**Patricia Caroline (916304)**

## Day 3 Handson 1.

Hands on – 1

Async Await usage - 1

Create a Console application that will print messages to the User The application should contain

· Two methods that return Asynchronous task

· The first method invokes the second method and awaits till the second method returns simple string. The string data returned by the second method should be stored in a string in the first method to display it

· Use Thread.Sleep in the second method to simulate the time delay

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading;

using System.Threading.Tasks;

namespace AsyncAwait11

{

public static class Program

{

public static async Task<string> StringReturn()

{

return "Hi, Welcome to Visual Studio";

}

public static async void Display()

{

Console.WriteLine("Please Wait!");

Thread.Sleep(2000);

var msgReceived = await StringReturn();

Console.WriteLine(msgReceived);

}

static void Main(string[] args)

{

Display();

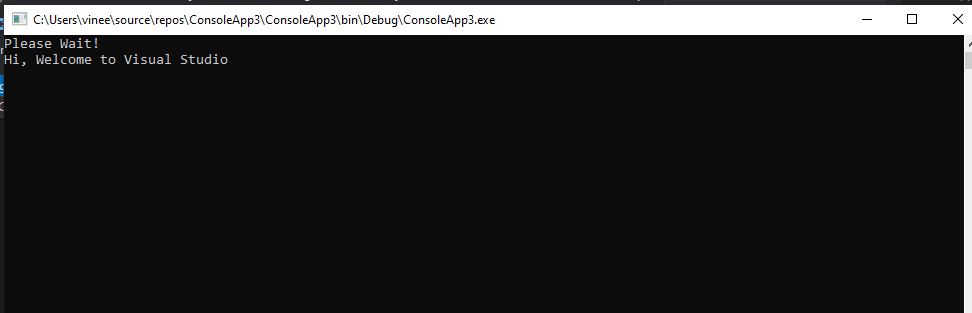
Console.ReadKey();

}

}

}

Output:



Async Await usage - 2

Use Windows forms application with Async Await usage

Create a Windows Forms application to read the number of characters in a text file(preferably a large file) and print that on a label

Use Async await concept to read the file content as the read operation shouldn't make the Windows form non-responding

Filecount.cs

using System;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.IO;

using System.Threading;

namespace filecount

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

public int CountChars()

{

int count = 0;

using (StreamReader streamReader = new StreamReader("C:\\Users\\vinee\\Desktop\\vivek.txt"))

{

string content = streamReader.ReadToEnd();

count = content.Length;

Thread.Sleep(2000);

}

return count;

}

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

{

}

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

{

}

private async void button1\_Click\_1(object sender, EventArgs e)

{

Task<int> task = new Task<int>(CountChars);

task.Start();

label1.Text = "File is processing";

int count = await task;

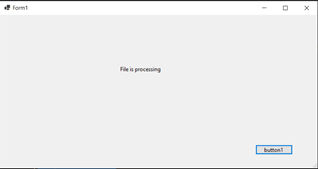
label1.Text = count.ToString() + " characters";

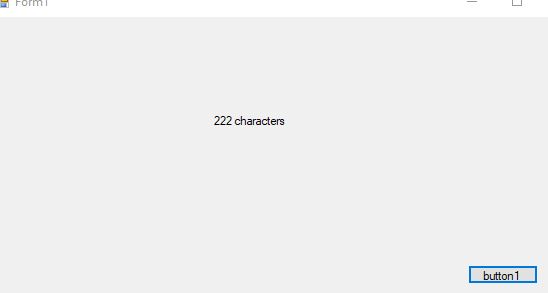
}

}

}

Output:





3.

Named parameters – Order of arguments as per the function and modify

Business case: Create a console application to print the details of a Cohort

Description:

Create a console application with method ‘GetCohortDetails’ that will take in the following input thru parameters:

· Cohort name

· GenC count

· Mode : OBL or PARC

· Track : JAVA or .Net

· Current Module

And print the details in the format – It is [Cohort name] with [GenC count] GenCs undergoing training for [Track] thru [Mode]. The current module of training is [Current Module]

Invoke the method ‘GetCohortDetails’

· Send all the parameters in the order of that in the method and check the output

· Send the parameters in incorrect order and check the output

· Use Named parameters to send the input data, NOT in the order in the method and check the output

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace GetCohortDetails

{

enum Mode

{

OBL,

PARC

}

enum Track

{

JAVA,

DOTNET

}

class Program

{

public static void GetCohortDetails(string cohortName, int gencCount, Mode mode, Track track, string currentModule)

{

Console.WriteLine("It is " + cohortName + " with " + gencCount + " GenCs undergoing training for " + track + " thru " + mode + ". The current module of training is " + currentModule);

}

static void Main(string[] args)

{

GetCohortDetails("CDE", 10, Mode.OBL, Track.JAVA, "java");

GetCohortDetails(cohortName: "CDE", gencCount: 16, mode: Mode.OBL, track: Track.JAVA, currentModule: "java");

GetCohortDetails("CDE", 20, Mode.PARC, Track.DOTNET, "dotnet");

GetCohortDetails(cohortName: "CDE", gencCount: 25, mode: Mode.PARC, track: Track.DOTNET, currentModule: "dotnet" +

"");

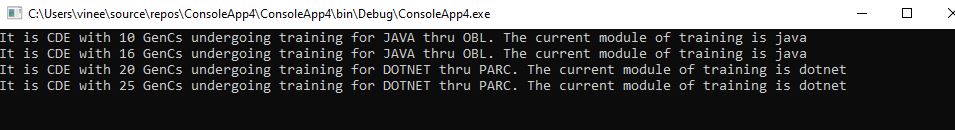
Console.ReadKey();

}

}

}

Output:



4.Optional parameters

Business case: Create a console application to print product details for an Amazon order

Description:

Create a console application with a method ‘OrderDetails’ that will take in the following input thru parameters:

· Seller name

· Product name

· Order quantity – Optional with default value 1

· Is returnable – Optional with default value TRUE

And print the details in the format – Here is the order detail – [Order quantity] number of [Product name] by [Seller name] is ordered. It’s returnable status is [Is returnable]

Invoke the method ‘OrderDetails’

· Send data for the required parameters only and check the output

Send data for the required and optional parameters and check the output

Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace OrderDetails

{

class Program

{

public static void OrderDetails(string sellerName, string productName, int orderQuantity = 1, bool isReturnable = true)

{

Console.WriteLine($"Here is the order detail –" + orderQuantity + " number of " + productName + " by " + sellerName + " is ordered. It’s returnable status is " + isReturnable);

}

static void Main(string[] args)

{

OrderDetails("vivek", "airconditioner");

Console.ReadKey();

}

}

}

Output: