OBJECTIVES:

1. Introduction to Visual Studio 2012/2015 and its environment.
2. Understanding Assembly.
3. Understanding and Developing Console Applications.
4. Practice Activities.

OBJECTIVE 1: Introduction to Visual Studio 2012/2015

Visual Studio

- Visual Studio is Microsoft proprietary IDE.
  Visual Studio 2015 use .NET framework 4.6 and C# 6.0 by default. Visual Studio provides following features and many more.
  - Powerful coding tools
  - Advanced Debugging
  - Web tools
  - Multiple languages
  - Ecosystem
  - Git Integration
  - Settings go with you

Powerful coding tools provide following features

- **IntelliSense**: It describes APIs as you type and uses auto completion to increase speed and accuracy.
- **Squiggly Lines**: Red lines under code demonstrating issues with code.
- **Peek to Definition**: Using Peek to Definition one can view and edit the code without switching away from the code that you're writing. Peek to Definition shows the target code in pop-up window.

  - ALT + F12 is used for Peek to Definition.
  - One can also open nested Peek to Definition windows.
  - Result list appears, if code that is selected has more than one definitions

- **Go to Definition:** Using Go to Definition one can view and edit the code without switching away from the code that you're writing. Go to Definition shows the target code in separate code window.
  - F12 is used for Go to Definition.
- **Navigate Within code**: These are the techniques for moving to particular locations into code more quickly.

Bookmark lines of code: You can use bookmarks to mark lines in your code so that you can quickly return to a specific location and jump back and forth between locations.

⇨
⇨
⇨

- To add a bookmark, place the cursor on the line you want to bookmark.
- Click the **Toggle** button, or press CTRL+K, this will add the bookmark.
- For Toggle button, go to: Choose Edit, Bookmarks, and Toggle Bookmark for setting a bookmark.
- All the bookmarks can then be viewed in bookmarks window.

⇨

- Select **View/Bookmark Window** to view book mark manager.

Search for symbol definitions in a file: searching within a solution to locate a symbol definitions and file names but search results don't include namespaces or local variables.

- To access this feature, on the menu bar, choose **Edit**, **Navigate To**.
- CTRL+, is used to find symbols.

Browse the overall structure of your code: In **Solution Explorer**, you can search and browse classes and their types and members in your projects.

- **Quick Find**: Finding symbols in the editor.
  - CTRL + F key combination is used to find keywords in editor.

Editor Find

Quick Find	Ctrl+F
Quick Find Next Result	Enter
Quick Find Previous Result	Shift+Enter
Quick Find Expand Drop Down	Alt+Down
Dismiss Find	Esc
Quick Replace	Ctrl+H
Quick Replace Replace Next	Alt+R
Quick Replace Replace All	Alt+A
Find in Files	Ctrl+Shift+F
Replace in Files	Ctrl+Shift+H

- **Editor zoom**: Zooming the code editor windows
  - Ctrl + Shift + >: for zooming in the code editor
  - Ctrl + Shift + <: for zooming out the code editor
- **Fix issues quickly**: Light bulb icons help you identify and fix common coding issues, in many cases "live" as you type your code, and take quick code actions (like refactoring, implementing interfaces and more) from right inside the editor.

- **Refactor with ease**: Refactoring is the process of improving your code after it has been written by changing the internal structure of the code without changing the external behavior of the code.

  Rename Refactoring: provides an easy way to rename identifiers for code symbols such as fields, local variables, methods, namespaces, properties, and types.

  - Rename can be used to change the names in comments and in strings and to change the declarations and calls of an identifier.
  - **CTRL + R,R** keys are used to apply rename refactoring.

  Extract Method Refactoring: is a refactoring operation that provides an easy way to create a new method from a code fragment in an existing member.

  - For extract method refactoring select any lines of code and go to Refactor Menu, click Extract Method.
  - Extract method dialog box appears.
  - Alternatively CTRL + R, M keys combination can be used for this purpose.
  - During extract method refactoring a new method is inserted into the code.

  Encapsulate Field Refactoring: enables you to quickly create a property from an existing field, and then seamlessly update your code with references to the new property.

  - To create the new property, the **Encapsulate Field** operation changes the access modifier for the field that you want to encapsulate to private, and then generates get and set accessors for that field.
  - CTRL + R, E keys are used for creating properties for fields.

- Select the field for and press CTRL + R, E for this operation.

Extract Interface Refactoring: Extract Interface is a refactoring operation that provides an easy way to create a new interface with members that originate from an existing class, struct, or interface.

⇨ - CTRL + R, I keys are used for extracting members from existing class, struct or interface to create other interface that will be placed into the new file.

Remove Parameters Refactoring: provides an easy way to remove parameters from methods, indexers, or delegates. Remove Parameters changes the declaration; at any locations where the member is called, the parameter is removed to reflect the new declaration.

⇨ - Keyboard shortcut CTRL+R, V to display the **Remove Parameters** dialog box.

- From there remove the parameters as you want, and those parameters will be remove from everywhere.

Reorder Parameters Refactoring: refactoring operation that provides an easy way to change the order of the parameters for methods, indexers, and delegates.

- **Reorder Parameters** changes the declaration, and at any locations where the member is called, the parameters are rearranged to reflect the new order.

- **CTRL + R, O** keyboard shortcut is used to invoke this refactoring operation.

○ **Use code snippets:** Snippets are pre-built code chunks used for rapid application development. One can use already created snippets or can also create their own snippets.

- 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 and pressing two times tab key.
- For example: if + tab + tab will insert if code snippet/chunk.

OBJECTIVE 2: Understanding Assembly

Assembly

- It is building block of .NET framework applications.
- It is collection of types and resources that forms a logical unit of functionality.
- It is fundamental unit of deployment, version control, reuse, activation scoping, and security permissions.
- When compiling c# application an executable file is created this file is known as assembly.
- This is the file that CLR runs.
- Assembly contains code in MSIL (Microsoft Intermediate language) format.
- All compilers of .NET framework compiles code into this format.
- An assembly can be of two types: executable (.exe) and data link library (.dll: library that contains code re-used by other programs).
- Tools that used to work on assemblies can be found at locations.
- C:\Windows\Microsoft.NET\Framework64\v4.0.30319
- C:\Program Files\Microsoft SDKs\Windows\v7.0A\bin

OBJECTIVE 3: Understanding and developing Console Application

Console Application

- Application created to be executed on console, display its output and takes input on/from console.
- Console application used to be invoked using command prompt by c-sharp compiler if application is created using c-sharp.

ACTIVITIES SECTION

ACTIVITY 1: Creating a basic Console Application without visual studio.

Open notepad editor.
Write following code in it.

```
using System;

namespace test{

        public class testClass{

                public static void Main(){

                        Console.WriteLine("Hello World!");

                        // Keep the console window open in debug mode.
                        Console.WriteLine("Press any key to exit.");
                        Console.ReadKey();

                }
        }
}
```

For code compilation open command prompt or developer console of visual studio 2015/2012.
In case of command prompt you need to set path for c-sharp compiler.
For path setting write

Set path =”C:\Windows\Microsoft.NET\Framework64\v4.0.30319”

Writing following command will create an assembly with extension .exe.

```
csc name_of_csharp_file.cs
```

Writing Name of exe file that is generated by previous command will invoke the program.

```
name_of_assembly_created
```

ACTIVITY 2: Taking arguments at run time from console.

Open notepad editor.
Write following code that will take arguments at run time from user.

```
using System;

namespace test{
        public class testClass{
                public static void Main(string[] args){
                        Console.WriteLine("Name: "+ args[0]);
                }

        }
}
```

- Compile code using c-sharp compiler.
- During assembly invocation provide single argument.
  For example: test2 Nisar

  Output will be: Name: Nisar

ACTIVITY 3: Create Console Application and take 3 values at run time, first value will be operator like + sign, second value will be first operand and third value will be second operand, perform the operation on the operands according to operator.

- int.Parse ("string value") is used to convert string value to integer.

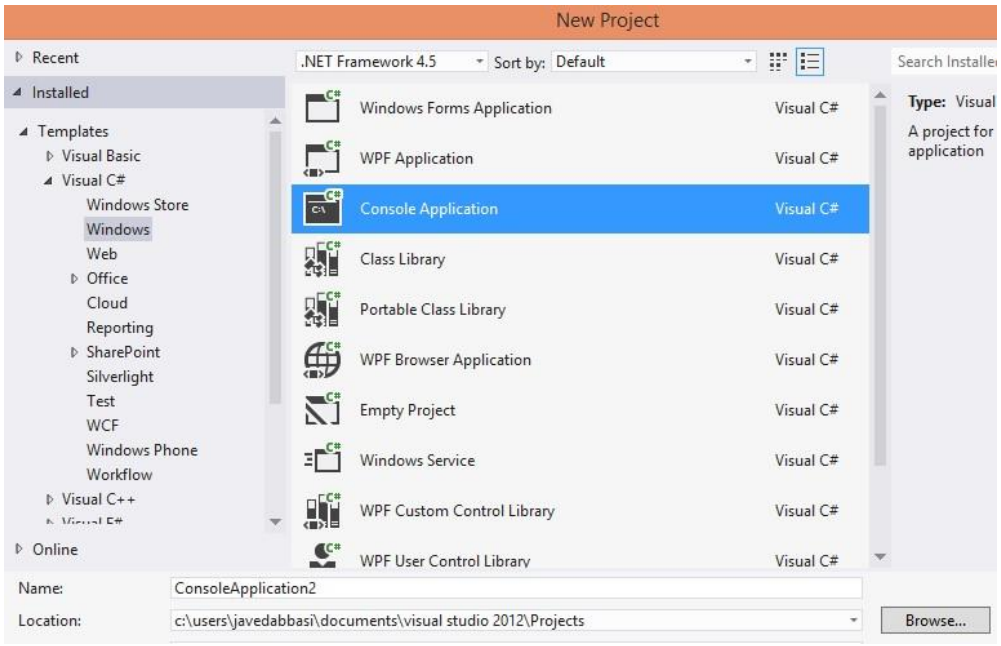
For example: if input is like this: basisCalculator + 2 2

Then output should be: Addition is: 4

Results portion: Place screen shot of result obtained through this activity.

ACTIVITY 4: Creating a basic Console Application using visual studio.

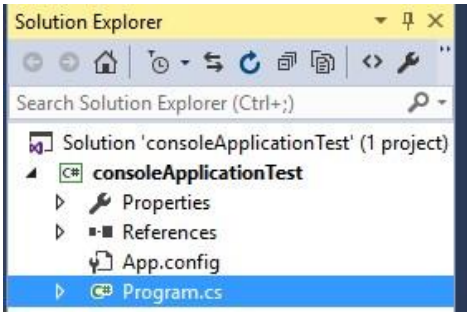
1. Open Microsoft Visual Studio 2012/2015.
2. Create a new console application project called ConsoleApplication1.
   - CTRL + SHIFT + N is shortcut after launching visual studio.



3. In the New Project dialog box, in the Installed Templates pane, expand Visual C#, and then click Windows.
4. In the Templates pane, click Console Application.
5. Provide following values for each of the properties in the dialog box, and then click OK
   - Name: ConsoleApplication
   - Location: default location of project

6. With step 5 a default class is added to a project named Program in that we have an entry point of program called Main method that is static in nature and can be called with the name of class and will be able to call only static methods in it.

```
namespace consoleApplicationTest
{
    class Program
    {

        static void Main(string[] args)
        {
            Console.Write("Hello World Example");
        }
    }
}
```



7. Run the program by pressing CTRL+F5 that is without debugging mode, it will take less time to execute the code as compared to compiling code in debugging mode.
8. If you want to run code using debugging mode press F5 only.

ACTIVITY 5: Modifying console application created in Activity 1.

⇨ Continue with activity 1.
⇨ In the Main method add following code, which will be used to read a line
⇨ of text from the keyboard.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Read a line of text from the keyboard
    line = Console.ReadLine();
}
```

⇨

ReadLine() method of Console class is used to read line of text.
Now again add the statement and comment shown in bold in the following code example, which echo the text back to the console by using the Console.WriteLine method.

```
static void Main(string[] args)
{
    // Buffer to hold a line as it is read in
    string line;

    // Read a line of text from the keyboard
    line = Console.ReadLine();

    // Write the results out to the console window
    Console.WriteLine(line);
```

Run the application and verify that it works as expected. You should be able to enter a line of text and see that line echoed to the console.

- On the Debug menu, click Start without Debugging (CTRL + F5).
- In the console window, type some text, and then press ENTER.
- Verify that the text that you typed is echoed to the console.
- Press ENTER to return to Visual Studio.

ACTIVITY 6: Create a console application like in activity 3 but here you have to take values at runtime using Console.ReadLine() method instead of taking it from string array.

Create console application that prompt for two values and operator from user in order to perform mathematical operations on operands. According to operator an operation should be performed on values.

int.Parse ("string value") is used to convert string value to integer.

⇨ **Results Portion: place snapshot of result and code in this portion.**

⇨

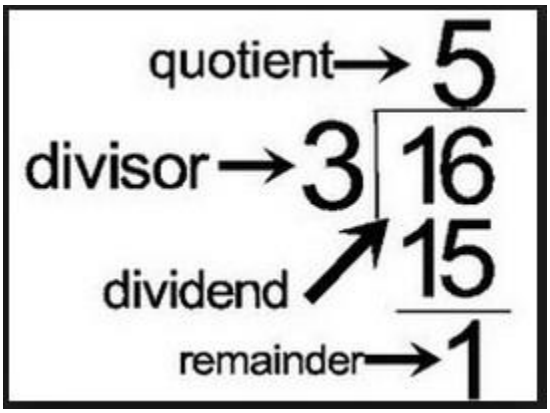
⇨

ACTIVITY 7: Write a static method named **DivideWithoutOperator** which takes two values dividend and divisor and displays result of division in console.

Note: Division operator should not be used for division.

Hint: Division is x number of times subtraction of divisor from dividend until you get remainder less than the divisor.

⇨



ACTIVITY 8: working with peek to definition and go to definition in visual studio.

Text is covered in objective 1.

Add new class named MathClass in project by right clicking on name of project visible in solution explorer.

```
namespace consoleApplicationTest
{
    class MathClass
    {
        public static void calc()...
    }
}
```

⇨
⇨ In Main method of Program class call calc() method.
⇨ By selecting signature of calc() method Press F12, this will directly jump to the calc() method from Main() method this is go to definition.
By selecting signature of calc() method Press ALT + F12, this will open a pop-up window at that position where you can modify the code of calc() method, this is peek to definition.
Observe the results of activity.

ACTIVITY 9: Setting Book Marks in the code.

⇨ Create console application using visual studio.
Write few lines of code in Program.cs file.
Press CTRL + K + K by standing at line of code, this will add book mark.
Press CTRL + K + K by standing at the same line to disable the book mark.

You can jump from one book mark to other by using book mark window.
Press CTRL + K + W to display book mark window.

ACTIVITY 10: Refactoring using visual studio.

Task 1: Extract Method Refactoring

It is a refactoring operation that provides an easy way to create a new method from a code fragment in an existing member.

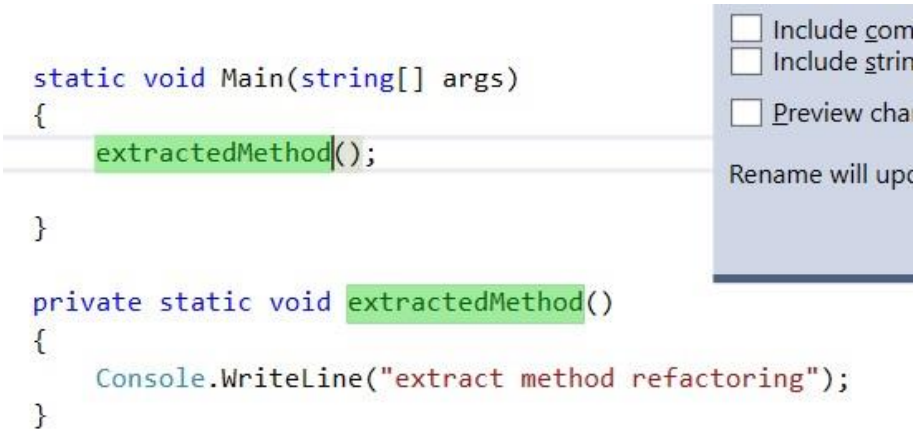
- **Create console application and write following code.**

```
class Program
{

    static void Main(string[] args)
    {
        Console.WriteLine("extract method refactoring");

    }


}
```

⇨
⇨
⇨

- Select this line of code from Main method.
- Press **CTRL + R, M** key for invoking window used in method refactoring.
- A new method will be created for which developer has to specify a name.

⇨



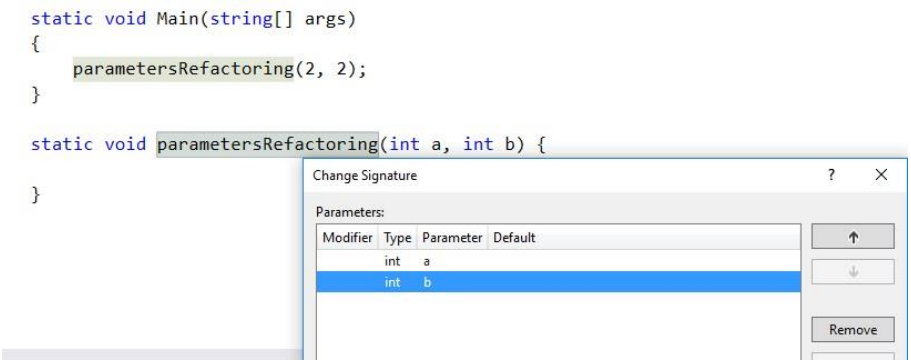
Task 2: <u>Remove Parameters Refactoring</u>

Provides an easy way to remove parameters from methods. Remove Parameters changes the declaration; at any locations where the member is called, the parameter is removed to reflect the new declaration.

- Write following code in console application.

```
class Program
{

    static void Main(string[] args)
    {
        parametersRefactoring(2, 2);
    }

    static void parametersRefactoring(int a, int b) {

    }

}
```

- By standing at signature of parametersRefactoring() method Press Keyboard shortcut CTRL+R, V to display the **Remove Parameters** dialog box.



- Remove the desired parameter and effects will be reflected throughout the code.

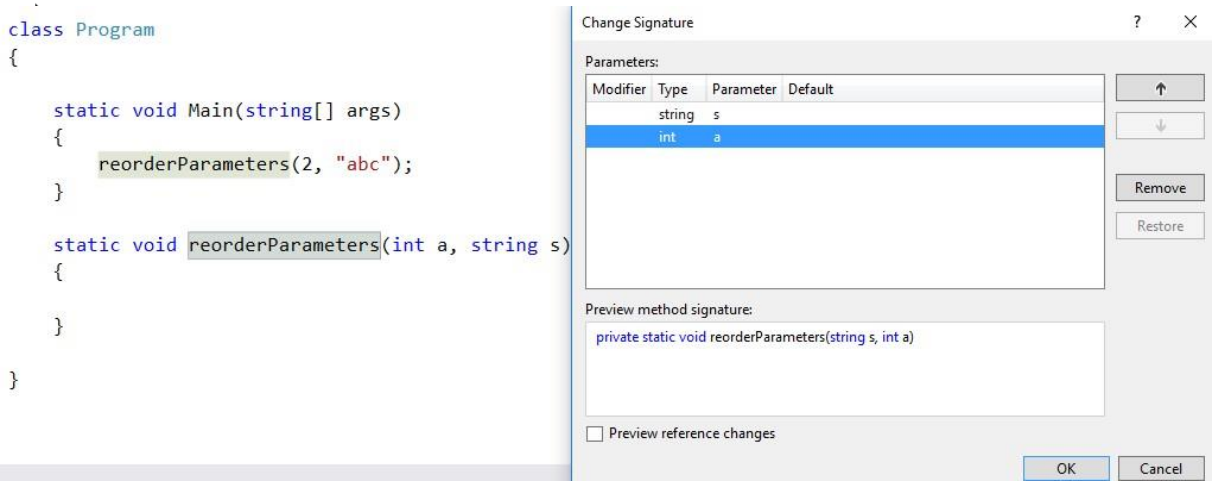
Task 3: <u>Reorder Parameters Refactoring</u>

Refactoring operation that provides an easy way to change the order of the parameters for methods.

- **Reorder Parameters** changes the declaration, and at any locations where the member is called, the parameters are rearranged to reflect the new order.

- Write following code in console application.

- **Press CTRL + R, O** keyboard shortcut to invoke refactoring operation.



- Re-order the parameters and results will be reflected everywhere in code.

```
class Program
{

    static void Main(string[] args)
    {
        reorderParameters("abc", 2);
    }

    static void reorderParameters(string s, int a)
    {

    }

}
```

Task 4. Rename Refactoring

Rename Refactoring: provides an easy way to rename identifiers for code symbols such as fields, local variables, methods, namespaces, properties, and types.

- Perform this activity for renaming fields, local variables and for each construct quoted in description independently.

⇨
⇨
⇨

⇨