MUHAMMAD ALI NASIK AWAN

OBJECTIVES:

1. Understanding code snippets, code snippet manager and creating code snippets.
2. Understanding documentation and creating documentation.
3. Understanding debugging and performing debugging in visual studio 2015.
4. Understanding and developing windows forms applications.
5. Practice Activities.

**OBJECTIVE 1:** Understanding code snippets, code snippet manager and creating code snippets.

# Code Snippets

* Snippets are pre-built code chunks/ lines of code used for rapid application development.
* One can use already created snippets or can also create their own snippets.
* There are two ways to add snippets.
  + Insert snippet: inline insertion of snippet.
  + Surround snippet: snippet that surrounds lines of code by selecting them and adding a snippet.

- For inserting code snippet, choose **Edit**, **IntelliSense**, **and Insert Snippet**

# -OR-

* Open the shortcut menu in a file and choose **Insert Snippet**.

**-OR-**

* One can insert code snippet by writing snippet keyword/shortcut and pressing two times tab key.

* For example: if + tab + tab will insert if code snippet/chunk. **-OR-**
* Pressing CTRL + K + X will open a pop up menu for insertion of snippet.

**-OR-**

* Pressing CTRL + K + S will open a pop up menu for insertion of surround snippet.

**OBJECTIVE 2:** Understanding documentation and creating documentation.

# Documentation

* It is description of types and type members of system software created to understand the code written for the project.
* For developing readable documentation three steps should be performed.
* At first step, one should write comments in xml elements for the desired lines of code using special comment field starts with ///.
* For example: writing ///<summary> comments goes here </summary>
* At second step one should create an xml file from the xml comments created for code.
* XML file can be created mainly in two ways:
  + Using compiler: using \doc option for c-sharp compiler.
  + Using visual studio: by checking documentation checkbox in properties of project.

* At third step one should use a tool like **sandcastle and sandcastle help file builder** to generate a readable file from xml file created in second step.

* XML file can also be created using third party extensions.
* For example: using GhostDoc extension of visual studio that can insert xml comment elements automatically.

* Pressing **CTRL + SHIFT + D** key combination will automatically insert xml comments in the code in case of extension (**GhostDoc**) that is installed with visual studio.

* With the help of same extension xml file and documentation can also be created.

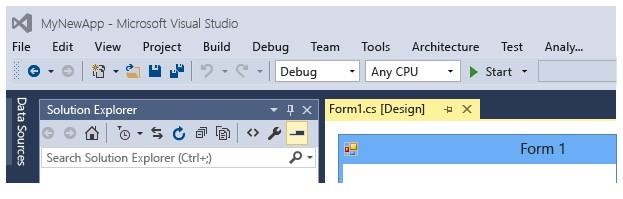
# RECOMMENDED XML TAGS FOR DOCUMENTATION

* <summary> Summary </summary>o The <summary> tag should be used to describe a type or a type member.
* <returns> Description </returns>o A description of the return value.
* <param name=”name”> Description </param>o Name: name of a method parameter. Enclose the name in double quotation marks (" ") o Description: A description for the parameter.
* <value> Property Description</value>o The <value> tag lets you describe the value that a property represents

**OBJECTIVE 3:** Understanding debugging and performing debugging in visual studio 2015.

**Debugging** It is finding errors and removing/resolving that errors from code.  There are two types of build configuration: Debug and Release.

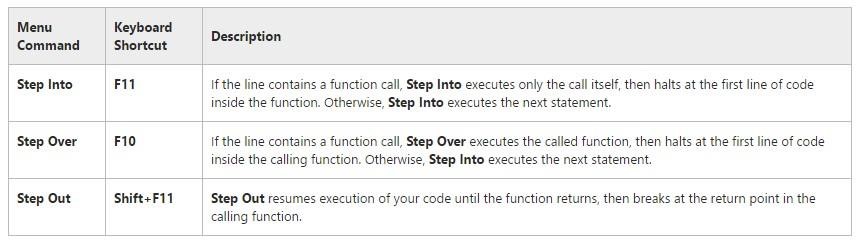
* Debug: produces a slower, larger executable that allows for a richer interactive run-time debugging experience but should not be shipped.
* In debug mode one can stop or pause the execution and execute the code in step wise manner by navigating consequences of every operation.
* The default build configuration is **Debug**.
* Release: builds a faster, more optimized executable that’s appropriate to ship (at least from the perspective of the compiler).



* For debugging code one can insert break points in a code.
* A breakpoint indicates where Visual Studio should suspend your running code so you can take a look at the values of variables, or the behavior of memory, or whether or not a branch of code is getting run. You do

NOT need to rebuild a project after setting and removing breakpoints.

* Along with breakpoints, **trace points** are a new debugger feature in Visual Studio. A trace point is a breakpoint with a custom action associated with it. When a trace point is hit, the debugger performs the specified trace point action instead of, or in addition to, breaking program execution.
* For inserting break point select line of code and right click for inserting break point.
* After inserting desired number of break points one can execute code in debugging mode.
* Following shortcuts can be used for stepping into code and so on.



**OBJECTIVE 4:** Understanding and developing windows forms applications.

# Windows Forms Applications

Windows Forms is the platform for developing windows applications based on .NET framework.

In .NET development, a Windows GUI application is called a Windows Forms (or *Winforms*) application.

# Form

* A form is a bit of screen, usually rectangular, that you can use to present information to the user and to accept input from the user.
* Forms can be standard windows, multiple document interface windows, dialog boxes, or display surfaces for graphical routines.
* Forms are objects that expose properties which define their appearance, methods which define their behavior, and events which define their interaction with the user.
* By setting the properties of the form and writing code to respond to its events, you customize the object to meet the requirements of your application.

* The form you create with the Windows Forms Designer is a class, and when you display an instance of the form at run time, this class is the template used to create the form.

* Additionally, forms are controls, because they inherit from the [Control c](https://msdn.microsoft.com/en-us/library/system.windows.forms.control(v=vs.71).aspx)lass.

**ACTIVITIES SECTION OBJECTIVE 1: ACTIVITIES**

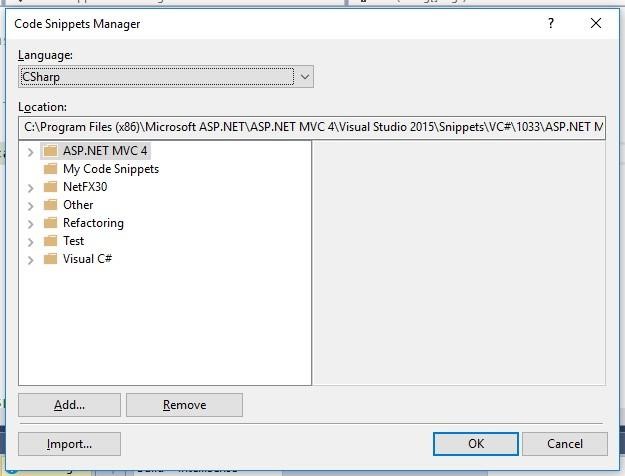
Understanding and creation of code snippets **ACTIVITY 1**:

# STEPS

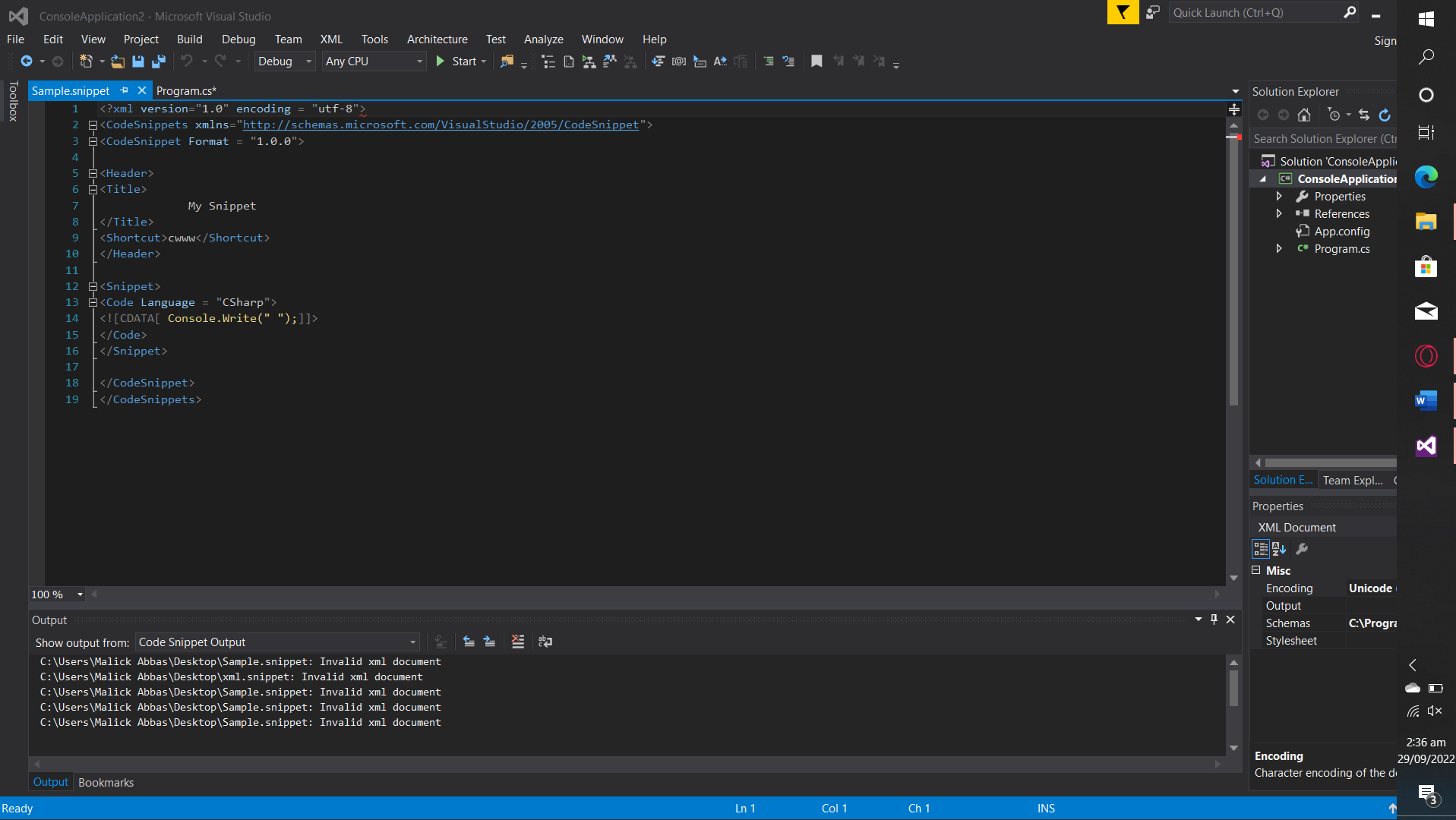
1. Create console application.
2. Create an xml file with extension .snippet.
3. Add following lines of code in the file.

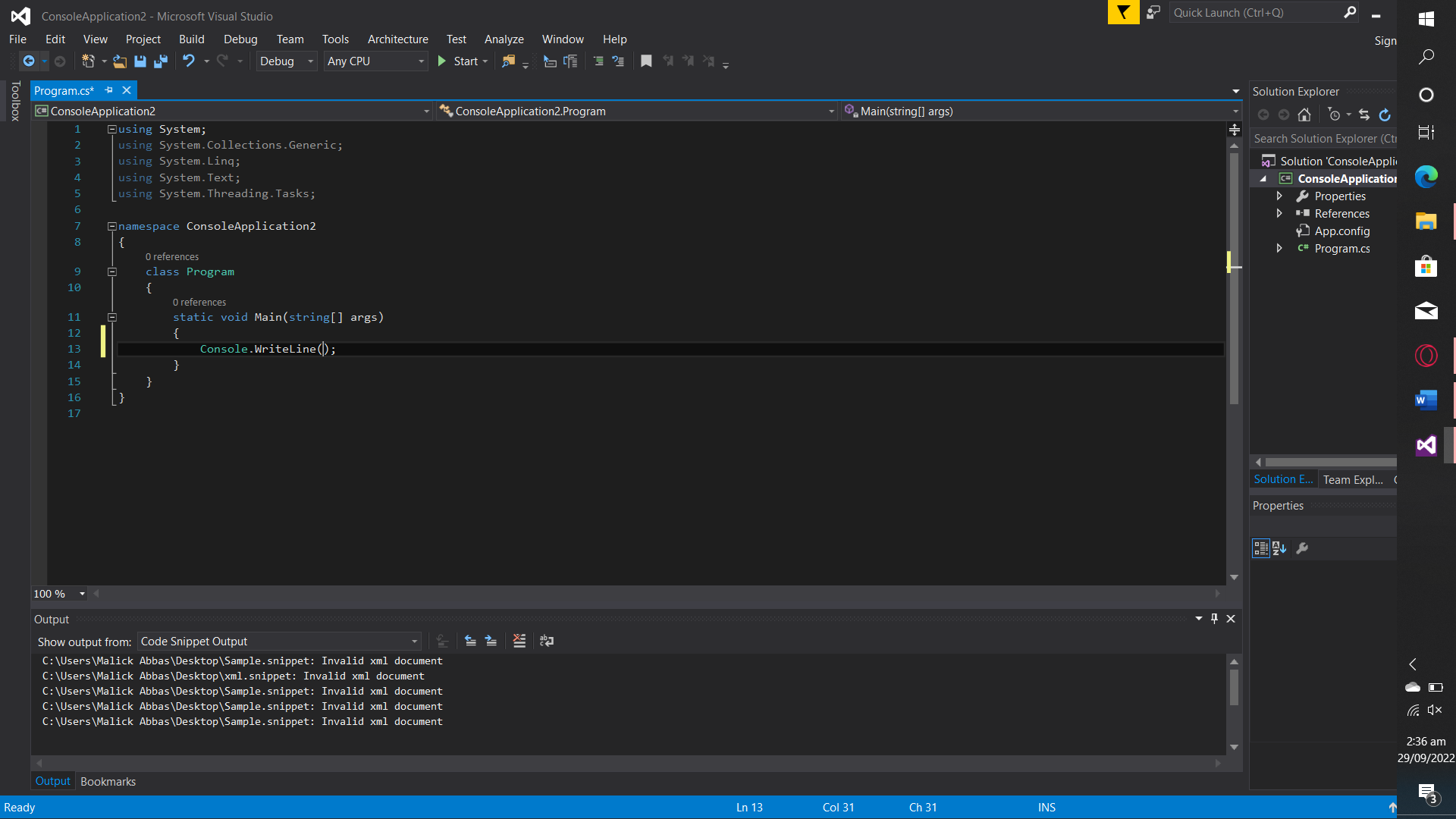


1. Open code snippets manager from Tools > Code Snippets Manager. -OR-
2. CTRL + K + B keys combination opens Code Snippets Manager directly.



1. Using Code snippets manager one can remove the existing and can add the new code snippets.
2. Select Visual C# from folders.
3. Press Import button.
4. Import the snippet that you have created, select location for it, and use the snippet in the code by typing cwww + two time tab key.





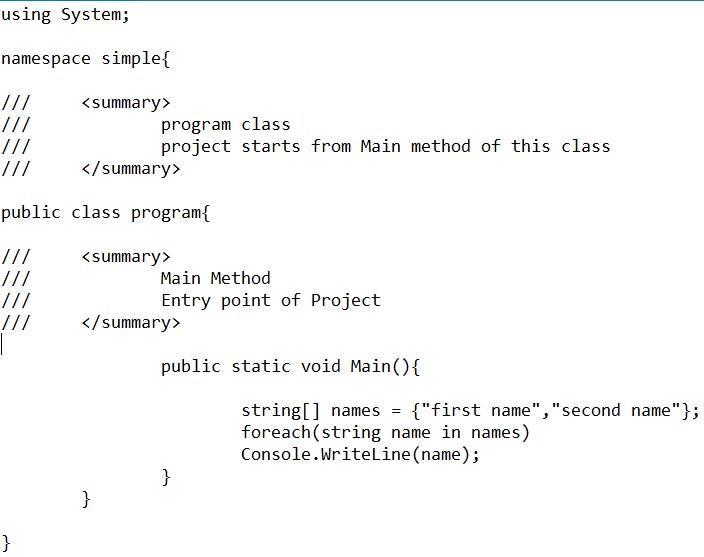
**OBJECTIVE 2: ACTIVITIES**

Understanding documentation and creating documentation.

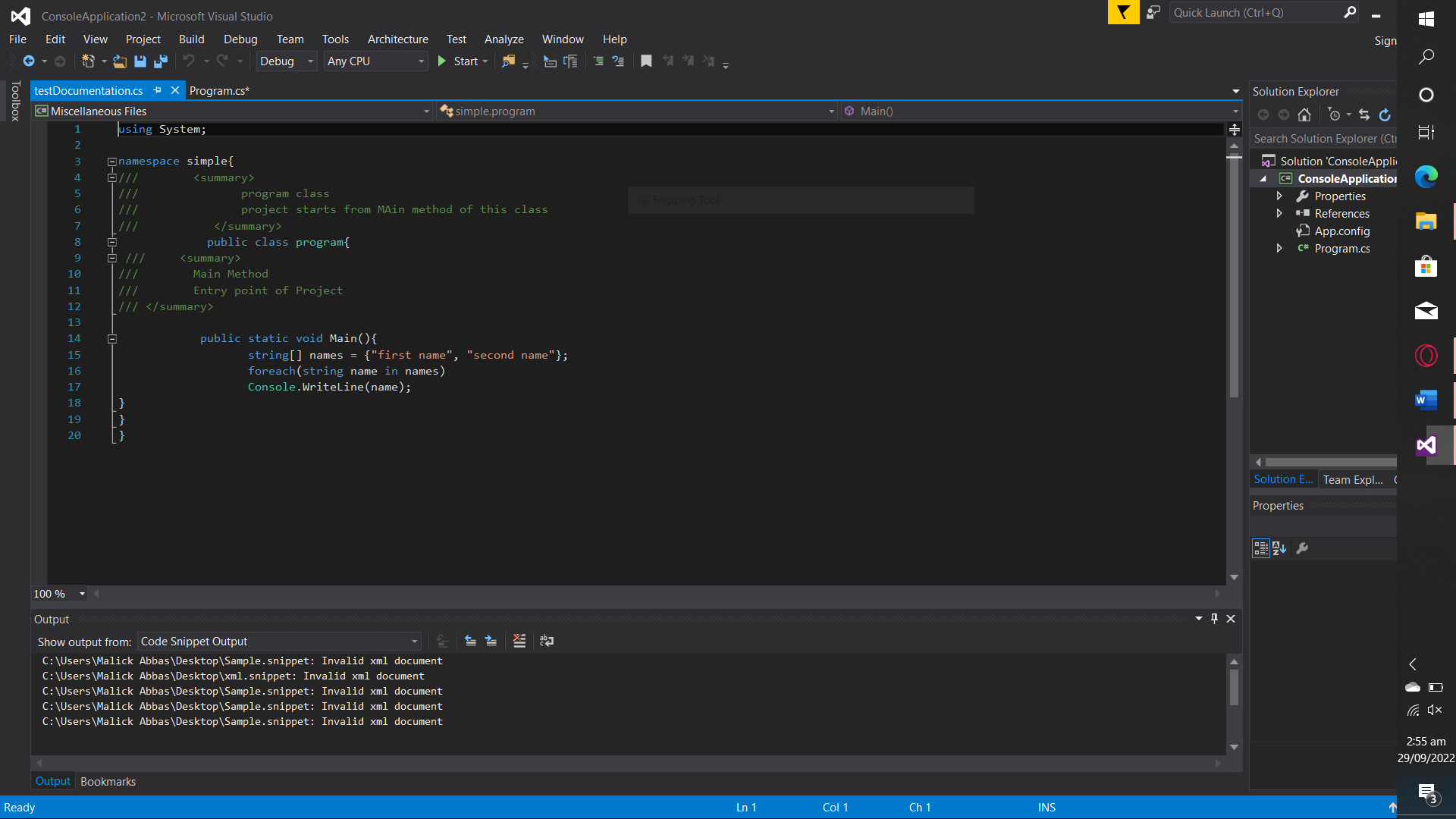
**ACTIVITY 2**

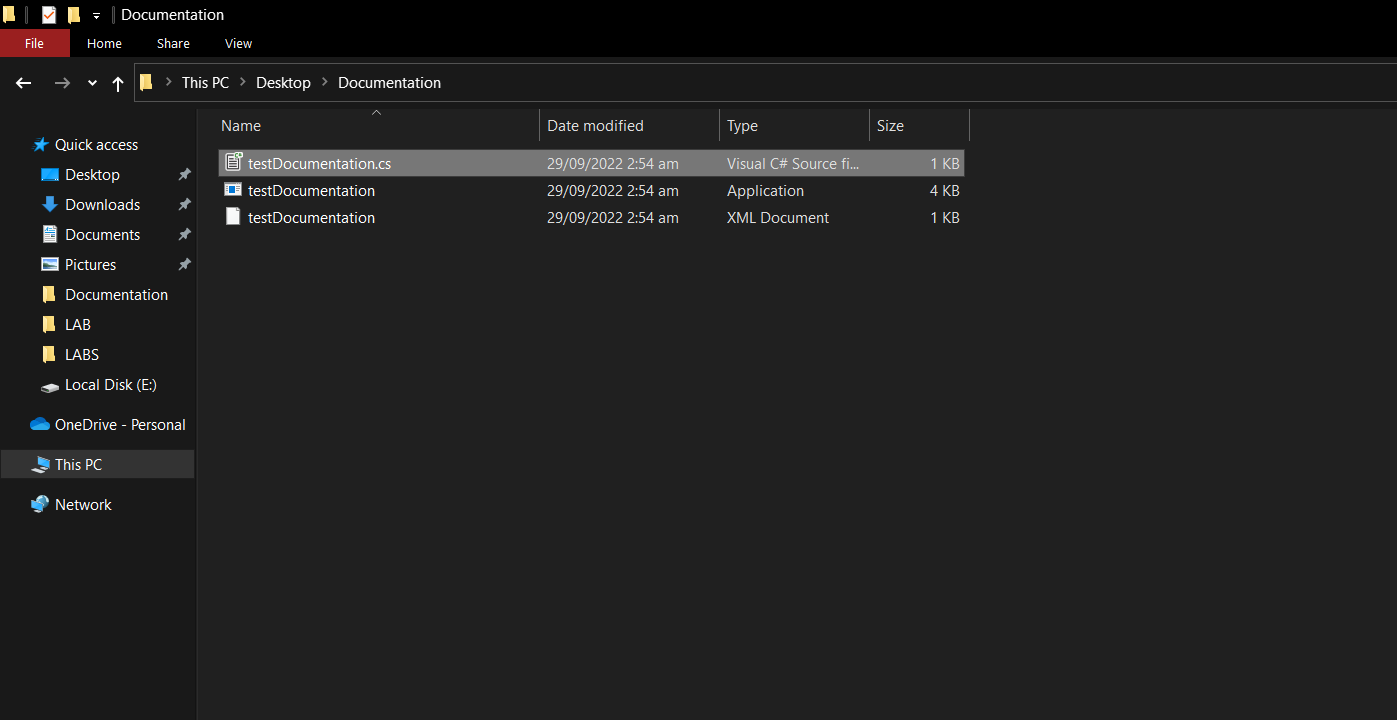
# STEPS

1. Create a Folder named Documentation
2. Create a file named testDocumentation.cs.
3. Open it with notepad.
4. Write following code in it.



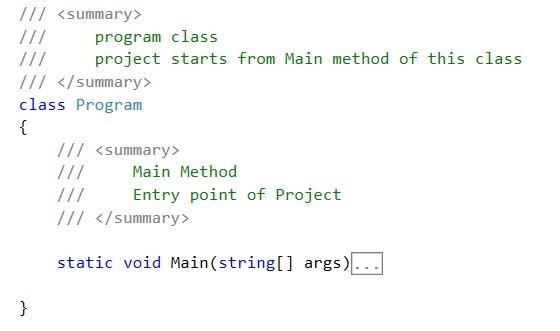
1. /// is special field comment to write documentation comments for code.
2. Compile code using c-sharp compiler into the same folder.
3. Documentation>csc /doc:name\_of\_the\_file.xml name\_of\_the\_file.cs
4. /doc is a flag that is providing extra information to c-sharp compiler for generating documentation of code.
5. Observe the results after compiling an xml file must be created into the folder Documentation with the name specified in command.
6. Now we can create .chm file or any other file from this xml file using tools like sand castle.



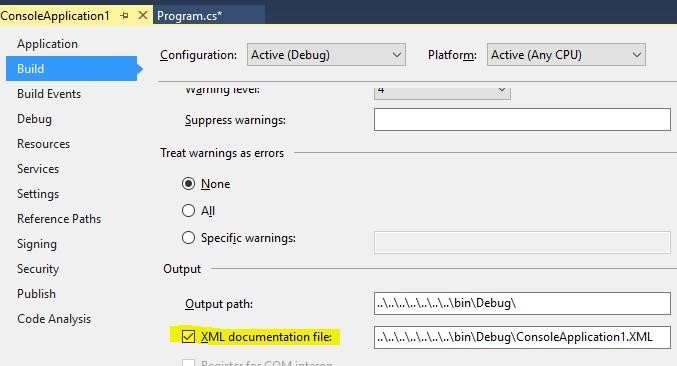
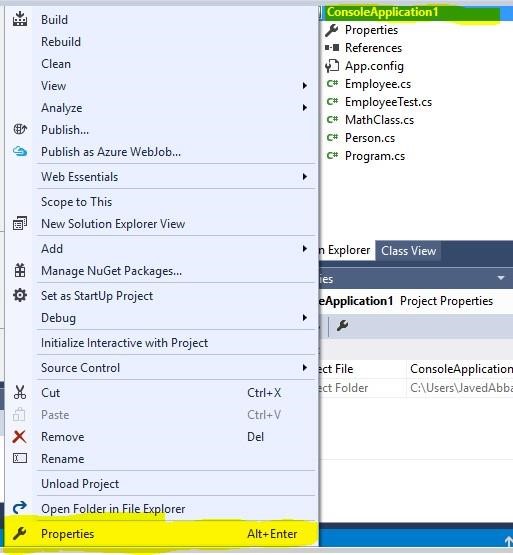


# ACTIVITY 3 STEPS

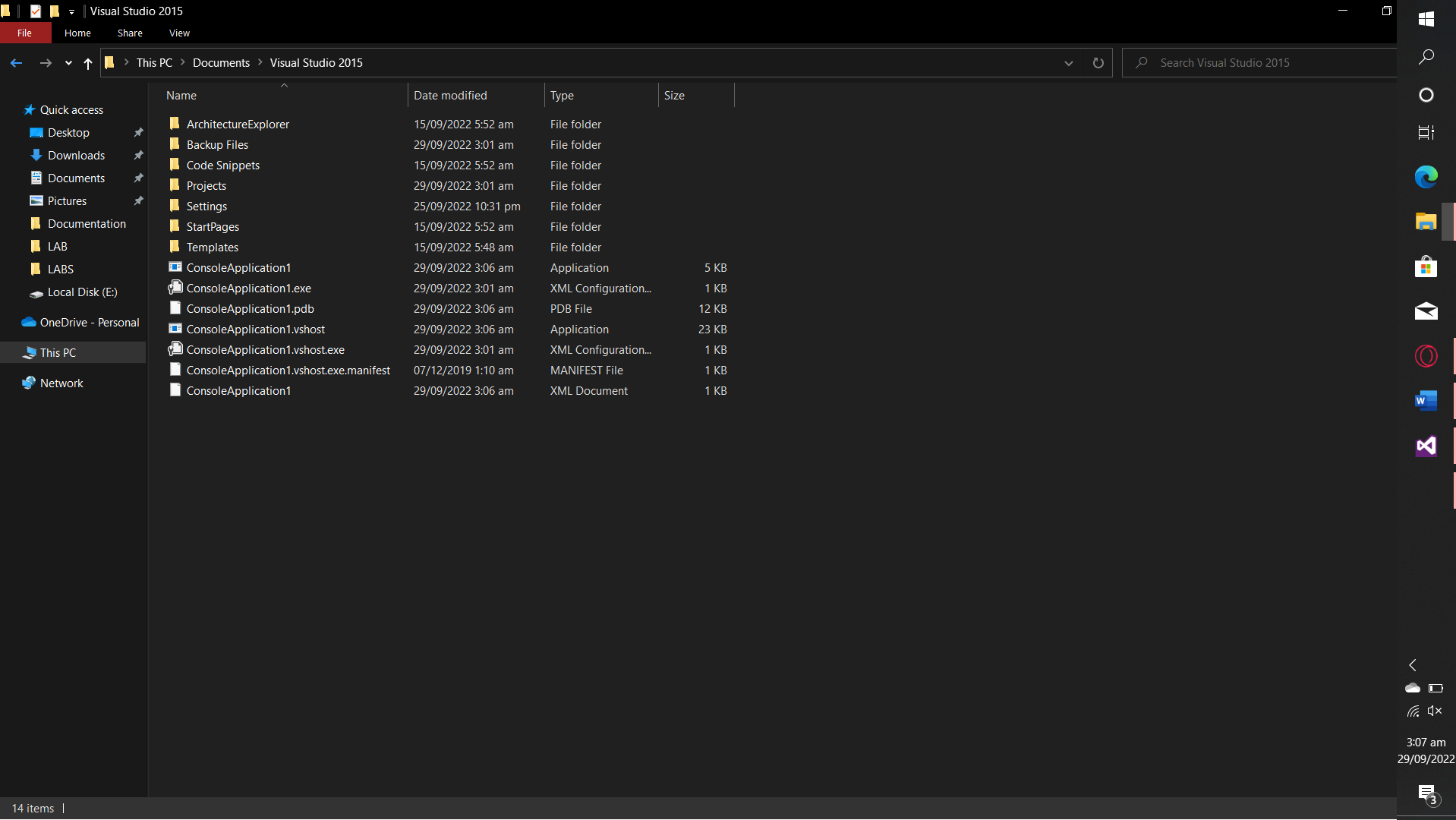
1. Create console application named ConsoleApplication1.
2. Write xml comments for documentation in default code generated by visual studio like in the file given below.



1. Go to the properties of a project by right clicking the project name that is ConsoleApplication1.
2. Select Build tab and check the XML documentation file checkbox for creating xml file.



1. After the build operation now xml file will automatically be created in to the destination folder.

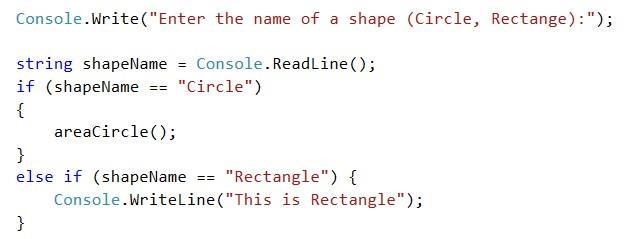


**OBJECTIVE 3: ACTIVITIES**

Understanding debugging and performing debugging in visual studio 2015.

# ACTIVITY 4: STEPS

* Create a console application.
* Write following code in Main method.



* Create static method named areaCircle() with any lines of code.
* Press F9 by standing at if statement this will add a break point at this line.
* Press F9 by standing at areaCircle() method call.
* Press F5 to execute application in Debug Mode.
* At runtime provide argument “Circle” in command prompt.
* Now observe following windows in visual studio.

* Autos window: shows values of variables and their type



* Locals window: shows values of local variables.



* Watch window: In this window, we can check values of local variables and also perform some other operations, like comparison of values and so on.



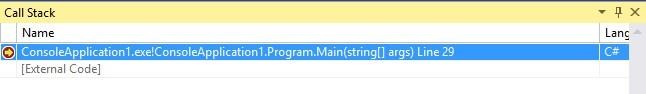
* + At first there will be nothing in this watch window.
  + Double click in the blank space below Name column to write something like name of variables and so on.o For example: As in given diagram two things are there:
  + First: name of shapeName variable and value associated with

it.

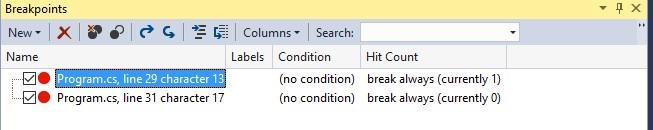
* + Second: Usage of equality comparison operator on value of shapeName and its result in Value column.



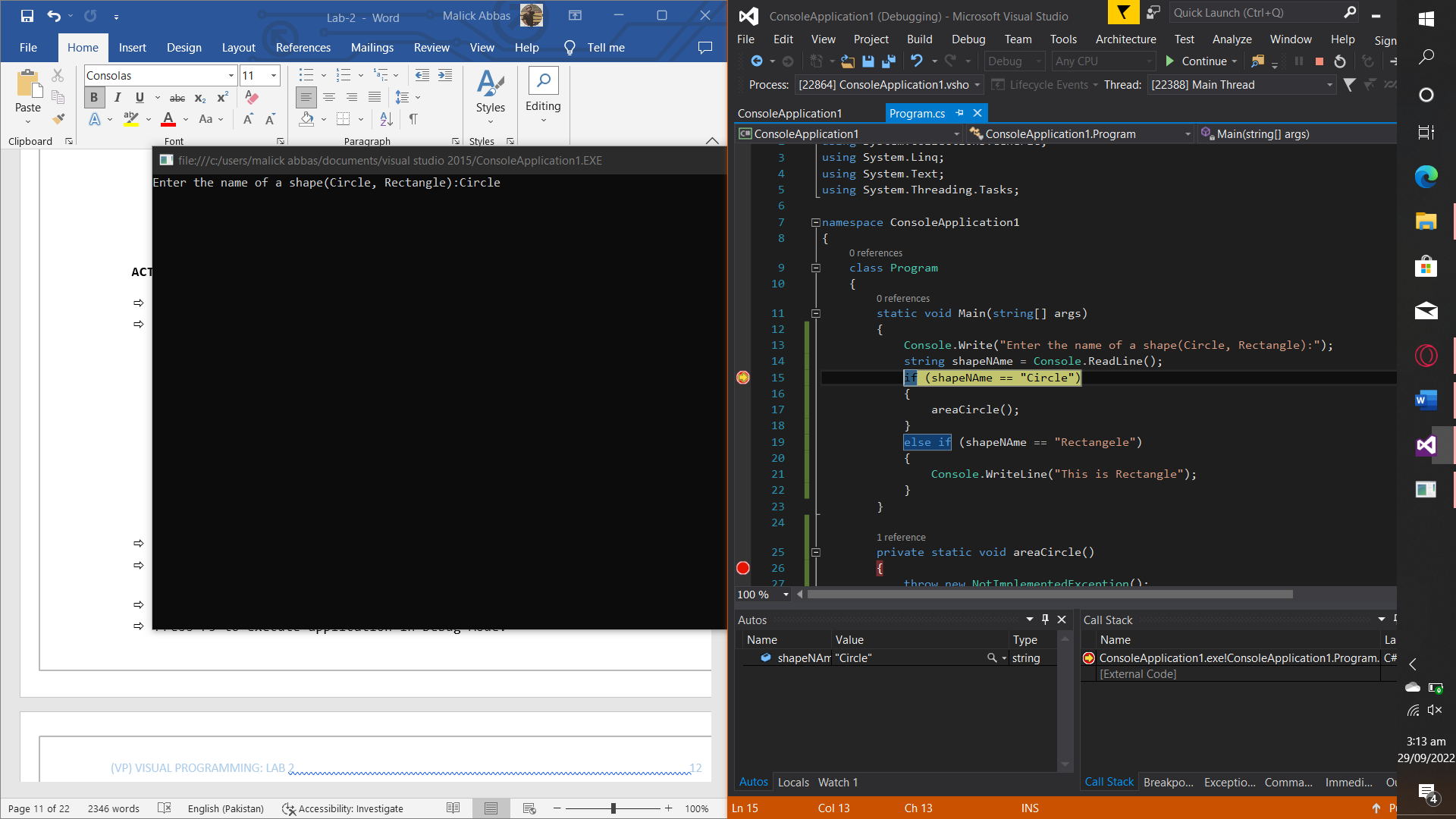
* Call Stack window: this shows stack of method calls, their hierarchy and line number.



* Break points window: shows list of break points in application along with labels, condition and hit count of break points.

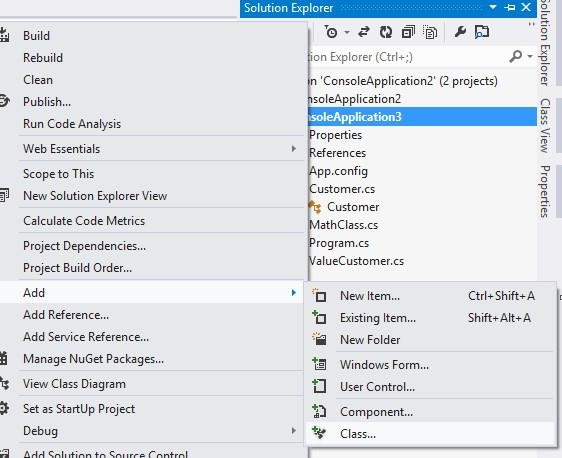


* **After observing all windows, Press Step into, Step over and Step out options and write notes on the functionality that you’ve observed.**



# ACTIVITY 5: THIS ACTIVITY IS RELATED TO FIRST LAB: STEPS

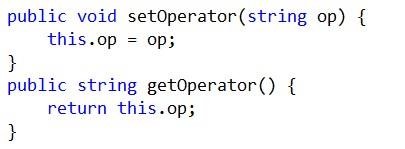
1. Create a console Application using visual studio 2012 or some other version if available.
2. Add new class named MathClass in application by right clicking on the name of console application in solution explorer.



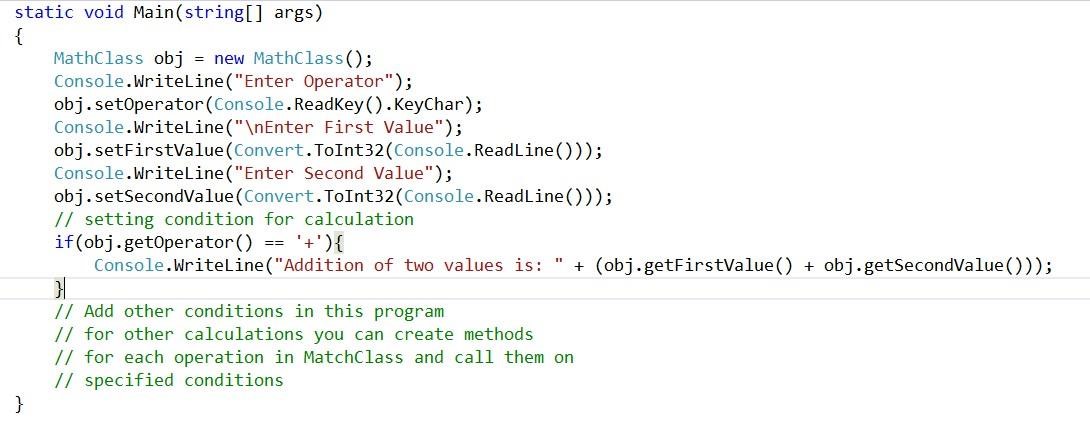
1. Create three variables in MathClass

* String op
* Int firstValue
* Int secondValue

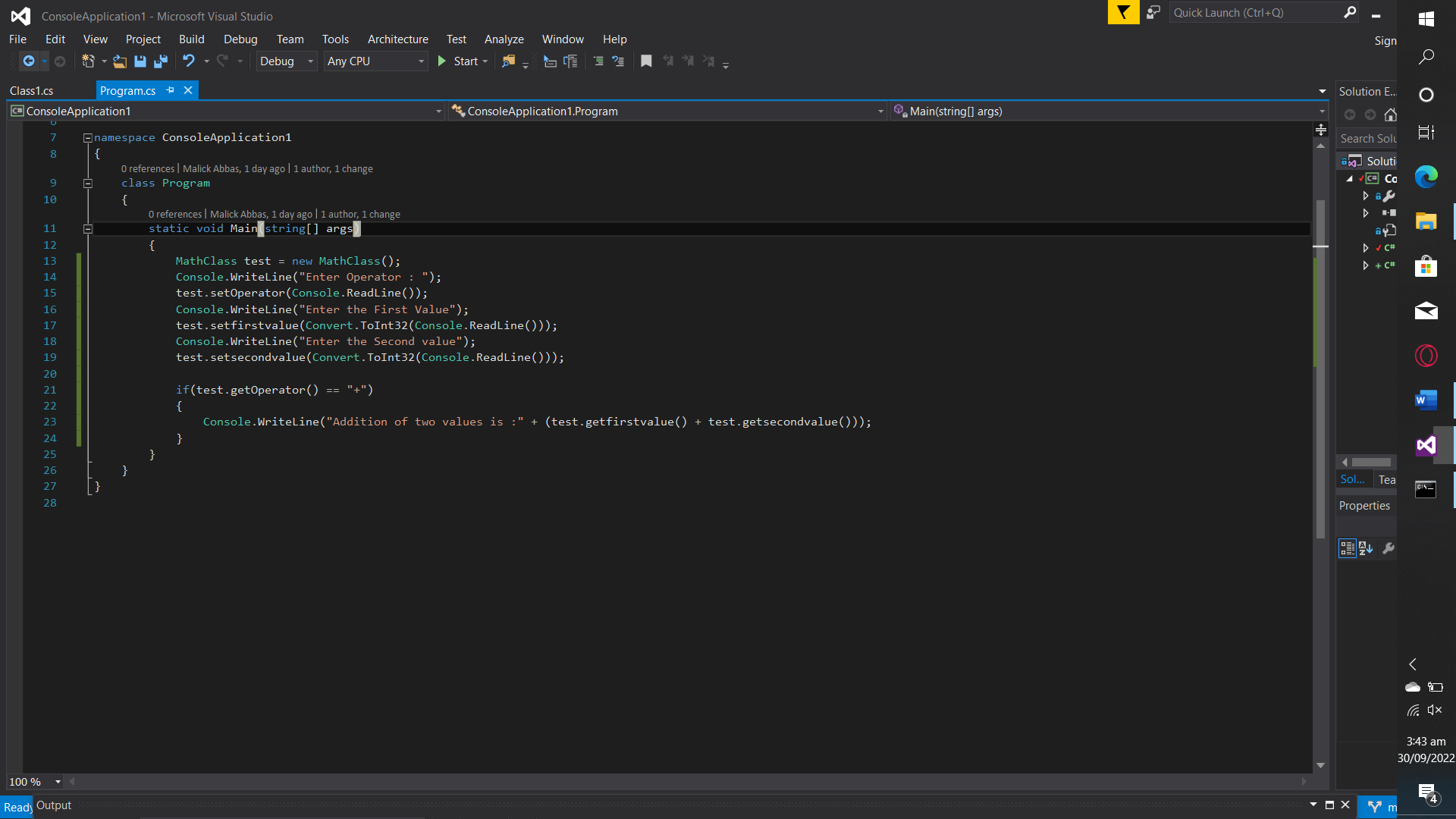
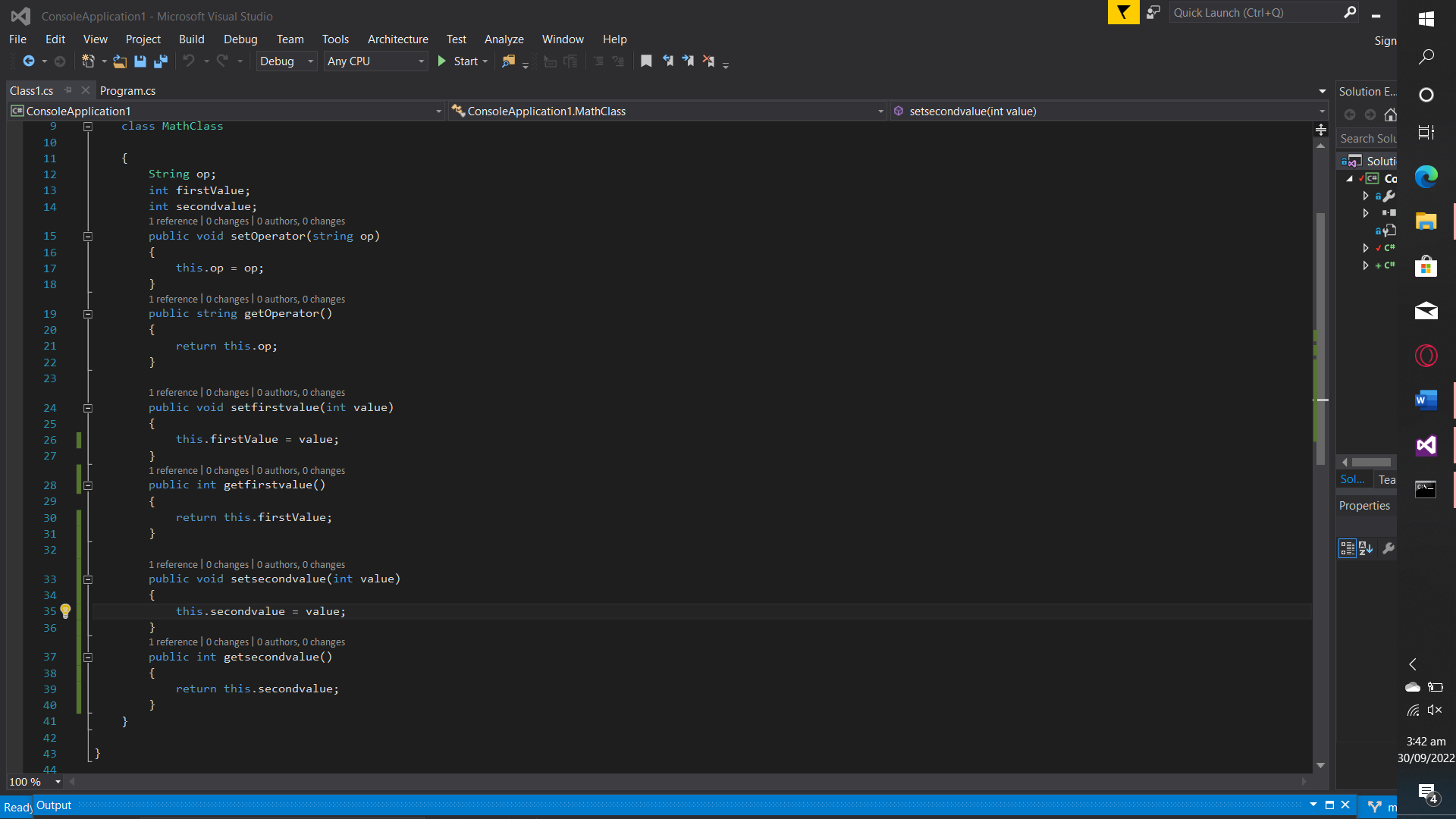
1. Create setter and getter methods for each of the variable like this one given in example.

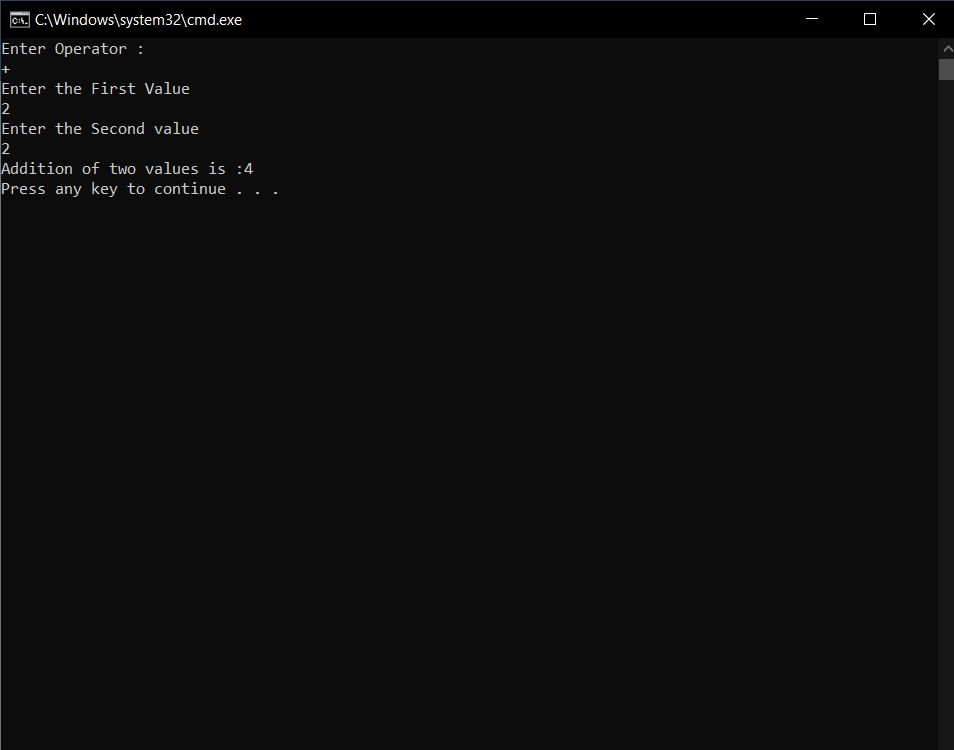


1. Here is a code of Main method in Program.cs class



1. Create Methods for other operations in the MathClass.
2. Execute the program.



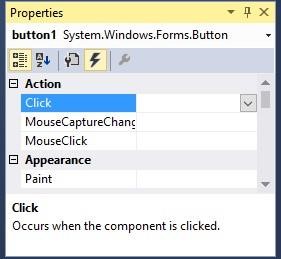
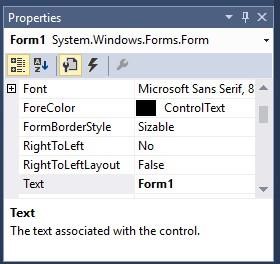


**OBJECTIVE 4: ACTIVITIES**

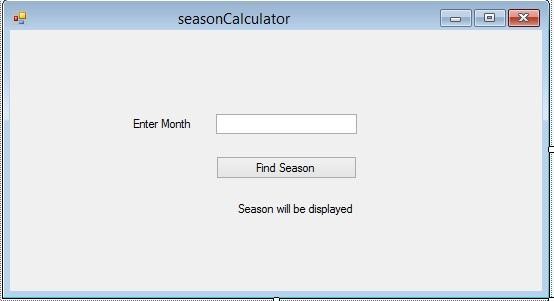
Understanding and developing windows forms applications.

# ACTIVITY 6 STEPS

1. Create your first Windows forms application named DemoWindowsForms.
2. Form1, a default form as a startup form will be created by Visual studio in your application.
3. Now add following controls to your form by dragging controls from toolbox (CTRL + ALT + X to view toolbox) to form and set their properties and events if any from properties windows.



|  |  |  |
| --- | --- | --- |
| Control | Set following properties | Events |
| Label1 | 1. Name = lblMonth 2. AutoSize = false 3. Text = “Enter Month Number ” |  |
| Label2 | 1. Name = lblSeason 2. AutoSize = false 3. Text = “Here Season for specific month will be displayed” |  |
| TextBox1 | 1. Name = txtMonth |  |
| Button1 | 1. Name = btnSeason 2. Text = “Find Season” | 1. Click Event : to add it double click the button |



1. Get value of txtMonth (TextBox ) in event handler method that will be invoked when you will click the Find Season button

5. You can define method name that should be invoked on btnName\_Click event of button using properties window in event section.

1. Write code to find out season using value of month variable.
2. Execute the application and validate the results.

CODE:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

int Month = Convert.ToInt32(txtMonth.Text);

if(Month == 12 || Month >= 1 && Month <=3)

{

label2.Text = "WINTER";

}

else if (Month >=4 && Month <= 6)

{

label2.Text = "SUMMER";

}

else if(Month >=7 && Month <= 9)

{

label2.Text = "SPRING";

}

else if(Month==10 || Month == 11)

{

label2.Text = "AUTUMN";

}

}

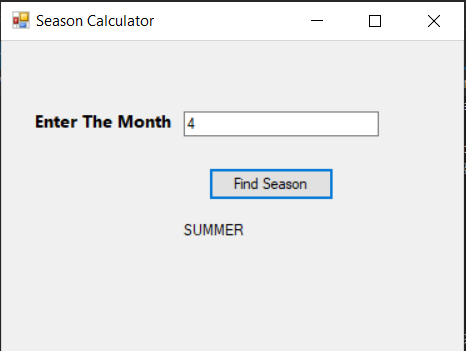
private void txtMonth\_TextChanged(object sender, EventArgs e)

{

}

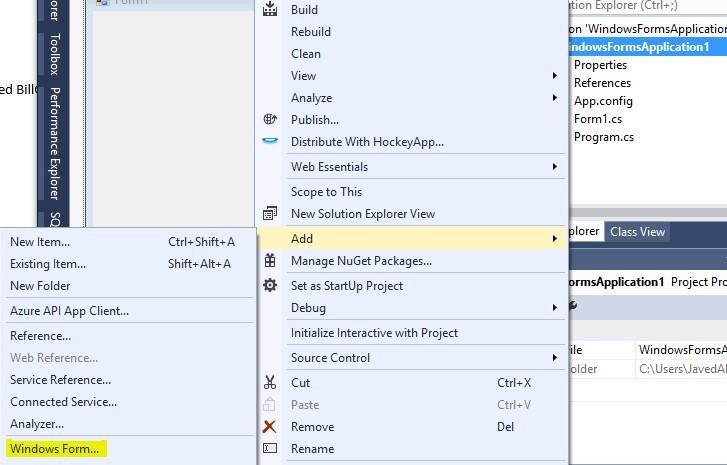
}

}

OUTPUT:  


# ACTIVITY 7: STEPS

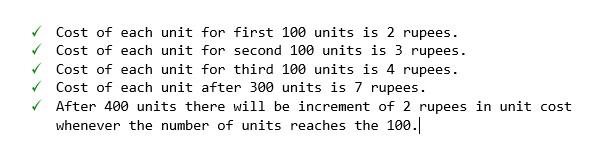
1. Add new form named BillCalculation in windows forms application created in previous activity by following steps given in snap shot.



1. Now add following controls on form.

|  |  |  |
| --- | --- | --- |
| Control | Set following properties | Events |
| Label1 | 1. Name = lblUnits 2. AutoSize = false 3. Text = “Enter Units ” |  |
| Label2 | 1. Name = lblBill 2. AutoSize = false 3. Text = “Here Bill will be displayed” |  |
| TextBox1 | 1. Name = txtUnits |  |
| Button1 | 1. Name = btnCalculateBill 2. Text = “Calculate Bill” | 2. Click Event : to add it double click the button |

1. Get value of units from textbox txtUnits.
2. Write code for calculation of bill from units following these conditions.



1. Execute the Program and validate the results.

Code:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

private void button1\_Click(object sender, EventArgs e)

{

int units = Convert.ToInt32(textBox1.Text);

int bill=0;

if(units >=1 && units <= 100)

{

bill = units \* 2;

label3.Text = "Rs" + bill.ToString();

}

else if (units > 100 && units <= 200)

{

bill = 200 + ((units - 100) \* 3);

label3.Text = "Rs" + bill.ToString();

}

else if (units > 200 && units <= 300)

{

bill = 200 + 300 + ((units - 200) \* 4);

label3.Text = "Rs" + bill.ToString();

}

else if (units > 300 && units <= 400)

{

bill = 200 + 300 + 400 +((units - 100) \* 7);

label3.Text = "Rs" + bill.ToString();

}

else if (units > 400)

{

int i = 9;

units = units - 400;

while (units > 100)

{

bill = 100 \* i;

units = units - 100;

i = i + 2;

}

bill = bill + (units \* i);

bill = bill + 200 + 300 + 400 + 700;

label3.Text = "Rs" + bill.ToString();

}

}

private void label3\_Click(object sender, EventArgs e)

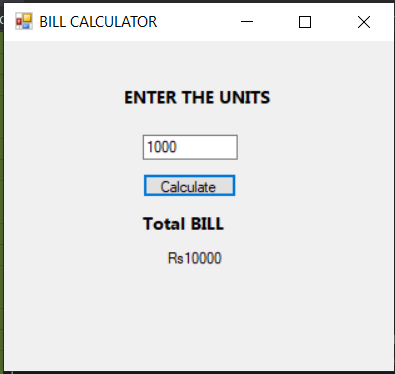
{

}

}

}

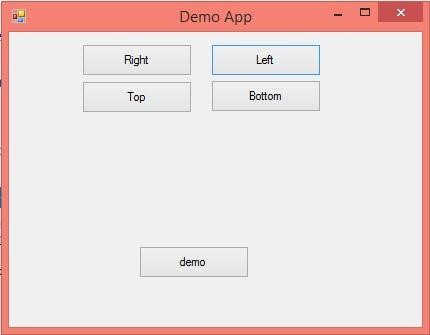
Output:



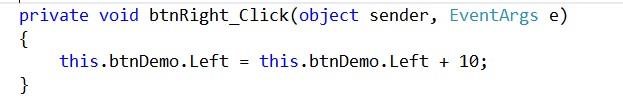
# ACTIVITY 8: STEPS

1. Create windows form application named DemoApplication using visual studio or some other version if available.
2. Change name of Form1 to frmDemo and text to Demo App.
3. Add following controls on the form:

|  |  |  |
| --- | --- | --- |
| Control | Set following properties | Events |
| Button1 | 1. Name = btnDemo 2. Text = “Demo” | No event |
| Button2 | 1. Name = btnRight 2. Text = “Right” | btnRight\_click() |
| Button3 | 1. Name = btnLeft 2. Text = “Left” | btnLeft\_click() |
| Button4 | 1. Name = “btnTop” 2. Text = “Top” | btnTop\_click() |
| Button5 | 1. Name = btnBottom 2. Text=”Bottom” | btnBottom\_click() |



1. Write following code in click event of **btnRight** button.



1. Execute the code and analyze the output.
2. Write code for other buttons to achieve functionality indicated by their labels.

**Note: Use Top property of button control to move it downwards.**

**CODE:**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApplication1

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

this.button5.Top = this.button5.Top - 10;

}

private void button2\_Click(object sender, EventArgs e)

{

this.button5.Left = this.button5.Left - 10;

}

private void button3\_Click(object sender, EventArgs e)

{

this.button5.Left = this.button5.Left + 10;

}

private void button4\_Click(object sender, EventArgs e)

{

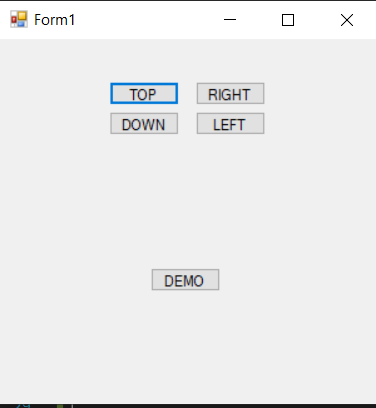
this.button5.Top = this.button5.Top + 10;

}

}

}

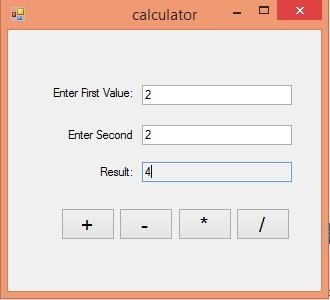
**OUTPUT:**



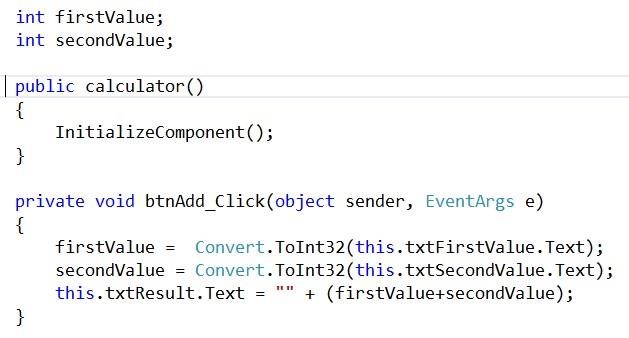
# ACTIVITY 9: STEPS

1. Add new form in DemoApplication for simple calculator.
2. Add following controls on form.

|  |  |  |
| --- | --- | --- |
| Control | Set following properties | Events |
| TextBox1 | 1. Name = “txtFirstValue” | No event |
| TextBox2 | 1. Name= “txtSecondValue” | No event |
| TextBox3 | 1. Name=”txtResult” 2. Readonly = true | No event |
| Button1 | 1. Name = btnAdd 2. Text = “+” | Double click button to generate click event automatically as below:  btnAdd\_click() |
| Button2 | 1. Name = btnSubtract 2. Text = “-” | btnSubtract\_click() |
| Button3 | 1. Name = “btnMultiply” 2. Text = “\*” | btnMultiply\_click() |
| Button4 | 1. Name = “btnDivide” 2. Text=”/” | btnDivide\_click() |



1. Add following code for Add operation and create code for other operations as well.



1. Write code for other buttons and add one extra button for clearing values of text boxes named btnClear.
2. Execute code and validate the results.

CODE:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Calculator

{

public partial class Form1 : Form

{

float first\_value=0, second\_value=0;

public Form1()

{

InitializeComponent();

}

private void label2\_Click(object sender, EventArgs e)

{

first\_value = Convert.ToInt32(textBox1.Text);

second\_value = Convert.ToInt32(textBox2.Text);

textBox3.Text = " " + (first\_value - second\_value);

}

private void button3\_Click(object sender, EventArgs e)

{

}

private void button2\_Click(object sender, EventArgs e)

{

first\_value = Convert.ToInt32(textBox1.Text);

second\_value = Convert.ToInt32(textBox2.Text);

textBox3.Text = " " + (first\_value \* second\_value);

}

private void button4\_Click(object sender, EventArgs e)

{

first\_value = Convert.ToInt32(textBox1.Text);

second\_value = Convert.ToInt32(textBox2.Text);

textBox3.Text = " " + (first\_value / second\_value);

}

private void button1\_Click(object sender, EventArgs e)

{

first\_value = Convert.ToInt32(textBox1.Text);

second\_value= Convert.ToInt32(textBox2.Text);

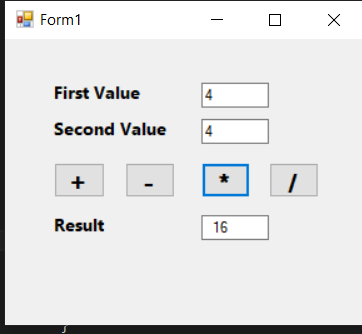
textBox3.Text = " " + (first\_value + second\_value);

}

}

}

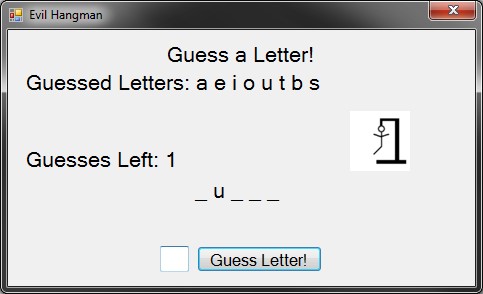
OUTPUT:



# ACTIVITY 10: Making a Hangman Game

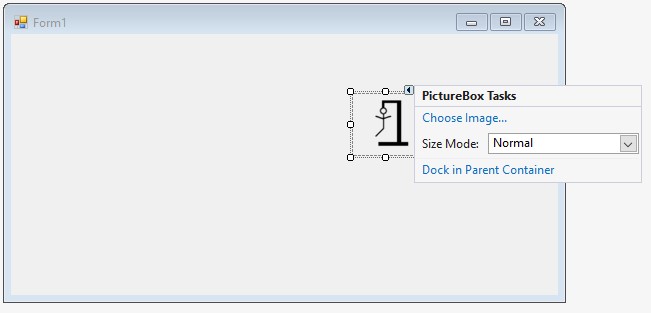
1. Create a new project named HangmanGame. 2. Add following controls on form

|  |  |  |
| --- | --- | --- |
| Control | Set following properties | Events |
| Label1 | Guess a Letter! |  |
| Label2 | Guessed Letters: |  |
| Label3 | Guesses Left |  |
| Label4 | (A random name with random spaces) |  |
| TextBox1 | 1. Name = GuessBox 2. MaxLength = 1 |  |
| Button1 | 1. Name = btnGuess 2. Text = “Guess Letter!” | btnGuess\_click() |
| PictureBox1 | 1. Name = picGuess |  |

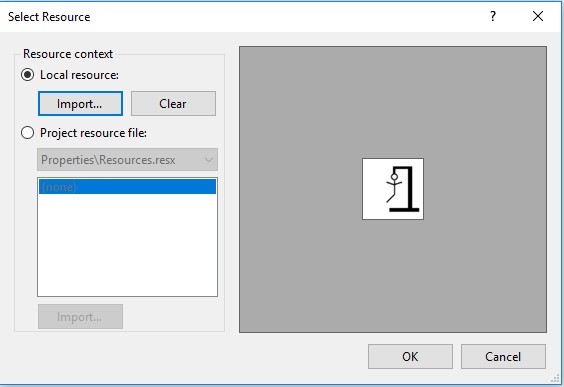


3. Use following steps to add images to the PictureBox 4. Drag and drop PictureBox on the form1.

1. Click on the play button to open a context menu.

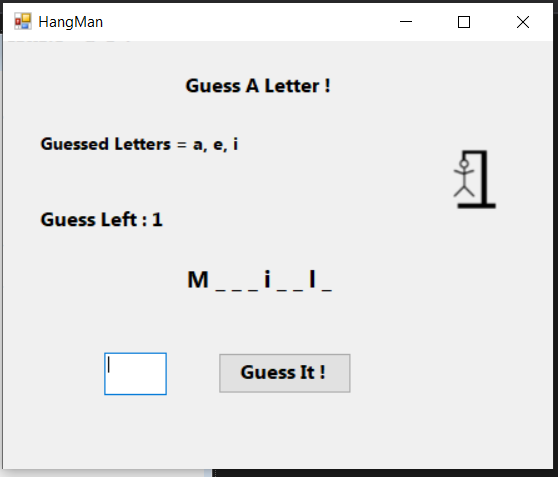


1. Click choose Image
2. On the new popup go for the Import Image from local resource



1. Click ok and the image will be imported in the selected PictureBox.

OUTPUT:



MUHAMMAD ALI NASIK AWAN