**Assignment of EAD**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**



**Submitted to:**

Dr.Shuja-ur-Rehman Baig

**Submitted by:**

Saad Ishtiaq

Roll no: BSEF18A025

***(DO NOT COPY.*** *This property belongs to***Saad Ishtiaq.** Beware***!!! )***

**Punjab University College of Information Technology,**

**PUCIT (Old Campus),**

**LAHORE**

**EAD HOMEWORK\_2**

Information is taken from

Provided Reference Book ,Google and Sir Shuja’s Lecture Notes.

**(BSSE & BSIT FALL 2018)**

**(Most Of the Examples are inspired by Sir Notes)**

**Q1- How does const differ from ReadOnly in C#?**

Ans:

|  |  |
| --- | --- |
| **Const** | **ReadOnly** |
| A **constant** member is defined at compile time and **cannot be changed at runtime**. Constants are declared as a field, using the const keyword and **must be initialized as they are declared**. | A **readonly** member is like a constant in that it represents an unchanging value. The difference is that a readonly member **can be initialized at runtime**, in a **constructor**, as well being able to be initialized as they are **declared**. |
| They can not be declared as static (they are implicitly static) | They can be either instance-level or static |
| **Example:**  public class MyClass  {  public const double PI1 = 3.14159;  } | **Example:**  public class MyClass1  {  public readonly double PI2 = 3.14159;  //or  public readonly double PI3;  public MyClass2()  {  PI3 = 3.14159;  }  } |

**Q2-Why we need Tuples?**

Ans:

**Tuple** is a generic static class that was added to C# 4.0 and it can hold any amount of elements, and they can be any type we want. So using tuple, we can return multiple values.One great use of tuple might be **returning multiple values from a method**.    
 A tuple allows you to combine multiple values of possibly different types into a single object **without** having to create **a custom class**. This can be **useful** if you want to write a method that for example returns three related values but you don't want to create a new class.

**Example:**

using System;

class Person

{

public string Name;

public int Age;

public string Country;

public (string MyName, int MyAge, string MyCountry) GetPersonDetails()

{

return (this.Name, this.Age, this.Country);

}

}

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Person p = new Person { Age = 12, Name = "Ali", Country = "Pakistan" };

(string name, int age, string country) = p.GetPersonDetails();

Console.WriteLine(name);

Console.WriteLine(age);

Console.WriteLine(country);

}

}

}

**Q3-Why we use out parameter in C#?**

Ans:

**out** is **used** to state that the **parameter** passed must be modified by the method. When a **parameter** passes with the **Out** keyword/**parameter** in the method, then that method works with the same **variable** value that is passed in the method call. If **variable** value changes, the method **parameter** value also changes.

**There are some additional features of C# Out parameter**

1. Out parameter can pass without its declaration and initialization.
2. Out parameter can use var type in the method parameter list.
3. It is not obligatory that Out parameter name should be same in both function definition and call.

**Example:**

using System;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Person p2 = new Person();

int a = 10;

Console.WriteLine($"before passing a ={a}");

p2.PassingParamters(out a);

Console.WriteLine($"after passing a ={a}");

}

}

}

class Person

{

public void PassingParamters(out int z)

{

z = 99;

z++;

}

}

**Q4-What are the advantages of using Partial Classes in C#?**

Ans:

* **Multiple Developer:** Using Partial Classes multiple developer can work on the same class easily.
* **Code Generator:** Partial classes are mainly used by code generator to keep different concerns separate
* **Partial Methods:** Using Partial Classes you can also define Partial methods as well where a developer can simply define the method and the other developer can implement that.
* **Partial Method Declaration only:** Even the code get compiled with method declaration only and if the implementation of the method isn't present compiler can safely remove that piece of code and no compile time error will occur.

**Example:**

using System;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Person3 p3 = new Person3();

p3.Name = "Saad";

p3.Age = 20;

Console.WriteLine(" Name: " + p3.Name + " Age: " + p3.Age);

}

}

}

partial class Person3

{

private string name;

public string Name

{

get

{

return name;

}

set

{

name = value;

}

}

partial class Person3

{

private int age;

public int Age

{

get

{

return age;

}

set

{

age = value;

}

}

}

**Q5-What is the output of the following program?**

using System;

public class Program

{

public static void Main()

{

void myFunc(string str, out string s)

{

s = str + "OOP"

+ "DSA";

}

string a = null;

myFunc("EAD", out a);

Console.WriteLine(a);

}

}

Ans:

Output will be -> EADOOPDSA

Because function is getting “EAD” as input from str variable and it is concatenating this string with “OOP” and “DSA” then returns in concatenated string in variable s which is of out type. Hence getting concatenated value in a, Then printing the string on console i.e “EADOOPDSA”.

**Q6- What are the advantages of Operator Overloading? Write any two.**

Ans:

Following are the advantages of operator overloading:

* Operator overloading enables programmers to use notation closer to the target domain. For example we can add two matrices by writing M1 + M2 rather than writing M1.add(M2).
* Operator overloading provides similar syntactic support of built-in types to user-defined types.
* Operator overloading makes the program easier to understand.

**Example:**

using System;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

Box Box1 = new Box(); // Declare Box1 of type Box

Box Box2 = new Box(); // Declare Box2 of type Box

Box Box3; // Declare Box3 of type Box

// box 1 specification

Box1.Length = 6.0;

Box1.Width = 7.0;

Box1.Height = 5.0;

// box 2 specification

Box2.Length = 12.0;

Box2.Width = 13.0;

Box2.Height = 10.0;

// Add two object as follows:

Box3 = Box1 + Box2;

Box3.PrintBoxDetails();

}

}

}

class Box

{

private double length;

private double width;

private double height;

public double Length { get => length; set => length = value; }

public double Width { get => width; set => width = value; }

public double Height { get => height; set => height = value; }

public void PrintBoxDetails()

{

Console.WriteLine($" length:{length}, width:{width} height:{height}");

}

// Overload + operator to add two Box objects.

public static Box operator +(Box b, Box c)

{

Box box = new Box();

box.length = b.length + c.length;

box.width = b.width + c.width;

box.height = b.height + c.height;

return box;

}

}

**Q7- Write a program in C# Sharp to create a file, write some text in the file using StreamWriter and read the file using StreamReader. Write the program in your compiler and attach the screenshot of your program along with its output.**

Ans:

using System;

using System.Collections.Generic;

using System.IO;

using System.Text;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

FileHandler.WriteFileusingStreamWriter();

FileHandler.ReadFileusingStreamReader();

}

}

}

namespace ConsoleApp1

{

internal class FileHandler

{

internal static void WriteFileusingStreamWriter()

{

FileStream fout = null;

try

{

fout = new FileStream("myFile.txt", FileMode.Create, FileAccess.Write);

StreamWriter stWriter = new StreamWriter(fout);

string str = "This Line is inserted by Saad";

stWriter.Write(str);

stWriter.Close();

}

catch (Exception ex)

{

Console.WriteLine(ex.GetType());

Console.WriteLine(ex.Message);

}

finally

{

fout.Close();

}

}

internal static void ReadFileusingStreamReader()

{

FileStream fout = null;

try

{

fout = new FileStream("myFile.txt", FileMode.Open, FileAccess.Read);

StreamReader stReader = new StreamReader(fout);

string line = String.Empty;

while ((line = stReader.ReadLine()) != null)

{

Console.WriteLine(line);

}

stReader.Close();

//ReadToEnd can also be used to read all at single time

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

finally

{

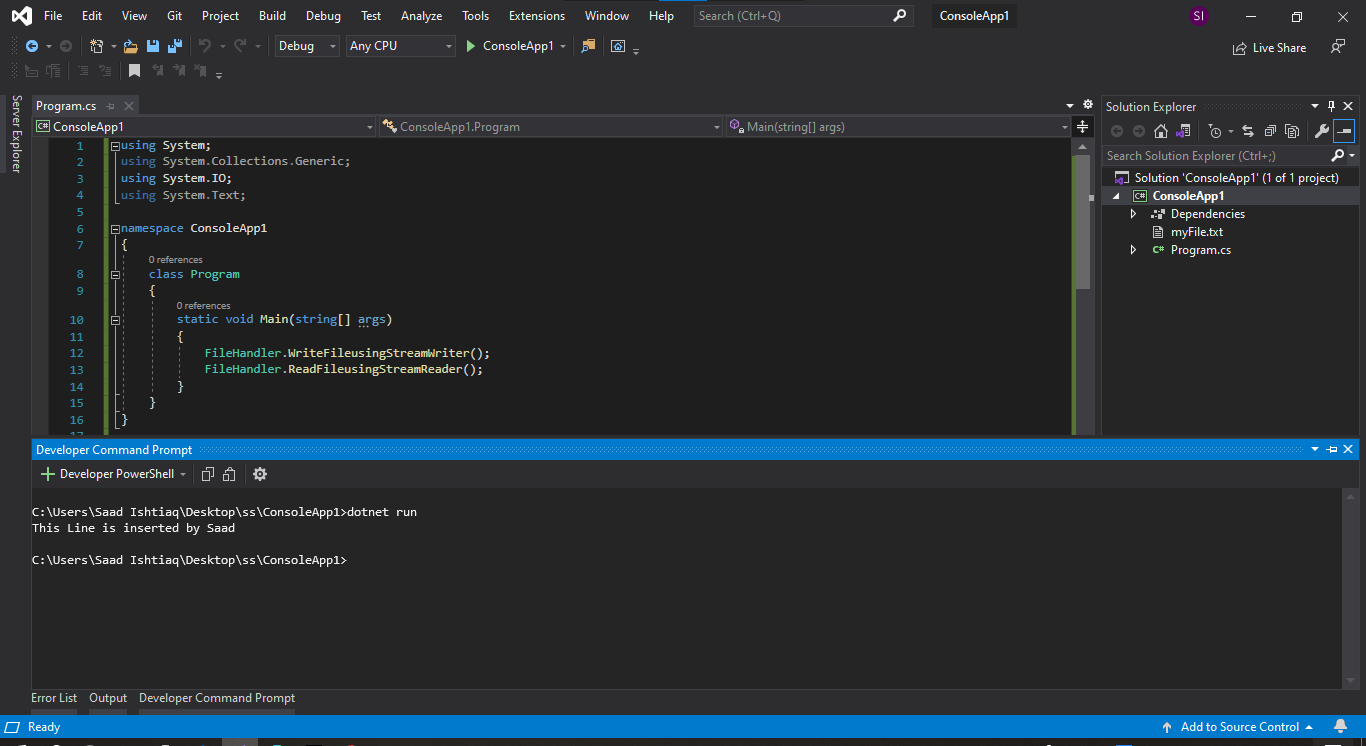
fout.Close();

}

}

}

}



**Q8- Write A function in C# that takes two numbers as parameters from user and swaps them. (Method return type should be void). Write the function in your compiler and attach the screenshot of your function along with its output.**

Ans:

using System;

namespace ConsoleApp1

{

class Program

{

static void Main(string[] args)

{

int x = 10;

int y = 20;

Console.WriteLine($"Value of x Before Swap: {x} \nValue of y Before Swap: {y} ");

MySwap(ref x,ref y);

Console.WriteLine($"Value of x After Swap: {x} \nValue of y After Swap: {y} ");

}

public static void MySwap( ref int a, ref int b)

{

int temp = a;

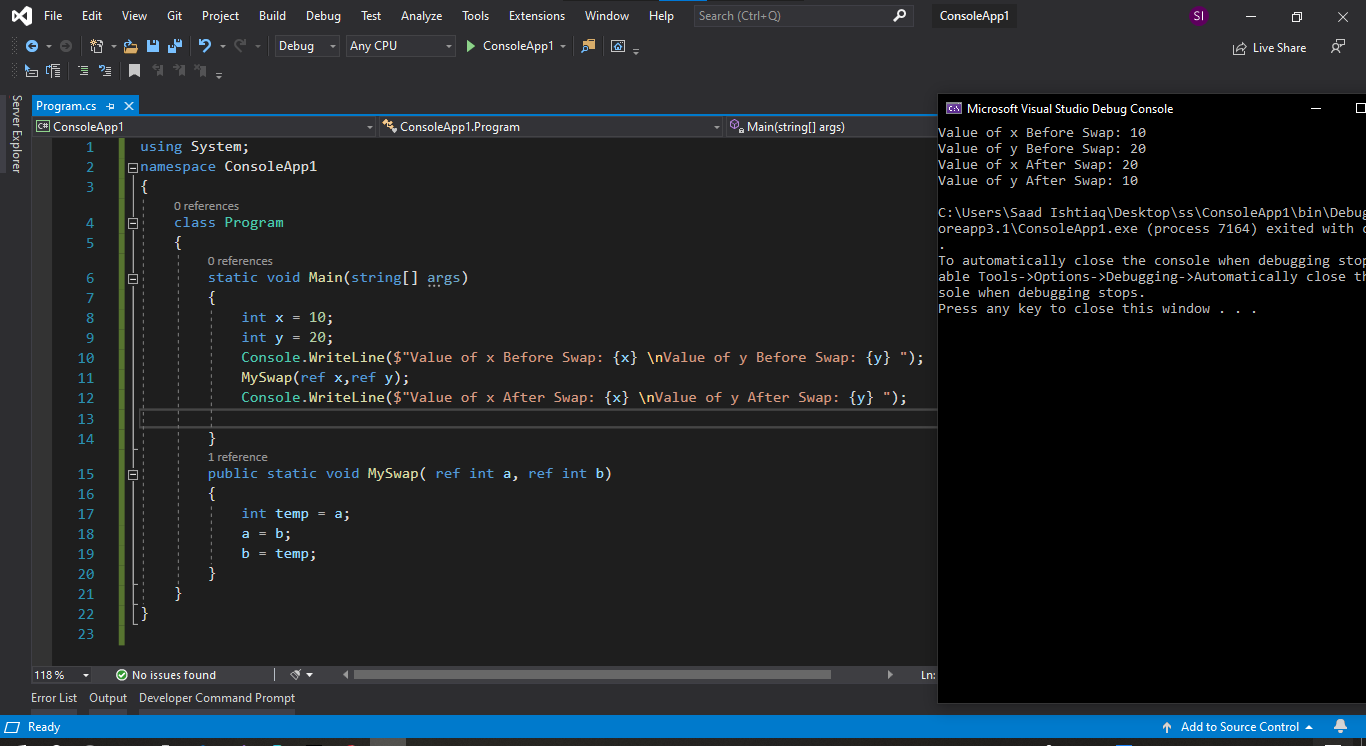
a = b;

b = temp;

}

}

}

****

**Q9- Write a class in C# that keeps counts of all its objects that are initialized during the execution of program. No variable outside the class should be responsible for maintaining count. Also this count cannot be changed by user himself but should only be modified by class. Write the program in your compiler and attach the screenshot of your program along with its output.**

**Ans:**

using System;

using System.Collections.Generic;

using System.Text;

namespace ConsoleApp1

{

public class CountClass

{

int value;

private static int count=0;

public CountClass(int x)

{

value = x;

count++;

}

public int Value => this.value;

public void ShowCount()

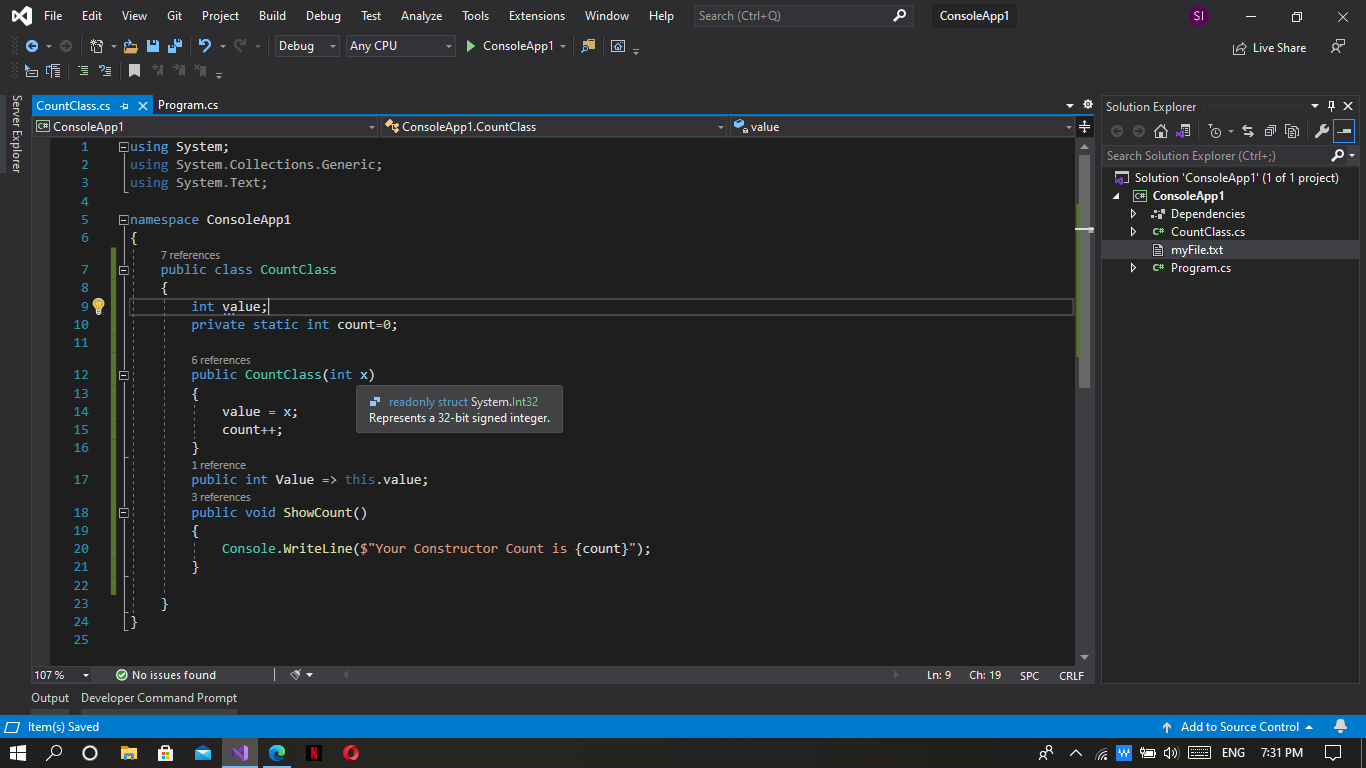
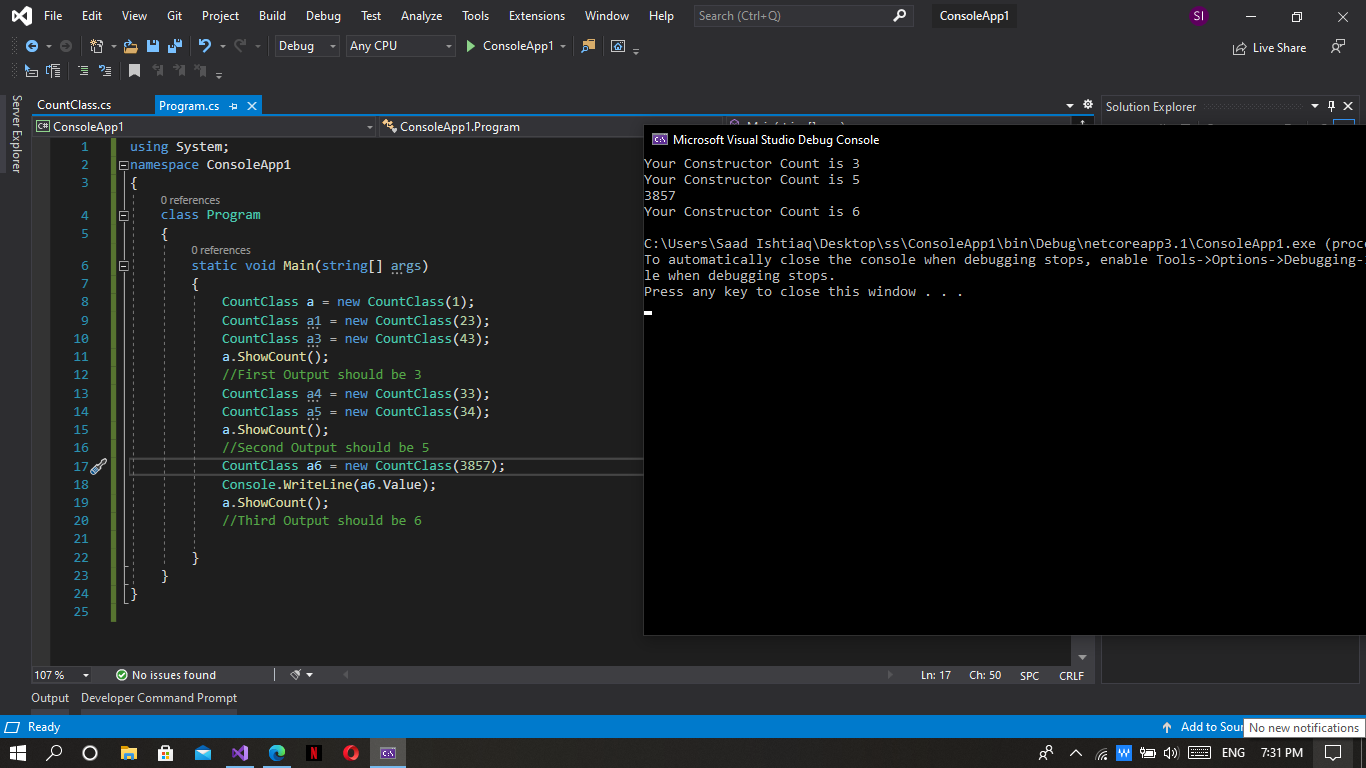
{

Console.WriteLine($"Your Constructor Count is {count}");

}

}

}

**Q10- What are the advantages of indexers in C#? Write any four.**

Ans:

**Advantages of Indexers:**

* They are used for overloading an [ ] operator.
* The syntax is simple an easy to user.
* It supports overloading for any user defines array in **C#**
* The access specifiers simply the code complexity.
* Indexer doesn’t require a name, and it is accessed using “this”.

**(**Site:Wideskills**)**