C Sharp

One Language For All Devices

Eng. / mina anwer Lewes
01221926646

Content :

Chapter One

Introduction to Our great IDE (Visual Studio).

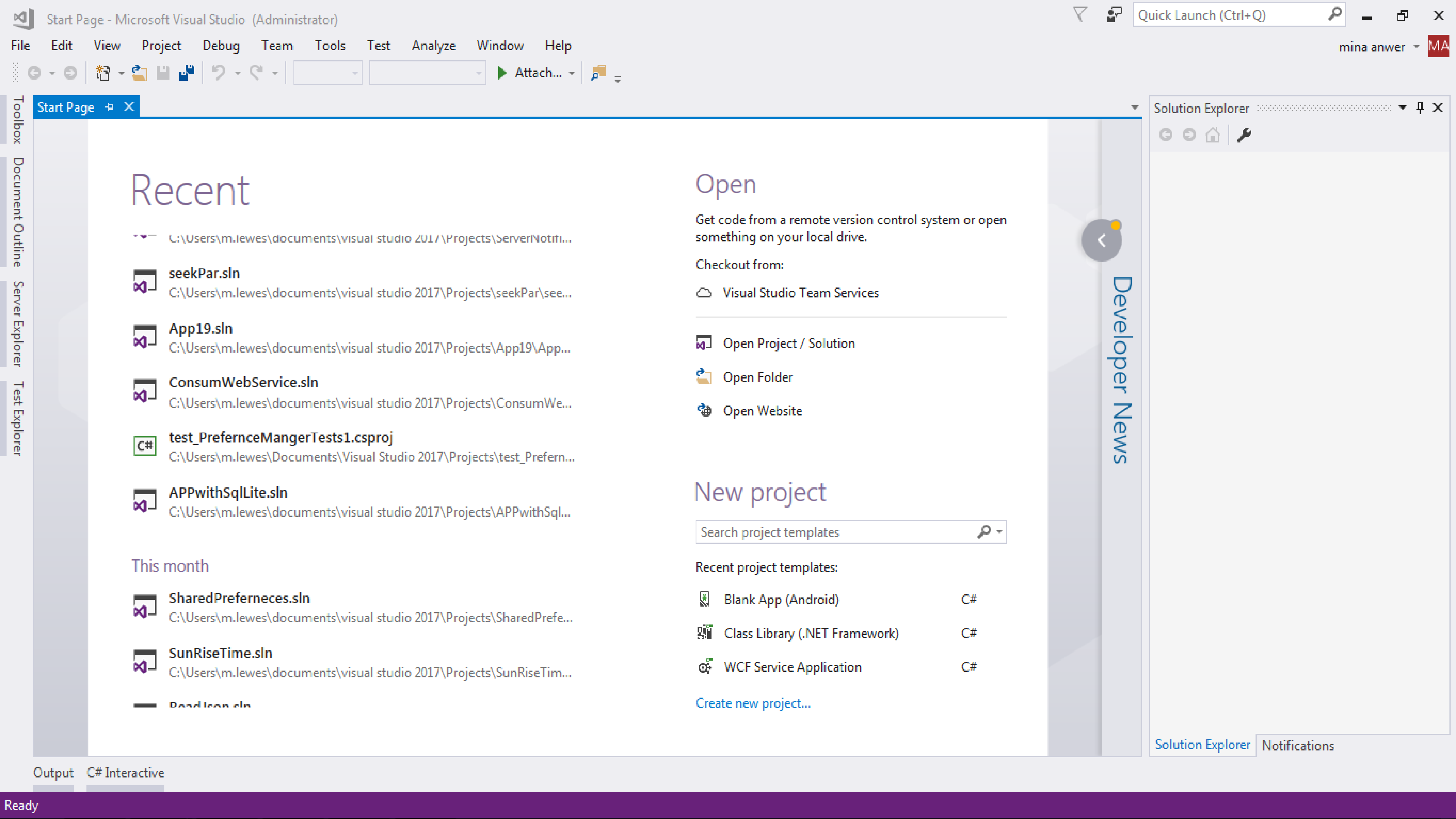
Chapter One Content :

1- install visual Studio 2013 , 2015 , 2017 .
2- Visual studio history .
3- Lets have a look .

Visual Studio 2013

2015

Visual Studio 2017



Recent

C:\Users\m.lewes\documents\visual studio 2017\Projects\ServerNotifi...

seekPar.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\seekPar\see...

App19.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\App19\App...

ConsumWebService.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\ConsumWe...

test_PrefernceMangerTests1.csproj
C:\Users\m.lewes\Documents\Visual Studio 2017\Projects\test_Prefern...

APPwithSqlLite.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\APPwithSql...

This month

SharedPreferneces.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\SharedPrefe...

SunRiseTime.sln
C:\Users\m.lewes\documents\visual studio 2017\Projects\SunRiseTim...

ReadJson.sln

Open

Get code from a remote version control system or open something on your local drive.

Checkout from:

Visual Studio Team Services

Open Project / Solution

Open Folder

Open Website

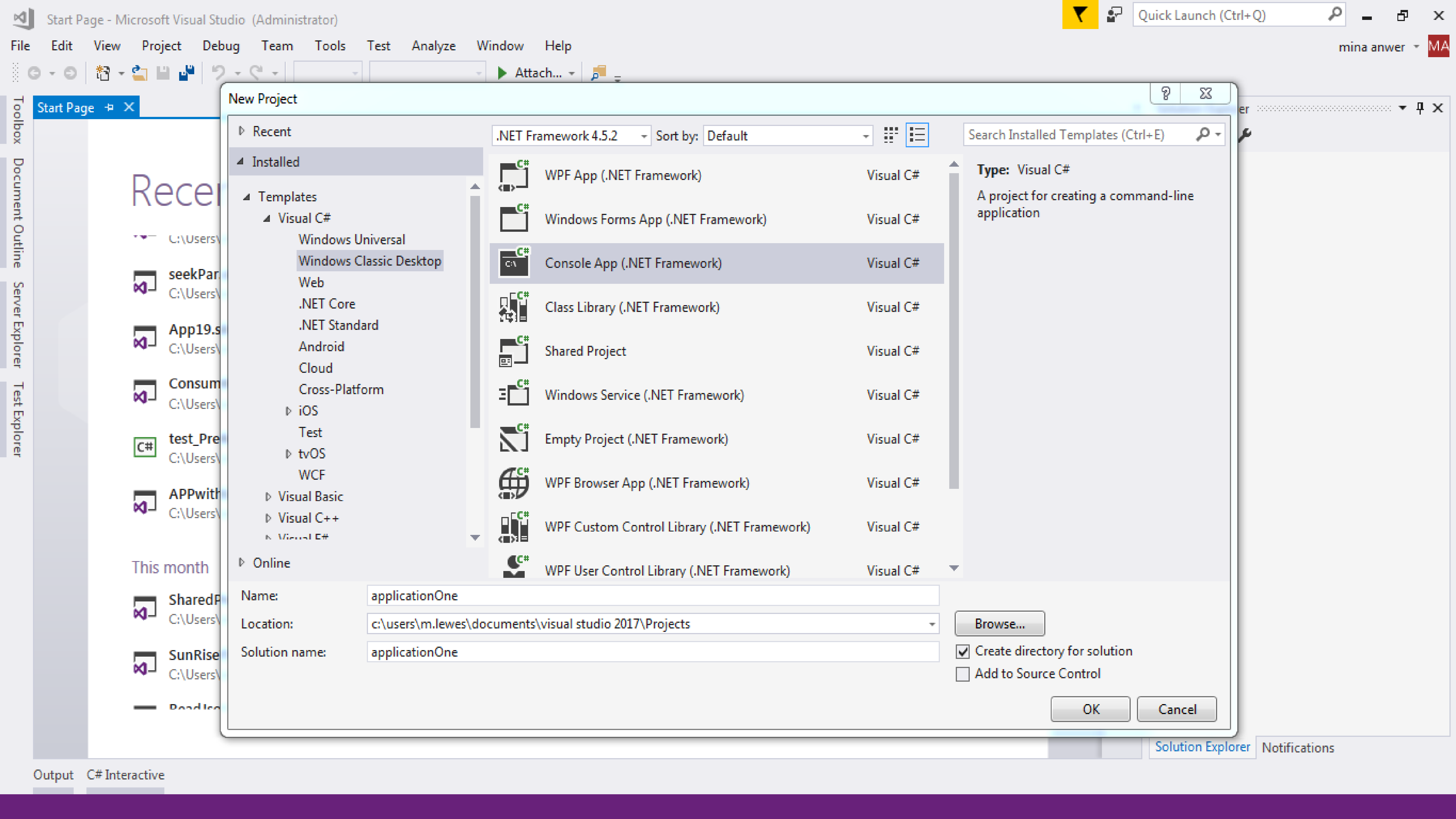
New project

Search project templates

Recent project templates:

Blank App (Android)	C#
Class Library (.NET Framework)	C#
WCF Service Application	C#

Create new project...



New Project

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows Universal
      - Windows Classic Desktop
      - Web
      - .NET Core
      - .NET Standard
      - Android
      - Cloud
      - Cross-Platform
      - iOS
      - Test
      - tvOS
      - WCF
    - Visual Basic
    - Visual C++
- Online

.NET Framework 4.5.2 Sort by: Default

Search Installed Templates (Ctrl+E)

WPF App (.NET Framework)	Visual C#
Windows Forms App (.NET Framework)	Visual C#
Console App (.NET Framework)	Visual C#
Class Library (.NET Framework)	Visual C#
Shared Project	Visual C#
Windows Service (.NET Framework)	Visual C#
Empty Project (.NET Framework)	Visual C#
WPF Browser App (.NET Framework)	Visual C#
WPF Custom Control Library (.NET Framework)	Visual C#
WPF User Control Library (.NET Framework)	Visual C#

Type: Visual C#

A project for creating a command-line application

Name: applicationOne

Location: c:\users\m.lewes\documents\visual studio 2017\Projects

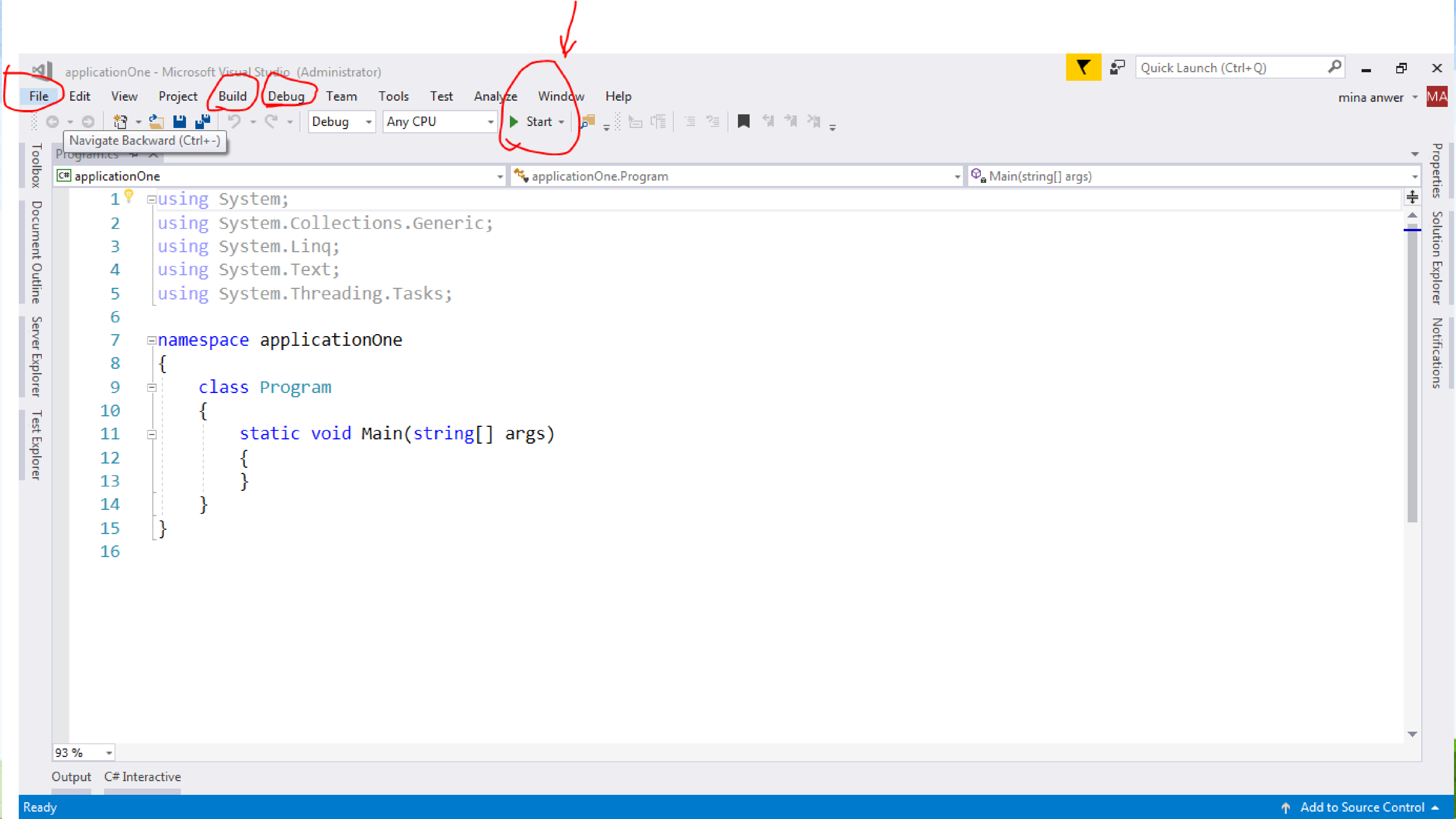
Solution name: applicationOne

Browse...

[✓] Create directory for solution

[ ] Add to Source Control

OK Cancel



```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace applicationOne
{
    class Program
    {
        static void Main(string[] args)
        {
        }
    }
}
```

Chapter Two

Programming Basics .

Chapter Two Content :

1- Hello World Program .

2- Data Types .

3- Operators .

4- Conditional Statements .

5- Looping .

6- Functions and Function Overloading .

7- Arrays .

8- operator overloading (self study) .

1- Hello World Program .

- Your First Hello World Program will be some thing Like this :

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication1
{
        class Program
       {
                 static void Main(string[] args)
                 {

                      Console.WriteLine("Hello, world!");
                      Console.ReadLine();
                 }
        }
}
```

1- Hello World Program .

- Hit F5 to Run your app .
- Hit F4 to Debug .
- Use all debugging tools to debug your code .

2- Data Types :

bool is one of the simplest data types. It can contain only 2 values

false or true. The bool type is important to understand when using logical operators like the if statement.

Int is short for integer, a data type for storing numbers without decimals. When working with numbers, int is the most commonly used data type. Integers have several data types within C#, depending on the size of the number they are supposed to store.

string is used for storing text, that is, a number of chars. In C#, strings are immutable, which means that strings are never changed after they have been created. When using methods which changes a string, the actual string is not changed - a new string is returned instead.

char is used for storing a single character.

Float is one of the data types used to store numbers which may or may not contain decimals.

And more ……..

2- Data Types :

- What is variable ?
  A variable can be compared to a storage room, and is essential for the programmer.
- In C#, a variable is declared like this:

```
<data type> <name>;
```

An example could look like this:

```
string name;
int age ;
```

demo :

```
using System;

namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            string firstName = “tota";
            string lastName = “ali";

            Console.WriteLine("Name: " + firstName + " " + lastName);

            Console.WriteLine("Please enter a new first name:");
            firstName = Console.ReadLine();

            Console.WriteLine("New name: " + firstName + " " + lastName);

            Console.ReadLine();
        }
    }
}
```

Home Work

We need application that Receive Username and Password from User and show welcome message to user :
welcome + Mr. + username

dead line : today 12 AM .

3- operators :

+ Arithmetic Operators

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

3- operators :

+ Relational Operators

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

3- operators :

+ Logical Operators :

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non zero then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

3- operators :

+ Bitwise Operators

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = 61, which is 1100 0011 in 2's complement due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15, which is 0000 1111

3- operators :

Read more about Operators :

+ Miscellaneous Operators

+ Operator Precedence in C#

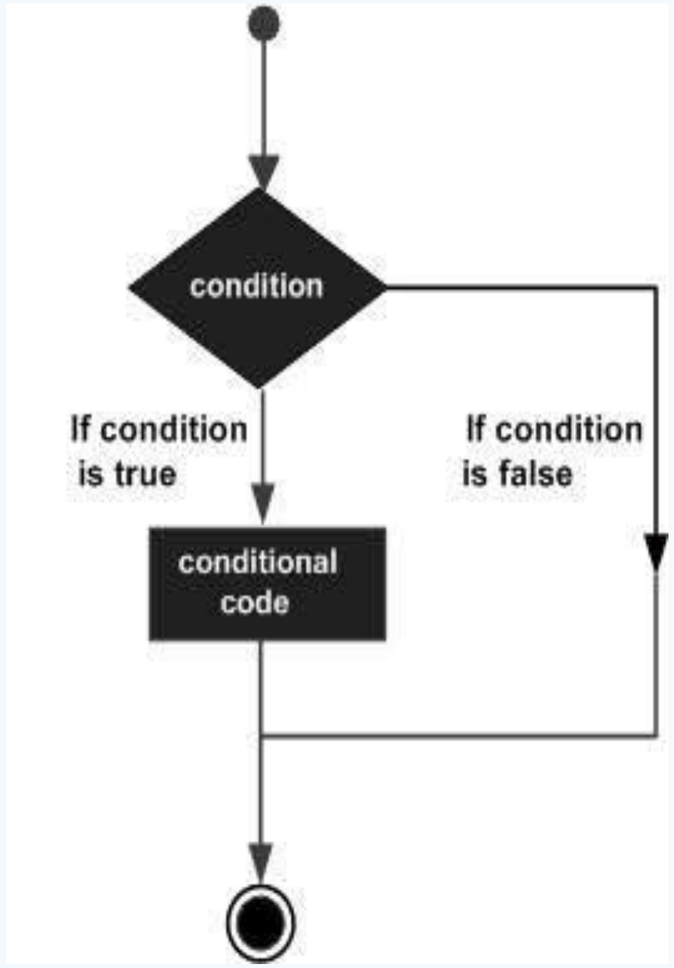
in :

https://www.tutorialspoint.com/csharp/csharp_operators.htm

4- Conditional Statements .

Following is the general form of a typical decision making structure found in most of the programming languages:

Statement	Description
if statement	An if statement consists of a Boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.
nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).



4- Conditional Statements .

The ? : Operator:

Exp1 ? Exp2 : Exp3 ;

Where Exp1, Exp2, and Exp3 are expressions.

Notice the use and placement of the colon.

The value of a ? expression is determined as follows: Exp1 is evaluated. If it is true, then Exp2 is evaluated and becomes the value of the entire ? expression. If Exp1 is false, then Exp3 is evaluated and its value becomes the value of the expression.

5- Looping

The while loop

The while loop is probably the most simple one, so we will start with that. The while loop simply executes a block of code as long as the condition you give it is true. A small example, and then some more explanation:

```
using System;
namespace ConsoleApplication1
 {
  class Program
   {
       static void Main(string[] args)
          {
             int number = 0;
                     while(number < 5)
                    {
                      Console.WriteLine(number);
                      number = number + 1;
                    }
                       Console.ReadLine();
           }}}
```

5- Looping

The do loop

The opposite is true for the do loop, which works like the while loop in other aspects through. The do loop evaluates the condition after the loop has executed, which makes sure that the code block is always executed at least once.

```
Do
{

    Console.WriteLine(number) ;
     number = number + 1;
}
while(number < 5);
```

5- Looping

The for loop

The for loop is a bit different. It's preferred when you know how many iterations you want, either because you know the exact amount of iterations, or because you have a variable containing the amount. Here is an example on the for loop.

```
using System;
 namespace ConsoleApplication1
{
           class Program
         {
               static void Main(string[] args)
                {
                     int number = 5;
                      for(int i = 0; i < number; i++)
                        Console.WriteLine(i);
                        Console.ReadLine();
                }
         }
 }
```

5- Looping

Loop Control Statements

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

Control Statement	Description
break statement	Terminates the loop or switch statement and transfers execution to the statement immediately following the loop or switch.
continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.

5- Looping

Infinite Loop

A loop becomes infinite loop if a condition never becomes false. The for loop is traditionally used for this purpose.

```
using System;
namespace Loops
 {
   class Program
     {
                static void Main(string[] args)
                  {
                         for (    ;      ;     )
                           {
                              Console.WriteLine("Hey! I am Trapped");
                            }
                     }
        }
  }
```

6- Functions and Function Overloading .

Functions :

A function allows you to encapsulate a piece of code and call it from other parts of your code. You may very soon run into a situation where you need to repeat a piece of code, from multiple places, and this is where functions come in. In C#, they are basically declared like this :

```
<visibility> <return type> <name>(<parameters>)
 {
    <function code>
 }
```

To call a function, you simply write its name, an open parenthesis, then parameters, if any, and then a closing parenthesis, like this:

```
DoStuff();
```

6- Functions and Function Overloading .

- *Demo " function to add two numbers " :*

```
public  int AddNumbers(int number1, int number2)
 {
     int result = number1 + number2;
     return result;
  }
```

To call this function

```
int result = AddNumbers(10, 5);
Console.WriteLine(result);
```

6- Functions and Function Overloading .

The ref modifier

Consider the following example:

```
static void Main(string[] args)
{
         int number = 20;
         AddFive(number);
         Console.WriteLine(number);
        Console.ReadKey();
 }

 static void AddFive(int number)
{
        number = number + 5;

 }
```

6- Functions and Function Overloading .

You will notice that the variable that incremented in function is not seen in main function that’s why we need to use ref keyword .

```
static void Main(string[] args)
{
         int number = 20;
         AddFive(number);
         Console.WriteLine(number);
        Console.ReadKey();
 }

 static void AddFive (ref int number)
{
        number = number + 5;

 }
```

6- Functions and Function Overloading .

The out modifier

A value passed to a ref modifier has to be initialized before calling the method - this is not true for the out modifier, where you can use un-initialized values. On the other hand, you can't leave a function call with an out parameter, without assigning a value to it Using the out modifier is just like using the ref modifier. Simply change the ref keyword to the out keyword.

6- Functions and Function Overloading .

The params modifier

```
static void Main(string[ ] args)
 {
          GreetPersons(0);
          GreetPersons(25, "John", "Jane", "Tarzan");
          Console.ReadKey( );
 }
  static void GreetPersons ( int someUnusedParameter, params string[ ] names)
{
        foreach(string name in names)
        Console.WriteLine("Hello, " + name);
}
```

7- Arrays.

Arrays are declared much like variables,

with a set of [ ] brackets after the datatype,

like this :

```
string[] names;
```

You need to instantiate the array to use it, which is done like this:

```
string[] names = new string[2];
```

The number (2) is the size of the array, that is, the amount of items we can put in it.

Putting items into the array is pretty simple as well:

```
names[0] = "John Doe";
```

7- Arrays .

```
using System;
using System. Collections;
namespace ConsoleApplication1
{
       class Program
      {
              static void Main(string[ ] args )
             {
                     string[ ] names = new string[2];
                     names[0] = "John Doe";
                      names[1] = "Jane Doe";
                     foreach   (string s in names)
                            Console.WriteLine(s);
                     Console.ReadLine();
             }
       }

}
```

Chapter Three

C# Fundamentals.

Chapter Three Content :

1- Classes .
2- Enumerations .

3- Nullables .

4- Structure .

5- Exception handling .

6- Debugging .

Introduction :

Besides or after finishing this chapter you need also to study my online
course about OOP :

https://www.youtube.com/watch?v=gJmskx4waBI&list=PL_3rspHWVSiG2RwYKF6dP6B99shuJ9WiS

```
Console.WriteLine    ( “  Have  Fun   !!!!   ”   )    ;
```

1- Classes :

A class is a group of related methods and variables.
A class describes these things, and in most cases, you create an instance of this class, now referred to as an object. On this object, you use the defined methods and variables. Of course, you can create as many instances of your class as you want to. Classes, and Object Oriented programming in general, is a huge topic. We will cover some of it in this chapter as well as in later chapters, but not all of it.

Encapsulation is defined 'as the process of enclosing one or more items within a physical or logical package'. Encapsulation, in object oriented programming methodology, prevents access to implementation details.
Encapsulation is implemented by using access specifiers. An access specifier defines the scope and visibility of a class member. C# supports the following access specifiers:

- **+ Public**
- **+ Protected**
- **+ Protected internal**
- **+ Private**
- **+ Internal**

1- Classes :

Lets see some Examples of classes :

```
static void Main (  string [  ] args )
  {
      Car  mycar  =  new Car( ) ;
      mycar.Name = "lamobergeny";
      mycar.Color = "blue";
      mycar.SerialNumber = 8798798456465464;
      string  data=  mycar.GetCarName();
      Console.WriteLine("car color is :" + data);
      Console.ReadLine();
  }
      class Car
      {
          public string Color ;
          public string Name;
          public long SerialNumber;
                public string GetCarName(  )
                {
                    return this.Name;
                }
        }
```

1- Classes :

Properties :

Properties allow you to control the accessibility of a classes variables, and is the recommended way to access variables from the outside in an object oriented programming language like C#. In our chapter on classes .
A property is much like a combination of a variable and a method .

```
class Car
  {
      public string Color { get; set; }
      public string Name { get; set; }
      public long SerialNumber { get; set; }

      public string GetCarName(  )
      {
         return this.Name;
      }
  }
```

1- Classes :

Constructors

Constructors are special methods ,
used when instantiating a class. A constructor can never return anything,
which is why you don't have to define a return type for it.
A normal method is defined like this .

```
 class Car
{
    public string Color { get; set; }
    public string Name { get; set; }
    public long SerialNumber { get; set; }

    public string GetCarName(  )
    { return this.Name; }

    public Car ()
    {            }

     public Car (string _color)
    {    this.Color = _color;    }

     public Car (string _color , string _name)
       {this.Color = _color; this.Name = _name; }

    public Car(string _color , string _name , int _serial )
      { this.Color = _color; this.Name = _name;this.SerialNumber = _serial; }
}
```

1- Classes :

Destructors :

Since C# is garbage collected, meaing that the framework will free the objects that you no longer use, there may be times where you need to do some manual cleanup. A destructor, a method called once an object is disposed, can be used to cleanup resources used by the object. Destructors doesn't look very much like other methods in C#. Here is an example of a destructor for our Car class:

```
~Car (   )
{
   Console.WriteLine("Out..");
 }
```

1- Classes :

Method overloading :

It allows the programmer do define several methods with the same name, as long as they take a different set of parameters. When you use the classes of the .NET framework, you will soon realize that method overloading is used all over the place. A good example of this, is the Substring() method of the String class. It is with an extra overload, like this

```
string Substring (int startIndex)
string Substring (int startIndex, int length)
String Substring (int startIndex, int length , double data)
```

1- Classes :

public - the member can be reached from anywhere. This is the least restrictive visibility. Enums and interfaces are, by default, publicly visible.

protected - members can only be reached from within the same class, or from a class which inherits from this class.

internal - members can be reached from within the same project only.

protected internal - the same as internal, except that also classes which inherits from this class can reach it members, even from another project.

private - can only be reached by members from the same class. This is the most restrictive visibility. Classes and structs are by default set to private visibility.

1- Classes :

Static members

in some cases, you might like to have a class which you may use without instantiating it, or at least a class where you can use members of it without creating an object for it. For instance, you may have a class with a variable that always remains the same, no matter where and how it's used. This is called a static member, static because it remains the same .

```
 public static int CalculateArea ( int width, int height)
{
   return width * height;
}

 public static class Rectangle
{
   public static int CalculateArea( int width, int height)
   {
       return width * height;
   }
}
```

1- Classes :

Inheritance :

One of the absolute key aspects of Object Oriented Programming (OOP), which is the concept that C# is built upon, is inheritance, the ability to create classes which inherits certain aspects from parent classes.

Lets see some Examples :

2- Enumerations :

Enumerations are special sets of named values which all maps to a set of numbers, usually integers. They come in handy when you wish to be able to choose between a set of constant values, and with each possible value relating to a number, they can be used in a wide range of situations. As you will see in our example, enumerations are defined above classes, inside our namespace

```
public enum Days { Monday, Tuesday, Wednesday, Thursday, Friday,
                   Saturday, Sunday }


public enum Days { Monday = 1, Tuesday, Wednesday, Thursday,
                   Friday, Saturday, Sunday }


 Days day = (Days)5;
 Console.WriteLine(day);
 Console.ReadLine();
```

3- Nullables:

C# provides a special data types, the nullable types, to which you can assign normal range of values as well as null values.
For example, you can store any value from -2,147,483,648 to 2,147,483,647 or null in a Nullable<Int32> variable. Similarly, you can assign true, false, or null in a Nullable<bool> variable. Syntax for declaring a nullable type is as follows:

```
static void Main(string[] args)
 {
     int? num1 = null;
     int? num2 = 45;
     double? num3 = new double?();
     double? num4 = 3.14157;

     bool? boolval = new bool?();

     // display the values

     Console.WriteLine("Nullables at Show: {0}, {1}, {2}, {3}", num1,
                        num2, num3, num4);
     Console.WriteLine("A Nullable boolean value: {0}", boolval );
     Console.ReadLine();
 }
```

4-Structures :

In C#, a structure is a value type data type. It helps you to make a single variable hold related data of various data types. The struct keyword is used for creating a structure.
To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member for your program.

```
struct Books
{
    public string title;
    public string author;
    public string subject;
    public int book_id;
};
```

```csharp
public static void Main(string[] args)
       {

           Books Book1;   /* Declare Book1 of type Book */
           Books Book2;   /* Declare Book2 of type Book */
           /* book 1 specification */
           Book1.title = "C Programming";
           Book1.author = "Nuha Ali";
           Book1.subject = "C Programming Tutorial";
           Book1.book_id = 6495407;
           /* book 2 specification */
           Book2.title = "Telecom Billing";
           Book2.author = "Zara Ali";
           Book2.subject = "Telecom Billing Tutorial";
           Book2.book_id = 6495700;
           /* print Book1 info */
           Console.WriteLine("Book 1 title : {0}", Book1.title);
           Console.WriteLine("Book 1 author : {0}", Book1.author);
           Console.WriteLine("Book 1 subject : {0}", Book1.subject);
           Console.WriteLine("Book 1 book_id :{0}", Book1.book_id);

           /* print Book2 info */
           Console.WriteLine("Book 2 title : {0}", Book2.title);
           Console.WriteLine("Book 2 author : {0}", Book2.author);
           Console.WriteLine("Book 2 subject : {0}", Book2.subject);
           Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);

           Console.ReadKey();

       }
```

4-Structures :

Features of C# Structures

Structures in C# are quite different from that in traditional C or C++. The C# structures have the following features:

- Structures can have methods, fields, indexers, properties, operator methods, and events.
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure. The default constructor is automatically defined and cannot be changed.
- Unlike classes, structures cannot inherit other structures or classes.
- Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces.
- Structure members cannot be specified as abstract, virtual, or protected.
- When you create a struct object using the New operator, it gets created and the appropriate constructor is called. Unlike classes , structs can be instantiated without using the New operator.
- If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.

4-Structures :

Class versus Structure :

- Classes and Structures have the following basic differences:
- classes are reference types and structs are value types
- structures do not support inheritance
- structures cannot have default constructor

- Homework :
  => 10 min Presentation :
  Compare between struct and class in C++ and C# . With clear Demo

- note :
  difference between struct and class in c# is commonly asked as an interview
  question .

5- Exception handling :

In every program, things go wrong sometimes. With C#, we're blessed with a good compiler, which will help us prevent some of the most common mistakes. Obviously it can't see every error that might happen, and in those cases, the .NET framework will throw an exception, to tell us that something went wrong.

```
using System;
using System.Collections;
namespace ConsoleApplication1
    {
        class Program
        {
            static void Main(string[] args)
            {
                int[] numbers = new int[2];
                numbers[0] = 23;
                numbers[1] = 32;
                numbers[2] = 42;
                foreach (int i in numbers)
                    Console.WriteLine(i);
                Console.ReadLine();
            }
        }
    }
```

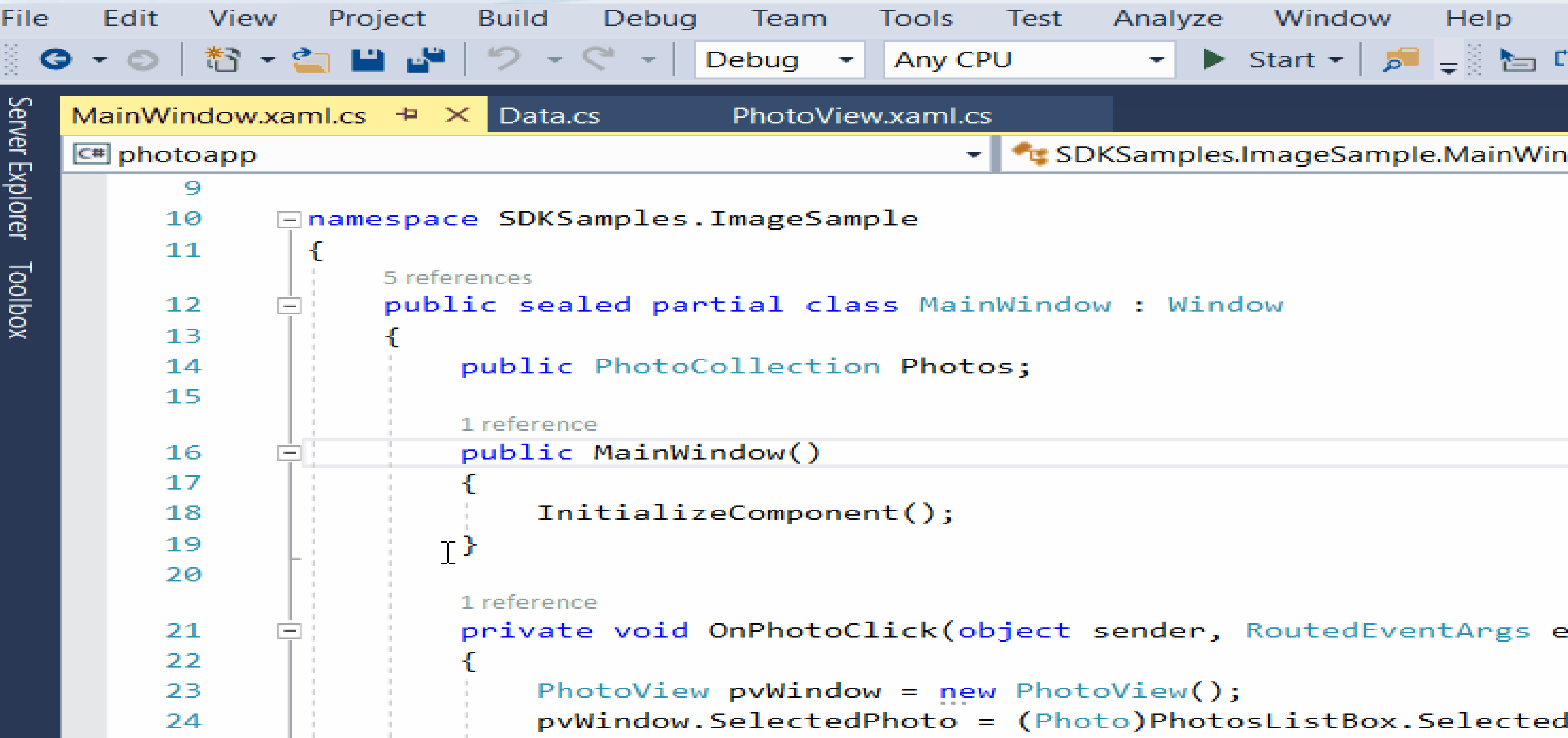
5- Exception handling :

You can use try with many catch statements and in the end you can use Finally statement

```
 int[] numbers = new int[2];
try
{
    numbers[0] = 23;
    numbers[1] = 32;
    numbers[2] = 42;

    foreach(int i in numbers)
        Console.WriteLine(i);
}
catch(IndexOutOfRangeException ex)
{
    Console.WriteLine("An index was out of range!");
}
catch(Exception ex)
{
    Console.WriteLine("Some sort of error occured: " + ex.Message);
}
finally
{
    Console.WriteLine("It's the end of our try block. Time to clean up!");
}
Console.ReadLine();
```

6- Debugging :

just Set a breakpoint and start the debugger


File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
Start
Server Explorer
Toolbox
MainWindow.xaml.cs
Data.cs
PhotoView.xaml.cs
photoapp
SDKSamples.ImageSample.MainWin

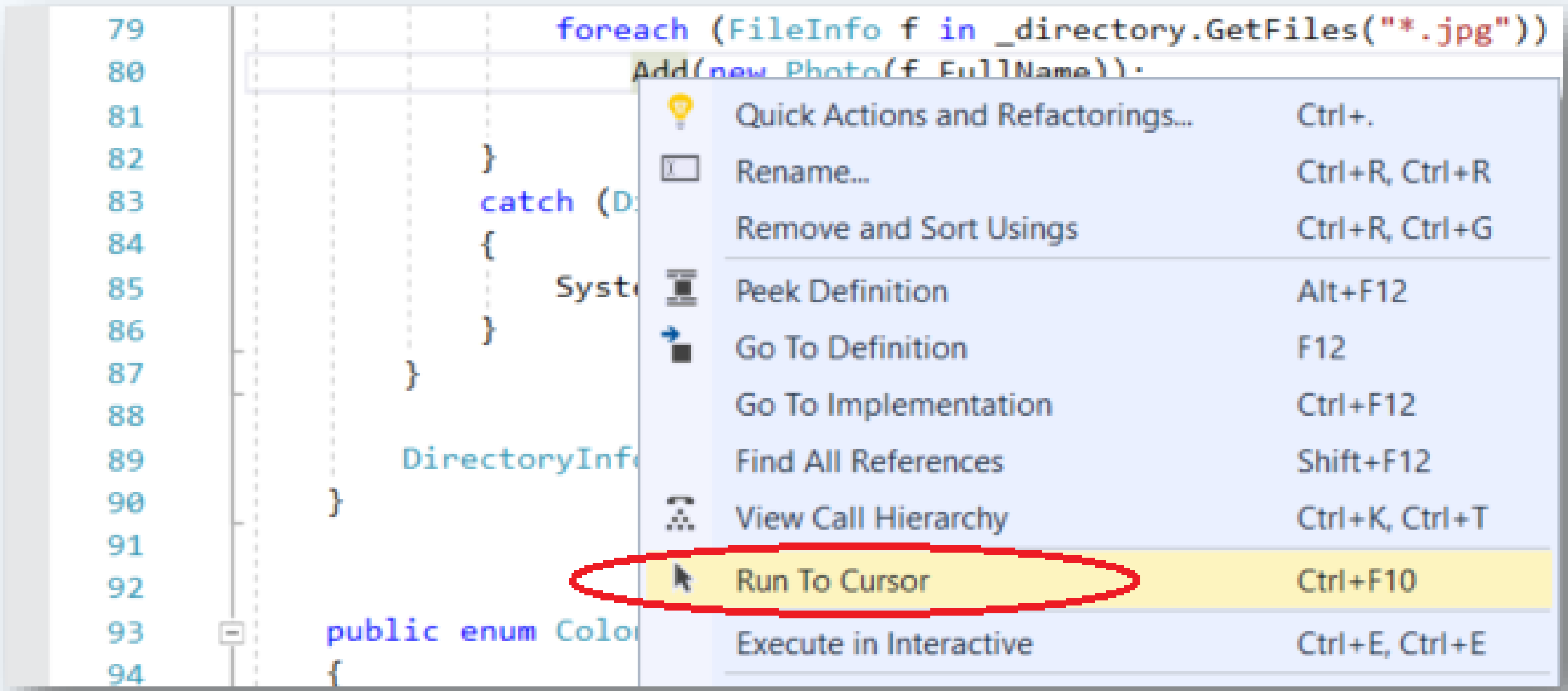
```
namespace SDKSamples.ImageSample
{
    5 references
    public sealed partial class MainWindow : Window
    {
        public PhotoCollection Photos;

        1 reference
        public MainWindow()
        {
            InitializeComponent();
        }

        1 reference
        private void OnPhotoClick(object sender, RoutedEventArgs e
        {
            PhotoView pvWindow = new PhotoView();
            pvWindow.SelectedPhoto = (Photo)PhotosListBox.Selected
```

6- Debugging :

Run to cursor

1.Choose the Stop Debugging red button or Shift + F5.

2.In the Update method, right-click the Add method call and choose Run to Cursor. This command starts debugging and sets a temporary breakpoint on the current line of code.



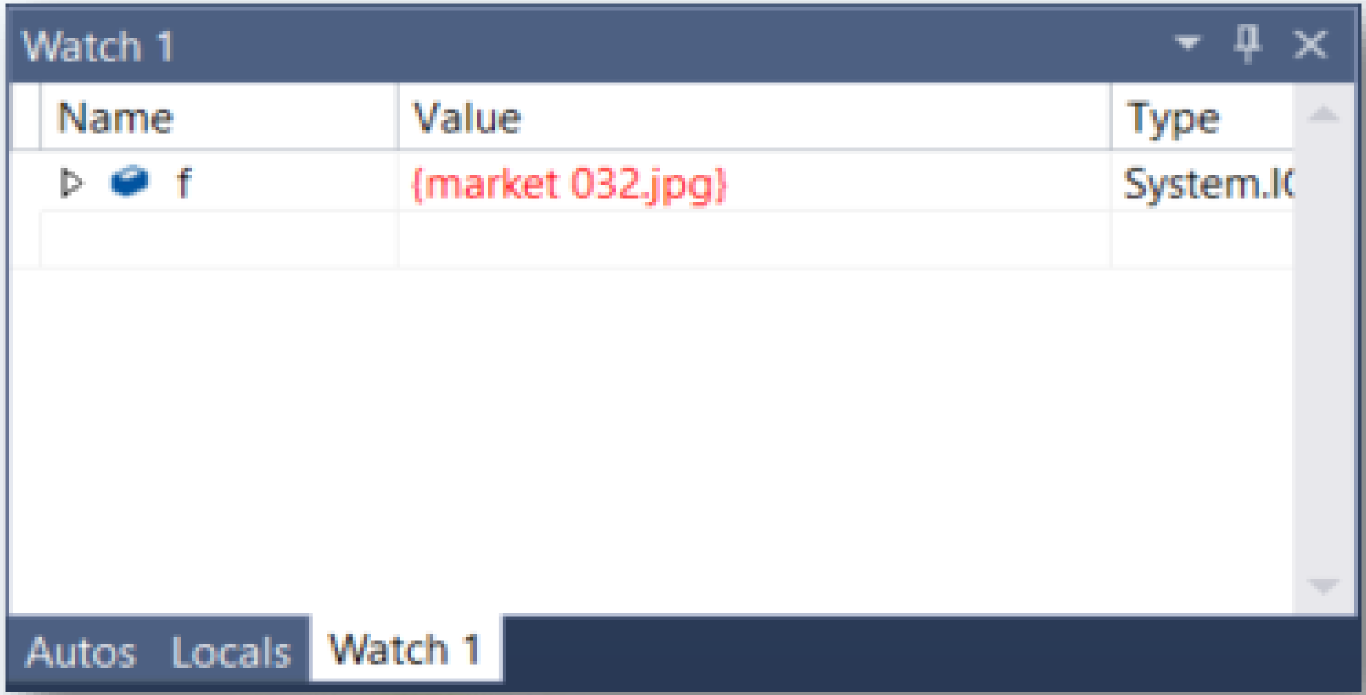
```
                foreach (FileInfo f in _directory.GetFiles("*.jpg"))
                    Add(new Photo(f.FullName));

            }
            catch (D
            {
                Syst
            }
        }

        DirectoryInf
    }


    public enum Colo
    {
```

Quick Actions and Refactorings... Ctrl+.
Rename... Ctrl+R, Ctrl+R
Remove and Sort Usings Ctrl+R, Ctrl+G
Peek Definition Alt+F12
Go To Definition F12
Go To Implementation Ctrl+F12
Find All References Shift+F12
View Call Hierarchy Ctrl+K, Ctrl+T
Run To Cursor Ctrl+F10
Execute in Interactive Ctrl+E, Ctrl+E

6- Debugging :

Set a watch

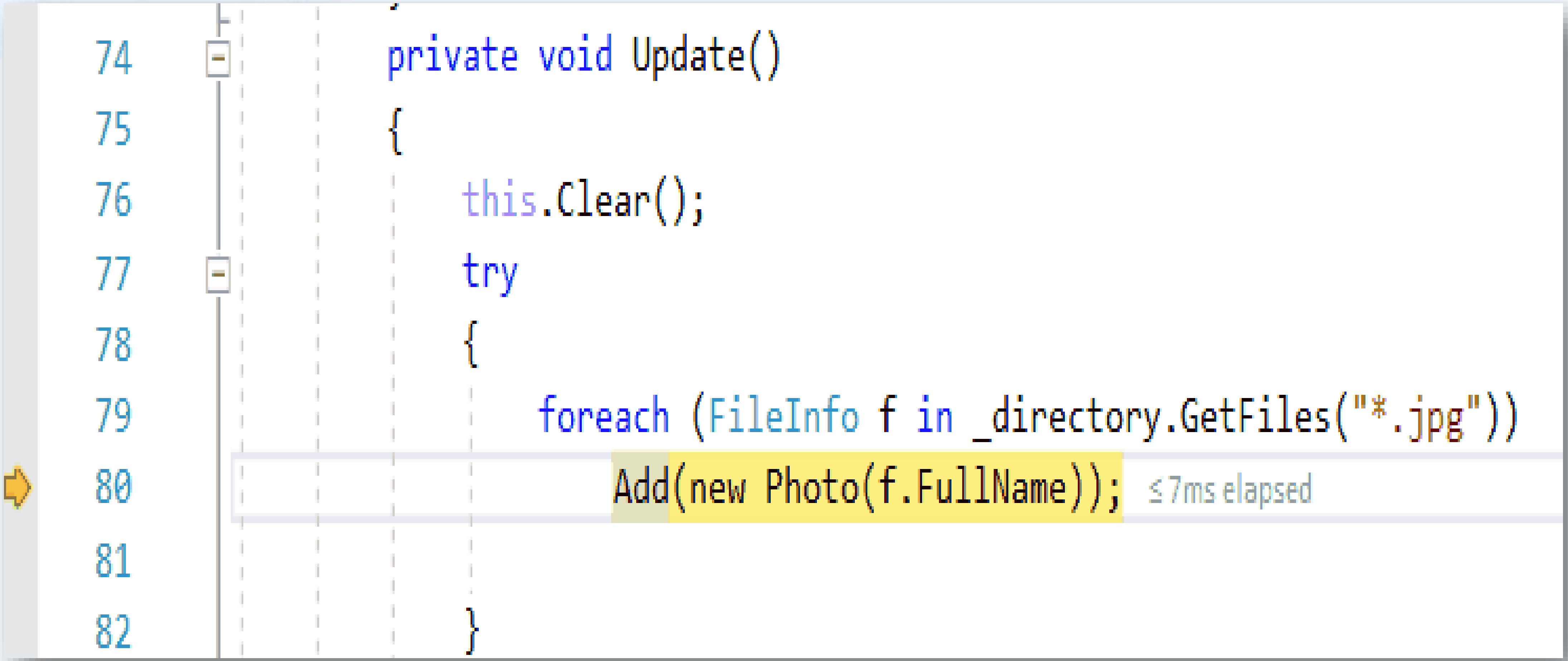
In the main code editor window, right-click the File object (f) and choose Add Watch.
You can use a Watch window to specify a variable (or an expression) that you want to keep an eye on.
Now, you have a watch set on the File object, and you can see its value change as you move through the debugger. Unlike the other variable windows, the Watch window always shows the variables that you are watching (they're grayed out when out of scope).


Watch 1

Name	Value	Type
f	{market 032.jpg}	System.I(

6- Debugging :

Change the execution flow



```
        private void Update()
        {
            this.Clear();
            try
            {
                foreach (FileInfo f in _directory.GetFiles("*.jpg"))
                    Add(new Photo(f.FullName)); ≤7ms elapsed

            }
```

6- Debugging :

- Read more about debugging from Microsoft https://docs.microsoft.com/en-us/visualstudio/debugger/debugger-tips-and-tricks
- Watch this online lesson from Microsoft virtual academy https://mva.microsoft.com/en-US/training-courses-embed/getting-started-with-visual-studio-2017-17798/Debugger-Feature-tour-of-Visual-studio-2017-sqwiwLD6D_1111787171

Chapter Four

C# Advanced.

Chapter Four Content :

1- Generics .

2- Interfaces .

3- Collections .

4- Delegets .

5- Events.

6- Threading

1- Generics

Generics allow you to delay the specification of the data type of programming elements in a class or a method, until it is actually used in the program. In other words, generics allow you to write a class or method that can work with any data type.

- It helps you to maximize code reuse, type safety, and performance.
- You can create generic collection classes. The .NET Framework class library contains several new generic collection classes in the System.Collections.Generic namespace. You may use these generic collection classes instead of the collection classes in the System.Collections namespace.
- You can create your own generic interfaces, classes, methods, events, and delegates.
- You may create generic classes constrained to enable access to methods on particular data types.
- You may get information on the types used in a generic data type at run-time by means of reflection.

1- Generics

A- Generics Method :

```
       static void Swap<T>(ref T lhs, ref T rhs)
        {
            T temp;
            temp = lhs;
            lhs = rhs;
            rhs = temp;
        }
```

```
Char a=‘n’,b=‘c’ ;
Int c=10 , d= 11 ;
Swap<int>(ref a, ref b);
 Swap<char>(ref c, ref d);
```

1- Generics

B- Generics Classes .

+ use simple List .

+ use List of Objects .

+ use Generics .

For demos look at GitHub Repo @ : Genrics.cs File

2- interface

we had a look at abstract classes. Interfaces are much like abstract classes and they share the fact that no instances of them can be created. However, interfaces are even more conceptual than abstract classes, since no method bodies are allowed at all. So an interface is kind of like an abstract class with nothing but abstract methods, and since there are no methods with actual code, there is no need for any fields.

Properties are allowed though, as well as indexers and events. You can consider an interface as a contract - a class that implements it is required to implement all of the methods and properties. However, the most important difference is that while C# doesn't allow multiple inheritance, where classes inherit more than a single base class, it does in fact allow for implementation of multiple interfaces!

```
 namespace Interfaces
{
 class Program
 {
    static void Main(string[] args)
    {
        List<Dog> dogs = new List<Dog>();
        dogs.Add(new Dog("Fido"));
        dogs.Add(new Dog("Bob"));
        dogs.Add(new Dog("Adam"));
        dogs.Sort();
        foreach (Dog dog in dogs)

        Console.WriteLine(dog.Describe());
        Console.ReadKey();
    }
  }
   interface IAnimal
  {
    string Describe();

    string Name {get; set;}
 }
```

```
class Dog : IAnimal, IComparable
{
   private string name;
  public Dog(string name)
 {this.Name = name;}

 public string Describe()
 {
   return "Hello, I'm a dog and my name is " + this.Name;
 }
 public int CompareTo(object obj)
 {
  if (obj is IAnimal)
   return this.Name.CompareTo((obj as IAnimal).Name);
  return 0;
 }
 public string Name
 {
   get { return name; }
   set { name = value; }
  }
}
```

3- Collections

There are two types of collections available in C#: non-generic collections and generic collections .Lets Start With Non-Generics Collections .

Non-generic		**Generic**
ArrayList	------------->	List
HashTable	------------->	Dictionary
SortedList	------------->	SortedList
Stack	------------->	Stack
Queue	------------->	Queue

3- Collections

Non-generic collections

Each element can represent a value of a different type.
Array Size is not fixed.
Elements can be added / removed at runtime.

Non-generic	Usage
ArrayList	ArrayList stores objects of any type like an array. However, there is no need to specify the size of the ArrayList like with an array as it grows automatically.
SortedList	SortedList stores key and value pairs. It automatically arranges elements in ascending order of key by default. C# includes both, generic and non-generic SortedList collection.
Stack	Stack stores the values in LIFO style (Last In First Out). It provides a Push() method to add a value and Pop() & Peek() methods to retrieve values. C# includes both, generic and non-generic Stack.
Queue	Queue stores the values in FIFO style (First In First Out). It keeps the order in which the values were added. It provides an Enqueue() method to add values and a Dequeue() method to retrieve values from the collection. C# includes generic and non-generic Queue.

Read More about Hashtable , and BitArray .

3- Collections

generic collections

Specific type
Array Size is not fixed
Elements can be added / removed at runtime.

List:

```
    using System.Collections.Generic;
    protected void Button1_Click(object sender, EventArgs e)
    {
            List<int> lst = new List<int>();
            lst.Add(100);
            lst.Add(200);
            lst.Add(300);
            lst.Add(400);
            foreach (int i in lst)
            {
                Console.Write(i+"<br>");
            }
     }
```

3- Collections

generic collections

Dictonary:

```
using System.Collections.Generic;

 protected void Button1_Click(object sender, EventArgs e)
 {
     Dictionary<int, string> dct = new Dictionary<int, string>();
     dct.Add(1, "cs.net");
     dct.Add(2, "vb.net");
     dct.Add(3, "vb.net");
     dct.Add(4, "vb.net");
     foreach (KeyValuePair<int, string> kvp in dct)
     {
         Console.Write(kvp.Key + " " + kvp.Value);
         Console.Write("<br>");
     }
  }
```

3- Collections

generic collections

SortedList:

```
 using System.Collections.Generic;

protected void Button3_Click(object sender, EventArgs e)
{
    SortedList<string, string> sl = new SortedList<string, string>();
    sl.Add("ora", "oracle");
    sl.Add("vb", "vb.net");
    sl.Add("cs", "cs.net");
    sl.Add("asp", "asp.net");

foreach (KeyValuePair<string, string> kvp in sl)
    {
        Response.Write(kvp.Key + " " + kvp.Value);
        Response.Write("<br>");
    }
}
```

3- Collections

generic collections

Stack:

```
using System.Collections.Generic;

protected void Button4_Click(object sender, EventArgs e)
{
    Stack<string> stk = new Stack<string>();
    stk.Push("cs.net");
    stk.Push("vb.net");
    stk.Push("asp.net");
    stk.Push("sqlserver");

    foreach (string s in stk)
    {
       Console.Write(s + "<br>");
    }
}
```

3- Collections

generic collections

Queue:

```
using System.Collections.Generic;

protected void Button1_Click(object sender, EventArgs e)
   {
        Queue<string> q = new Queue<string>();

          q.Enqueue("cs.net");
          q.Enqueue("vb.net");
          q.Enqueue("asp.net");
          q.Enqueue("sqlserver");

      foreach (string s in q)
      {
         Response.Write(s + "<br>");
      }
   }
```

4- Delegets

Look to our Code Reference and Recorded Lecture ..

5- Events

Look to our Code Reference and Recorded Lecture ..

6- Threading

Look to our Code Reference and Recorded Lecture ..

Chapter Five

Object Oriented Programming Concepts.

Chapter Four Content :

1- C# Namespaces .

2- class Vs Object .

3- Inheritance :

4- Encapsulation :

5- Abstraction :

6-Polymorphism :

7- interface

8- Association vs Aggregation vs Composition

1- C# Namespaces .

A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace :

```
namespace namespace_name
{
 // code declarations
}
```

You need to define variable with its full name like this in case of ambiguity between two name spaces :

```
namespace_name.item_name;
```

```
namespace first_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside first_space");
        }
    }
}
namespace second_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside second_space");
        }
    }
}
class TestClass
{
    static void Main(string[] args)
    {
        first_space.namespace_cl fc = new first_space.namespace_cl();
        second_space.namespace_cl sc = new second_space.namespace_cl();
        fc.func();
        sc.func();
        Console.ReadKey();
    }
}
```

2- class Vs Object .

- **Class** and **object** holds Avery deep concept that's why we move from traditional structure programming to object oriented programming .
- lets try to answer some important questions .

+ what is OOP ??
+ why we move from traditional structure programming to OOP ?
+ class vs object ?
+ door , table , room , pen is this object or classes ?
+ what is main OOP Concepts ?

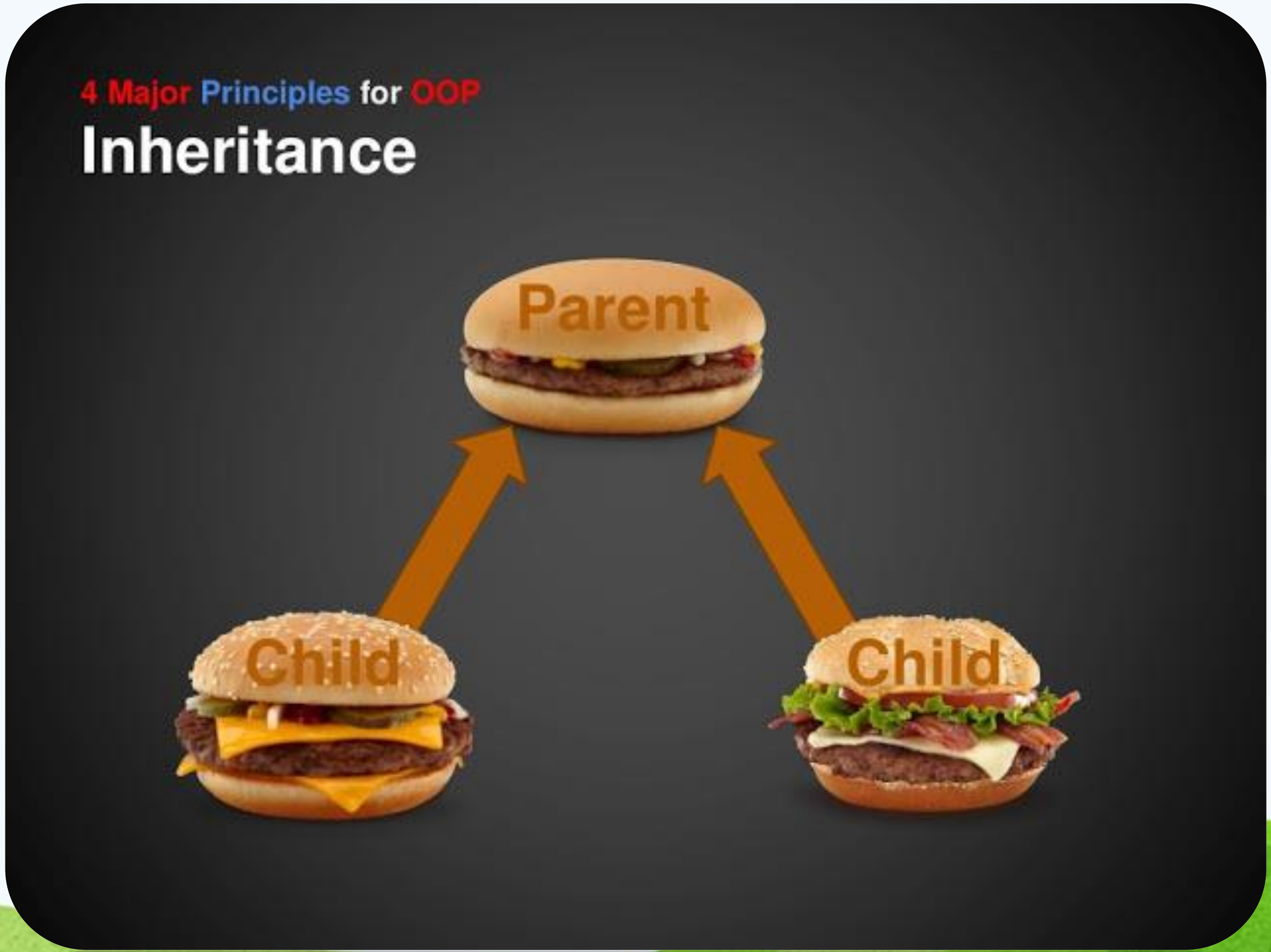
OOP Four Gods :

1. Inheritance
2. Abstraction
3. Encapsulation
4. Polymorphism.

3- Inheritance :

Inheritance :

The ability of a new class to be created, from an existing class by extending it, is called inheritance.



3- Inheritance :

Notes :

- in C# class can inherit from one class only .
  (some other languages permit with that) .
  - multi Level inheritance are allowed .
  - A inherit from B , B inherit from C , ……etc.
  - all C# classes inherit from class Object .

For all code demo see our get-Hup Repos link .

4- Encapsulation :

Encapsulation is obvious in

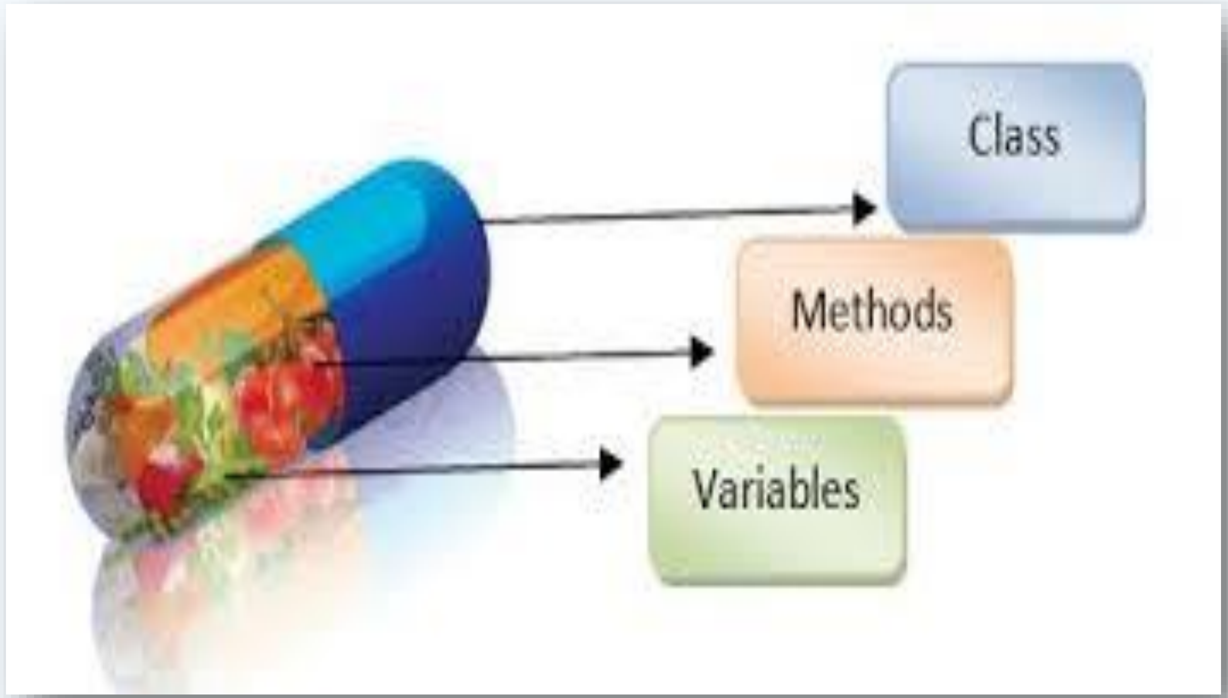
- access modifiers :

public

private

protected

internal



4- Encapsulation :

Encapsulation OR (Data Hiding)

The encapsulation is the inclusion-within a program object-of all the resources needed for the object to function, basically, the methods and the data.
In OOP the encapsulation is mainly achieved by creating classes, the classes expose public methods and properties. A class is kind of a container or capsule or a cell, which encapsulate a set of methods, attribute and properties to provide

its indented functionalities to other classes.
In that sense, encapsulation also allows a class to change its internal implementation without hurting the overall functioning of the system. That idea of encapsulation is to hide how a class does its business, while allowing other classes to make requests of it.

5- Abstraction :

Abstraction :

The word abstract means a concept or an idea not associated with any specific instance.

In programming we apply the same meaning of abstraction by making classes not associated with any specific instance.

The abstraction is done when we need to only inherit from a certain class, but not need to instantiate objects of that class.

In such case the base class can be regarded as "Incomplete". Such classes are known as an "Abstract Base Class".

5- Abstraction

What is Abstract class ??

Abstract classes is classes cannot be instantiated. It can only be used as a super-class for other classes that extend the abstract class. Abstract class is the concept and implementation gets completed when it is being realized by a subclass. In addition to this a class can inherit only from one abstract class and must override all its methods/properties that are declared to be abstract and may override virtual methods/ properties.

6-Polymorphism :

- Polymorphism : (تعدد الوجوه)
- Static Polymorphism
  - Function overloading
  - Operator overloading
- Dynamic Polymorphism

8- Association vs Aggregation vs Composition

- **Association :**

Association is "*a*" relationship among objects determine what an object instance can cause another to perform an action on its behalf. We can also say that an association defines the multiplicity among the objects. We can define a one-to-one, one-to-many, many-to-one and many-to-many relationship among objects. Association is a more general term to define a relationship among objects. Association means that an object "uses" another object.

8- Association vs Aggregation vs Composition

- **Aggregation**

  is a special type of Association. Aggregation is "*the*" relationship among objects. We can say it is a direct association among the objects. In Aggregation, the direction specifies which object contains the other object. There are mutual dependencies among objects.

  For example, departments and employees, a department has many employees but a single employee is not associated with multiple departments.

8- Association vs Aggregation vs Composition

- **Composition :**

is special type of Aggregation. It is a strong type of Aggregation. In this type of Aggregation the child object does not have their own life cycle. The child object's life depends on the parent's life cycle. Only the parent object has an independent life cycle. If we delete the parent object then the child object(s) will also be deleted. We can define the Composition as a "Part of" relationship.

For example, the company and company location, a single company has multiple locations. If we delete the company then all the company locations are automatically deleted. The company location does not have their independent life cycle, it depends on the company object's life (parent object).

Home Work

We need a 3 pages sheet describing the difference between interface and abstract class .

With very clear demo !!

References :

1- C++ How To Program Deitel and Deitel seventh Edition .
2- Clr via C# Jeffery Richter Second Edition .
3- OOP Fundamentals Code Project
link : https://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep .

4- useful website to study fundamentals :
https://channel9.msdn.com/Series/C-Fundamentals-for-Absolute-Beginners/01
5- http://www.c-sharpcorner.com/UploadFile/736bf5/collection-in-C-Sharp/