control has a set of properties which describe the control.



If you follow all of the above steps and run your program in Visual Studio, you will get the following output

Output:-



In the output, you can clearly see that the Windows Form is displayed. You can also see 'Hello World' is displayed on the form.

Adding Controls to a form

We had already seen how to add a control to a form when we added the label control in the earlier section to display “Hello World.”

Let’s look at the other controls available for Windows forms and see some of their common properties.

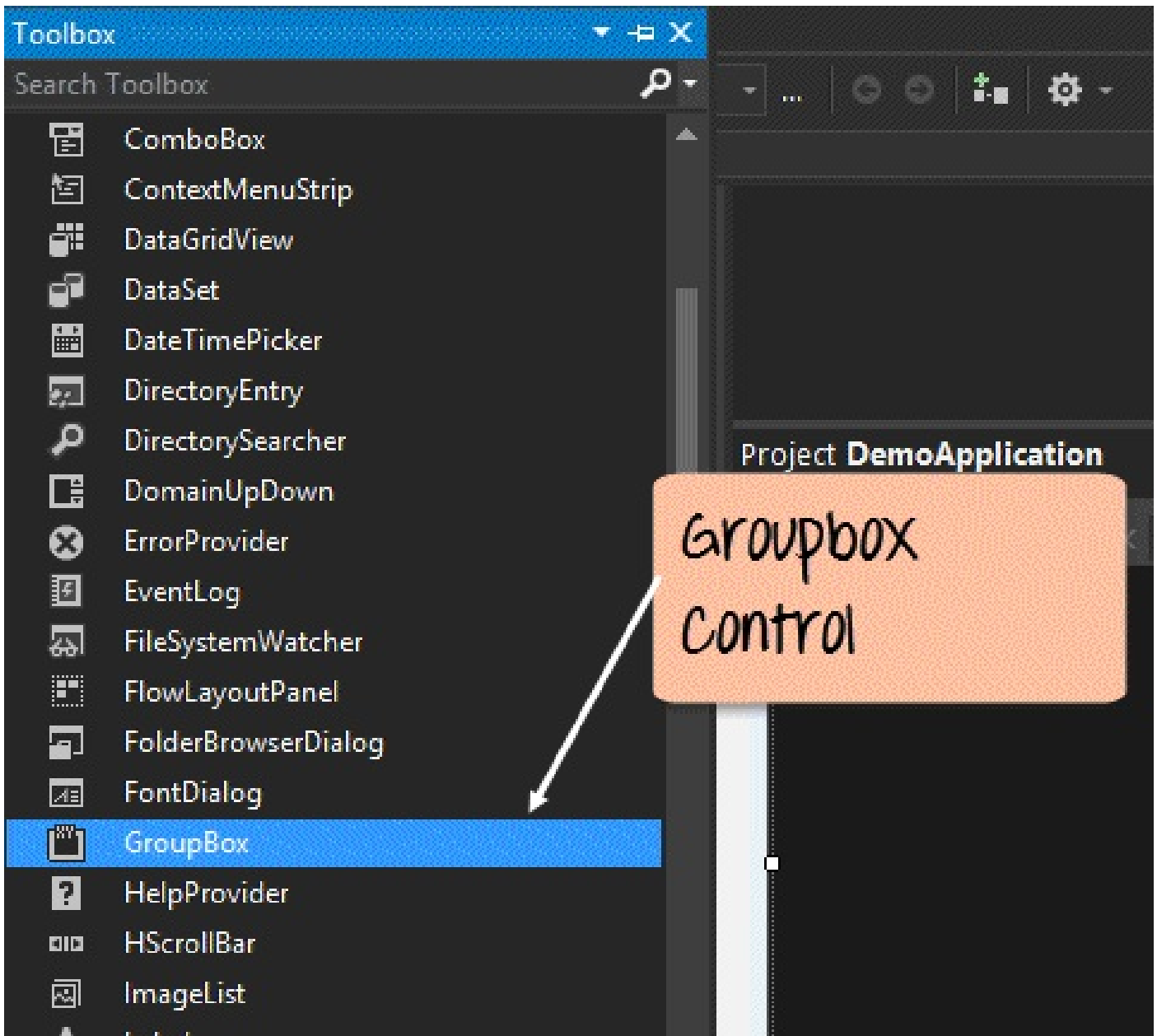
In our example, we will create one form which will have the following functionality.

1. The ability for the user to enter name and address.
2. An option to choose the city in which the user resides in
3. The ability for the user to enter an option for the gender.
4. An option to choose a course which the user wants to learn. There will choices for both C# and ASP.Net

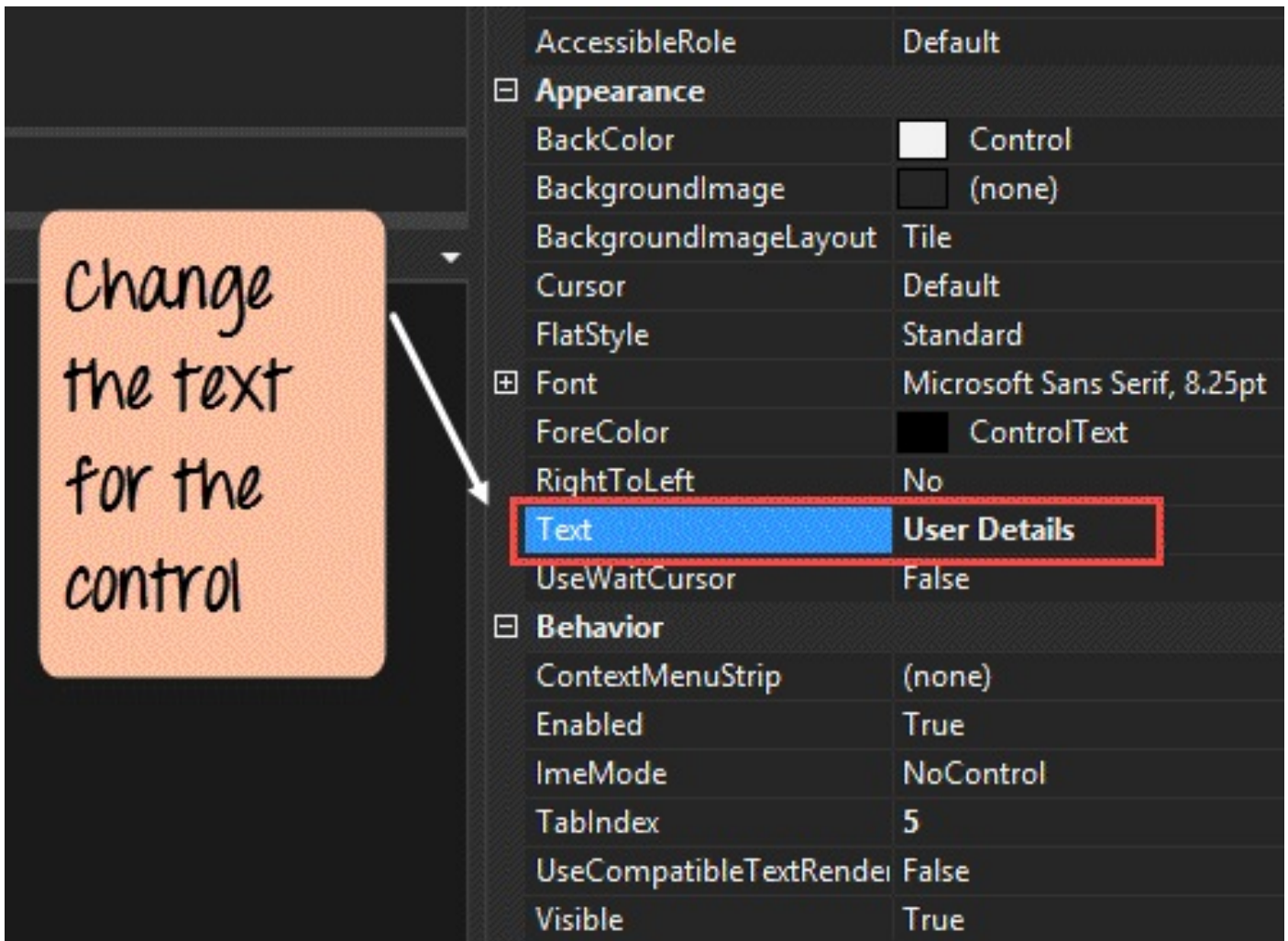
So let’s look at each control in detail and add them to build the form with the above-mentioned functionality.

Group Box – A group box is used for grouping logical controls into a section. Let’s take an example, if you had a collection of controls for entering details such as name and address of a person. Ideally, these are details of a person, so you would want to have these details in a separate section on the Form. For this purpose, you can have a group box. Let’s see how we can implement this with an example shown below

Step 1) The first step is to drag the Groupbox control onto the Windows Form from the toolbox as shown below

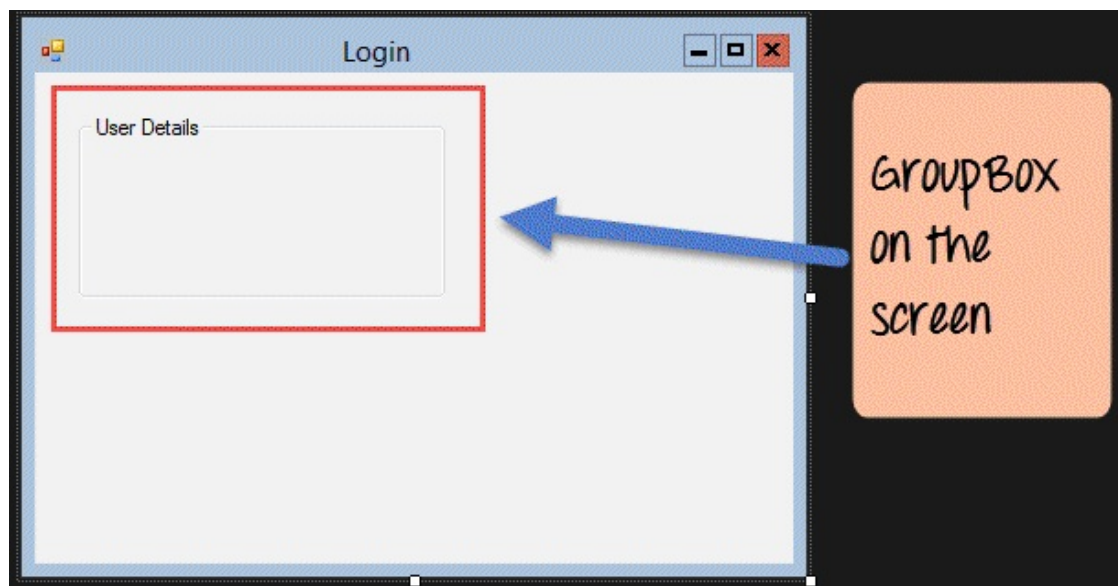


Step 2) Once the groupbox has been added, go to the properties window by clicking on the groupbox control. In the properties window, go to the Text property and change it to “User Details”.



Once you make the above changes, you will see the following output

Output:-



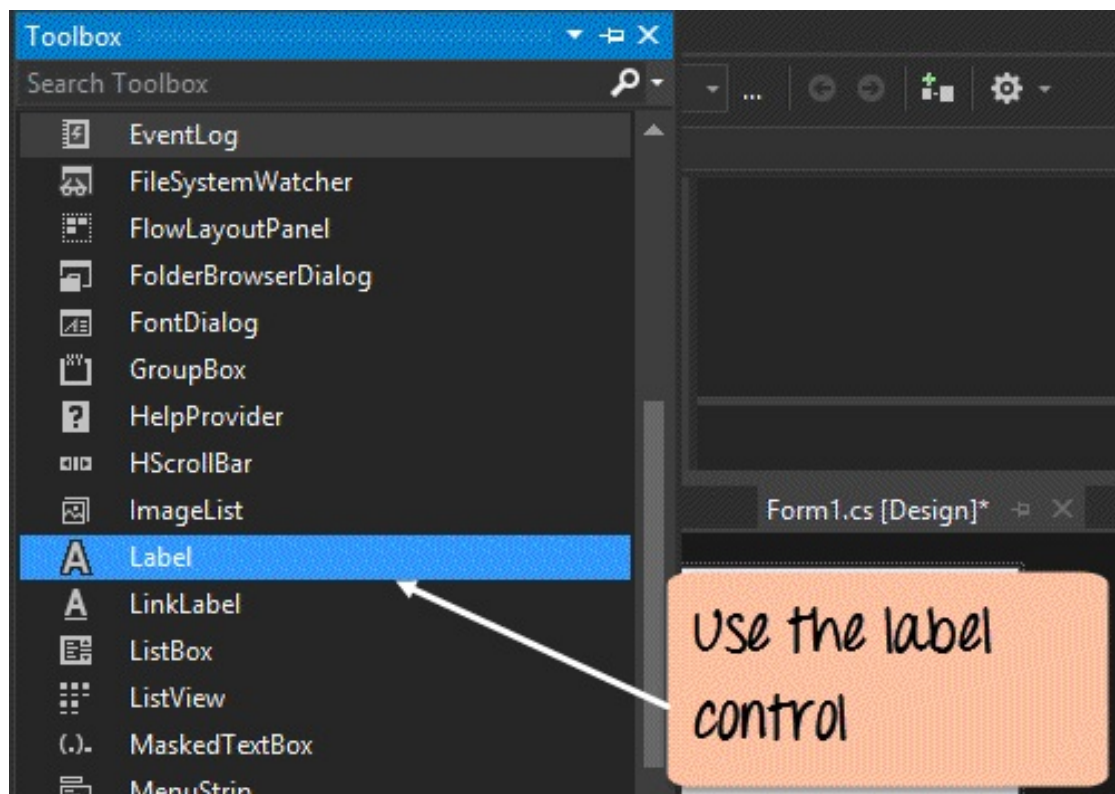
In the output, you can clearly see that the Groupbox was added to the form. You can also see that the text of the groupbox was changed to "User Details."

Label Control – Next comes the Label Control. The label control is used to display a text or a message to the user on the form. The label control is normally

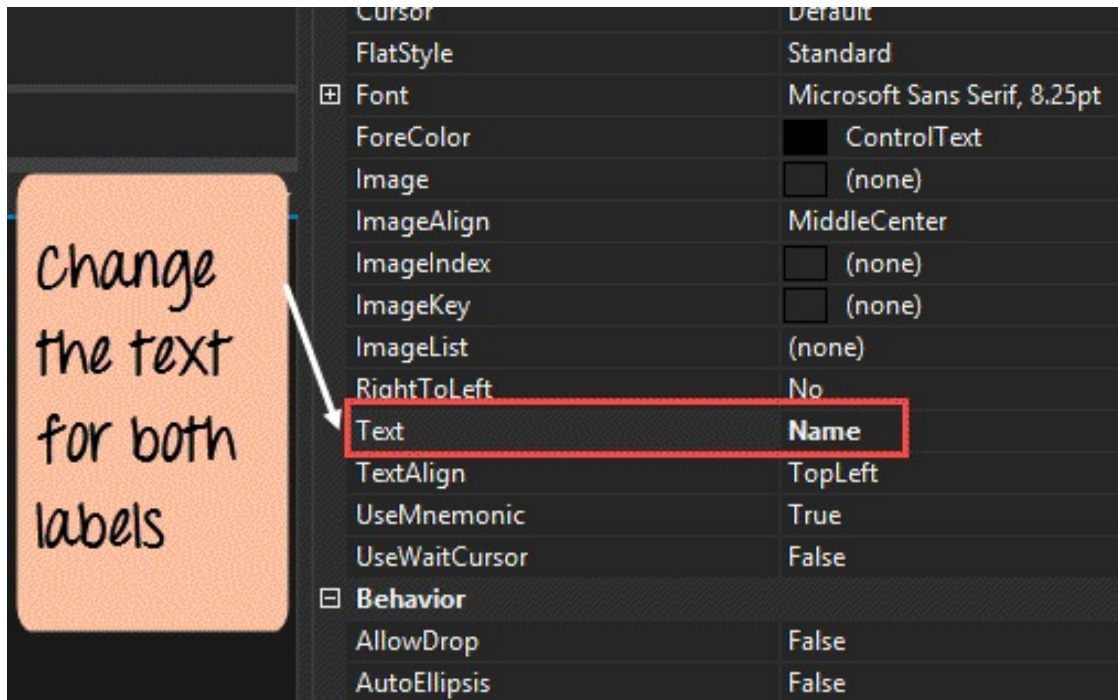
used along with other controls. Common examples is wherein a label is added along with the textbox control.

The label gives an indication to the user on what is expected to fill up in the textbox. Let's see how we can implement this with an example shown below. We will add 2 labels, one which will be called 'name' and the other called 'address.' They will be used in conjunction with the textbox controls which will be added in the later section.

Step 1) The first step is to drag the label control on to the Windows Form from the toolbox as shown below. Make sure you drag the label control 2 times so that you can have one for the 'name' and the other for the 'address'.

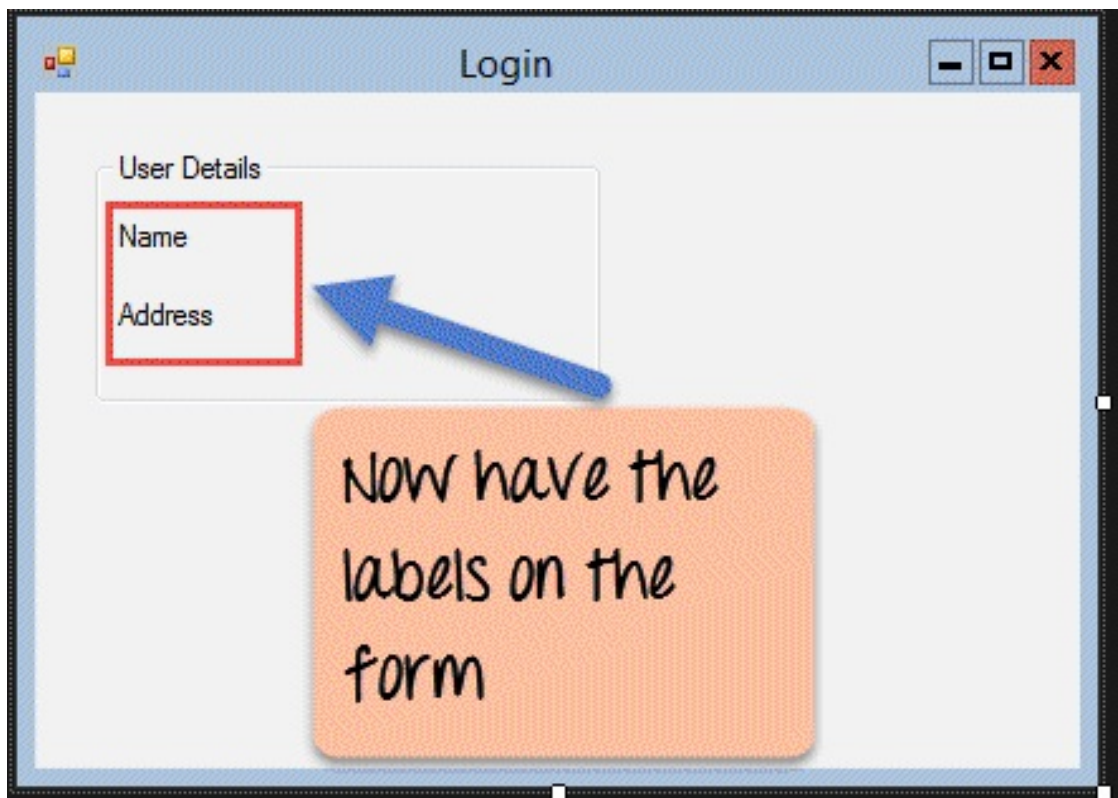


Step 2) Once the label has been added, go to the properties window by clicking on the label control. In the properties window, go to the Text property of each label control.



Once you make the above changes, you will see the following output

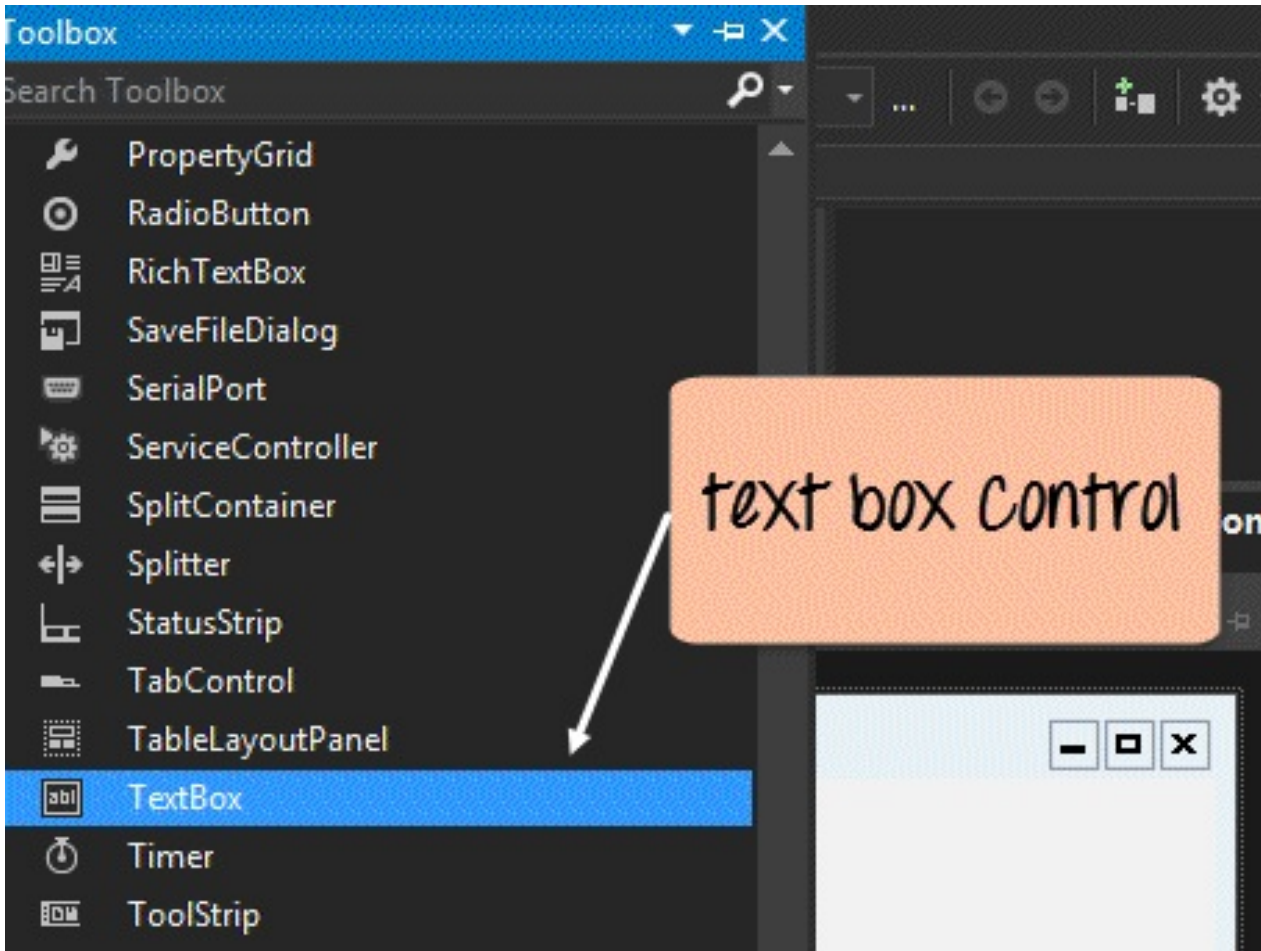
Output:-



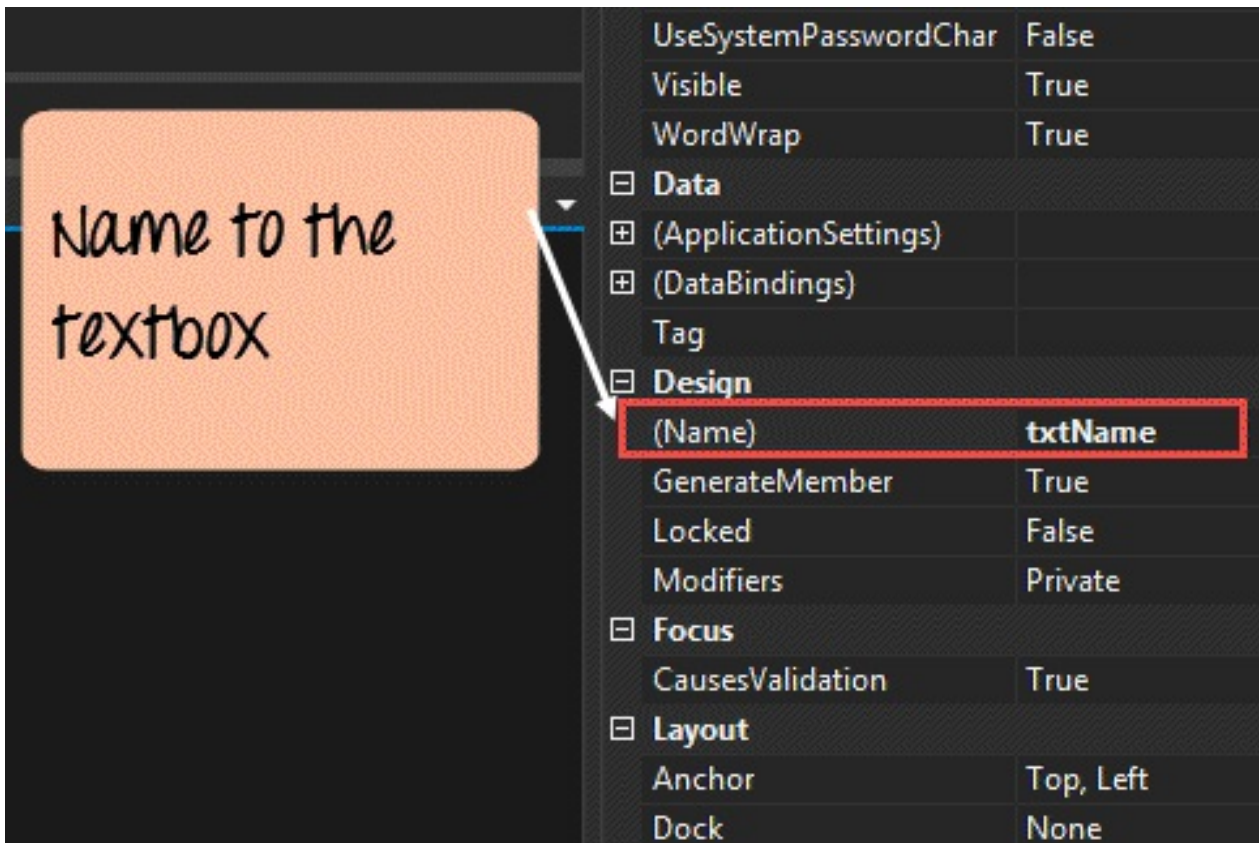
You can actually see the label controls added to the form.

- **Textbox** – A textbox is used for allowing a user to enter some text on the forms application. Let's see how we can implement this with an example shown below. We will add 2 textboxes to the form, one for the Name and the other for the address to be entered for the user

Step 1) The first step is to drag the textbox control onto the Windows Form from the toolbox as shown below

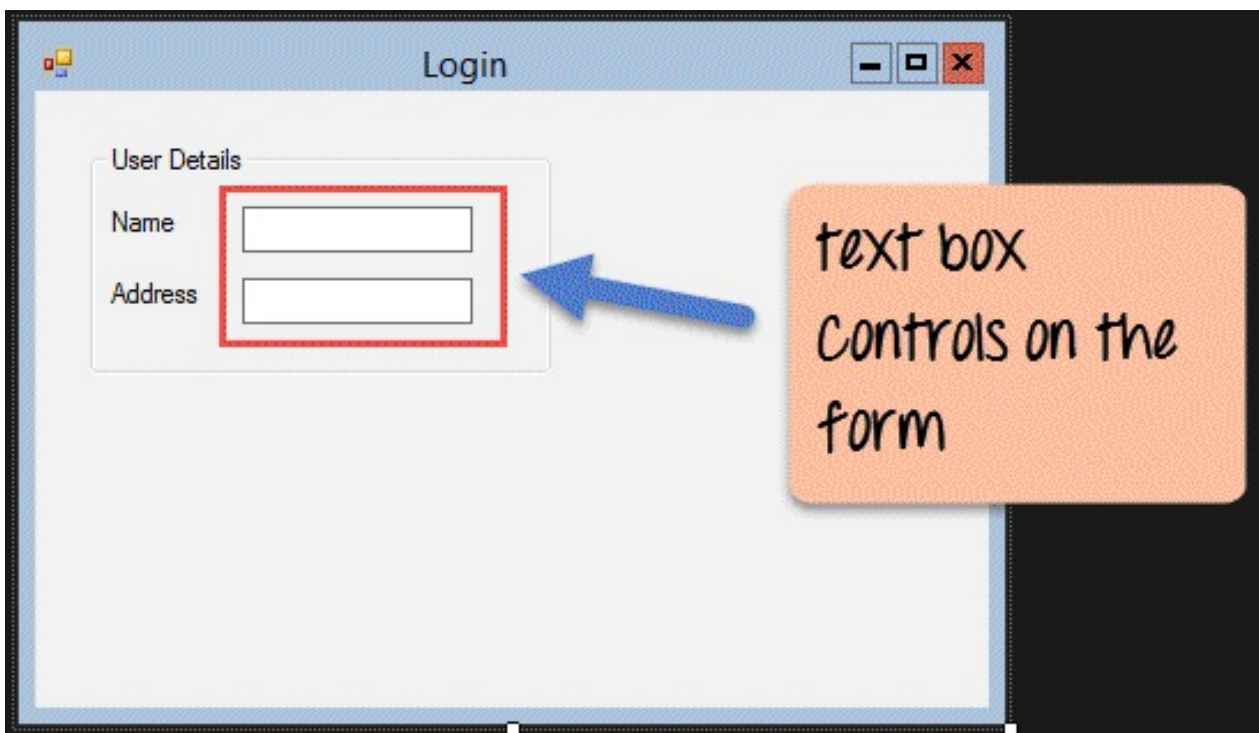


Step 2) Once the text boxes have been added, go to the properties window by clicking on the textbox control. In the properties window, go to the Name property and add a meaningful name to each textbox. For example, name the textbox for the user as txtUser and that for the address as txtAddress. A naming convention and standard should be made for controls because it becomes easier to add extra functionality to these controls, which we will see later on.



Once you make the above changes, you will see the following output

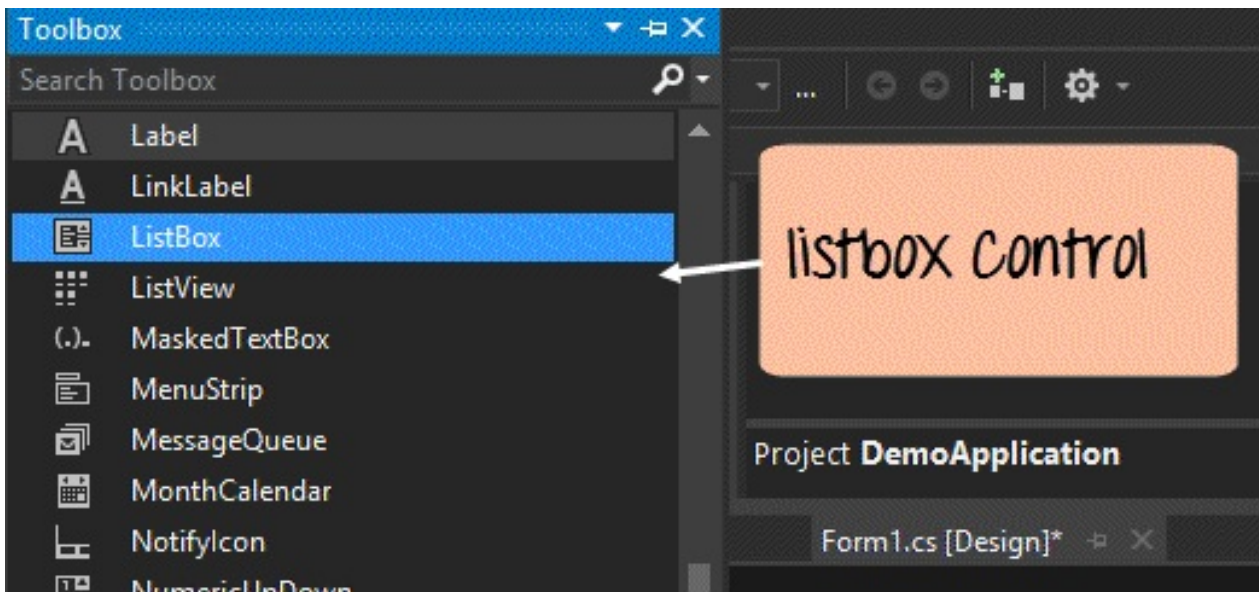
Output:-



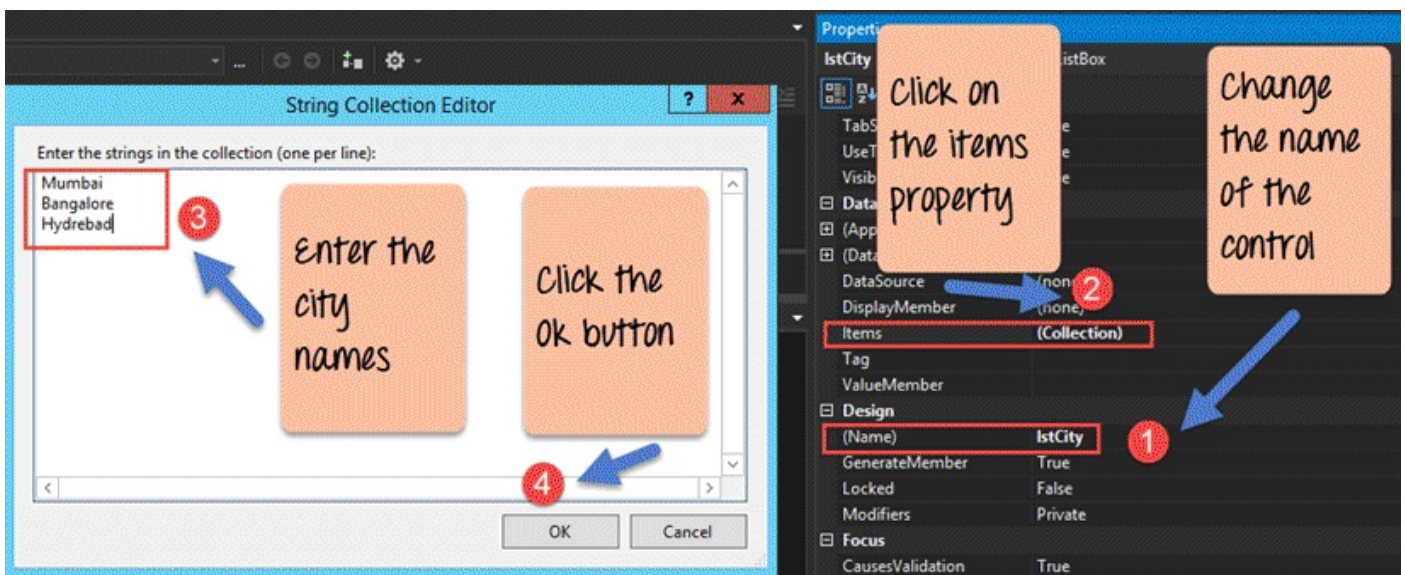
In the output, you can clearly see that the Textboxes were added to the form.

List box – A Listbox is used to showcase a list of items on the Windows form. Let's see how we can implement this with an example shown below. We will add a list box to the form to store some city locations.

Step 1) The first step is to drag the list box control onto the Windows Form from the toolbox as shown below



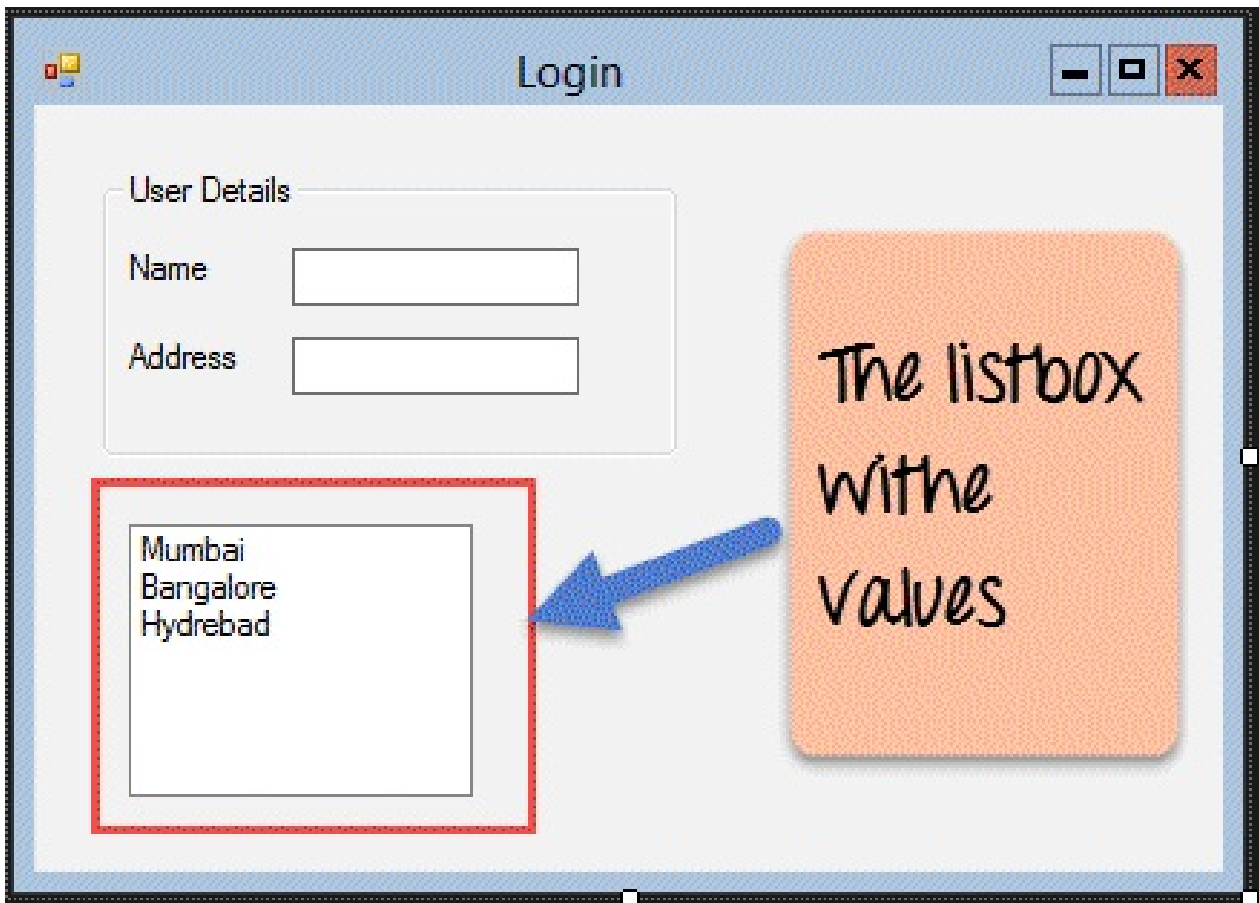
Step 2) Once the list box has been added, go to the properties window by clicking on the list box control.



1. First, change the property of the Listbox box control , in our case we have changed this to IstCity
2. Click on the Items property. This will allow you to add different items which can show up in the list box. In our case, we have selected items “collection”.
3. In the String Collection Editor, which pops up, enter the city names. In our case, we have entered “Mumbai”, “Bangalore” and “Hyderabad”.
4. Finally, click on the ‘OK’ button.

One you make the above changes, you will see the following output

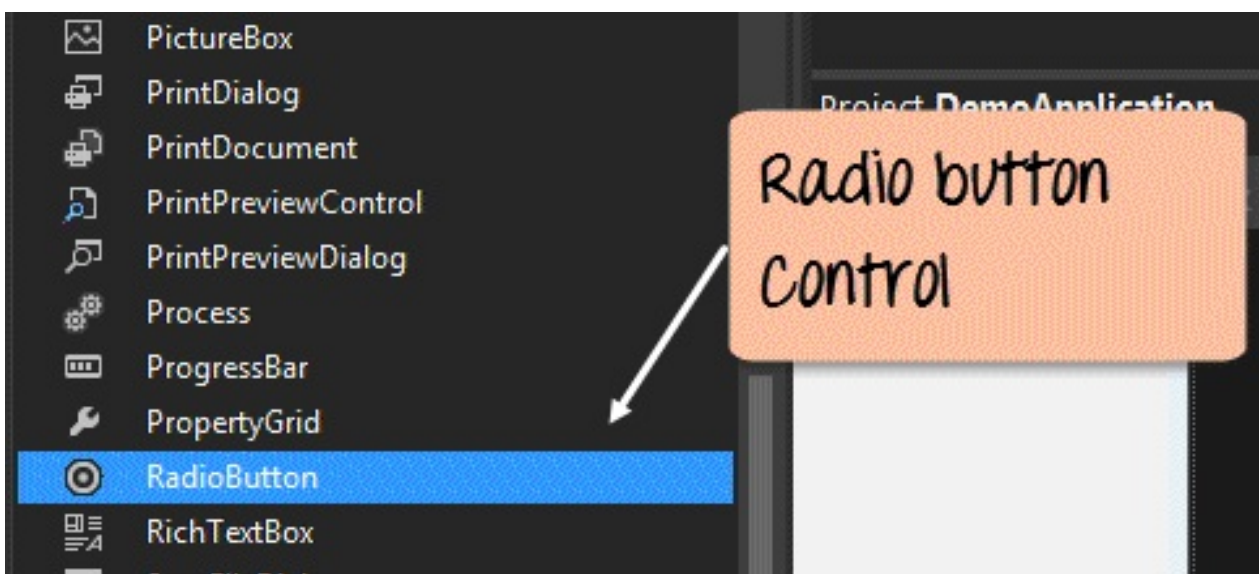
Output:-



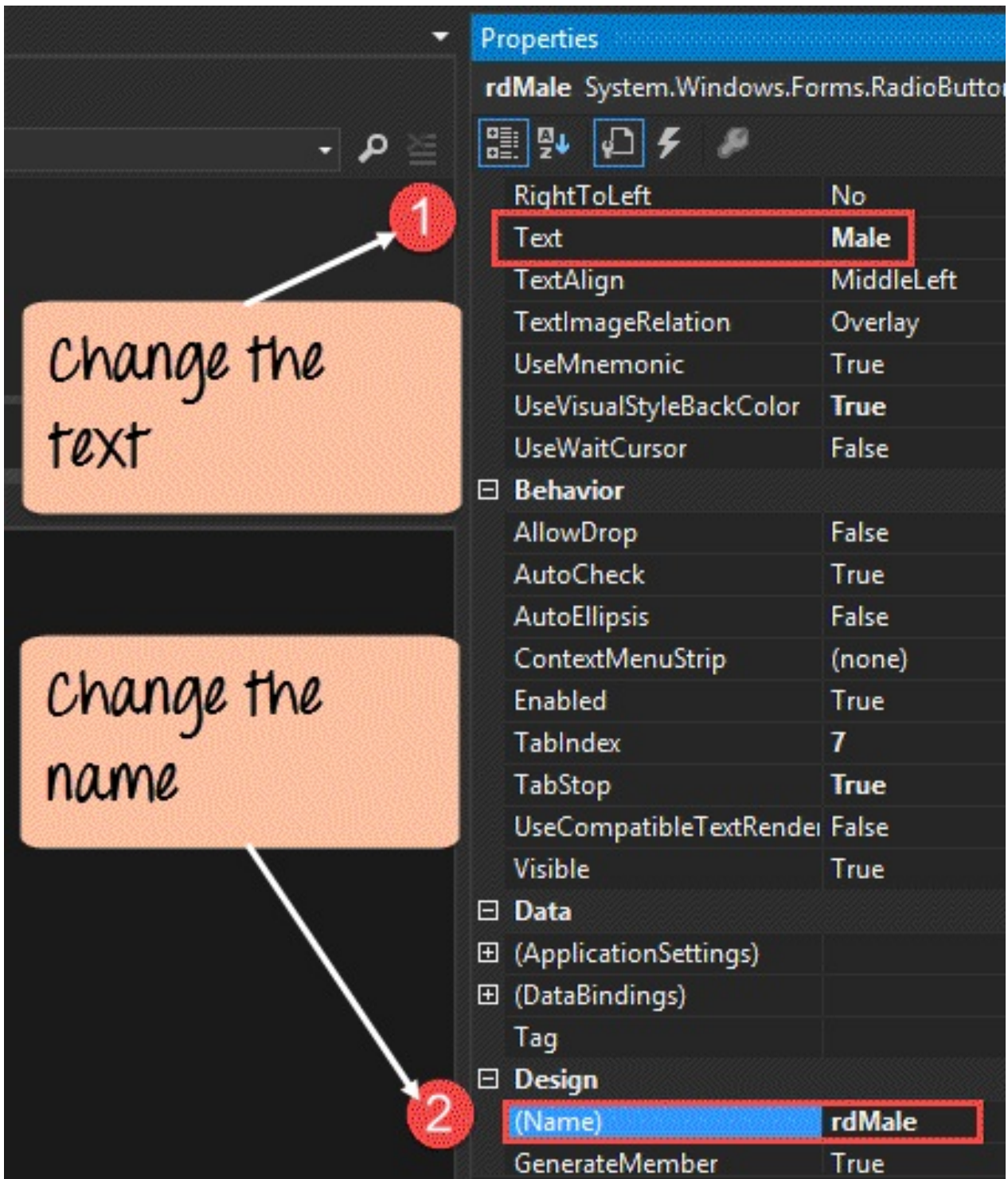
In the output, you can clearly see that the Listbox was added to the form. You can also see that the list box has been populated with the city values.

RadioButton - A Radiobutton is used to showcase a list of items out of which the user can choose one. Let's see how we can implement this with an example shown below. We will add a radio button for a male/female option.

Step 1) The first step is to drag the 'radiobutton' control onto the Windows Form from the toolbox as shown below.



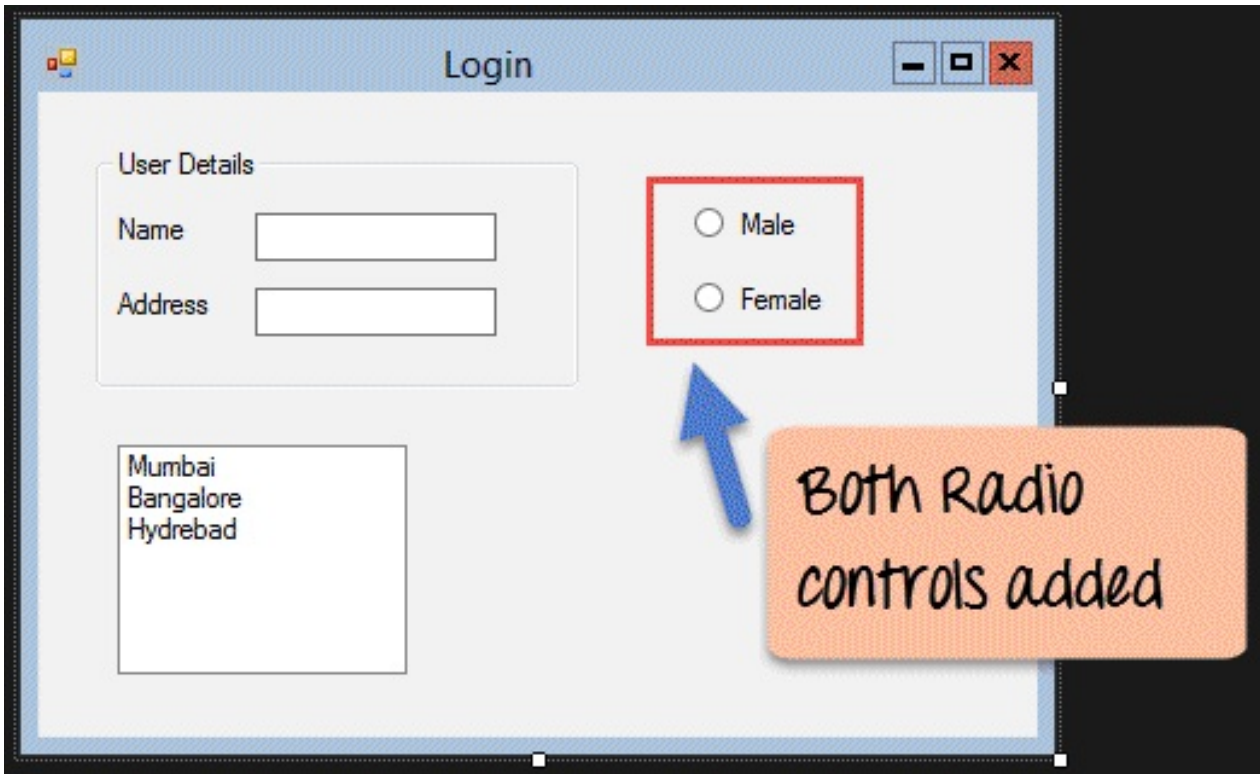
Step 2) Once the Radiobutton has been added, go to the properties window by clicking on the Radiobutton control.



1. First, you need to change the text property of both Radio controls. Go the properties windows and change the text to male of one radiobutton and the text of the other to female.
2. Similarly, change the name property of both Radio controls. Go the properties windows and change the name to 'rdMale' of one radiobutton and to 'rdfemale' for the other one.

One you make the above changes, you will see the following output

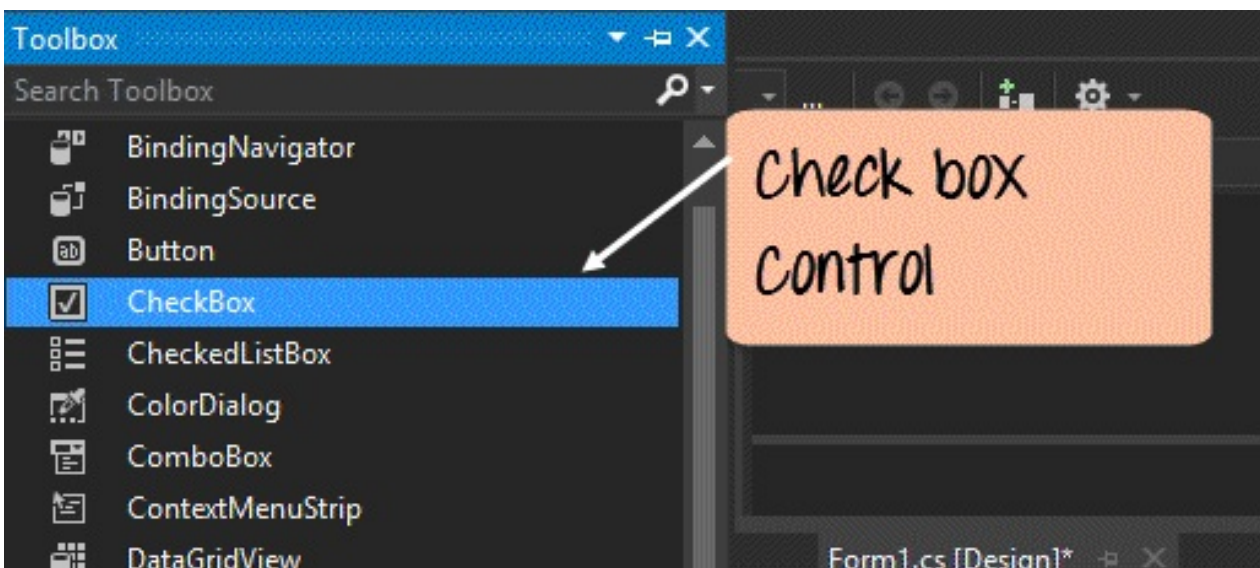
Output:-



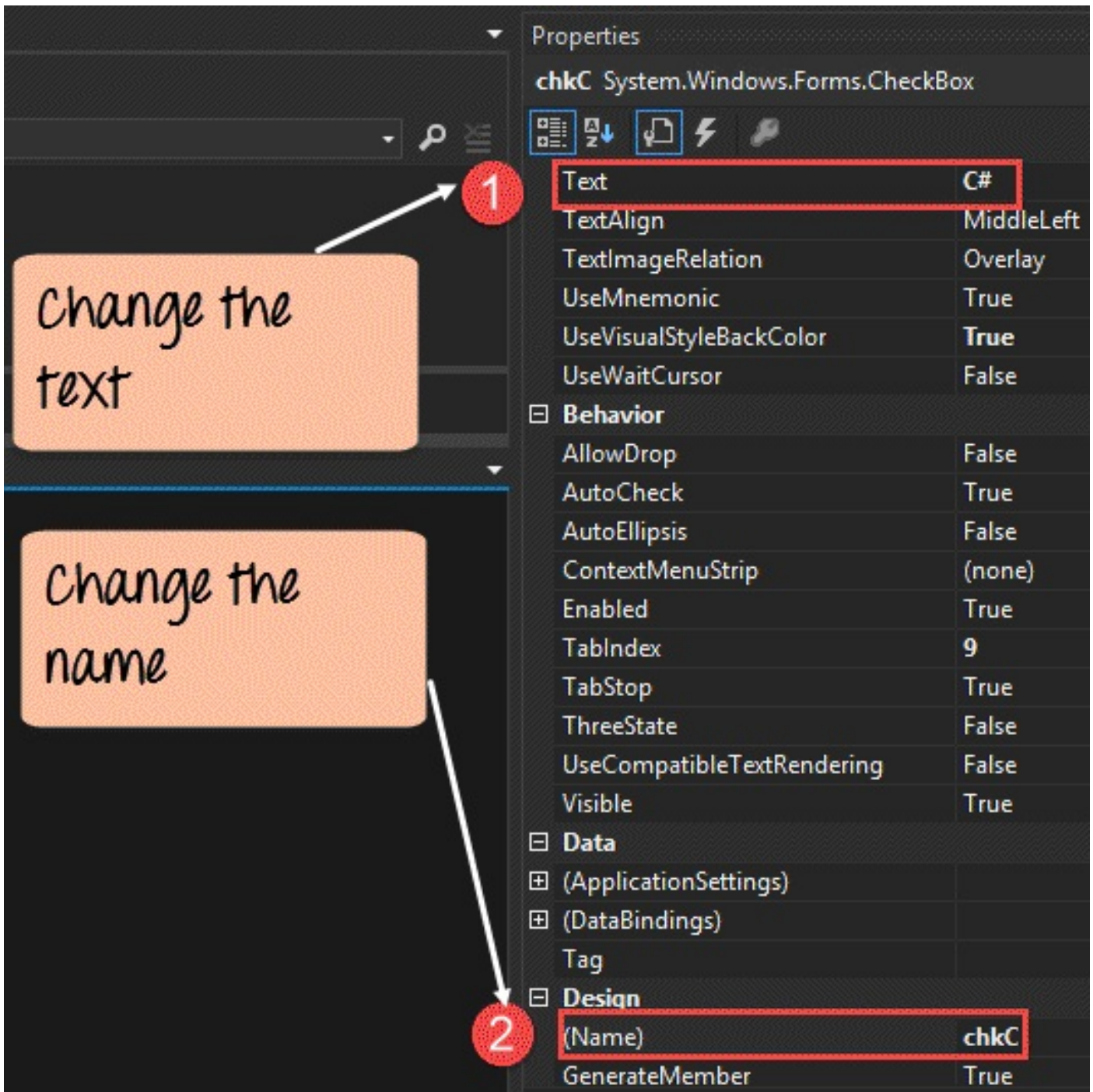
You will see the Radio buttons added to the Windows form.

Checkbox - A checkbox is used to provide a list of options in which the user can choose multiple choices. Let's see how we can implement this with an example shown below. We will add 2 checkboxes to our Windows forms. These checkboxes will provide an option to the user on whether they want to learn C# or ASP.Net.

Step 1) The first step is to drag the checkbox control onto the Windows Form from the toolbox as shown below



Step 2) Once the checkbox has been added, go to the properties window by clicking on the Checkbox control.

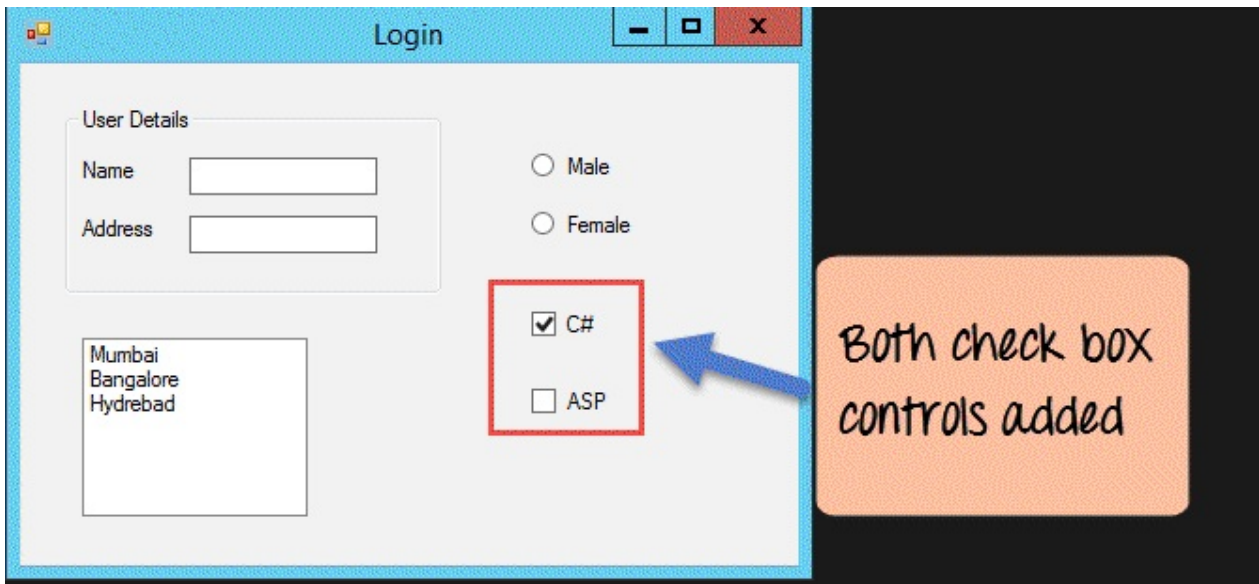


In the properties window,

1. First, you need to change the text property of both checkbox controls. Go the properties windows and change the text to C# and ASP.Net.
2. Similarly, change the name property of both Radio controls. Go the properties windows and change the name to chkC of one checkbox and to chkASP for the other one.

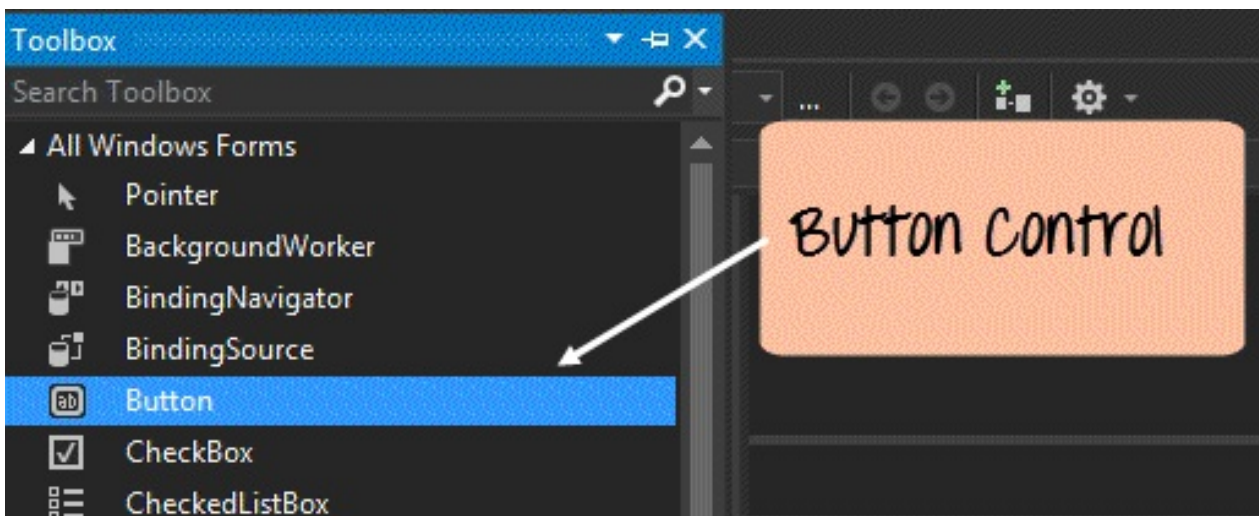
One you make the above changes, you will see the following output

Output:-

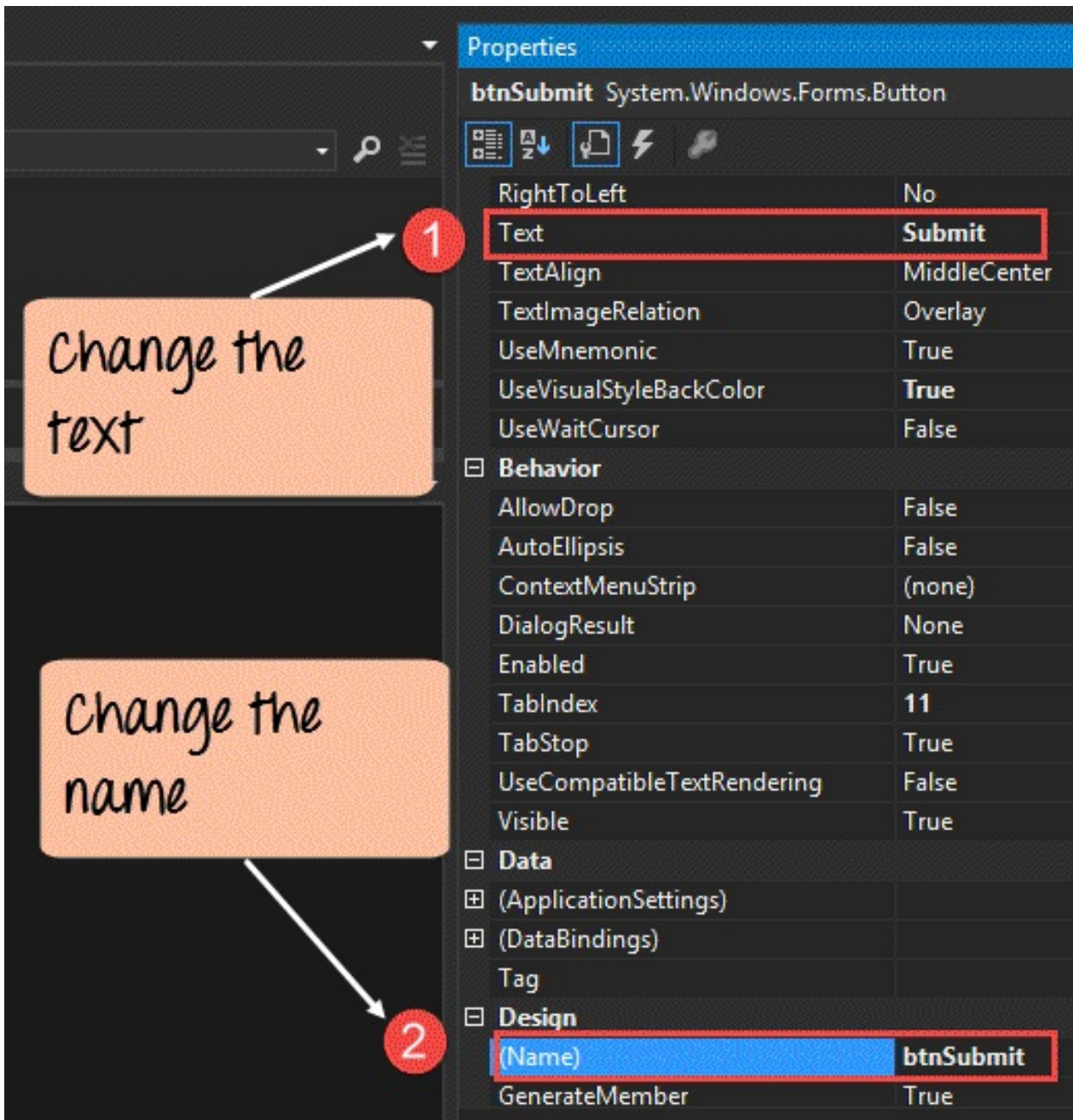


Button - A button is used to allow the user to click on a button which would then start the processing of the form. Let's see how we can implement this with an example shown below. We will add a simple button called 'Submit' which will be used to submit all the information on the form.

Step 1) The first step is to drag the button control onto the Windows Form from the toolbox as shown below



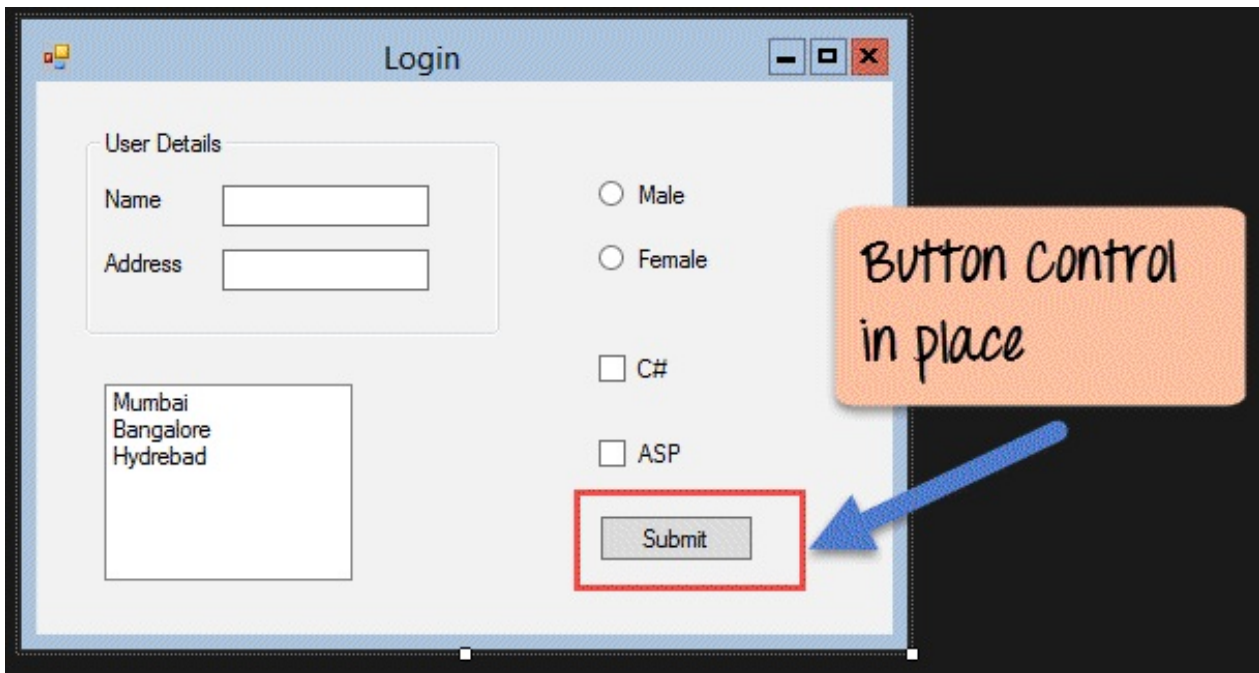
Step 2) Once the Button has been added, go to the properties window by clicking on the Button control.



1. First, you need to change the text property of the button control. Go the properties windows and change the text to 'submit'.
2. Similarly, change the name property of the control. Go the properties windows and change the name to 'btnSubmit'.

One you make the above changes, you will see the following output

Output:-



Congrats, you now have your first basic Windows Form in place. Let's now go to the next topic to see how we can do Event handling for Controls.

C# Event Handling for Controls

When working with windows form, you can add events to controls. An event is something that happens when an action is performed. Probably the most common action is the clicking of a button on a form. In Windows forms , you can add code which can be used to perform certain actions when a button is pressed on the form.

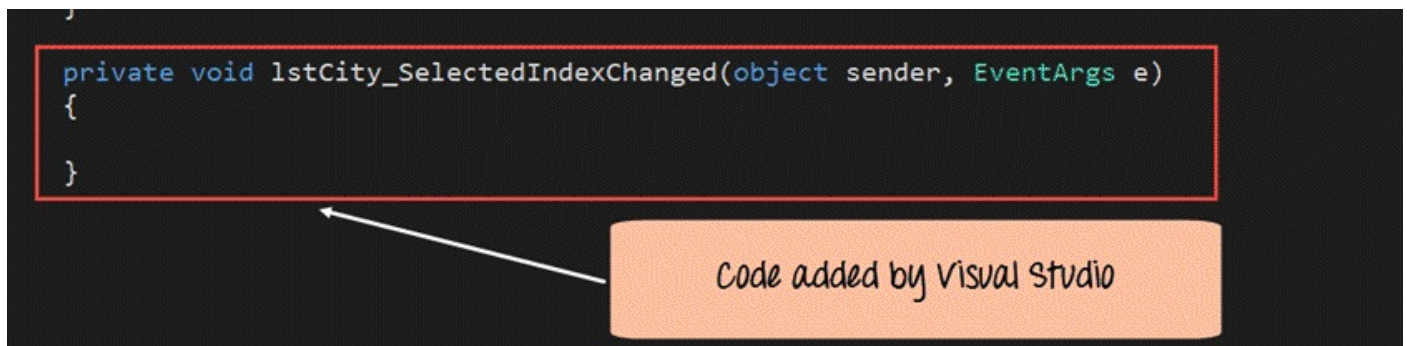
Normally when a button is pressed on a form, it means that some processing should take place.

Let's take a look at one of the event and how it can be handled before we go onto the button event scenario.

The below example will showcase an event for the Listbox control. So whenever an item is selected in the listbox control, a message box should pop up which shows the item selected. Let's perform the following steps to achieve this.

Step 1) Double click on the Listbox in the form designer. By doing this, Visual Studio will automatically open up the code file for the form. And it will automatically add an event method to the code. This event method will be triggered, whenever any item in the listbox is selected.

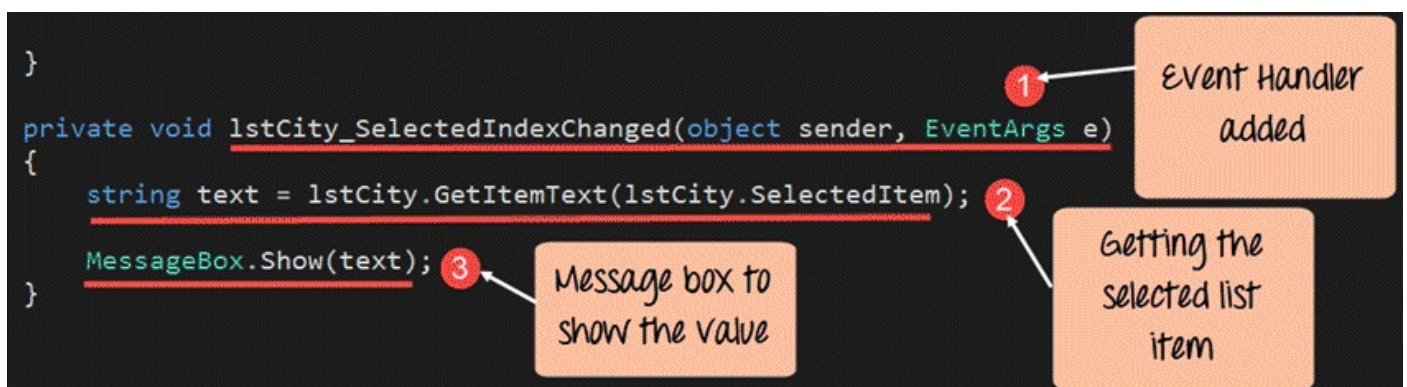
```
private void lstCity_SelectedIndexChanged(object sender, EventArgs e)
{
}
```



The image shows a code snippet for the `lstCity_SelectedIndexChanged` event handler. The code is enclosed in a red box. Below the code, there is a callout box with the text "Code added by Visual Studio" and an arrow pointing to the code block.

Above is the snippet of code which is automatically added by Visual Studio, when you double-click the List box control on the form. Now let's add the below section of code to this snippet of code, to add the required functionality to the listbox event.

```
}
private void lstCity_SelectedIndexChanged(object sender, EventArgs e)
{
    string text = lstCity.GetItemText(lstCity.SelectedItem);
    MessageBox.Show(text);
}
```

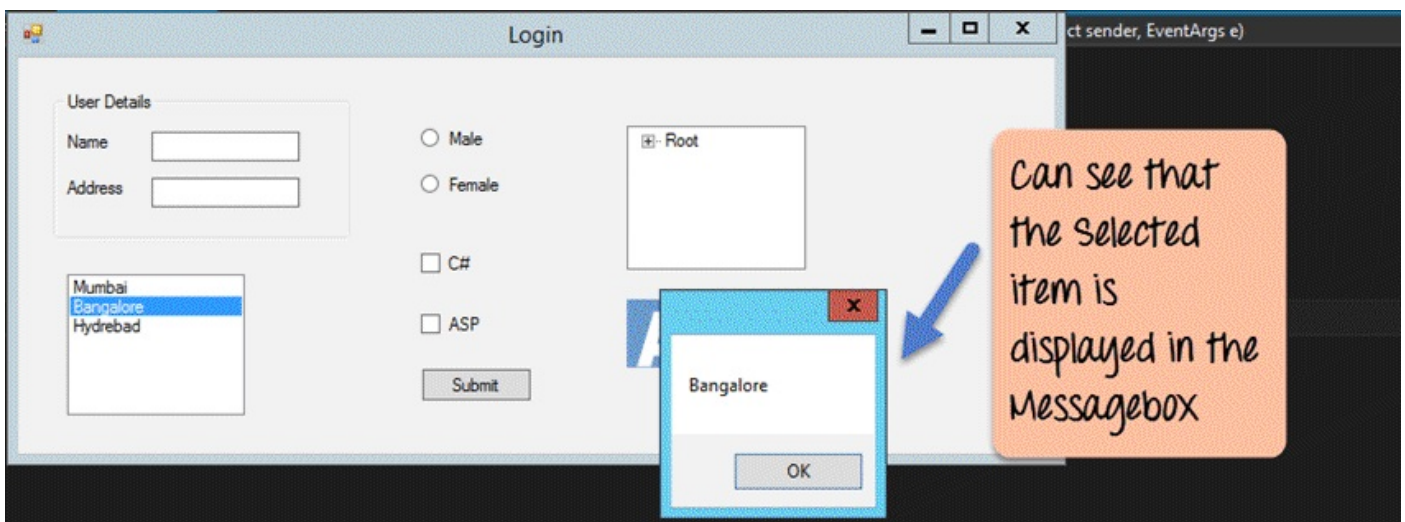


The image shows the same code snippet as above, but with additional annotations. Three callout boxes are present: 1. "Event Handler added" with a red circle containing the number 1, pointing to the event signature. 2. "Getting the selected list item" with a red circle containing the number 2, pointing to the line `string text = lstCity.GetItemText(lstCity.SelectedItem);`. 3. "Message box to show the value" with a red circle containing the number 3, pointing to the line `MessageBox.Show(text);`.

1. This is the event handler method which is automatically created by Visual Studio when you double-click the List box control. You don't need to worry about the complexity of the method name or the parameters passed to the method.
2. Here we are getting the SelectedItem through the lstCity.SelectedItem property. Remember that lstCity is the name of our Listbox control. We then use the GetItemText method to get the actual value of the selected item. We then assign this value to the text variable.
3. Finally, we use the MessageBox method to display the text variable value to the user.

Once you make the above changes, and run the program in Visual Studio you will see the following output

Output:-



From the output, you can clearly see that when any item from the list box is selected, a message box will pop up. This will show the selected item from the listbox.

Now let's look at the final control which is the button click Method. Again this follows the same philosophy. Just double click the button in the Forms Designer and it will automatically add the method for the button event handler. Then you just need to add the below code.

```
    MessageBox.Show(text);
}

private void btnSubmit_Click(object sender, EventArgs e)
{
    String name = txtName.Text;
    String address = txtAddress.Text;
    MessageBox.Show (name + address);
}

```

1. Button Click method

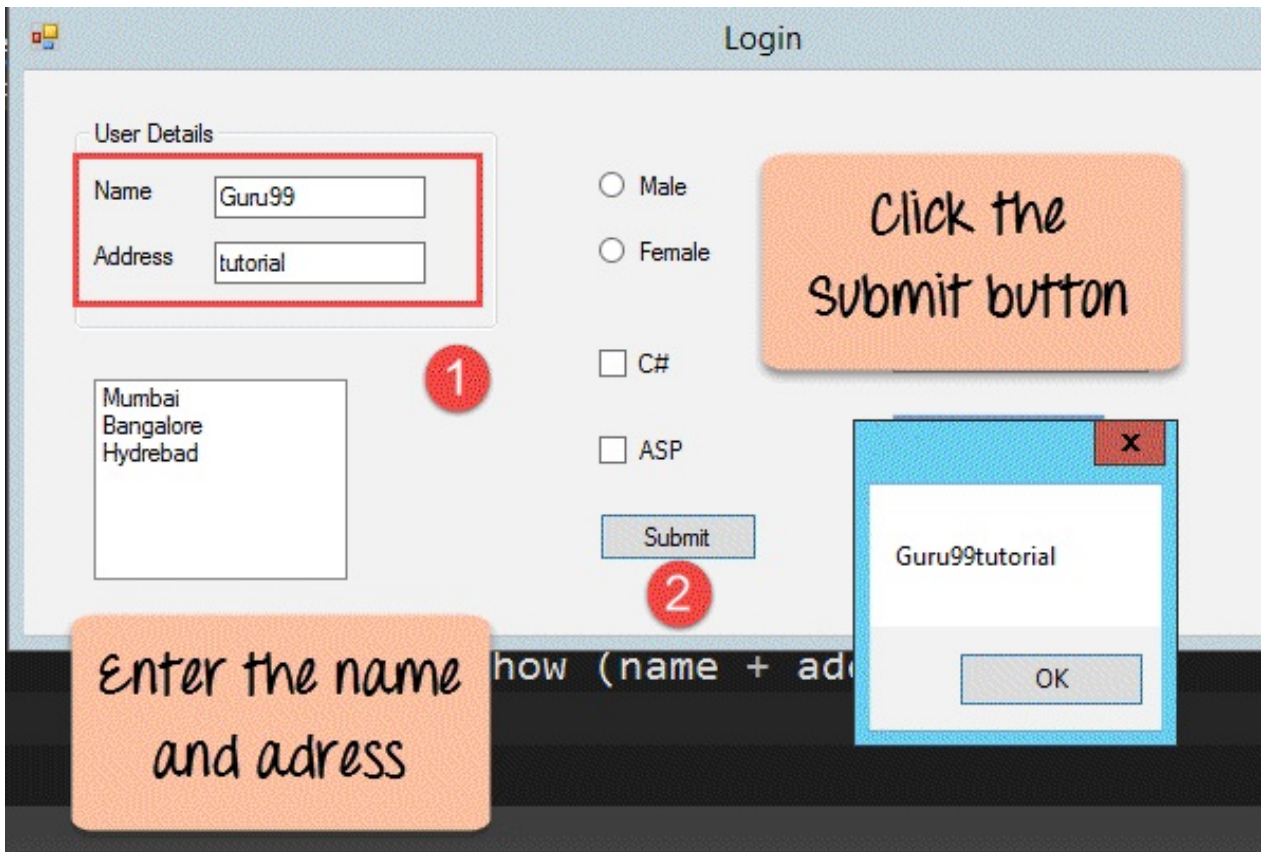
2. Getting the name and address value

3. Message box to show the name and address

1. This is the event handler method which is automatically created by Visual Studio when you double click the button control. You don't need to worry on the complexity of the method name or the parameters passed to the method.
2. Here we are getting values entered in the name and address textbox. The values can be taken from the text property of the textbox. We then assign the values to 2 variables, name, and address accordingly.
3. Finally, we use the MessageBox method to display the name and address values to the user.

Once you make the above changes, and run the program in Visual Studio you will see the following output

Output:-



1. First, enter a value in the name and address field.
2. Then click on the Submit button

Once you click the Submit button , a message box will pop, and it will correctly show you what you entered in the user details section.

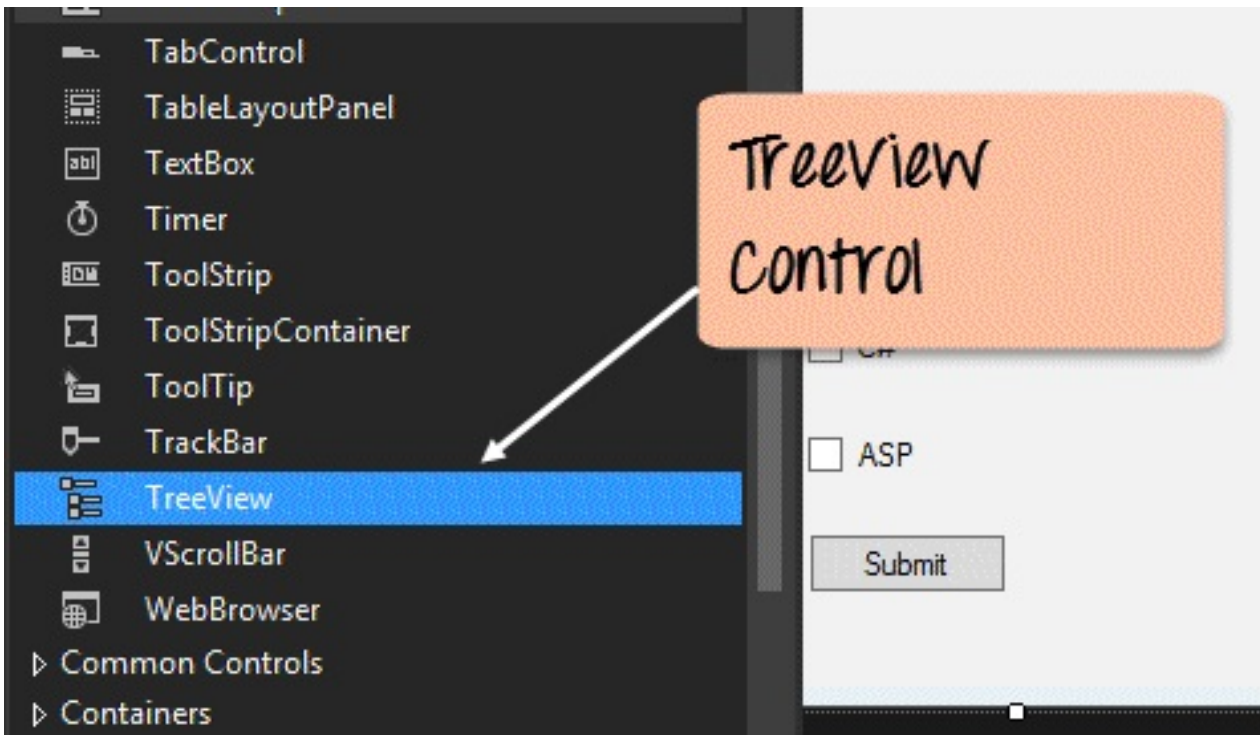
Other Controls

There are 2 further controls we can look at, one is the ‘Tree Control’ and the other is the ‘Image control’. Let’s look at examples of how we can implement these controls

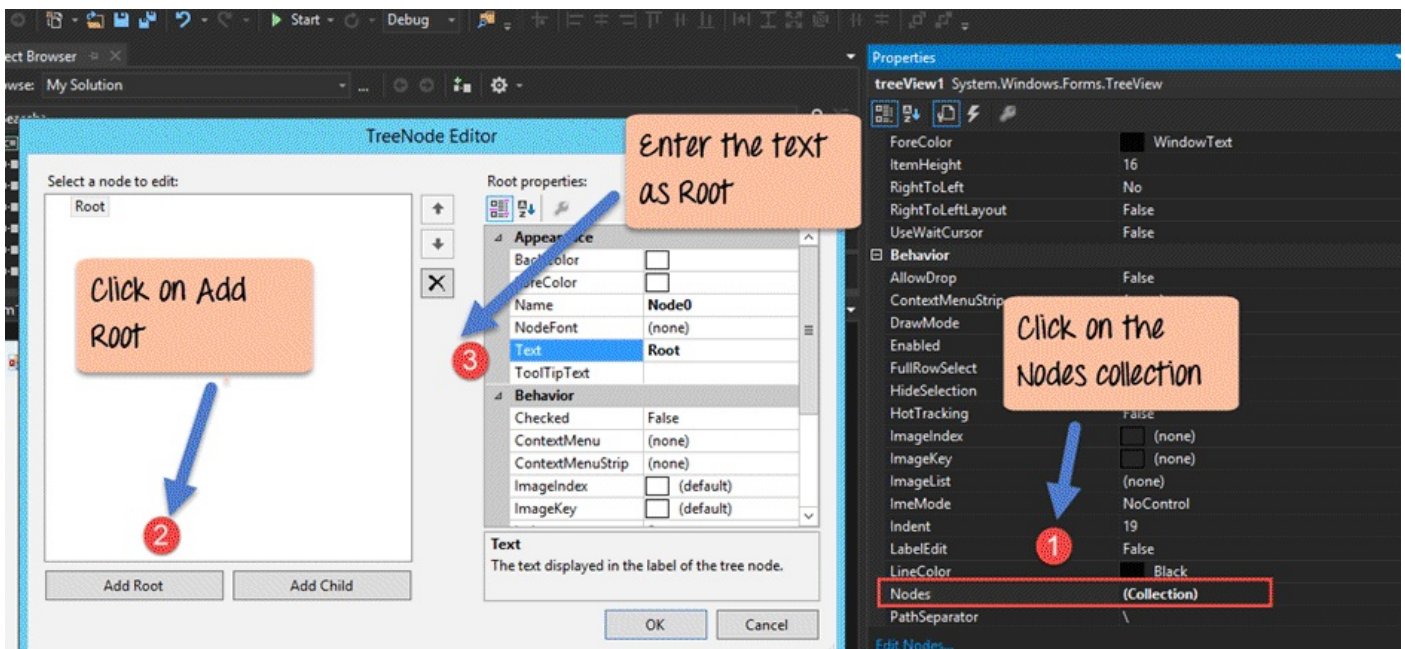
- **TreeControl** – The tree control is used to list down items in a tree like fashion. Probably the best example is when we see the Windows Explorer itself. The folder structure in Windows explorer is like a tree like structure.

Let’s see how we can implement this with an example shown below.

Step 1) The first step is to drag the Tree control onto the Windows Form from the toolbox as shown below

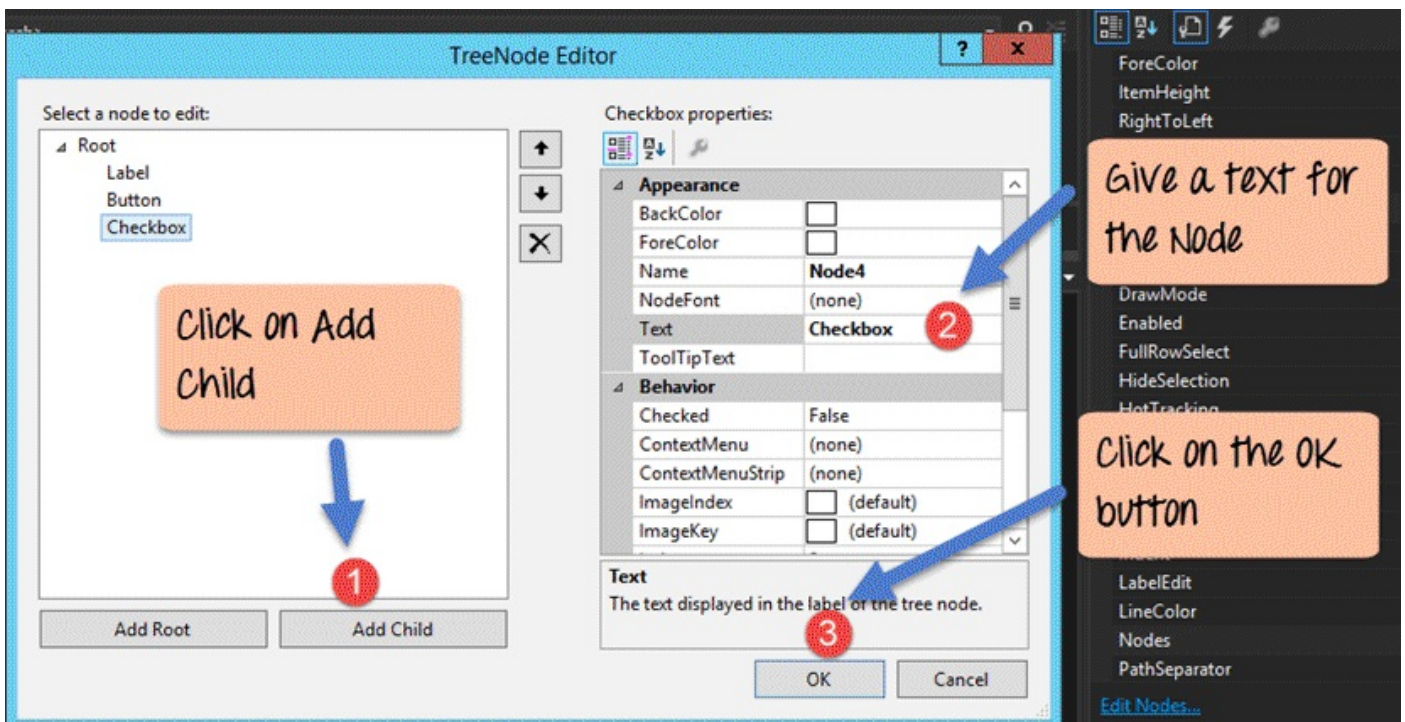


Step 2) The next step is start adding nodes to the tree collection so that it can come up in the tree accordingly. First, let’s follow the below sub-steps to add a root node to the tree collection.



1. Go to the properties toolbox for the tree view control. Click on the Node's property. This will bring up the TreeNode Editor
2. In the TreeNode Editor click on the Add Root button to add a root node to the tree collection.
3. Next, change the text of the Root node and provide the text as Root and click 'OK' button. This will add Root node.

Step 3) The next step is start adding the child nodes to the tree collection. Let's follow the below sub-steps to add child root node to the tree collection.

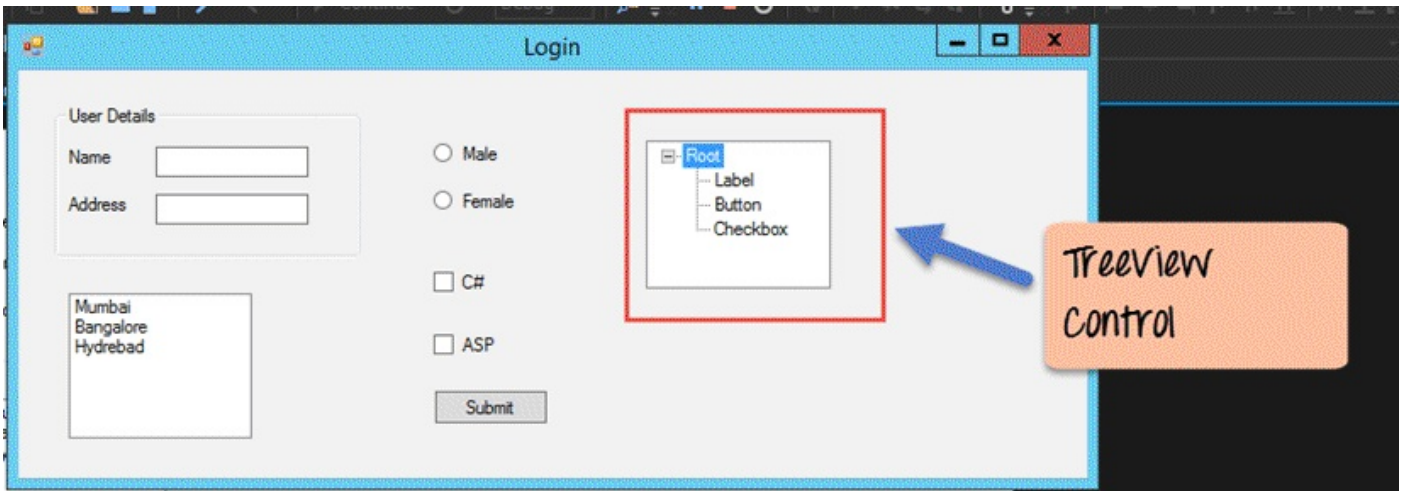


1. First, click on the Add child button. This will allow you to add child nodes to the Tree collection.

2. For each child node, change the text property. Keep on repeating the previous step and this step and add 2 additional nodes. In the end, you will have 3 nodes as shown above, with the text as Label, Button, and Checkbox respectively.
3. Click on the OK button

Once you have made the above changes, you will see the following output.

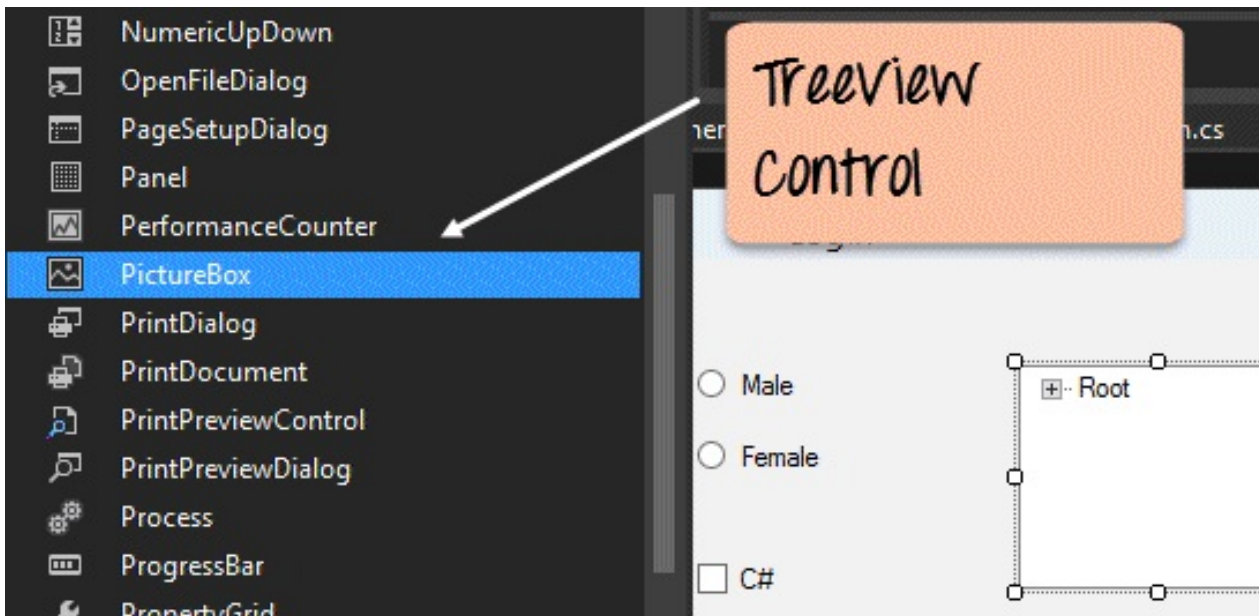
Output:-



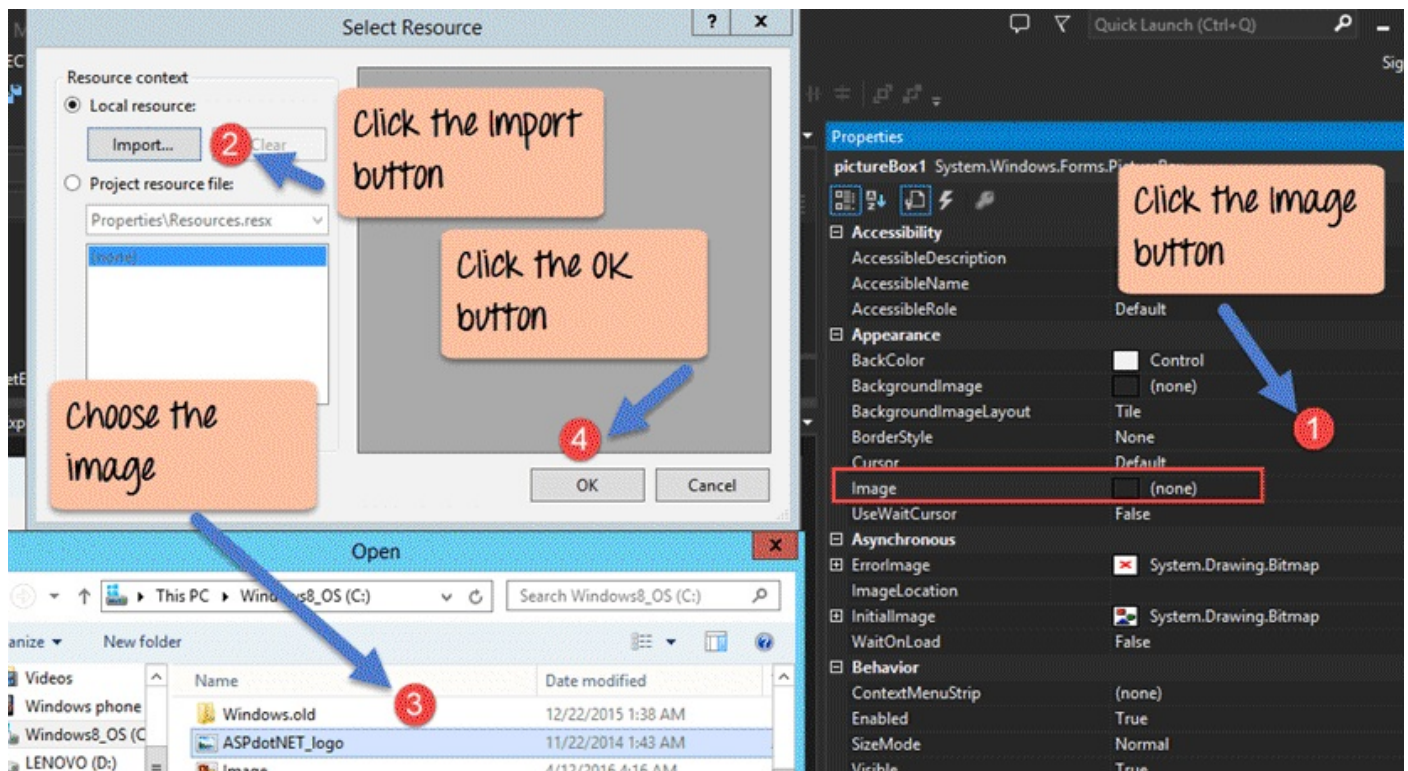
You will actually be able to see the Tree view added to the form. When you run the Windows form application, you can expand the root node and see the child nodes in the list.

- **PictureBox Control** – This control is used to add images to the Windows Forms. Let's see how we can implement this with an example shown below.

Step 1) The first step is to drag the PictureBox control onto the Windows Form from the toolbox as shown below



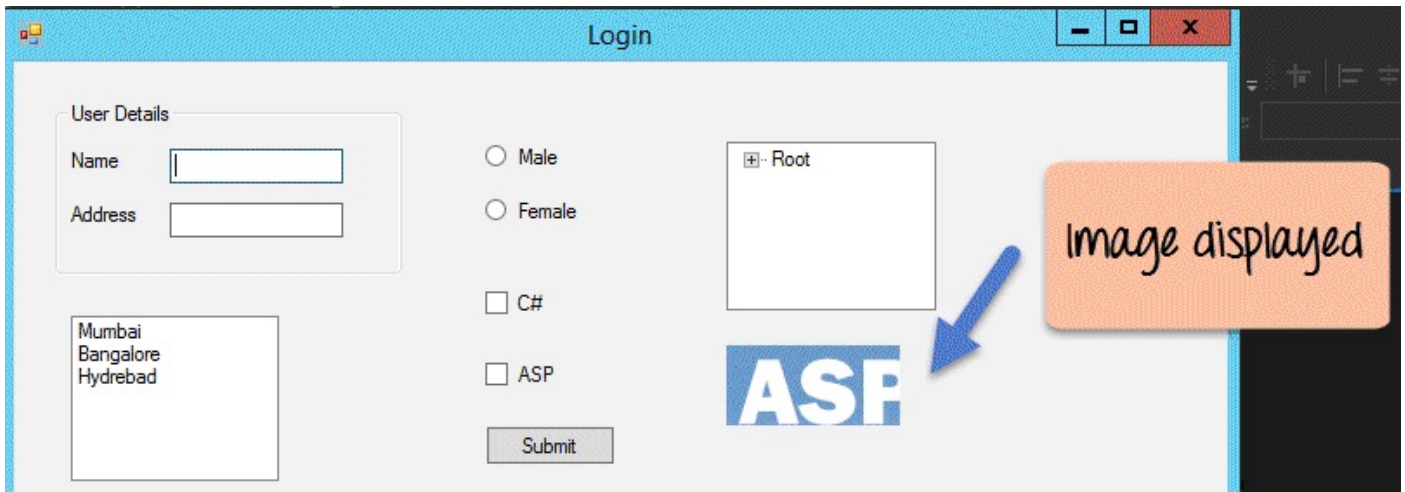
Step 2) The next step is to actually attach an image to the picture box control. This can be done by following the below steps.



1. First, click on the Image property for the PictureBox control. A new window will pop out.
2. In this window, click on the Import button. This will be used to attach an image to the picturebox control.
3. A dialog box will pop up in which you will be able to choose the image to attach the picturebox
4. Click on the OK button

One you make the above changes, you will see the following output

Output:-



From the output, you can clearly see that an image is displayed on the form.

Summary

- A Windows forms application is one that runs on the desktop of a computer. Visual Studio along with C# can be used to create a Windows Forms application.
- Controls can be added to the Windows forms via the Toolbox in Visual Studio. Controls such as labels , checkboxes , radio buttons, etc. can be added to the form via the toolbox.
- One can also use advanced controls like the treeview control and the picturebox control.
- Event handlers are used to respond to events generated from controls. The most common one is the one added for the button clicked event.

Chapter 7: Database Access

Accessing Data from a database is one of the important aspects of any programming language. It is an absolute necessity for any programming language to have the ability to work with databases. C# is no different.

It has the ability to work with different types of databases. It can work with the most common databases such as Oracle and Microsoft SQL Server.

It also has the ability to work with new forms of databases such as MongoDB and MySQL.

Fundamentals of Database connectivity

C# and .Net has the ability to work with a majority of databases, the most common being Oracle and Microsoft SQL Server. But with every database, the logic behind working with all of them is mostly the same.

In our examples, we will look at working the Microsoft SQL Server as our database. For learning purposes, one can download and use the **Microsoft SQL Server Express Edition**, which is a free database software provided by Microsoft.

In working with databases, the following are the concepts which are common across all databases.

1. **Connection** – To work with the data in a database, the first obvious step is the connection. The connection to a database normally consists of the below-mentioned parameters.
 - a. **Database name or Data Source** – The first important parameter is the database name to which the connection needs to be established. Each connection can only work with one database at a time.
 - b. **Credentials** – The next important aspect is the username and password which needs to be used to establish a connection to the database. It ensures that the username and password have the necessary privileges to connect to the database.
 - c. **Optional parameters** - For each database type, you can specify optional parameters to provide more information on how .net should handle the connection to the database. For example, one can specify a parameter for how long the connection should stay active. If no operation is performed for a specific period of time, then the parameter would determine if the connection has to be closed.
2. **Selecting data from the database** – Once the connection has been established, the next important aspect is to fetch the data from the database. C# has the ability to execute 'SQL' select command against the database. The 'SQL' statement can be used to fetch data from a specific table in the database.
3. **Inserting data into the database** – C# can also be used to insert records into the database. Values can be specified in C# for each row that needs to be inserted into the database.
4. **Updating data into the database** – C# can also be used to update existing records into the database. New values can be specified in C# for each

row that needs to be updated into the database.

5. **Deleting data from a database** – C# can also be used to delete records into the database. Select commands to specific which rows need to be deleted can be specified in C#.

Ok, now that we have seen the theory of each operations, let's jump into the further sections to look at how we can perform database operations in C#.

Connections in .Net

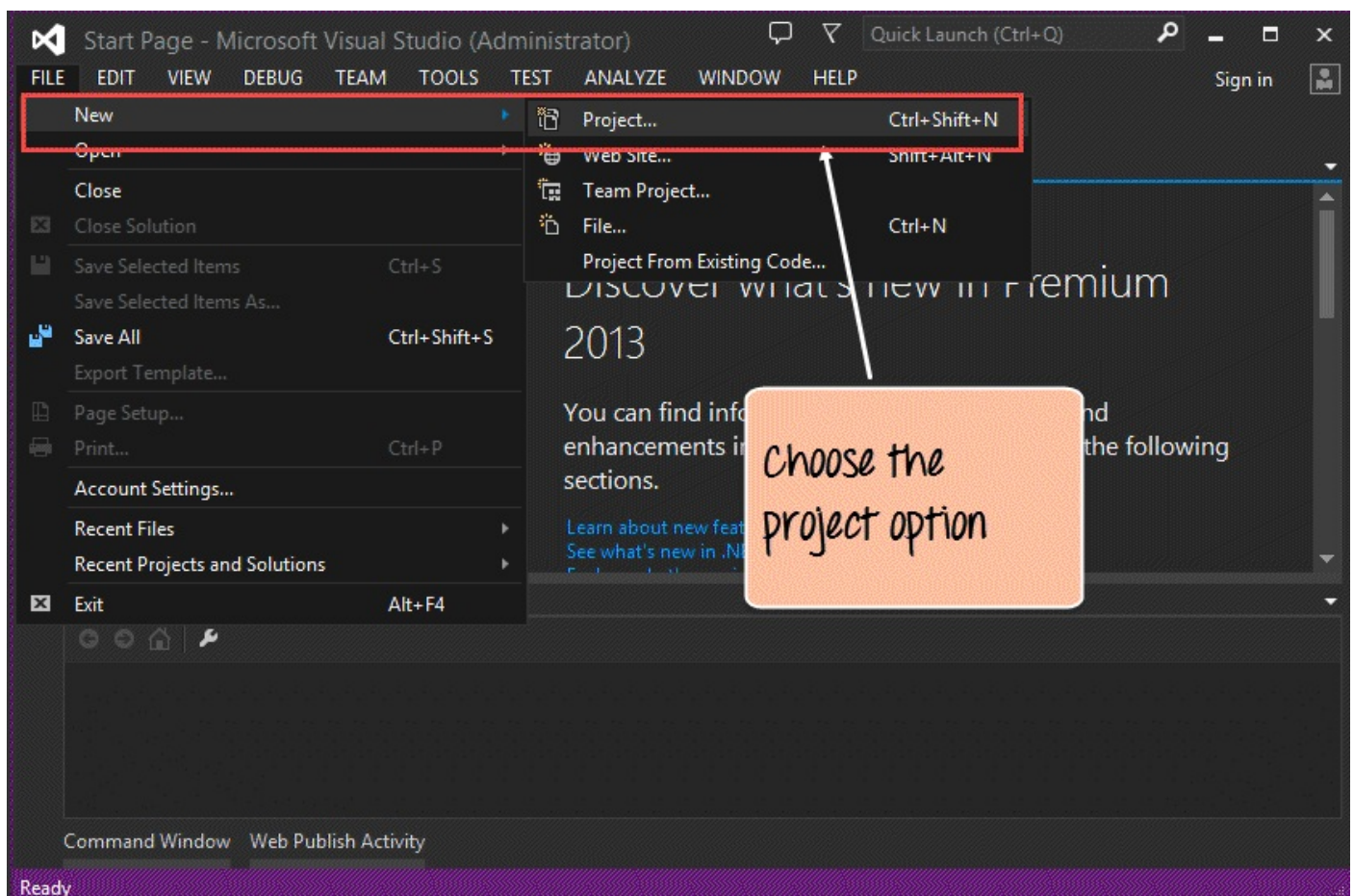
Let's now look at the code, which needs to be kept in place to create a connection to a database. In our example, we will connect to a database which has the name of Demodb. The credentials used to connect to the database are given below

- **Username – sa**
- **Password – demo123**

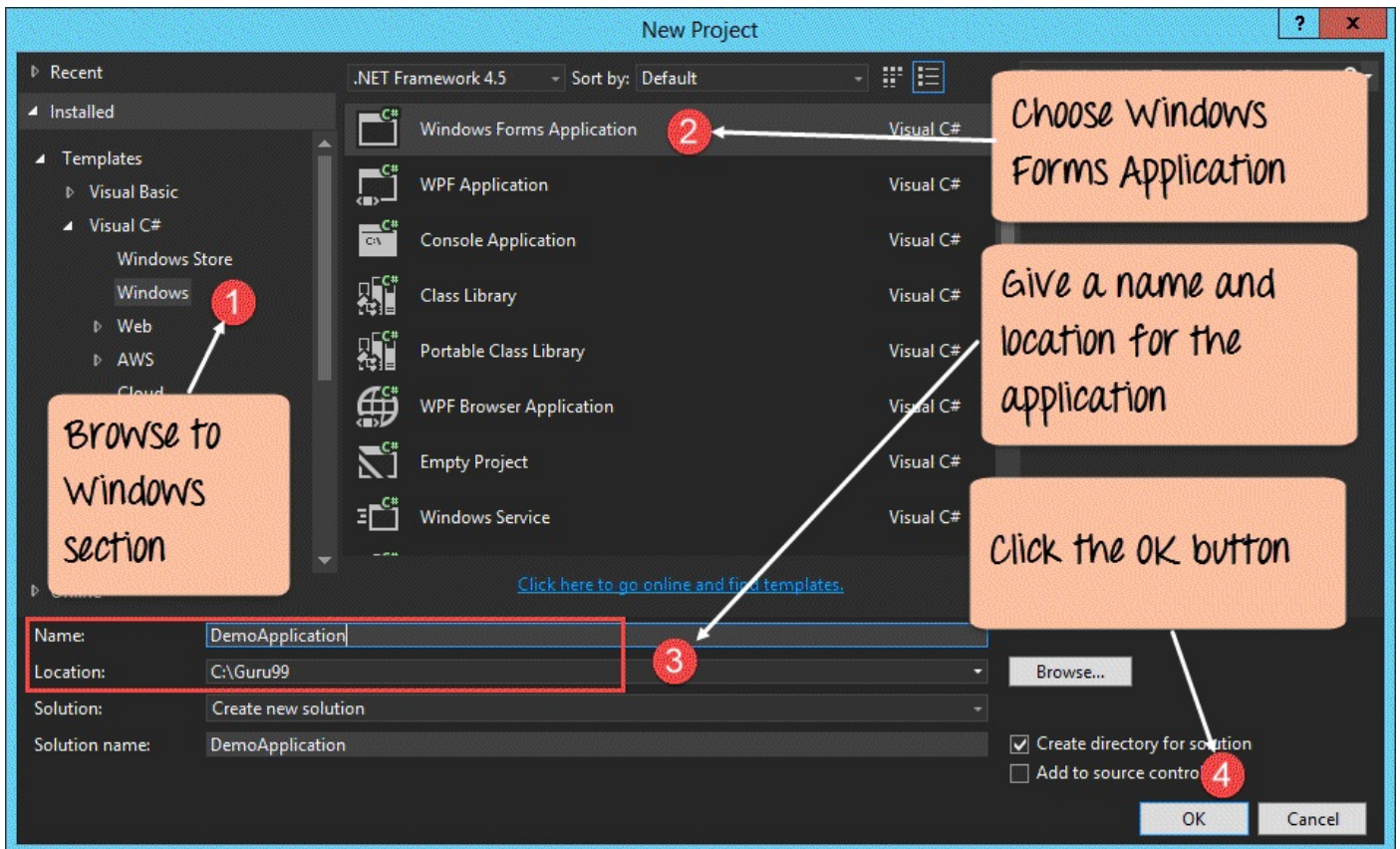
We will see a simple Windows forms application to work with databases. We will have a simple button called “Connect” which will be used to connect to the database.

So let's follow the below steps to achieve this

Step 1) The first step involves the creation of a new project in Visual Studio. After launching Visual Studio, you need to choose the menu option New->Project.

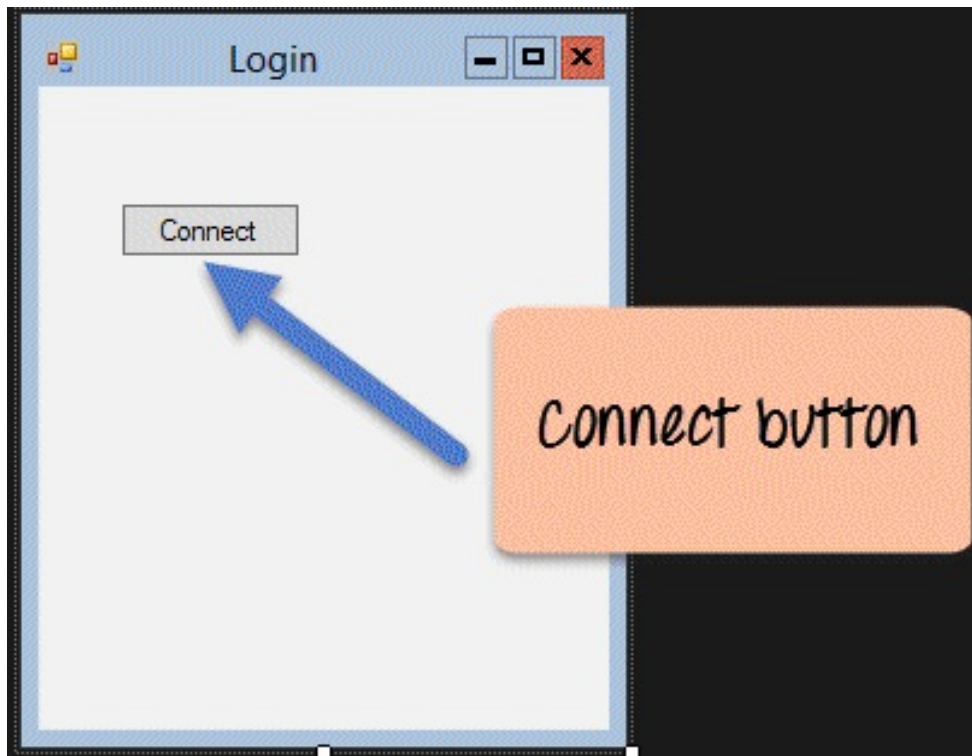


Step 2) The next step is to choose the project type as a Windows Forms application. Here, we also need to mention the name and location for our project.



1. In the project dialog box, we can see various options for creating different types of projects in Visual Studio. Click the Windows option on the left-hand side.
2. When we click the Windows options in the previous step, we will be able to see an option for Windows Forms Application. Click this option.
3. We then give a name for the application which in our case is “DemoApplication”. We also need to provide a location to store our application.
4. Finally, we click the ‘OK’ button to let Visual Studio to create our project.

Step 3) Now add a button from the toolbox to the Windows form. Put the text property of the Button as Connect. This is how it will look like



Step 4) Now double click the form so that an event handler is added to the code for the button click event. In the event handler, add the below code.

```

Set
connection
string
    private void button1_Click(object sender, EventArgs e)
    {
        string connetionString;
        SqlConnection cnn ;
        connetionString = @"Data Source=WIN-50GP30FGO75;Initial Catalog=Demodb
        ;User ID=sa;Password=demo123";
        cnn = new SqlConnection(connetionString);
        cnn.Open();
        MessageBox.Show ("Connection Open ! ");
        cnn.Close();
    }
    
```

Variable declaration

Assign connection

open connection

Close connection

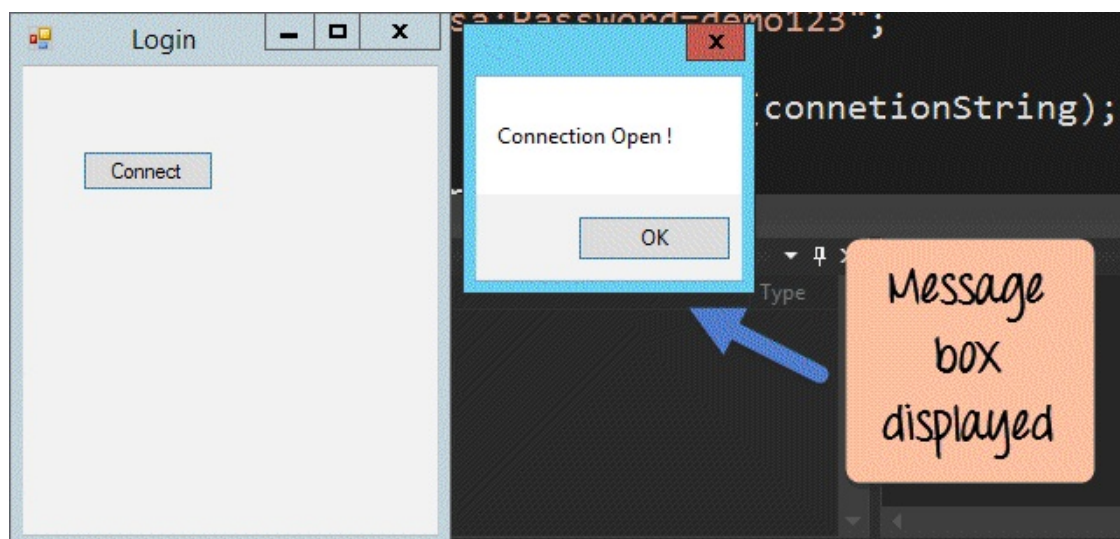
Code Explanation:-

1. The first step is to create variables, which will be used to create the connection string and the connection to the SQL Server database.
2. The next step is to actually create the connection string. The connecting string needs to be specified in a correct way in order for C# to understand the connection string. The connection string consists of the following parts
 - a. Data Source – This is the name of the server on which the database resides. In our case, it resides on a machine called WIN- 50GP30FGO75.
 - b. The Initial Catalog is used to specify the name of the database

- c. The UserID and Password are the credentials required to connect to the database.
3. Next, we assign the connecting string to the variable cnn. The variable cnn, which is of type SqlConnection is actually used to establish the connection to the database.
4. Next, we use the Open method of the cnn variable to open a connection to the database. We then just display a message to the user that the connection is established.
5. Once the operation is completed successfully, we then close the connection to the database. It is always a good practice to close the connection to the database if nothing else is required to be done on the database.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



When you click on “connect” button, from the output you can clearly see that the database connection was established. Hence, the message box was displayed.

Access data

To showcase how data can be accessed using C#, let us assume that we have the following artifacts in our database.

1. A table called demotb. This table will be used to store the ID and names of various Tutorials.
2. The table will have 2 columns, one called "TutorialID" and the other called "TutorialName."
3. For the moment, the table will have 2 rows as shown below.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code in our form, so that we can query for this data and display the information via a MessageBox. Note that all the code entered below is a continuation of the code written for the data connection in the previous section.

Step 1) Let's split the code into 2 parts so that it will be easy to understand for the user.

- The first will be to construct our "select" statement, which will be used to read the data from the database.
- We will then execute the "select" statement against the database and fetch all the table rows accordingly.

```
cnn.Open();  
SqlCommand command;  
SqlDataReader dataReader;  
String sql,Output="";  
  
sql = "Select TutorialID,TutorialName from demotb";  
command = new SqlCommand(sql, cnn);  
dataReader = command.ExecuteReader();  
  
while (dataReader.Read())  
{  
Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "\n";  
}  
  
MessageBox.Show(Output);
```

Code Explanation:-

1. The first step is to create the following variables
 - a. SQLCommand – The 'SQLCommand' is a class defined within C#. This class is used to perform operations of reading and writing into the database. Hence, the first step is to make sure that we create a variable type of this class. This variable will then be used in subsequent steps of reading data from our database.
 - b. The datareader object is used to get all the data specified by the SQL query. We can then read all the table rows one by one using the data reader.
 - c. We then define 2 string variables, one is "SQL" to hold our SQL command string. The next is the "Output" which will contain all the table values.
2. The next step is to actually define the SQL statement, which will be used against our database. In our case, it is "Select TutorialID, TutorialName from demotb". This will fetch all the rows from the table demotb.
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string.
4. Next, we will execute the data reader command, which will fetch all the rows from the demotb table.
5. Now that we have all the rows of the table with us, we need a mechanism to access the row one by one. For this, we will use the while statement. The while statement will be used to access the rows from the data reader one at a time. We then use the GetValue method to get the value of TutorialID and TutorialName.

Step 2) In the final step, we will just display the output to the user and close all the objects related to the database operation.

```
while (dataReader.Read())
{
    Output = Output + dataReader.GetValue(0) + " - " + dataReader.GetValue(1) + "\n";
}
MessageBox.Show(Output);
dataReader.Close();
command.Dispose();
cnn.Close();
}
```

1

2

Display the output to the user

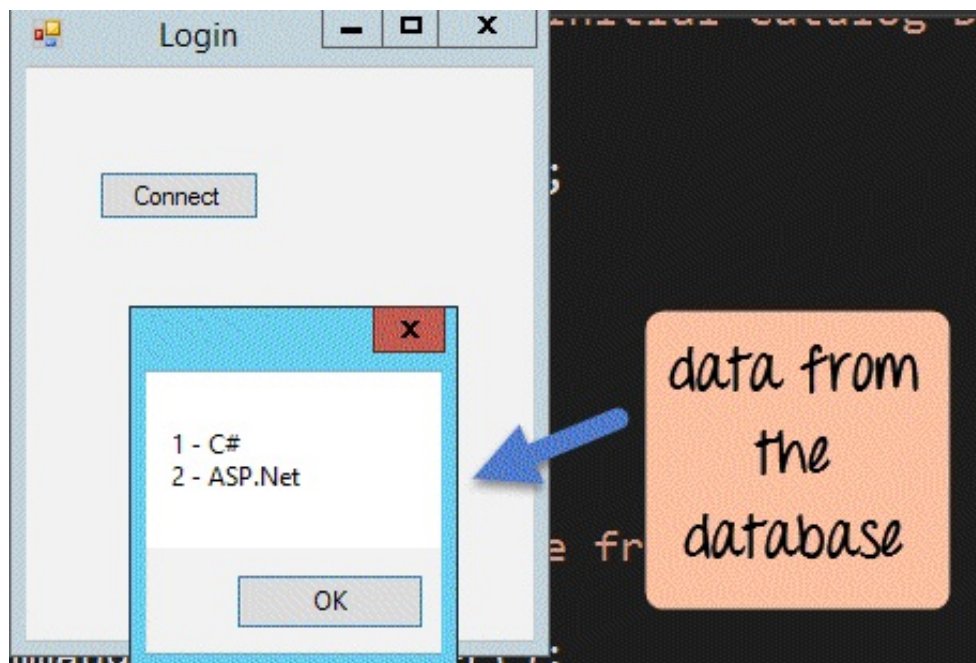
Close all objects

Code Explanation:-

1. We will continue our code by displaying the value of the Output variable using the MessageBox. The Output variable will contain all the values from the demotb table.
2. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is run using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



From the output, you can clearly see that the program was able to get the values from the database. The data is then displayed in the message box.

C# Insert Database

Just like Accessing data, C# has the ability to insert records into the database as well. To showcase how to insert records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net

Let's change the code in our form, so that we can insert the following row into the table

TutorialID	TutorialName
3	VB.Net

So let's add the following code to our program. The below code snippet will be used to insert an existing record in our database.

```
SqlCommand command;
SqlDataAdapter adapter = new SqlDataAdapter();
String sql="";

sql = "Insert into demotb (TutorialID,TutorialName) values(3,' " + "VB.Net" + "')";

command = new SqlCommand(sql, cnn);

adapter.InsertCommand = new SqlCommand(sql, cnn);
adapter.InsertCommand.ExecuteNonQuery();

command.Dispose();
cnn.Close();
```

Define variables

Define the insert statement

Define the sqlcommand

Associate the insert command

Close all objects

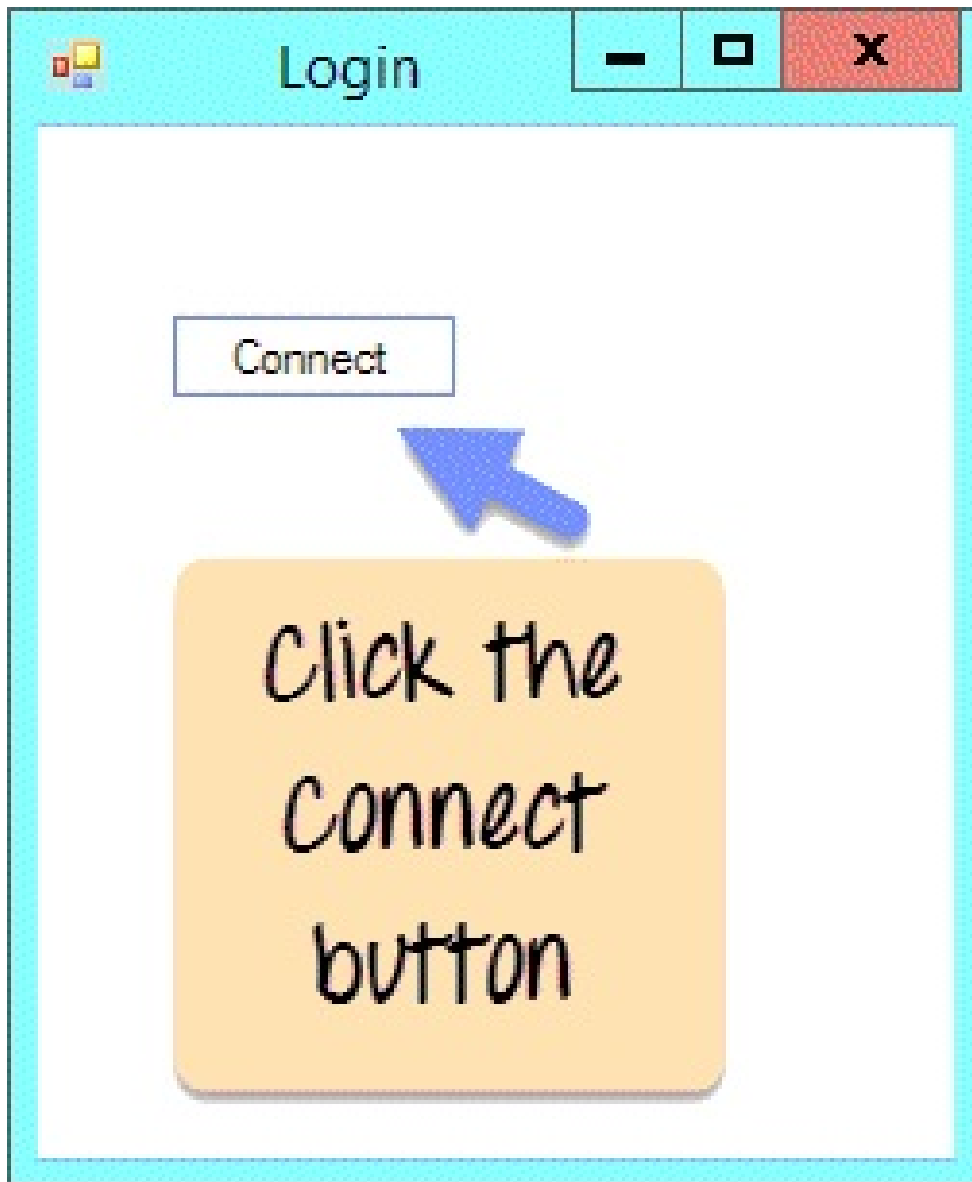
Code Explanation:-

1. The first step is to create the following variables
 - a. **SqlCommand** – This data type is used to define objects which are used to perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
 - b. The dataadapter object is used to perform specific SQL operations such as insert, delete and update commands.
 - c. We then define a string variable, which is “SQL” to hold our SQL command string.

2. The next step is to actually define the SQL statement which will be used against our database. In our case, we are issuing an insert statement, which will insert the record of TutorialID=1 and TutorialName=VB.Net
3. Next, we create the command object which is used to execute the SQL statement against the database. In the SQL command, you have to pass the connection object and the SQL string
4. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the ExecuteNonQuery method which is used to execute the Insert statement against our database. The 'ExecuteNonQuery' method is used in C# to issue any DML statements against the database. By DML statements, we mean the insert, delete, and update operation. In C# , if you want to issue any of these statements against a table , you need to use the ExecuteNonQuery method.
5. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you actually go to SQL Server Express and see the rows in the demotb table, you will see the row inserted as shown below

A screenshot of the SQL Server Express Results window. The window has two tabs: "Results" and "Messages". The "Results" tab is active, showing a table with three columns: "TutorialName" and "TutorialID". The table contains three rows of data. The third row, with "VB.Net" and "3", is highlighted with a red border. A blue arrow points from an orange callout box to the third row. The callout box contains the text "Row added" in a handwritten font.

	TutorialName	TutorialID
1	C#	1
2	ASP.Net	2
3	VB.Net	3

C# Update Database

Just like Accessing data, C# has the ability to update existing records from the database as well. To showcase how to update records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net

Let's change the code in our form, so that we can update the following row. The old row value is TutorialID as "3" and Tutorial Name as "VB.Net". Which we will update it to "VB.Net complete" while the row value for Tutorial ID will remain same.

Old row

TutorialID	TutorialName
3	VB.Net

New row

TutorialID	TutorialName
3	VB.Net complete

So let's add the following code to our program. The below code snippet will be used to update an existing record in our database.

```
SqlCommand command;  
SqlDataAdapter adapter = new SqlDataAdapter();  
String sql="";  
  
sql = "Update demotb set TutorialName='"+"VB.Net Complete"+"'' where TutorialID=3";  
  
command = new SqlCommand(sql, cnn);  
adapter.UpdateCommand = new SqlCommand(sql, cnn);  
adapter.UpdateCommand.ExecuteNonQuery();  
  
command.Dispose();  
cnn.Close();
```

1. Define variables

2. Define the update statement

3. Define the sqlcommand

4. Associate the update command

5. Close all objects

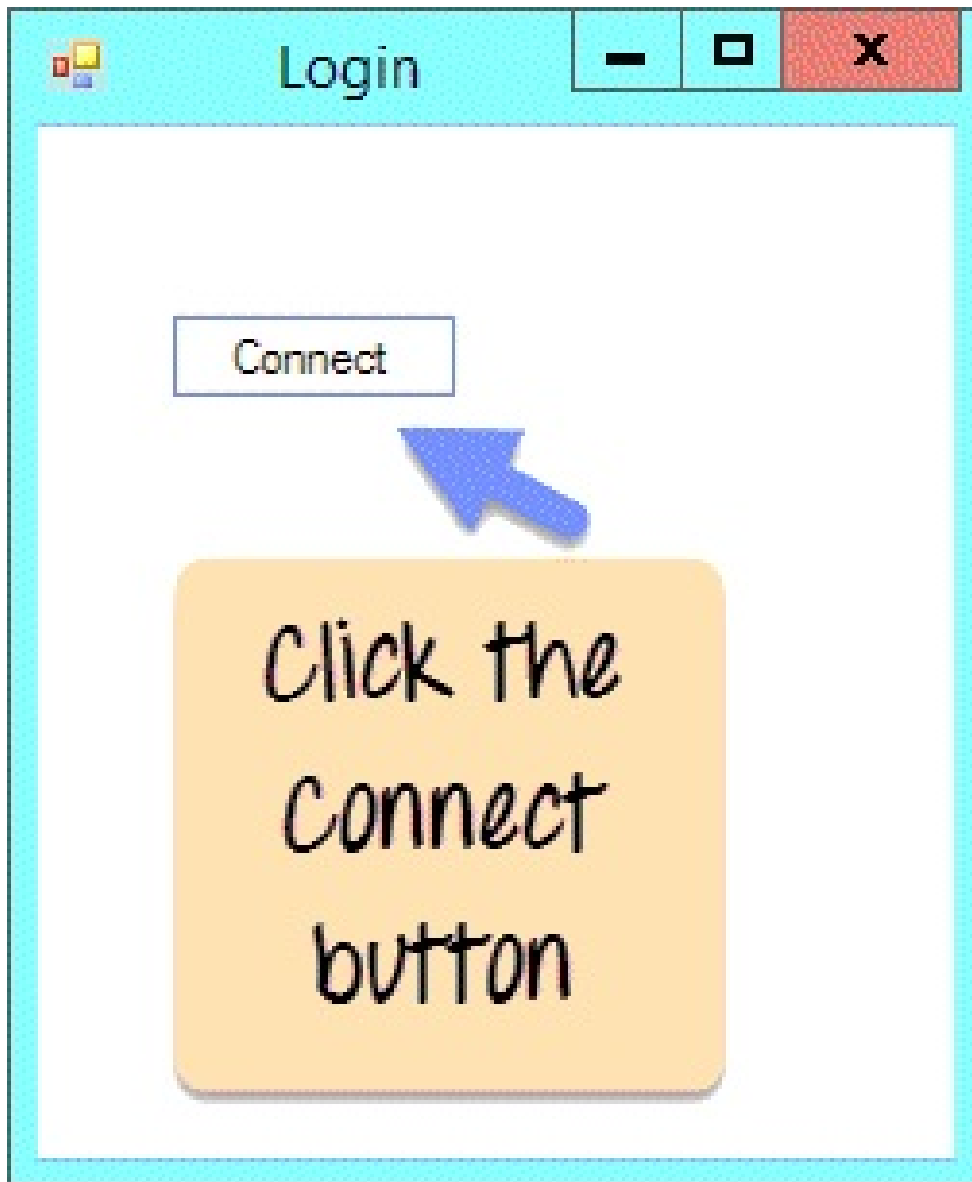
Code Explanation:-

1. The first step is to create the following variables

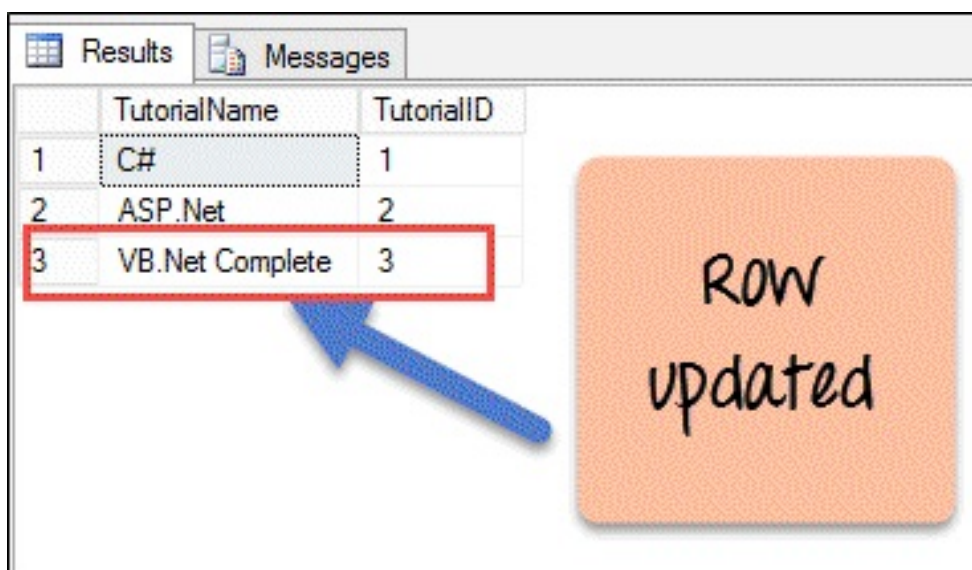
- a. SqlCommand – This data type is used to define objects which are used to perform SQL operations against a database. This object will hold the SQL command which will run against our SQL Server database.
 - b. The dataadapter object is used to perform specific SQL operations such as insert, delete and update commands.
 - c. We then define a string variable, which is SQL to hold our SQL command string.
2. The next step is to actually define the SQL statement which will be used against our database. In our case we are issuing an update statement, this will update the Tutorial name to “VB.Net Complete” while the TutorialID is unchanged and kept as 3.
 3. Next, we will create the command object, which is used to execute the SQL statement against the database. In the SQL command, you have passed the connection object and the SQL string.
 4. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the ExecuteNonQuery method which is used to execute the Update statement against our database.
 5. We finally close all the objects related to our database operation. Remember this is always a good practice.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you actually go to SQL Server Express and see the rows in the demotb table, you will see the row was successfully updated as shown below.



Deleting Records

Just like Accessing data, C# has the ability to delete existing records from the database as well. To showcase how to delete records into our database, let's take the same table structure which was used above.

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net complete

Let's change the code in our form, so that we can delete the following row

TutorialID	TutorialName
3	VB.Net complete

So let's add the following code to our program. The below code snippet will be used to delete an existing record in our database.

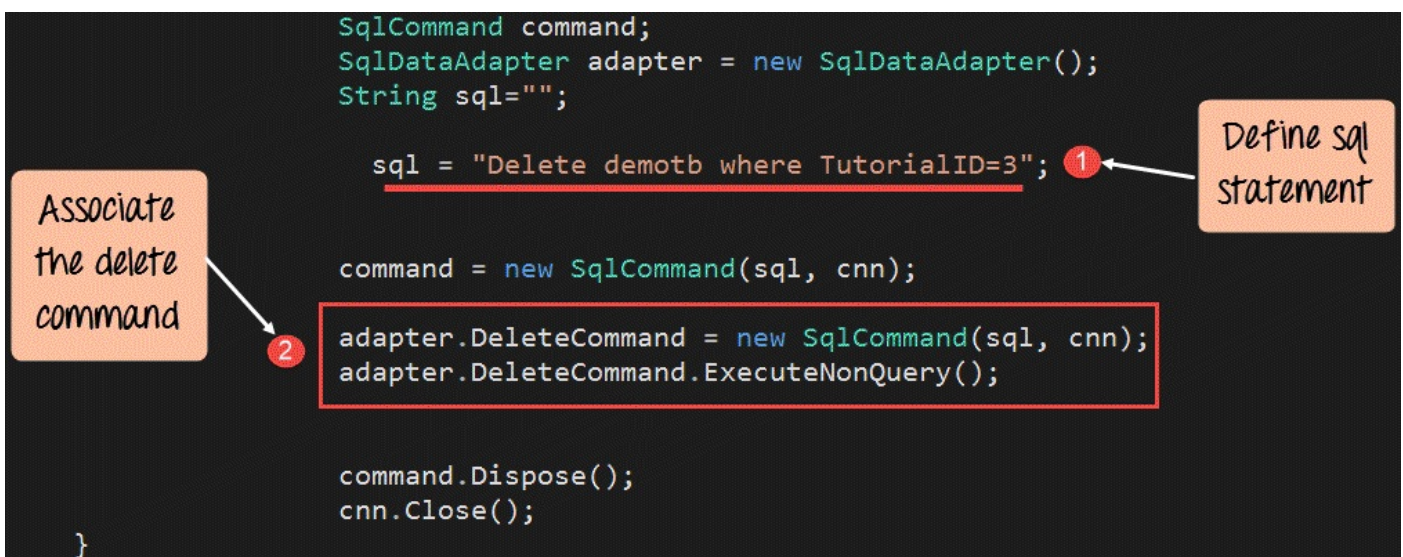
```
SqlCommand command;
SqlDataAdapter adapter = new SqlDataAdapter();
String sql="";

sql = "Delete demotb where TutorialID=3";

command = new SqlCommand(sql, cnn);

adapter.DeleteCommand = new SqlCommand(sql, cnn);
adapter.DeleteCommand.ExecuteNonQuery();

command.Dispose();
cnn.Close();
}
```

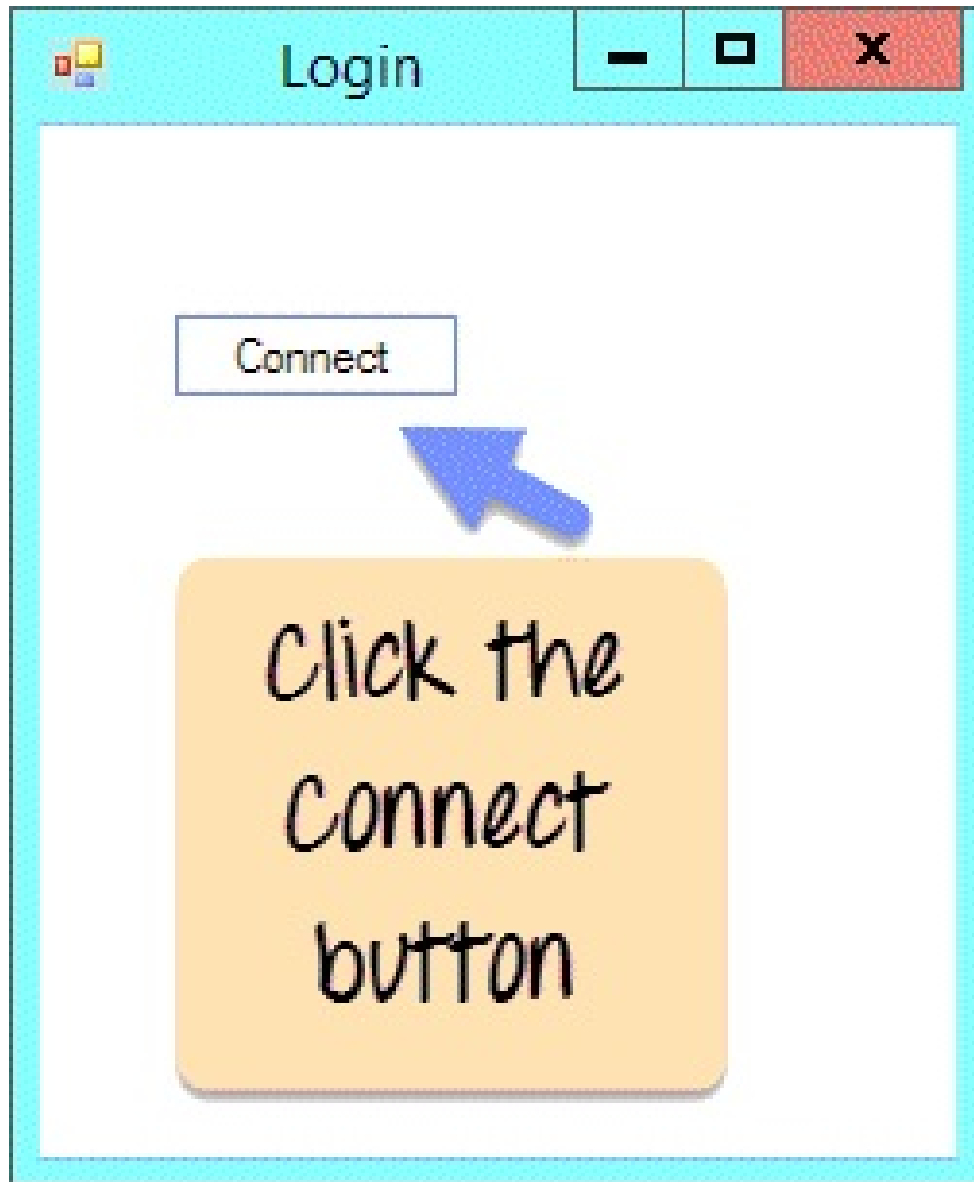


Code Explanation:-

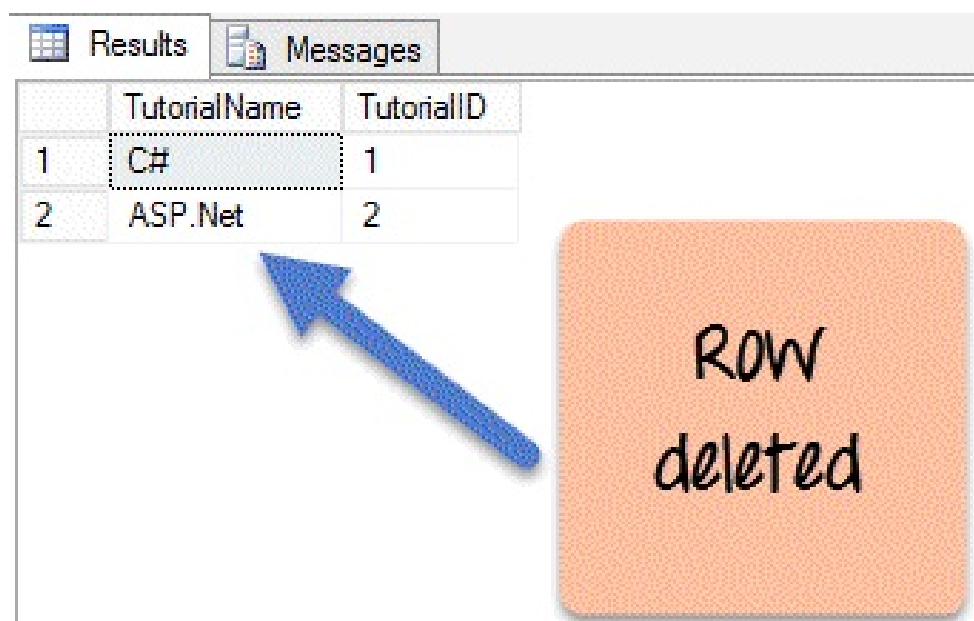
1. The Key difference in this code is that we are now issuing the delete SQL statement. The delete statement is used to delete the row in the demotb table in which the TutorialID has a value of 3.
2. In our data adapter command, we now associate the insert SQL command to our adapter. We also then issue the ExecuteNonQuery method which is used to execute the Delete statement against our database.

When the above code is set, and the project is executed using Visual Studio, you will get the below output. Once the form is displayed, click the Connect button.

Output:-



If you actually go to SQL Server Express and see the rows in the demotb table, you will see the row was successfully deleted as shown below.



Connecting Controls to Data

In the earlier sections, we have seen how to we can use C# commands such as SQLCommand and SqlDataReader to fetch data from a database. We also saw how we read each row of the table and use a messagebox to display the contents of a table to the user.

But obviously, users don't want to see data sent via message boxes and would want better controls to display the data. Let's take the below data structure in a table

TutorialID	TutorialName
1	C#
2	ASP.Net
3	VB.Net complete

From the above data structure, the user would ideally want to see the TutorialID and Tutorial Name displayed in a textbox. Secondly, they might want to have some sort of button control which could allow them to go to the next record or to the previous record in the table. This would require a bit of extra coding from the developer's end.

The good news is that C# has the ability to reduce the additional coding effort by allowing binding of controls to data. What this means is that C# can automatically populate the value of the textbox as per a particular field of the table.

So, you can have 2 textboxes in a windows form. You can then link one text box to the TutorialID field and another textbox to the TutorialName field. This linking is done in the Visual Studio designer itself, and you don't need to write extra code for this.

Visual Studio will ensure that it actually writes the code for you to ensure the linkage works. Then when you run your application, the textbox controls will automatically connect to the database, fetch the data and display it in the textbox controls. No coding is required from the developer's end to achieve this.

Let's look at a code example of how we can achieve binding of controls.

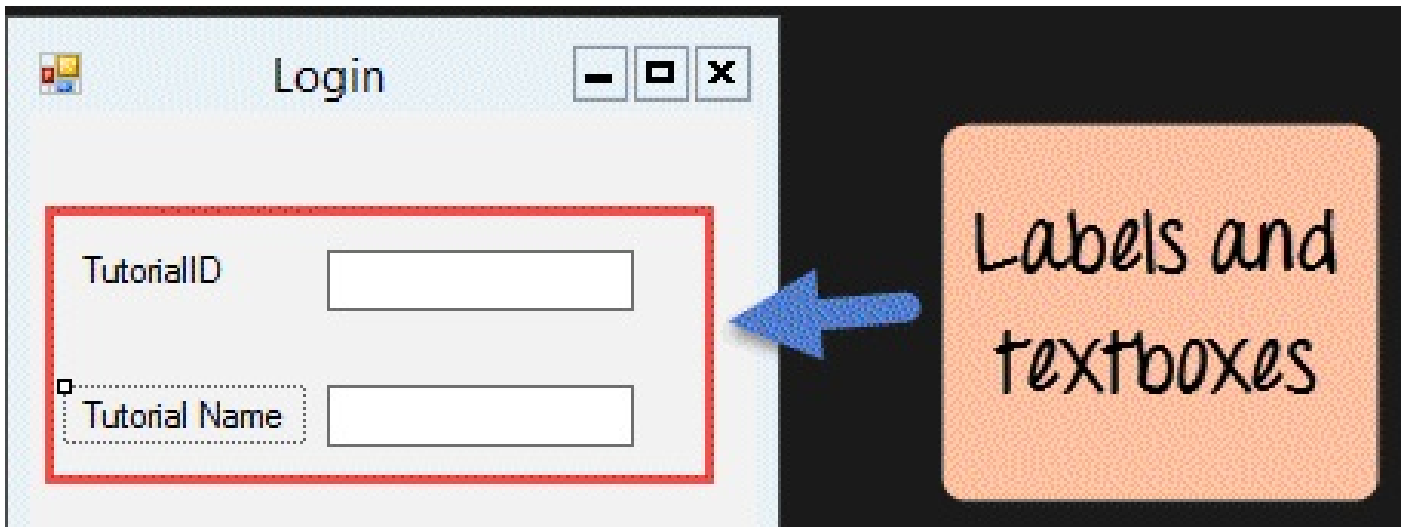
In our example, we are going to create 2 textboxes on the windows form. They are going to represent the Tutorial ID and Tutorial Name respectively. They will be bound to the Tutorial ID and TutorialName fields of the database accordingly.

Let's follow the below-mentioned steps to achieve this.

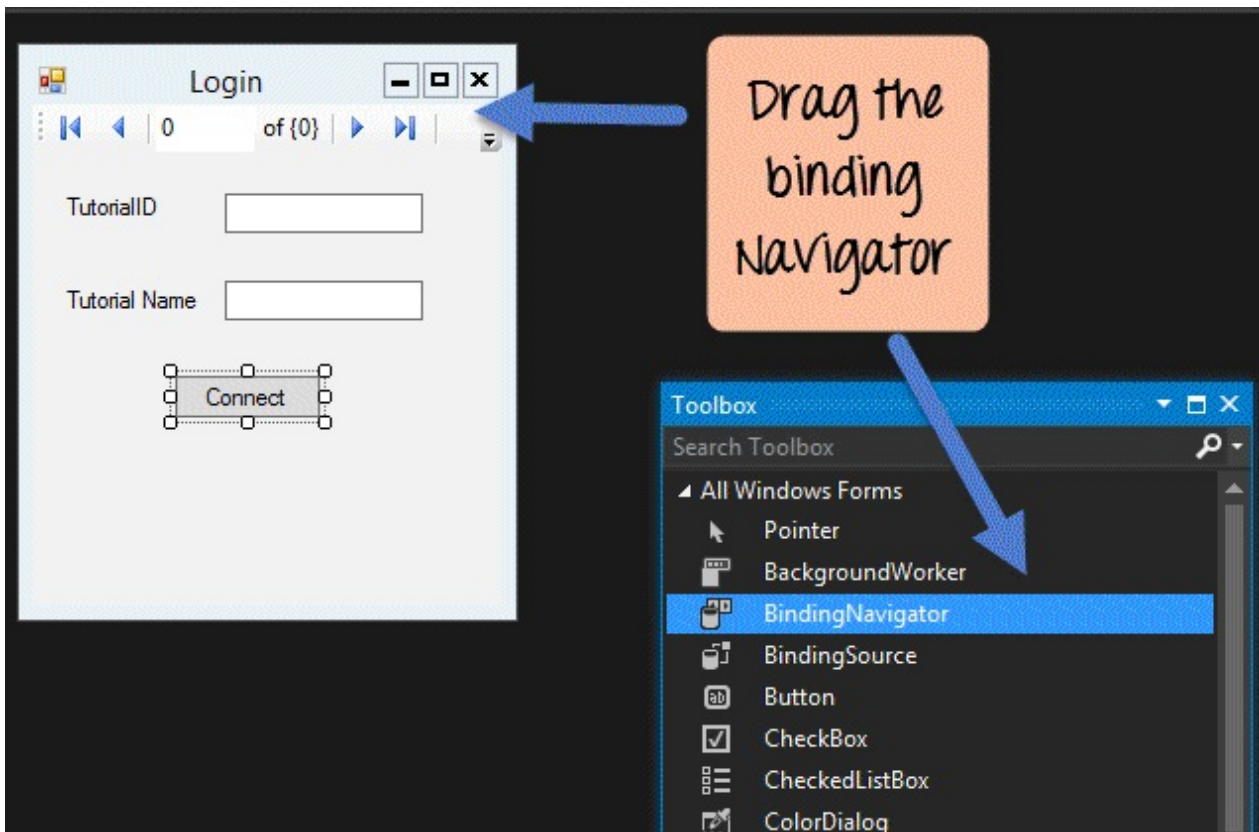
Step 1) Construct the basic form. In the form drag and drop 2 components- labels and textboxes. Then carry out the following substeps

1. Put the text value of the first label as TutorialID
2. Put the text value of the second label as TutorialName
3. Put the name property of the first textbox as txtID
4. Put the name property of the second textbox as txtName

Below is the how the form would look like once the above-mentioned steps are performed.



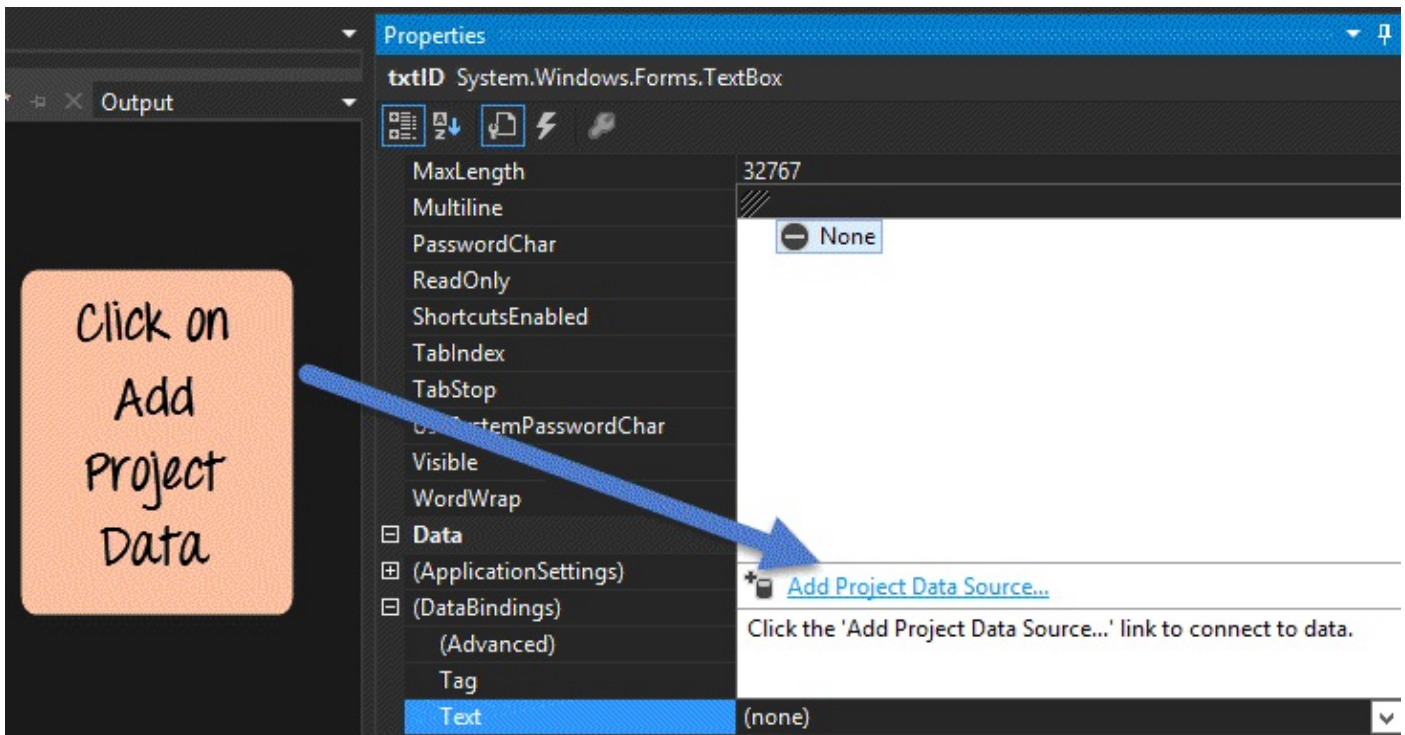
Step 2) The next step is to add a binding Navigator to the form. The binding Navigator control has the ability to automatically navigate through each row of the table. To add the binding navigator, just go to the toolbox and drag it to the form.



Step 3) The next step is to actually add a binding to our database. This can be done by going to any of the Textbox control and clicking on the DataBindings->Text property. The Binding Navigator is used to establish a link from your application to a database.

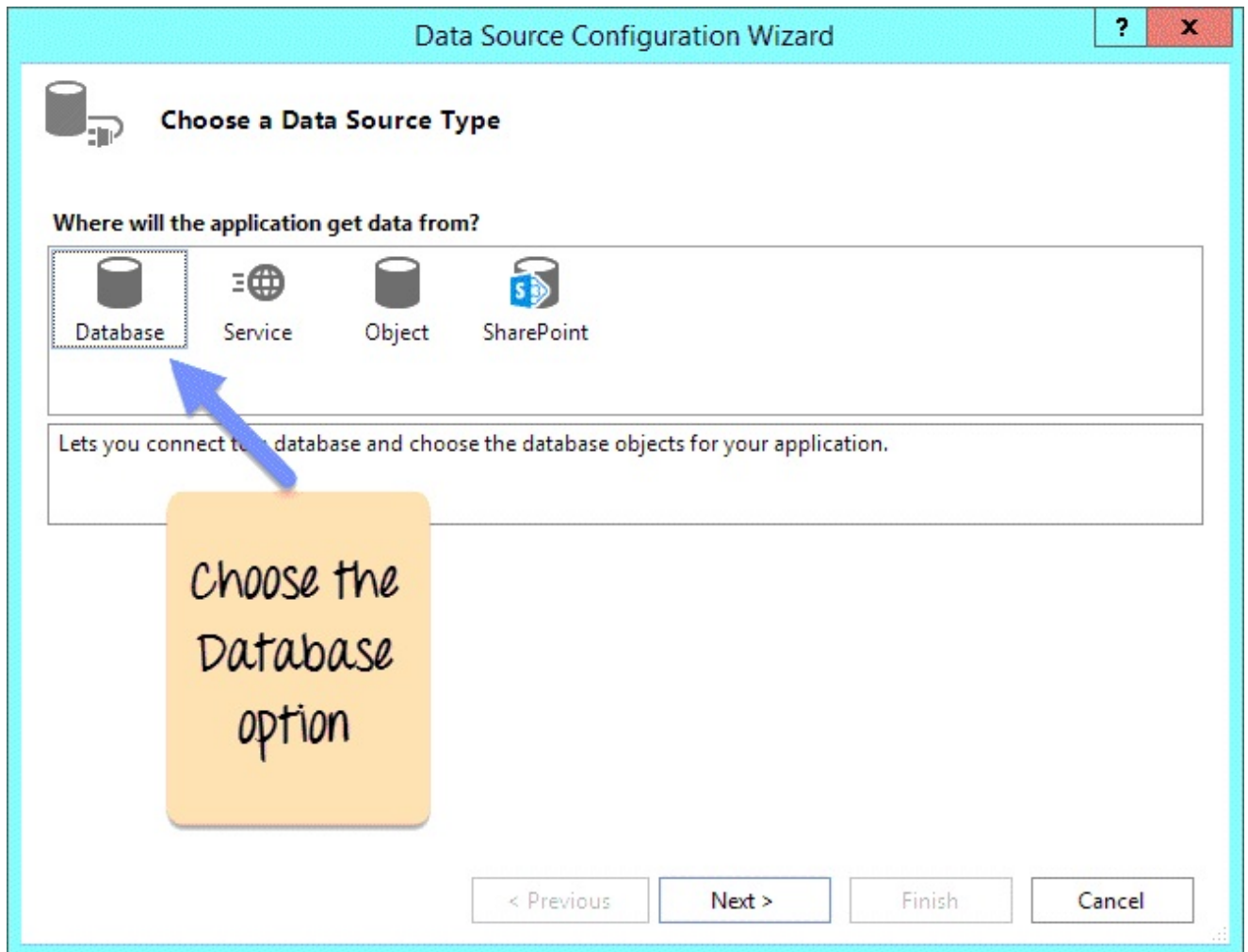
When you perform this step, Visual Studio will automatically add the required code to the application to make sure the application is linked to the database. Normally the database in Visual Studio is referred to as a Project Data Source. So to ensure the connection is established between the application and the database, the first step is to create a project data source.

The following screen will show up. Click on the link- “Add Project Data Source”. When you click on the project data source , you will be presented with a wizard, this will allow you to define the database connection.

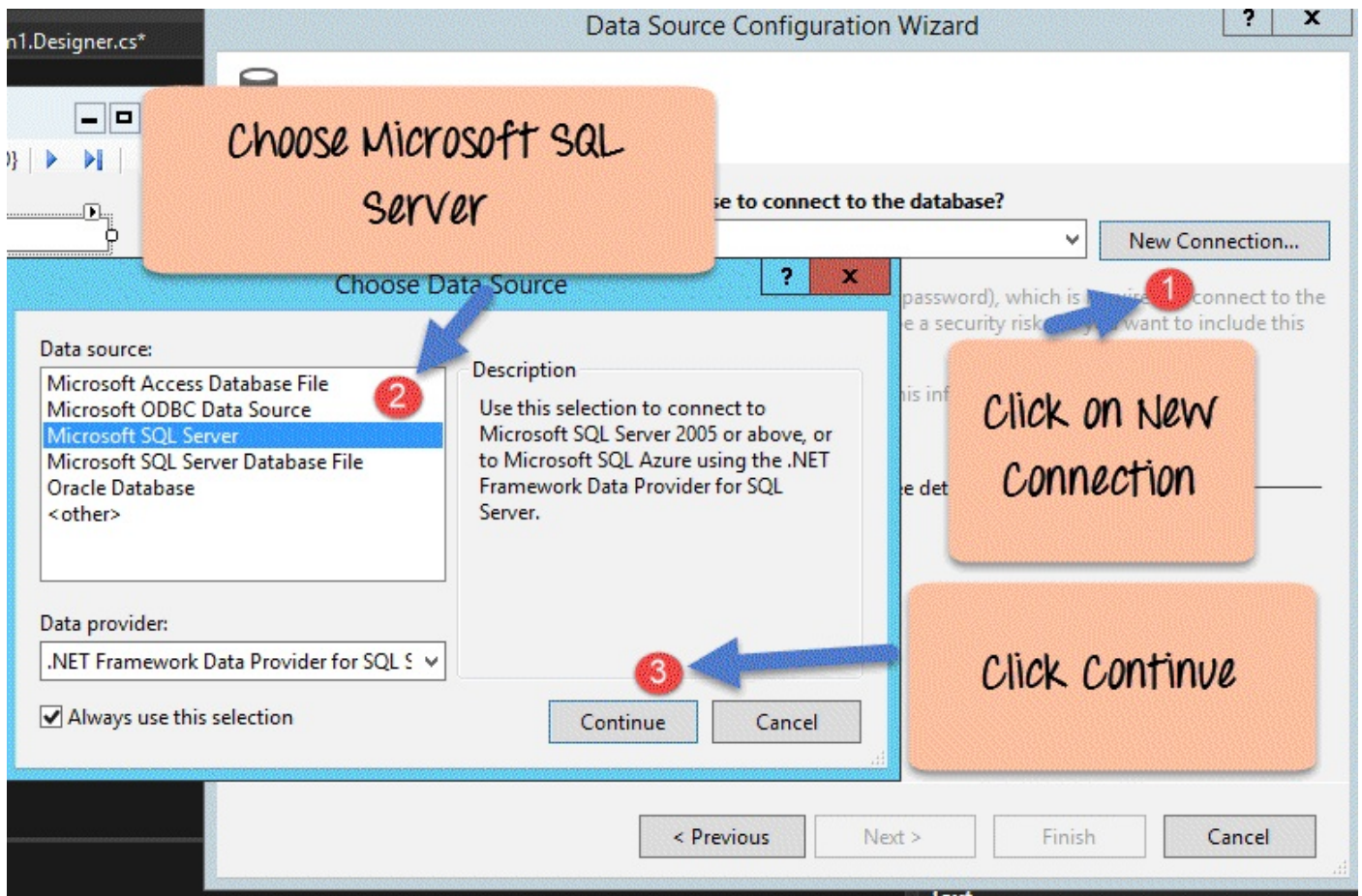


Step 4) Once you click on the Add Project Data Source link , you will be presented with a wizard which will be used to create a connection to the demotb database. The following steps show in detail what needs to be configured during each step of the wizard.

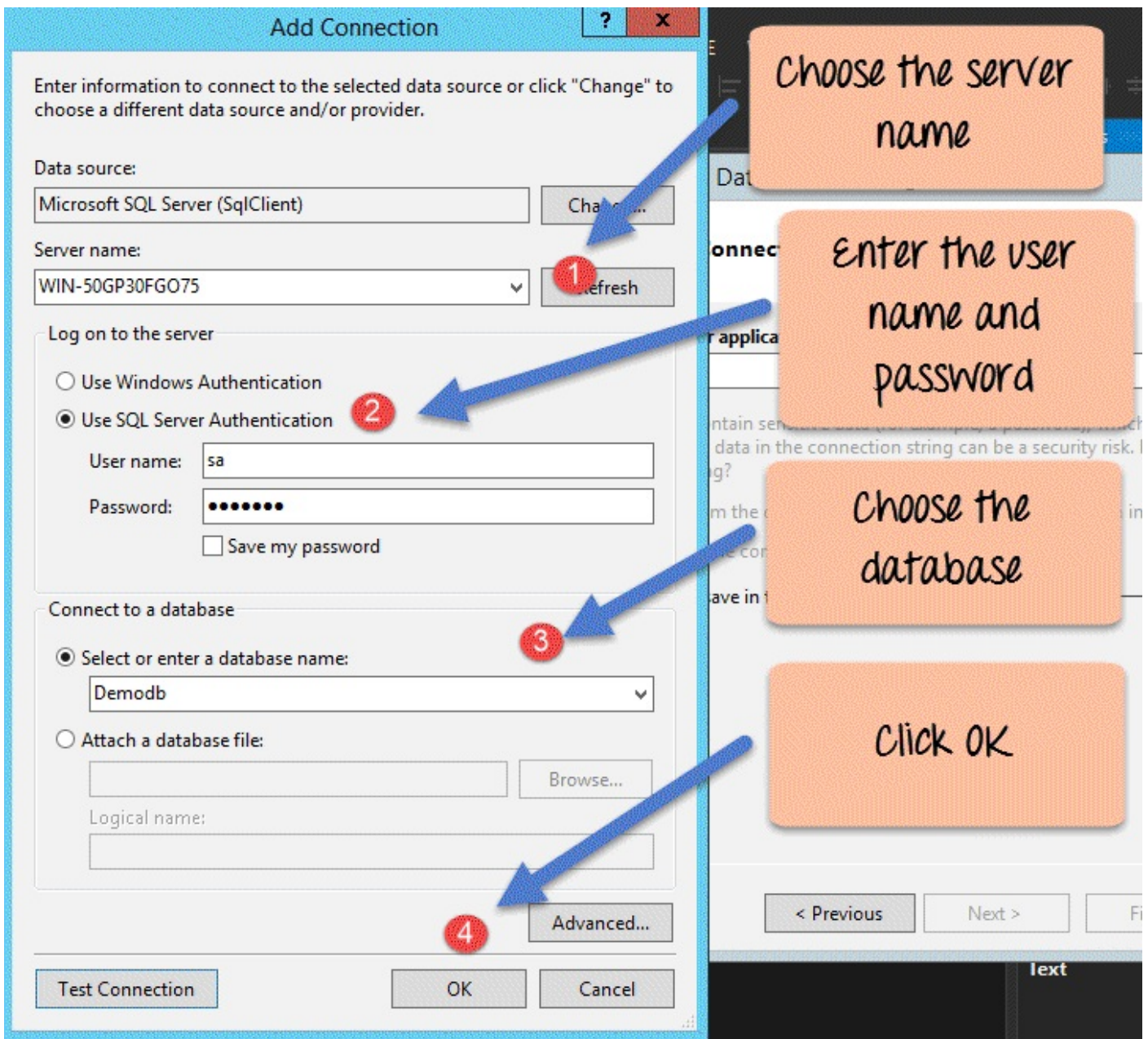
1. In the screen which pops up , choose the Data Source type as Database and then click on next button.



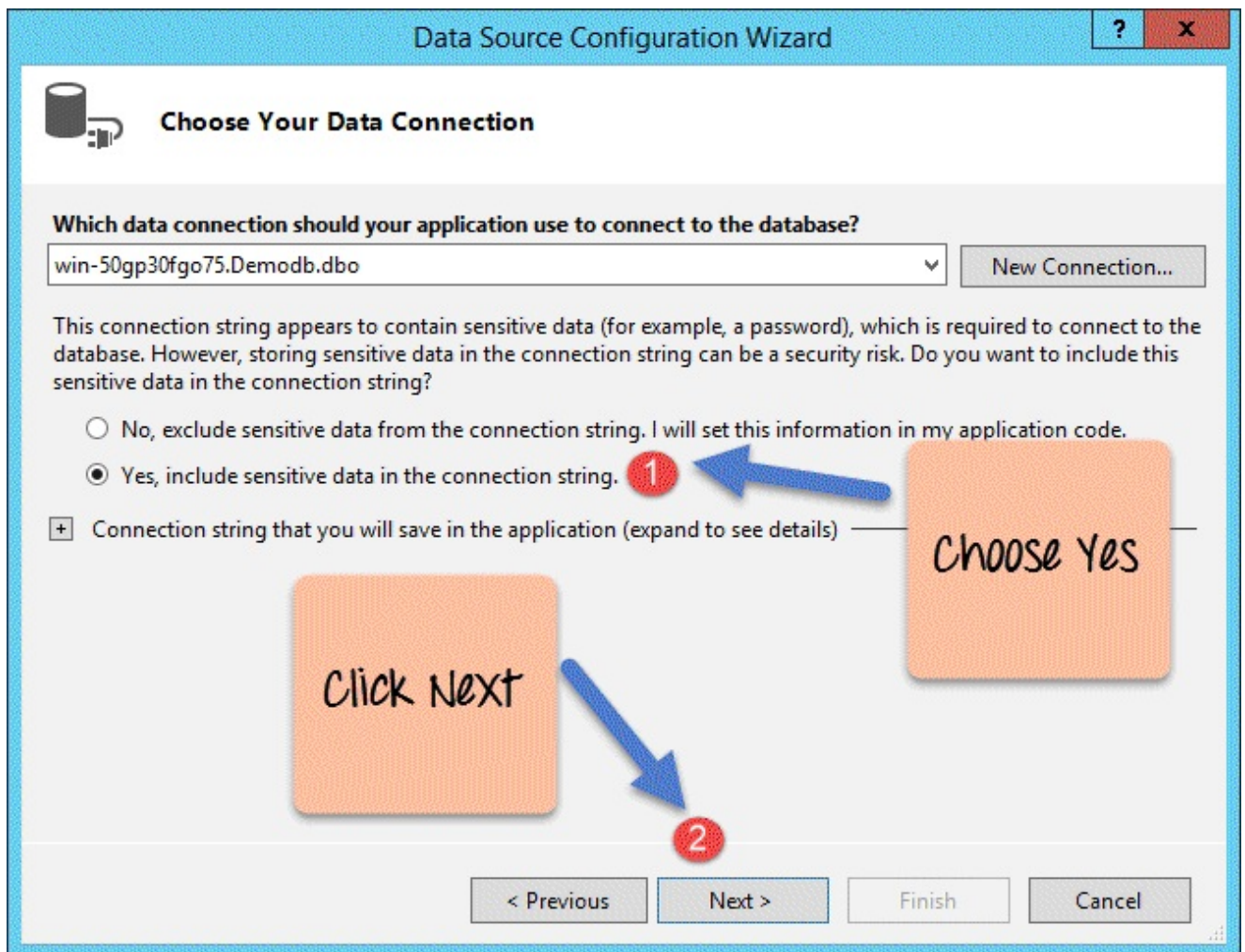
2. In the next screen, you need to start the creation of the connection string to the database. The connection string is required for the application to establish a connection to the database. It contains the parameters such as server name, database name, and the name of the driver.
1. Click on the New connection button
2. Choose the Data Source as Microsoft SQL Server
3. Click the Continue button.



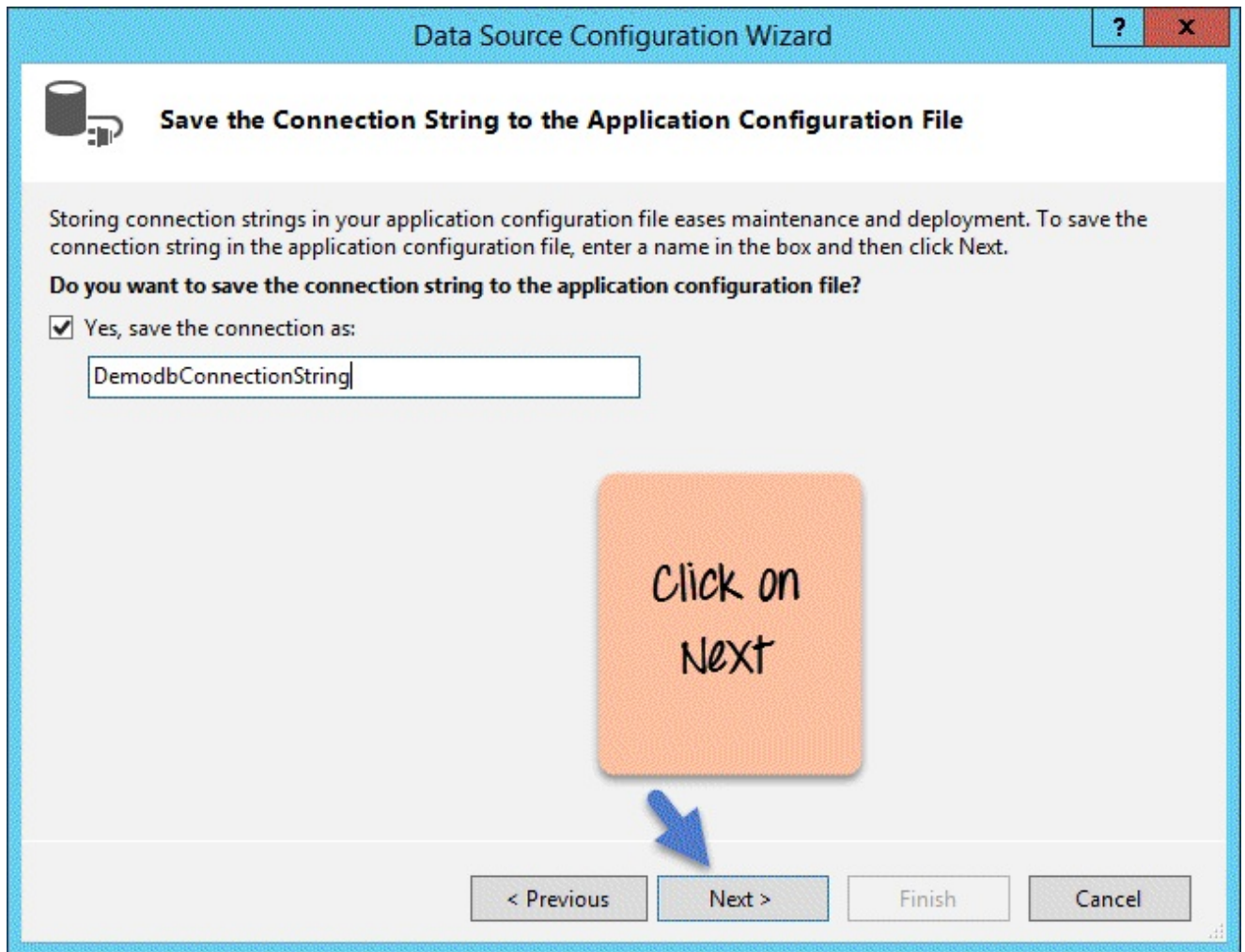
3. Next, you need to add the credentials to connect to the database
 - a. Choose the server name on which the SQL Server resides
 - b. Enter the user id and password to connect to the database
 - c. Choose the database as demotb
 - d. Click the 'ok' button.



4. In this screen, we will confirm all the settings which were carried on the previous screens.
 - a. Choose the option “Yes” to include sensitive data in the connection string
 - b. Click on the “Next” button.

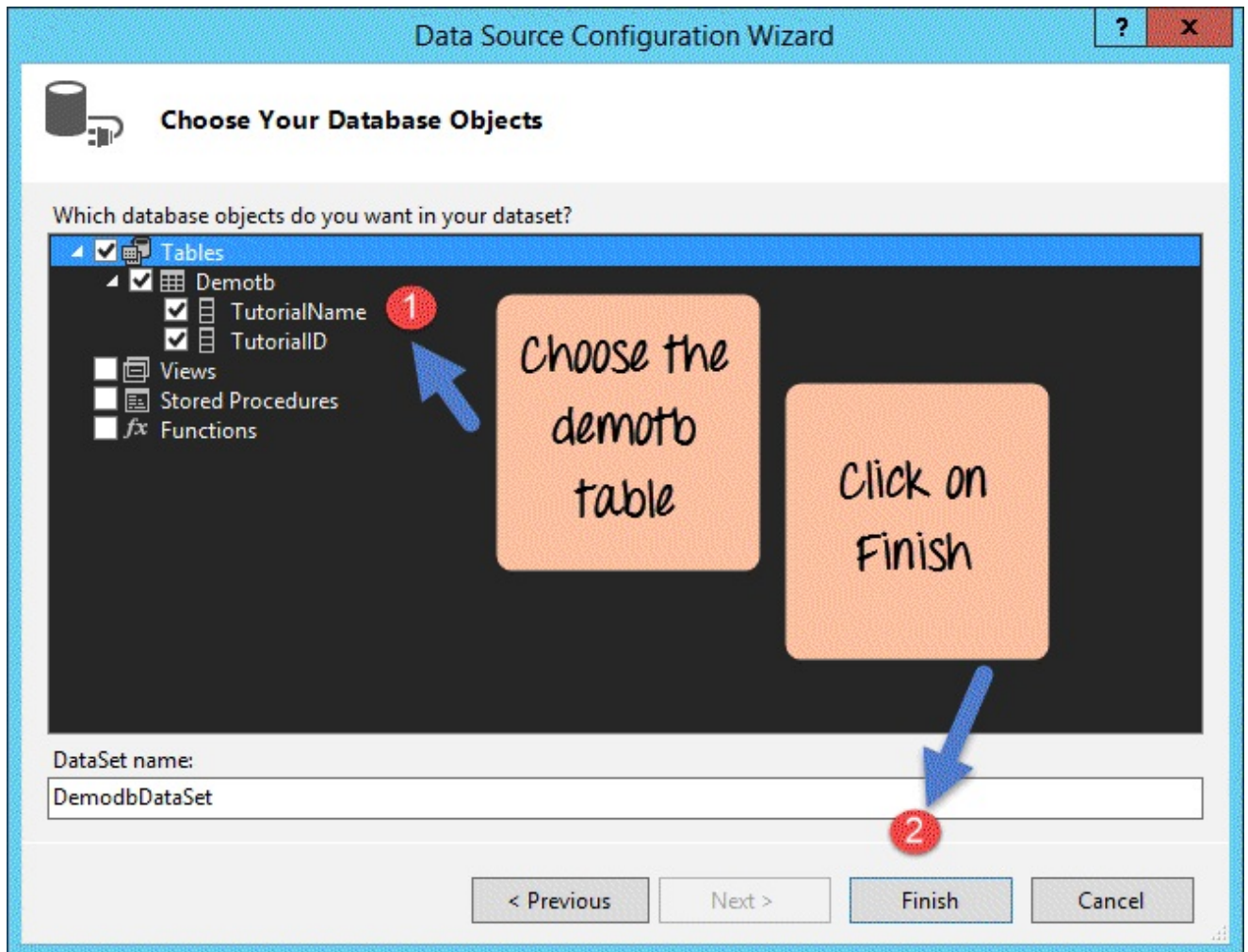


5. In the next screen, click on the “Next” button to confirm the creation of the connection string



6. In this step,

1. Choose the tables of Demotb, which will be shown in the next screen.
2. This table will now become an available data source in the C# project

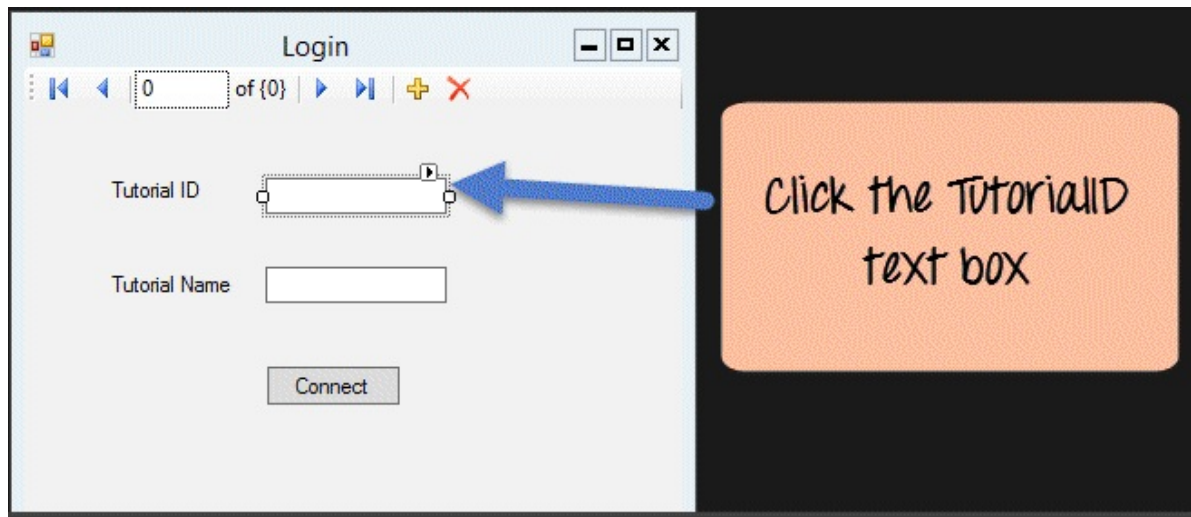


When you click the Finish button, Visual Studio will now ensure that the application is able to query all the rows in the table Demotb.

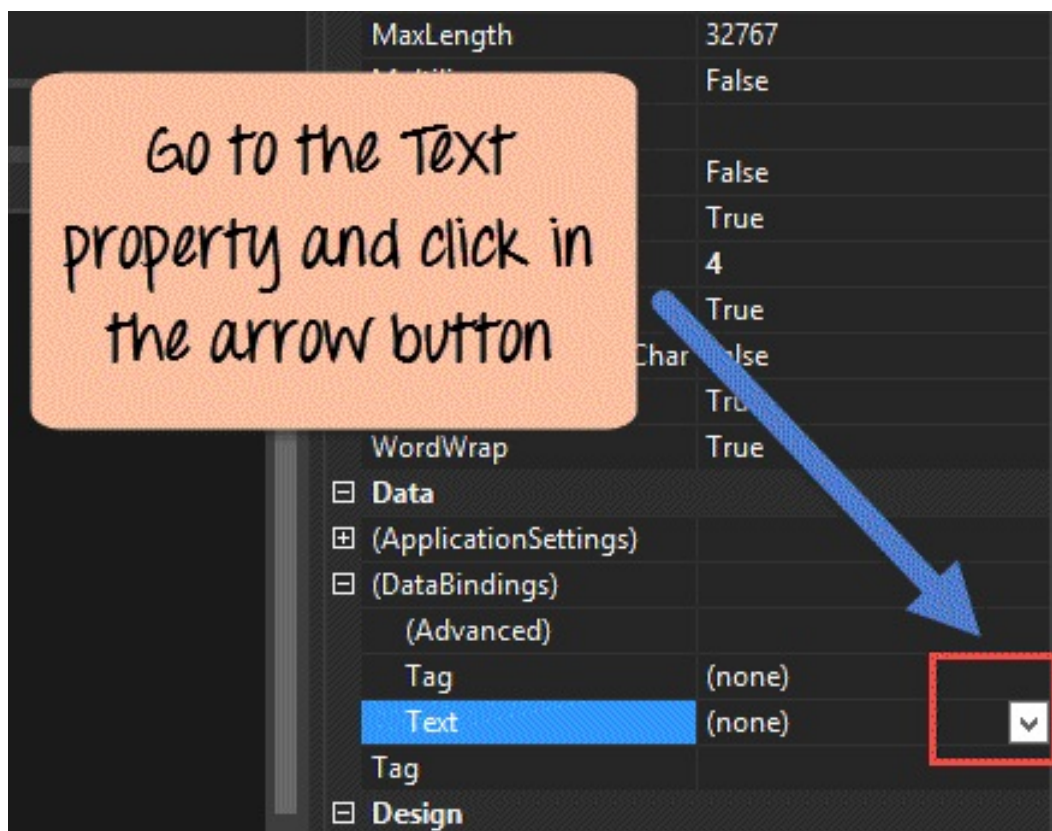
Step 5) Now that the data source is defined, we now need to connect the TutorialID and TutorialName textbox to the demotb table. When you click on the Text property of either the TutorialID or TutorialName textbox, you will now see that the binding source to Demotb is available.

For the first text box choose the Tutorial ID. Repeat this step for the second textbox and choose the field as TutorialName. The below steps shows how we can navigate to each control and change the binding accordingly.

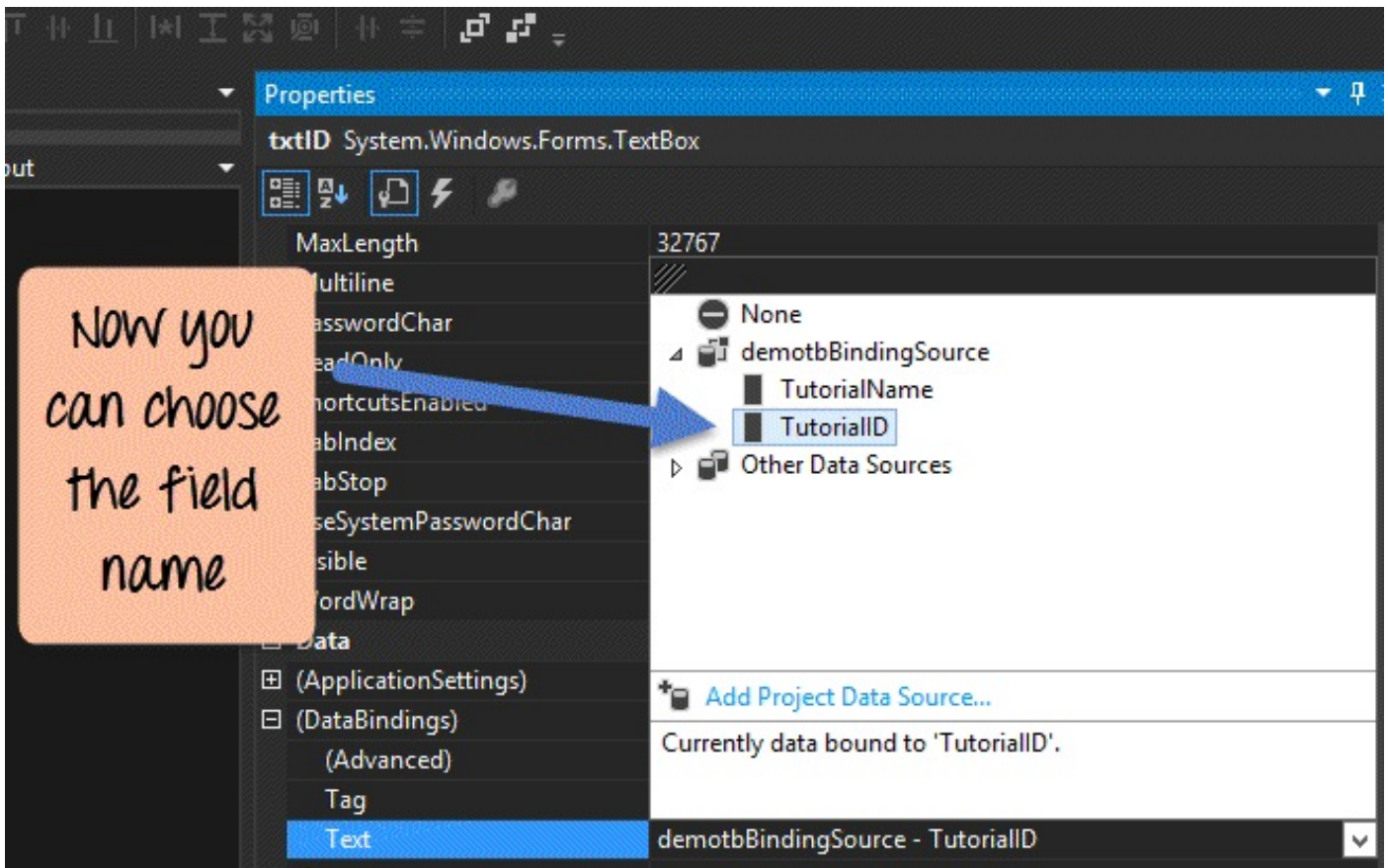
1. Click on the Tutorial ID control.



2. In the Properties window , you will see the properties of the TutorialID textbox. Go to the text property and click on the down arrow button.



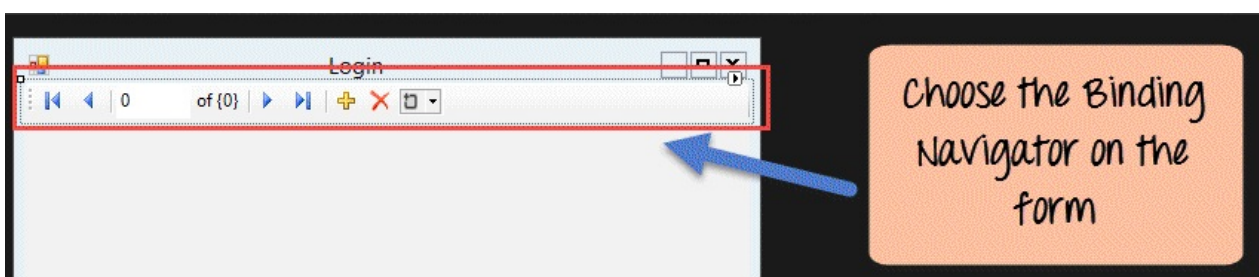
3. When you click the down arrow button , you will see the demotbBinding Source option. And under this , you will see the options of TutorialName and TutorialID. Choose the Tutorial ID one.



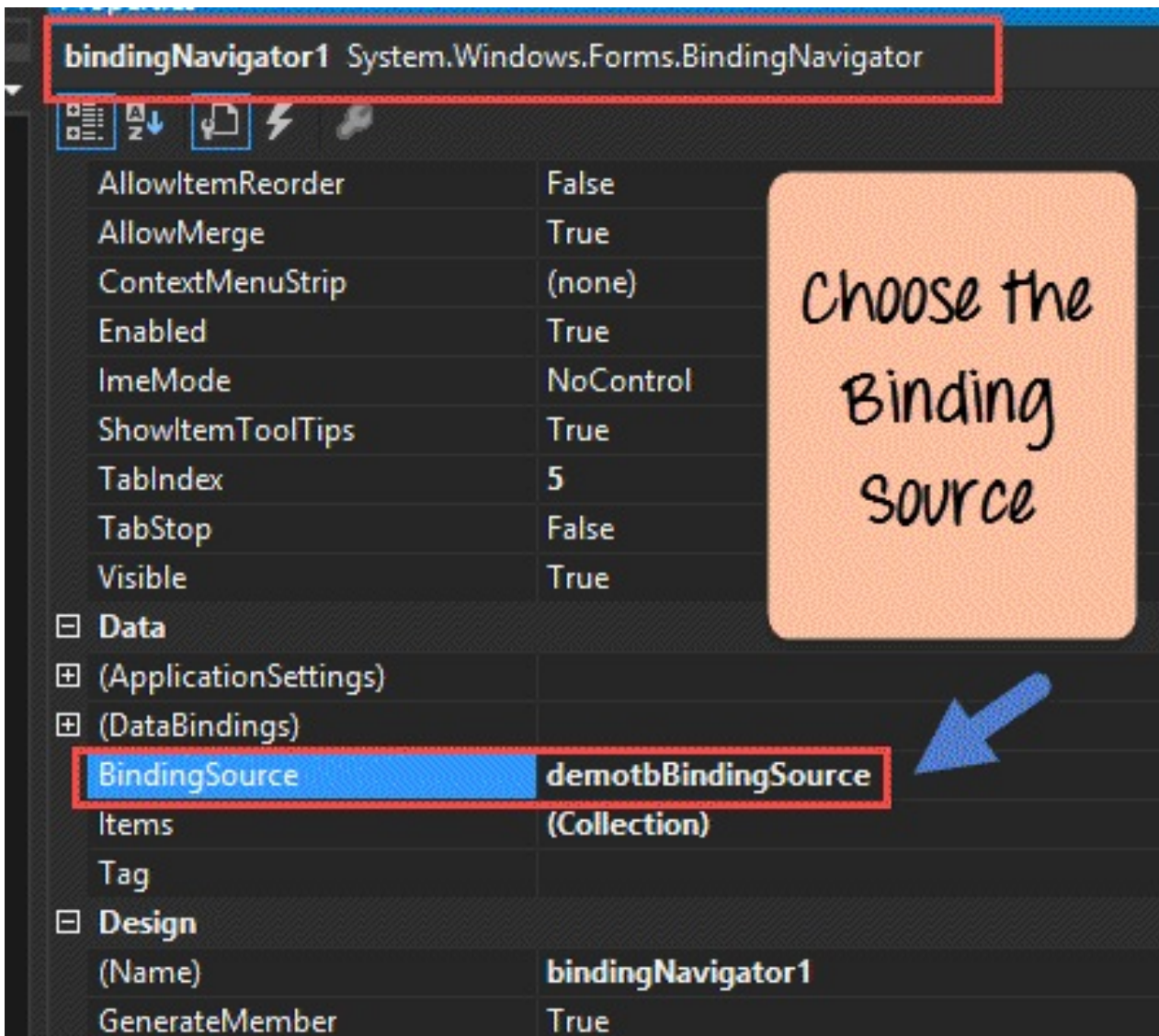
Repeat the above 3 steps for the Tutorial Name text box.

1. So click on the Tutorial Name text box
2. Go to the properties window
3. Choose the Text property
4. Choose the TutorialName option under demotbBindingSource

Step 6) Next we need to change the Binding Source property of the BindingNavigator to point to our Demotb data source. The reason we do this is because the Binding Navigator needs to also know which table it needs to refer to. The Binding Navigator is used to select the next or previous record in the table. So even though the data source is added to the project as a whole and to the text box control, we still need to ensure the Binding Navigator also has a link to our data source. In order to do this, we need to click the Binding navigator object, go to the Binding Source property and choose the one that is available

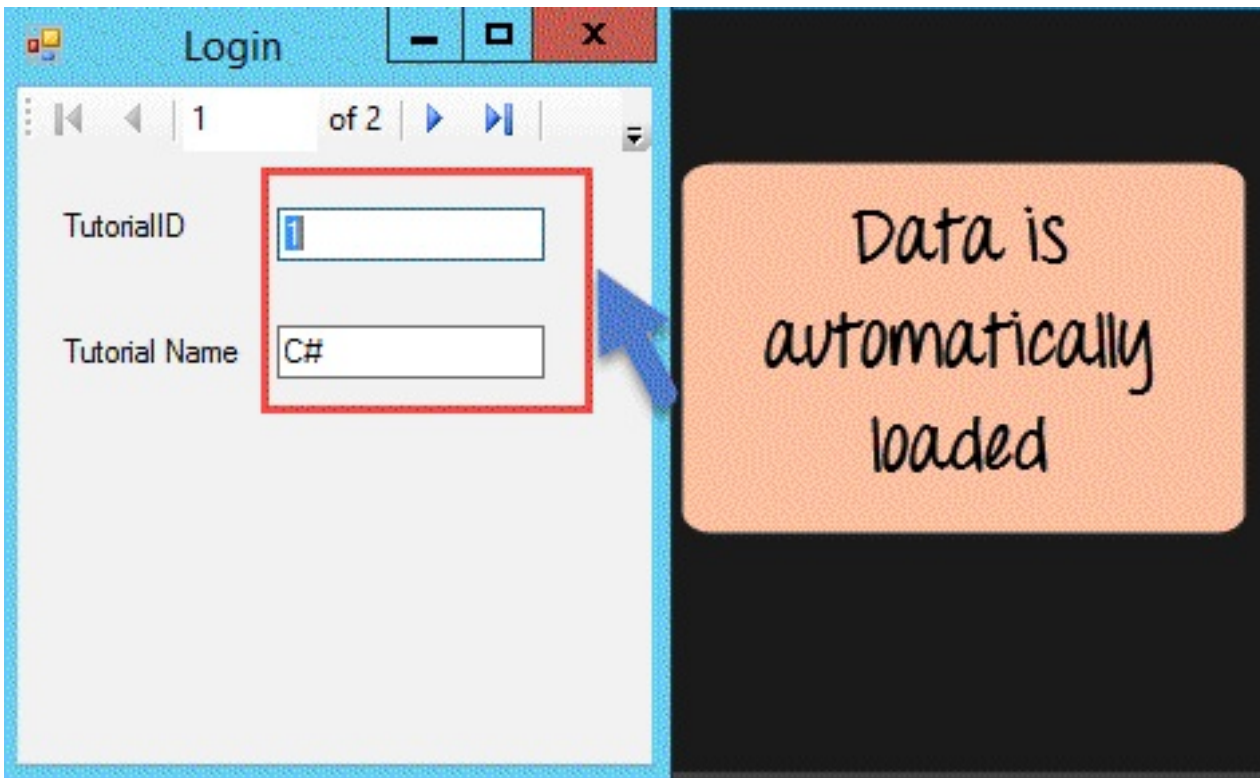


Next, we need to go to the Properties window so that we can make the change to Binding Source property.

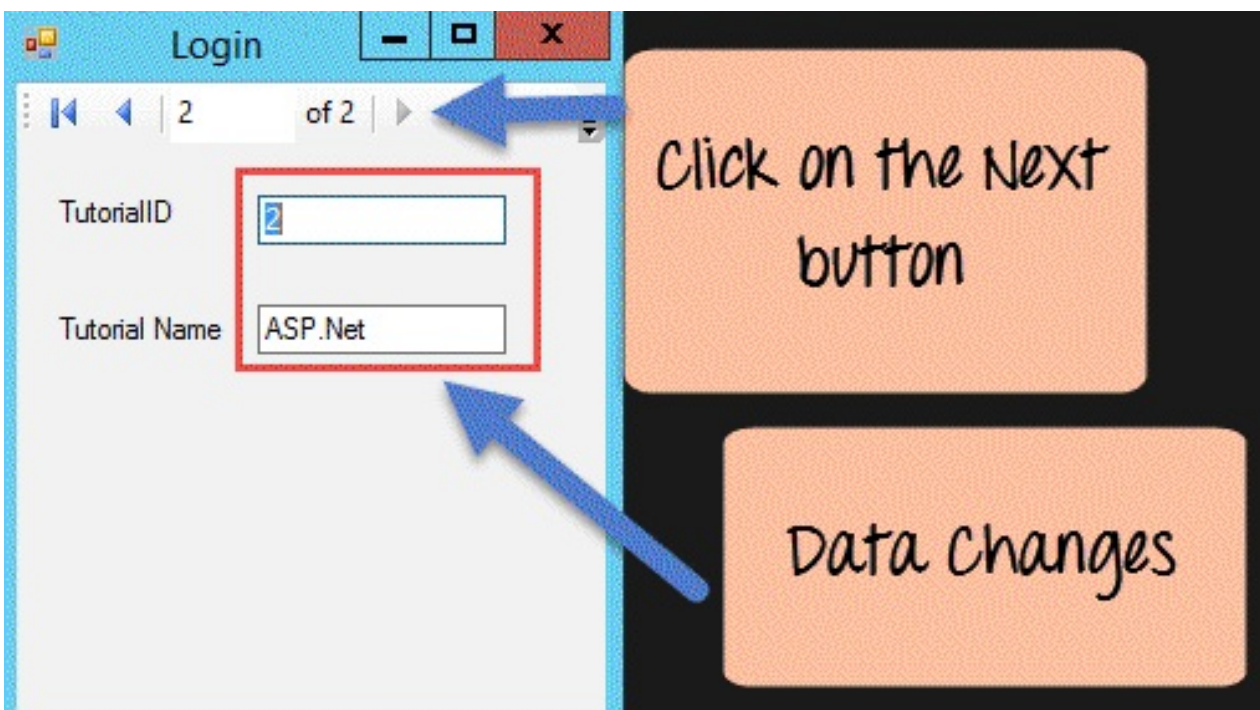


When all of the above steps are executed successfully, you will get the below-mentioned output.

Output:-



Now when the project is launched, you can see that the textboxes automatically get the values from the table.



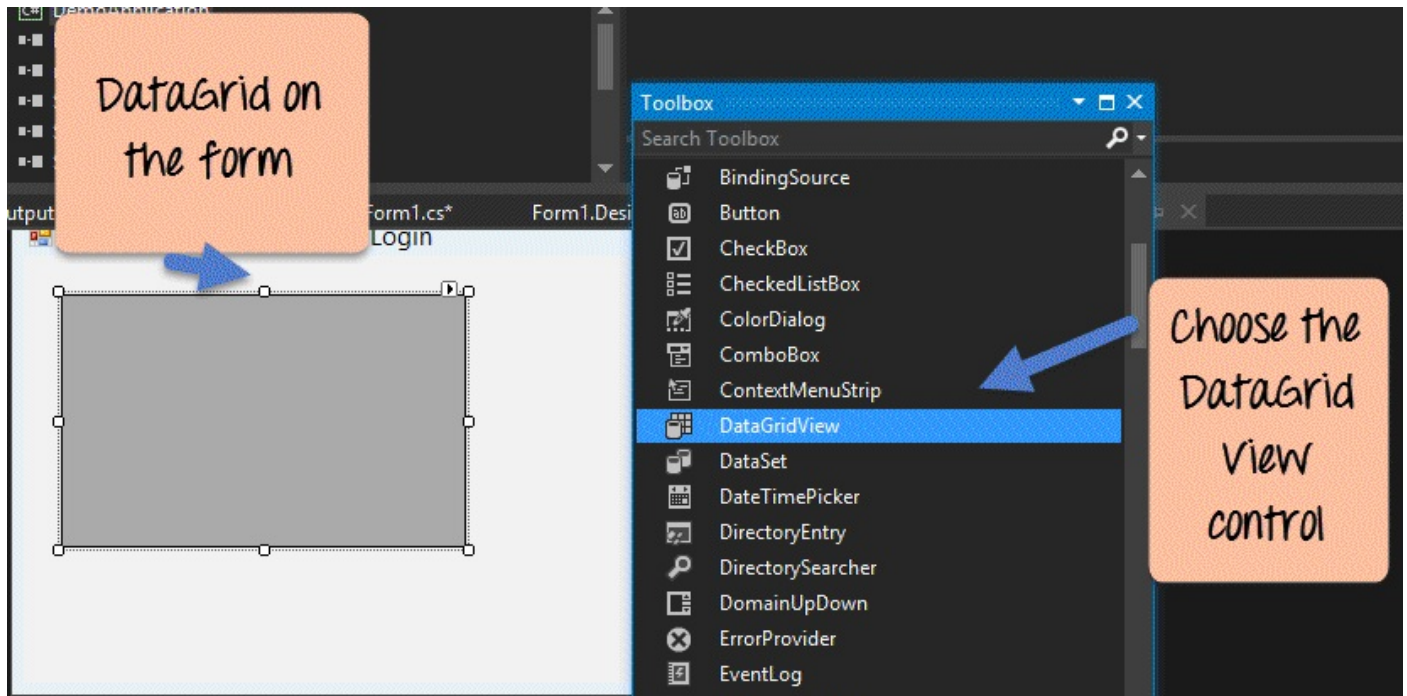
When you click the Next button on the Navigator, it automatically goes to the next record in the table. And the values of the next record automatically come in the text boxes

Using DataGrids

Data Grids are used to display data from a table in a grid like format. When a user see's table data, they normally prefer seeing all the table rows in one shot. This can be achieved if we can display the data in a grid on the form.

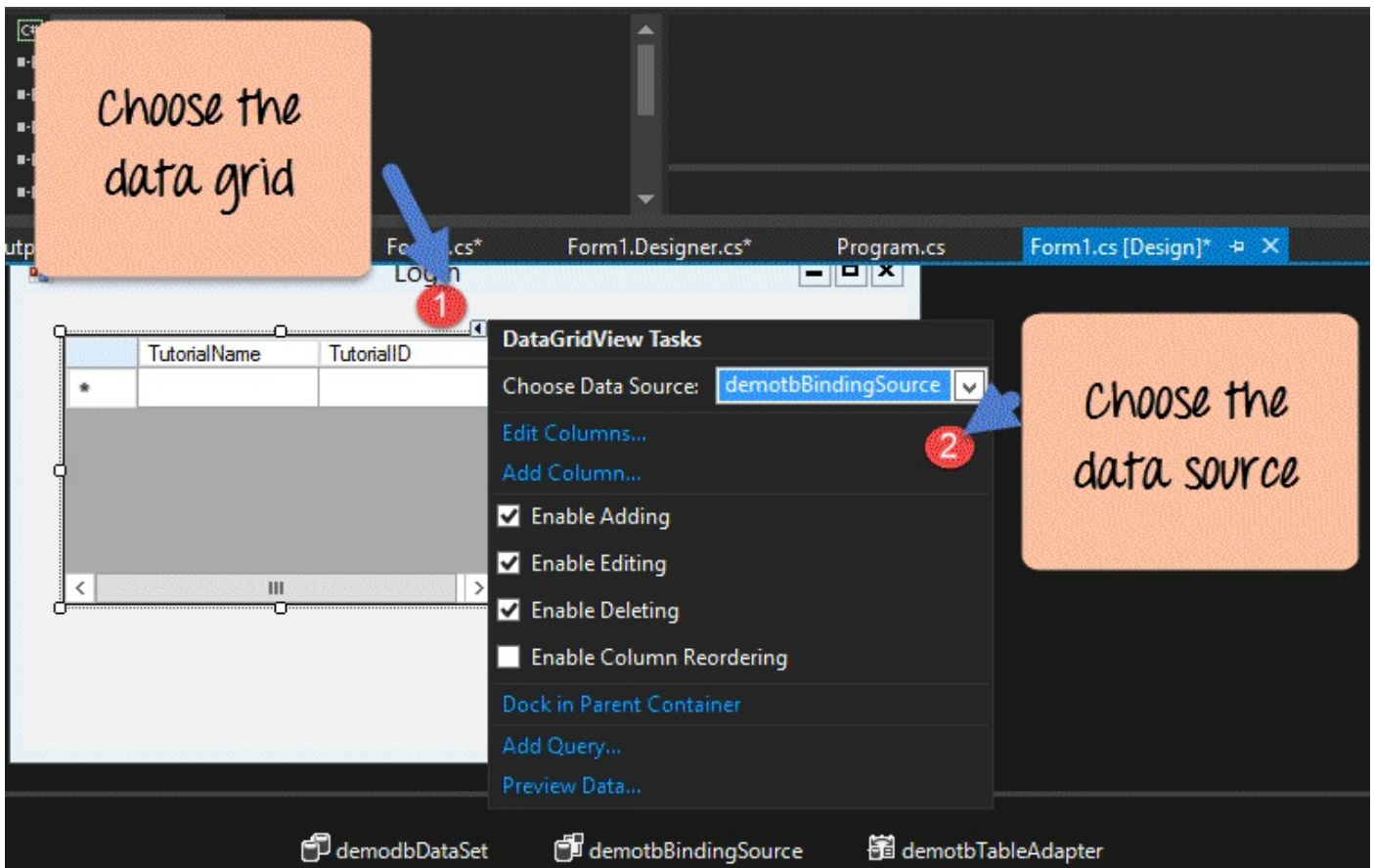
C# and Visual Studio have inbuilt data grids, this can be used to display data. Let's take a look at an example of this. In our example, we will have a data grid, which will be used to display the Tutorial ID and Tutorial Name values from the demotb table.

Step 1) Drag the DataGridView control from the toolbox to the Form in Visual Studio. The DataGridView control is used in Visual Studio to display the rows of a table in a grid-like format.



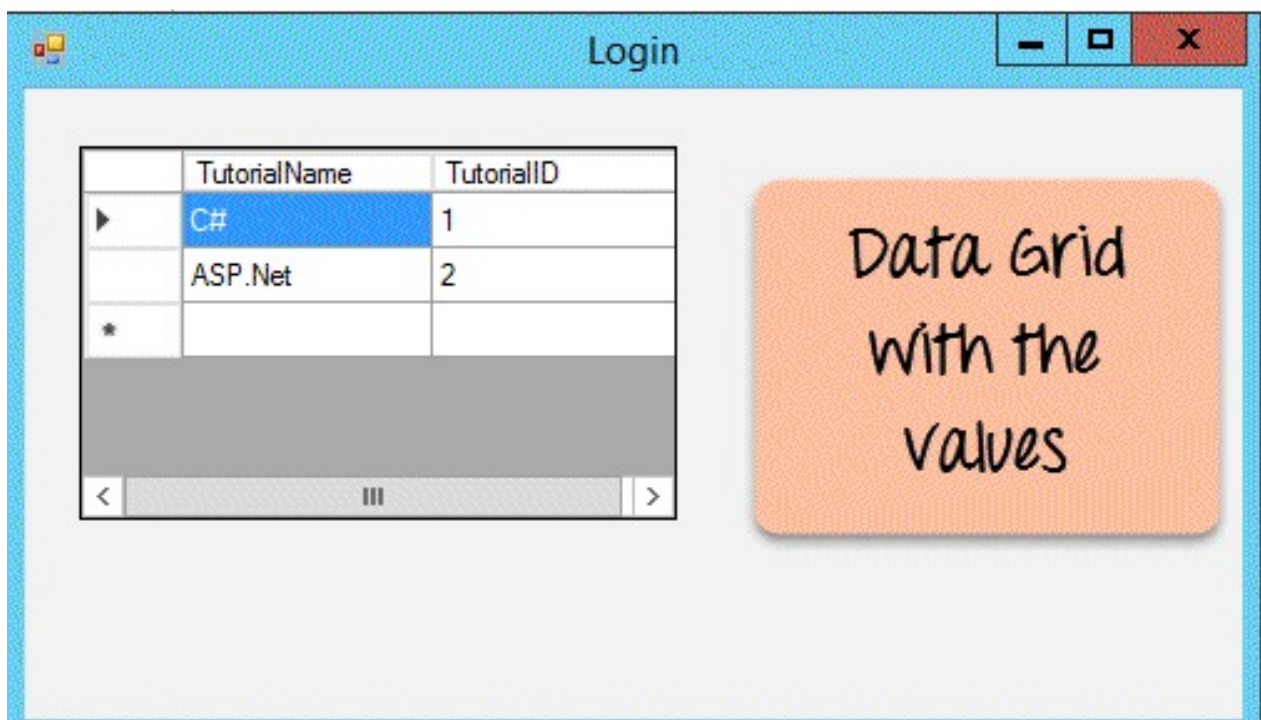
Step 2) In the next step, we need to connect our data grid to the database. In the last section, we had created a project data source. Let's use the same data source in our example.

1. First, you need to choose the grid and click on the arrow in the grid. This will bring up the grid configuration options.
2. In the configuration options, just choose the data source as demotbBindingSource which was the data source created in the earlier section.



If all the above steps are executed as shown, you will get the below-mentioned output.

Output:-



From the output, you can clearly see that the grid was populated by the values from the database.

Summary

- C# has the ability to work with databases such as Oracle and Microsoft SQL Server.
- C# has all the commands which are required to work with databases. This involves establishing a connection to the database. You can perform operations such as select, update, insert and delete using the commands in C#.
- The datareader object in C# is used to hold all the data returned by the database. The While loop in C# can be used to read the data rows one at a time.
- The data adapter object is used to perform SQL operations such as insert, delete, and update against the database.
- C# has the ability to bind controls to the various fields in a table. They are bound by defining a data source in C#. The data source is used to pull the data from the database and populate them in the controls.
- The binding navigator is used to automatically navigate through the rows in a table.
- The data grid in C# has the ability to connect to the database and display all the values from the table in a grid-like format.

Chapter 8: File Operations

C# has a wide array of file operations. These operations include opening a file, reading or writing to a file.

There can be an instances wherein you want to work with files directly, in which case you would use the file operations available in C#.

Some of the basic file operations are mentioned below.

1. Reading – This operation is the basic read operation wherein data is read from a file.
2. Writing - This operation is the basic write operation wherein data is written to a file. By default, all existing contents are removed from the file, and new content is written.
 1. Appending – This operation also involves writing information to a file. The only difference is that the existing data in a file is not overwritten. The new data to be written is added at the end of the file.

This tutorial focuses on how to work with files in C#.

Basic File I/O Commands

C# and .Net has the ability to work with files with the help of several File I/O commands. Let's have a look at some of these commands. For our example, we will assume that we have a file in the D drive called Example.txt.

The file will be a simple text file and have 2 lines as shown below

- Guru99 - .Net
- Guru99 -C#

For our example, we will create a simple Console application and work with our File I/O commands. The console application is the basic one which was created in the earlier chapters. In the console application, all code is written to the Program.cs file.

1. **File.Exists** – The File exists method is used to check if a particular file exists. So now let's see the code which can be used to check if our Example.txt file actually exists or not. Enter the below code in the program.cs file.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"D:\Example.txt"; 1
        if (File.Exists(path)) 2
        {
            Console.WriteLine("File Exists"); 3
        }
        Console.ReadKey();
    }
}
```

String variable with the file name

Check the existence of the file

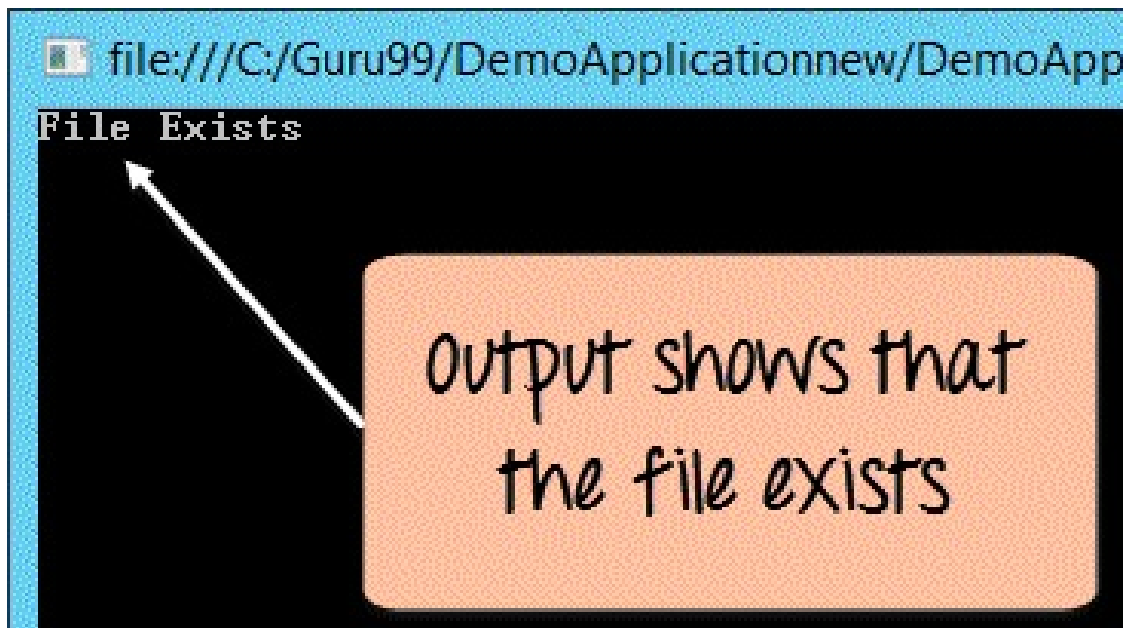
Display a confirmation message

Code Explanation:-

1. First, we are setting a string variable with the path to our Example.txt file.
2. Next, we use the File.Exists method to check if the file really exists or not. If the File exists, a true value will be returned.
3. If we get a true value and the file does exist, then we write the message "File Exists" to the console.

When the above code is set, and the project is executed using Visual Studio, you will get the below output.

Output:-



From the above output, you can clearly see that the File.Exists command was executed successfully, and the correct message was displayed in the console window.

2. **File.ReadAllLines** – The method is used to read all the lines one by one in a file. The lines are then stored in a string array variable. Let's look at an example. Enter the below code in the program.cs file.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"D:\Example.txt";
        string[] lines;
        lines=File.ReadAllLines(path);
        Console.WriteLine(lines[0]);
        Console.WriteLine(lines[1]);
        Console.ReadKey();
    }
}
```

String array

Reading all the lines from the file

Writing the lines to the console

1

2

3

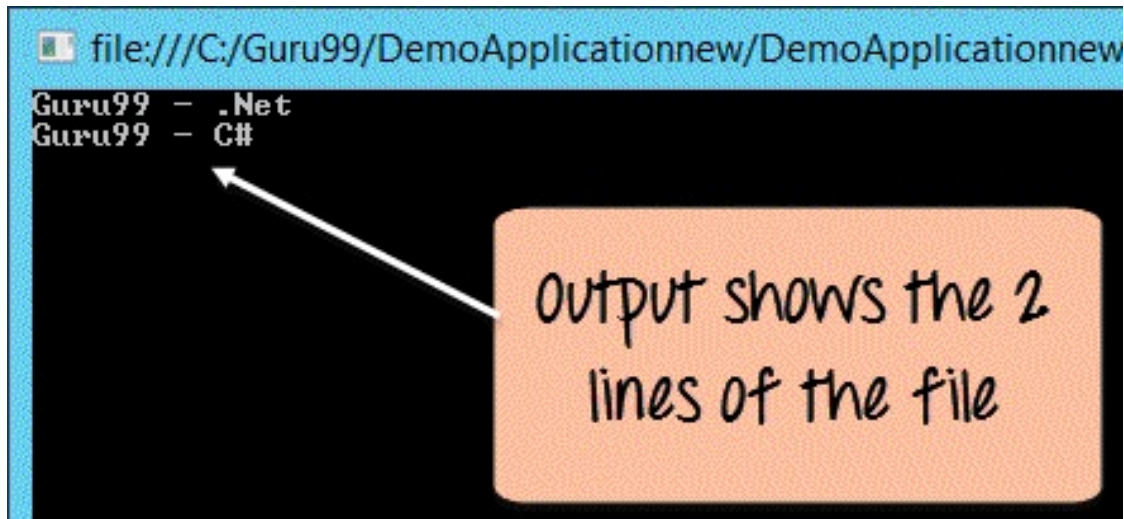
Code Explanation:-

1. First, we are declaring a string array variable. This will be used to store the result which will be returned by the File.ReadAllLines method.
2. Next, we use the File.ReadAllLines method to read all the lines from our text file. The result is then passed to the lines variable.

3. Since we know that our file contains only 2 lines, we can access the value of the array variables via the `lines[0]` and `lines[1]` command.

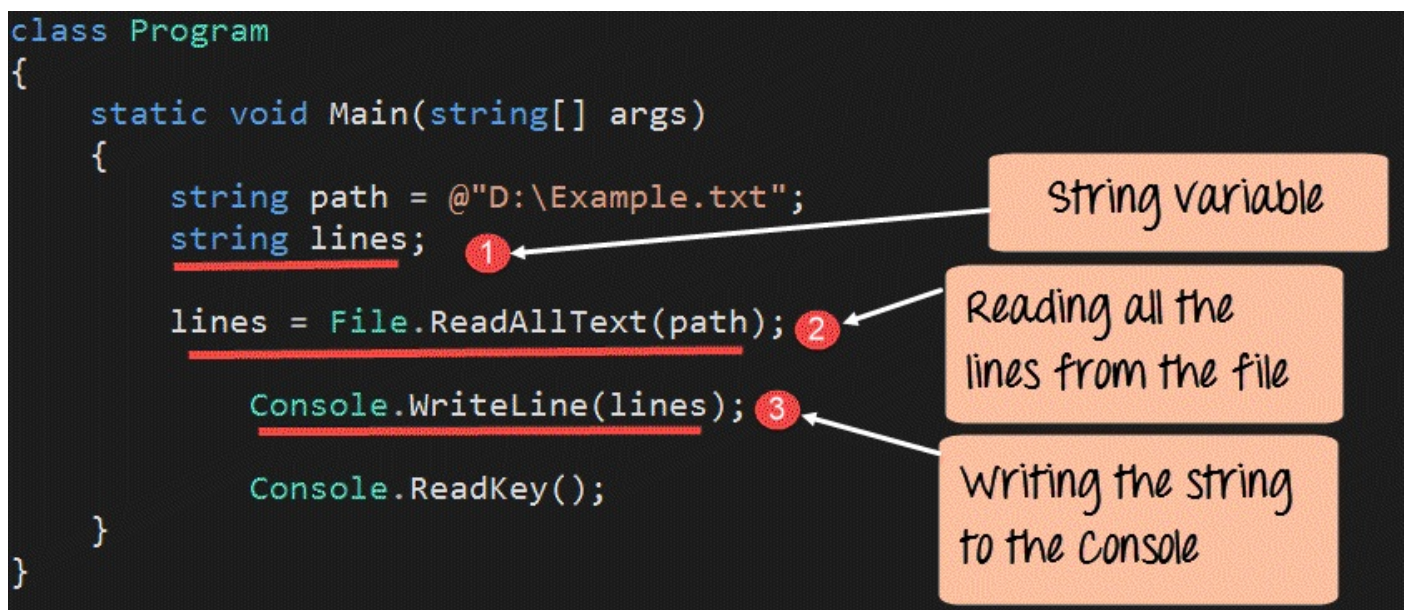
When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



From the output, you can clearly see that the `File.ReadAllLines` command returned both the lines from our file `Example.txt`

3. **File.ReadAllText** – This method is used to read all the lines in a file at once. The lines are then stored in a string variable. Let's look at an example. Enter the below code in the `program.cs` file.



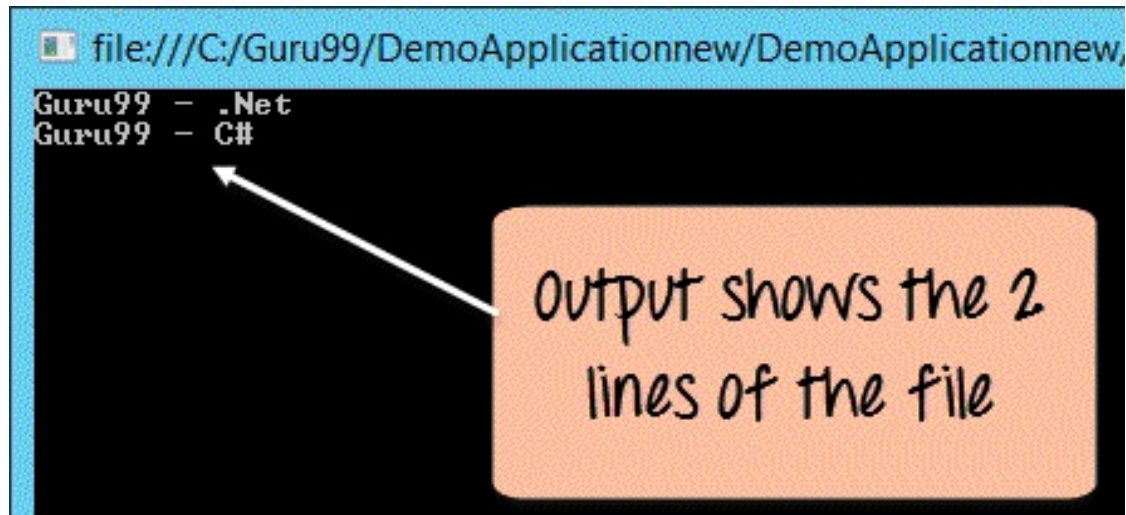
Code Explanation:-

1. First, we are declaring a string variable called `Lines`. This will be used to store the result which will be returned by the `File.ReadAllText` method.

2. Next, we use the File.ReadAllText method to read all the lines from our text file. The result is then passed to the lines variable.
3. We can directly use the Console.WriteLine method to display the value of the Lines variable.

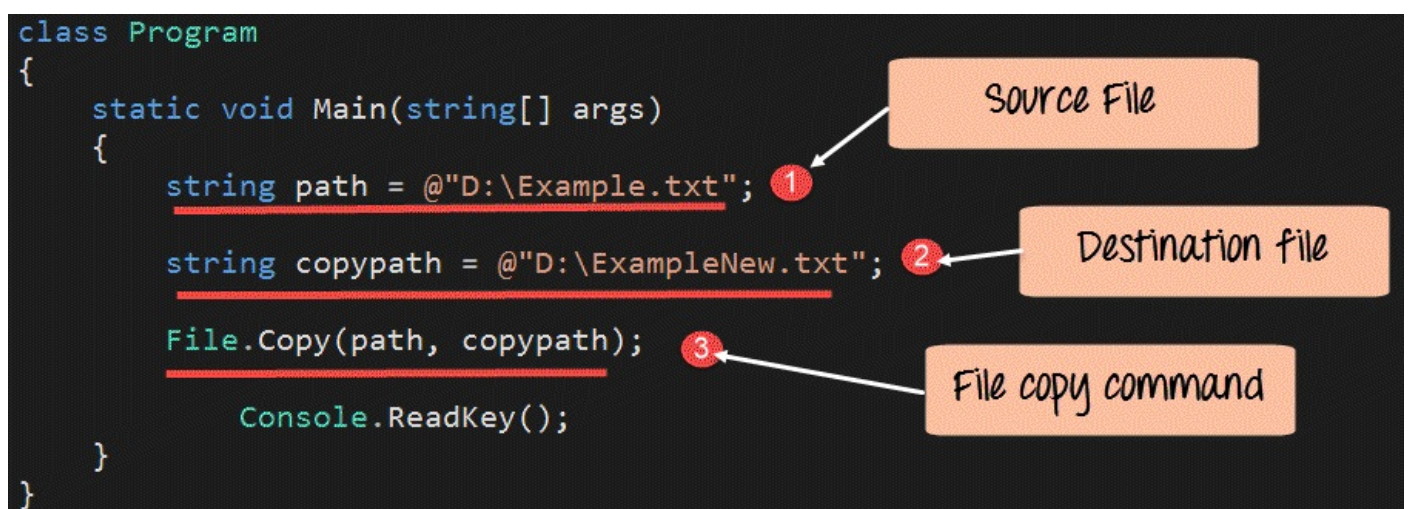
When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



From the output, you can clearly see that the File.ReadAllText command returned both the lines from our file Example.txt

4. **File.Copy** – The method is used to make a copy of an existing file. Let's look at an example. Enter the below code in the program.cs file.



Code Explanation:-


1. First, we are declaring a string variable called path. This will be the location of our Example.txt file. This file will be the source file used for the copy operation.

2. Next, we are declaring a string variable called `copypath`. This will be the location of a new file called `ExampleNew.txt` file. This will be the destination file in which the contents will be written from the source file `Example.txt`.
3. We then call the `File.Copy` method to copy the file `Example.txt` file to the file `ExampleNew.txt`.

When the above code is set, and the project is run using Visual Studio, the file `Example.txt` will be copied to `ExampleNew.txt`.

5. **File.Delete** – The method is used to delete an existing file. Let's look at an example. Enter the below code in the `program.cs` file.

```
class Program
{
    static void Main(string[] args)
    {
        string path = @"D:\Example.txt"; 1
        File.Delete(path); 2
        Console.ReadKey();
    }
}
```



Code Explanation:-

1. First, we are declaring a string variable called `path`. This will be the location of our `Example.txt` file. This is the file which will be deleted.
2. Next, we are calling the `File.Delete` method to delete the file.

When the above code is set, and the project is run using Visual Studio, the file `Example.txt` will be deleted from the D drive.

Streams – Reading and Writing to files

In C# file operations, normally streams are used to read and write to files. A stream is an additional layer created between an application and a file. The stream is used to ensure smooth read and write operations to the file.

Streams are normally used when reading data from large files. By using streams, the data from large files is broken down into small chunks and sent to the stream. These chunks of data can then be read from the application.

The reason for breaking it down into small chunks is because of the performance impact of reading a big file at one shot. If you were to read the data from say, a 100 MB file at one shot, your application could just hang and become unstable. The best approach is then to use streams to break the file down into manageable chunks.

So when a write operation is carried out on the file, the data to be written, is first written to the stream. From the stream, the data is then written to the file. The same goes for the read operation. In the read operation, data is first transferred from the file to the stream. The data is then read from the application via the stream. Let's look at an example of how we can read and write using streams.

For our example, we will assume that we have a file in the D drive called Example.txt. The file will be a simple text file and have 2 lines as shown below

- Guru99 - .Net
- Guru99 -C#

For our example, we will create a simple Console application and work with File streams

1. **Stream Reader** – The stream reader is used to read data from a file using streams. The data from the file is first read into the stream. Thereafter the application reads the data from the stream.

Let's look at an example of how we can use streams for reading data from a file. Enter the below code in the program.cs file.

```
{
    string path = @"D:\Example.txt";
    using (StreamReader sr = File.OpenText(path)){
        string s = "";
        while ((s = sr.ReadLine()) != null)
        {
            Console.WriteLine(s);
        }
    }
}
Console.ReadKey();
```

Defining the stream reader

Defining temporary string

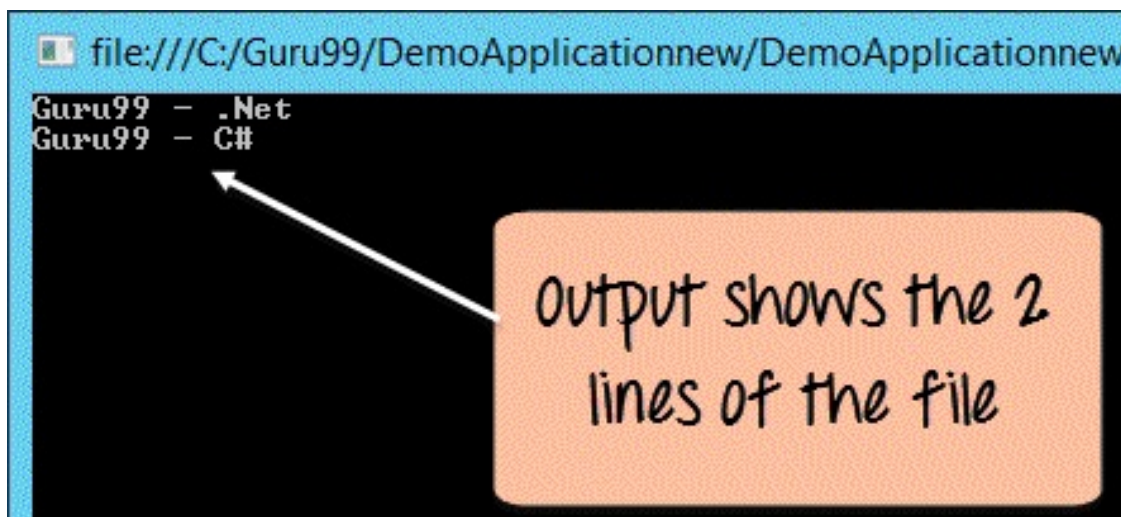
Reading each line of the stream

Code Explanation:-

1. First, we are declaring a stream reader object. The stream reader object is used in C# to define a stream from the file to the application. The data will be pushed from the file to the stream whenever data is read from the file. The File.OpenText is used to open the file “Example.txt” in read-only mode. The handler to the file is then sent to the stream reader object.
2. Next, we are defining a temporary variable ‘s’ which will be used to read all the data from the file.
3. We then use the stream reader method ReadLine to read each line from the stream buffer. When we perform this operation, each line will be first transferred from the file to the buffer. Then the string line will be transferred from the buffer to the variable ‘s’. We then write the contents of the string ‘s’ to the console.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



From the output, you can clearly see that the Stream Reader read both the lines from the file. Finally, the lines of the string read from the stream were sent to the Console.

2. **Stream Writer** – The stream writer is used to write data to a file using streams. The data from the application is first written into the stream. Thereafter the stream writes the data to the file. Let's look at an example of how we can use streams for writing data from a file. Enter the below code in the program.cs file.

```
{
    string path = @"D:\Example.txt";
    using (StreamWriter sr = File.AppendText(path))
    {
        sr.WriteLine("Guru99 - ASP.Net");
        sr.Close();
        Console.WriteLine(File.ReadAllText(path));
    }
    Console.ReadKey();
}
```

The code is annotated with four numbered steps in red circles and arrows pointing to the corresponding lines of code:

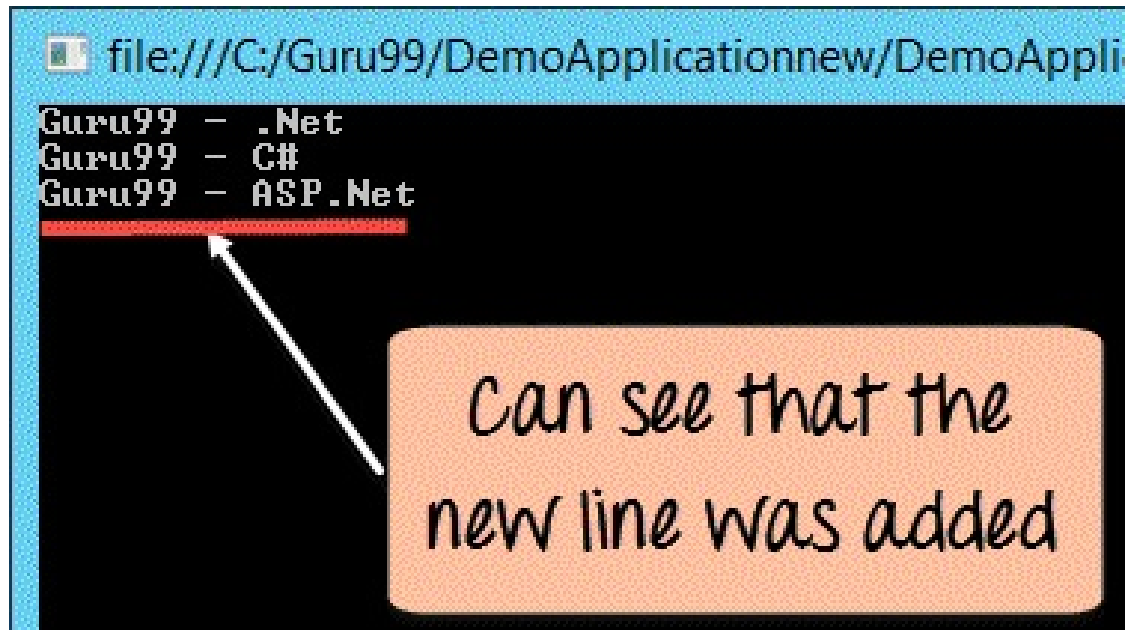
- 1. **Open StreamWriter object**: Points to the `using (StreamWriter sr = File.AppendText(path))` line.
- 2. **Write to the stream**: Points to the `sr.WriteLine("Guru99 - ASP.Net");` line.
- 3. **Close the stream**: Points to the `sr.Close();` line.
- 4. **Read the contents of the file**: Points to the `Console.WriteLine(File.ReadAllText(path));` line.

Code Explanation:-

1. First, we are declaring a stream writer object. The stream writer object is used in C# to define a stream. The stream is then used to write data from the application to the file. The data will be pushed from the application to the stream whenever data needs to be written. The `File.AppendText` command is used to open the file “Example.txt” in an append mode. The handler to the file is then sent to the stream writer object.
2. We are using the stream write method `Writeline` to write the line “Guru99 – ASP.Net” to the stream. From the stream, the line will then be written to the file.
3. We then close the stream writer after writing to the file. It's normally a good practice to close file handlers when the file is no longer required for writing purposes.
4. Finally, we are reading the contents of the file again and writing it to the console log. This is to check as to whether the line was really written to the file.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



```
file:///C:/Guru99/DemoApplicationnew/DemoAppli
Guru99 - .Net
Guru99 - C#
Guru99 - ASP.Net
```

Can see that the
new line was added

From the output, you can see that the line “Guru99 – ASP.Net” was added to the file successfully. All the 3 lines of text can be clearly seen in the console.

C# Serialization

The concept of Serialization and deserialization is used whenever data pertaining to objects have to be sent from one application to another. Serialization is used to export application data into a file. The destination application then uses deserialization to extract the data from the application for further use.

Serialization is a concept in which C# class objects are written or serialized to files. Let's say you had a C# class called Tutorial. And the class has 2 properties of ID and Tutorial's name.

Serializing can be used to directly write the data properties of the Tutorial class to a file. Deserialization is used to read the data from the file and construct the Tutorial object again.

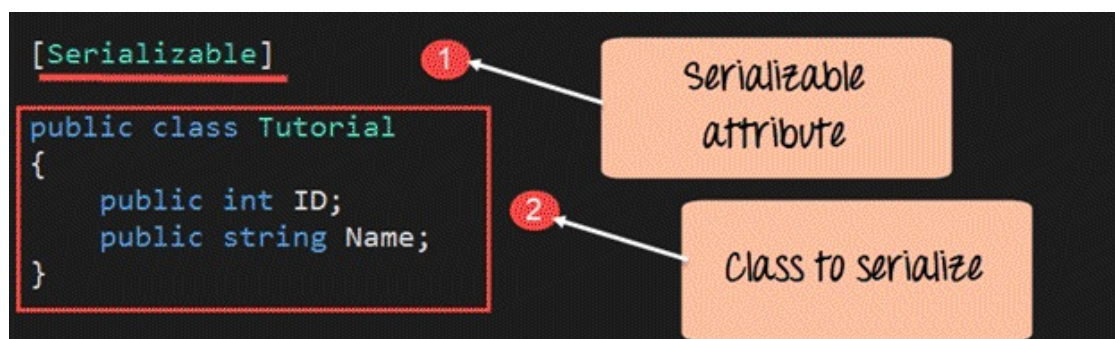
Let's look at an example of how we can achieve this.

In our example, we are going to perform the below high-level steps in the code

1. Create a class called Tutorial which has 2 properties, namely ID and Name
2. We will then create an object from the class and assign a value of "1" to the ID property and a value of ".Net" to the name property.
3. We will then use serialization to serialize the above object to a file called Example.txt
4. Finally, we will use deserialization to deserialize the object from the file and display the values in the Console.

Enter the below code in the program.cs file of the console application.

Step 1) The first step is to add the class which will be used for serialization



Code Explanation:-

1. The class which needs to be serialized needs to have the [Serializable] attribute. This is a keyword in C#. This keyword is then attached to the

Tutorial class. If you don't mention this attribute, you will get an error when you try to serialize the class.

2. Next is the definition of the class which will be serialized. Here we are defining a class called "Tutorial" and providing 2 properties, one is "ID" and the other is "Name."

Step 2) In this step, first we will create the object of the Tutorial class and serialize it to the file called Example.txt

The screenshot shows a C# program in a Visual Studio-like environment. The code is as follows:

```
static void Main(string[] args)
{
    Tutorial obj = new Tutorial();
    obj.ID = 1;
    obj.Name = ".Net";

    IFormatter formatter = new BinaryFormatter();
    Stream stream = new FileStream("D:\\Example.txt", FileMode.Create, FileAccess.Write);

    formatter.Serialize(stream, obj);
    stream.Close();
}
```

Annotations on the image:

- 1**: Points to the object initialization code: `Tutorial obj = new Tutorial(); obj.ID = 1; obj.Name = ".Net";`. The annotation box says "Initialize an object".
- 2**: Points to the file stream creation code: `IFormatter formatter = new BinaryFormatter(); Stream stream = new FileStream("D:\\Example.txt", FileMode.Create, FileAccess.Write);`. The annotation box says "Create the file stream".
- 3**: Points to the serialization code: `formatter.Serialize(stream, obj); stream.Close();`. The annotation box says "Serialize the object".

Code Explanation:-

1. First, we create an object of the Tutorial class. We then assign the value of "1" to ID and ".net" to the name property.
2. We then use the formatter class which is used to serialize or convert the object to a binary format. The data in the file in serialization is done in binary format. Next, we create a file stream object. The file stream object is used to open the file Example.txt for writing purposes. The keywords `FileMode.Create` and `FileMode.Write` is used to specifically mention that the file should be opened for writing purposes.
3. Finally, we use the `Serialize` method to transfer the binary data to the file. We then close the stream, since the write operation is complete.

Step 3) Finally to ensure that the data is present in the file, we use deserialization to deserialize the object from the file.

```
stream = new FileStream("D:\\Example.txt", FileMode.Open, FileAccess.Read);
Tutorial objnew = (Tutorial)formatter.Deserialize(stream);
Console.WriteLine(objnew.ID);
Console.WriteLine(objnew.Name);
Console.ReadKey();
}
```

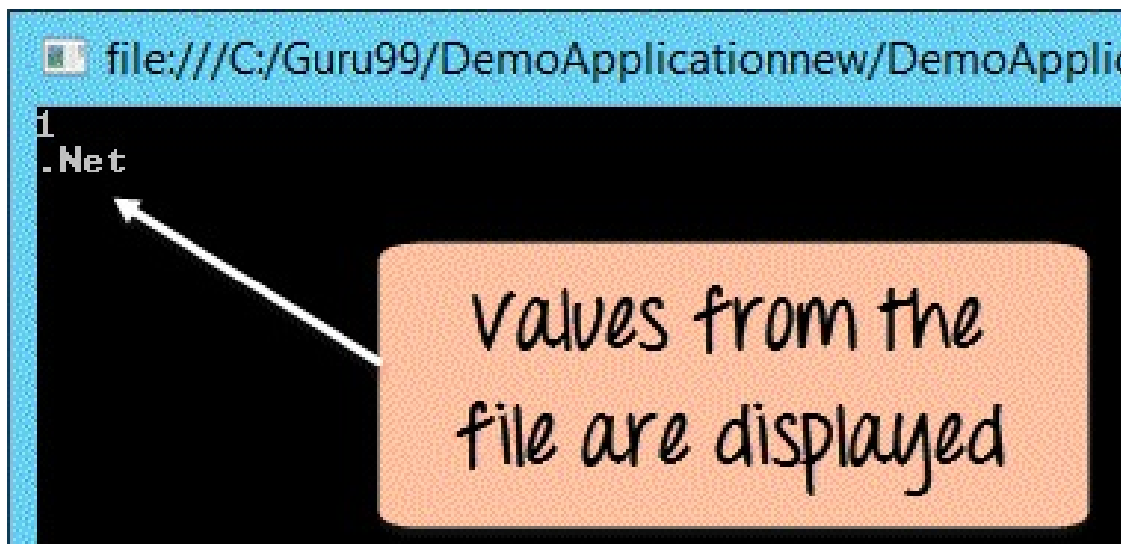
The diagram consists of three orange callout boxes with white text and arrows pointing to specific lines of code. Callout 1 points to the `FileStream` constructor. Callout 2 points to the `Deserialize` method. Callout 3 points to the `WriteLine` statements.

Code Explanation:-

1. We create the object “stream” to open the file Example.txt in read only mode.
2. We then use the formatter class which is used to deserialize the object, which is stored in the Example.txt file. The object returned is set to the object objnew.
3. Finally, we display the properties of the object “objnew” to the console using the “ID” and “name” properties.

When the above code is set, and the project is run using Visual Studio, you will get the below output.

Output:-



You can see from the above output that the values from the file were deserialized properly and displayed in the console.

Summary

- C# has a number of File operations which can be performed on files. Most of these operations are part of the class File.
- If you want to read data from a file, you can use the File.ReadAllText or File.ReadAllLines methods.

- Streams are used as an intermediate level between the application and the file.
 - A `StreamReader` is used whenever data is required to be read from a file.
 - A `StreamWriter` is used whenever data needs to be written to a file.
- Serialization is used to write class objects to files. De-Serialization is used to recover the objects from the file.