***DAY 20 ASSIGNMENT PRESENTED***

***BY***

***POTUKANUMA JEEVITHA***

***18-02-2022***

|  |
| --- |
| **1.Research and understand scope of variables in C#.** |
| The part of the program where a particular variable is accessible is termed as the scope of that variable. A variable can be defined in a class, method, loop etc.,  C# scope rules of variables can be divided into 3 categories:   * Class level scope * Method level scope * Block level scope |
| The scope of variable “b” is only within the code block:    The variable is inside the forloop, so the scope of the variable is inside the forloop only.    Here, variable is outside the forloop. |

|  |
| --- |
| **2. What are delegates in C# . write the points discussed about delegates in the class.** |
| **Keyword:-** “delegate”   * Delegate is a function pointer. * Using delegates we can call (or) point to one or more methods. * When declaring a delegate: * Return and parameters must match with the methods you want to point using the delegate.   **Benefits:**   * Using single call from delegate, all your methods pointing to delegate will be called.   **Types :**   * Single-cast delegate. * Multi-cast delegate. |
| **Write C# code to illustrate the usage of delegates.** |
| **Code:**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Author: JEEVITHA  //Purpose: example C# code for MULTI-CAST DELEGATE  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  namespace project\_2  {  public delegate void MyCaller(int a , int b);  internal class Program  {  public static void Add(int a, int b)  {  Console.WriteLine(a + b);  }  public static void Mul(int a, int b)  {  Console.WriteLine(a \* b);  }  public static void Div(int a, int b)  {  Console.WriteLine(a / b);  }  static void Main(string[] args)  {  MyCaller mc = new MyCaller(Add);  mc += Mul;  mc += Div;  // 10, 20  mc(10, 20);  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  //12,24  mc(12, 24);  Console.WriteLine("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  //30,14  mc(30, 14);  Console.ReadLine();  }  }  } |
| **Output:** |

|  |
| --- |
| **3.What are nullable types in C#.** |
| * The nullable type allows you to assign a null value to a variable.   **Properties of nullable types:**   * **Nullable <T>** type allows assignment of null to value types. * **?** operator is a shorthand syntax for nullable types. * Use **value** property to get the value of nullable type. * Use **HasValue** property to check whether value is assigned to nullable type or not. * **Static Nullable** class is a helper class to compare nullable types. |
| **Code:**  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Author: JEEVITHA  //Purpose: WACP to illustrate nullable type  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  namespace project\_3  {  internal class Program  {  static void Main(string[] args)  {  byte? input = 20;  if (input.HasValue)  Console.WriteLine(input \* input);  else  Console.WriteLine("No vaiue");  Console.ReadLine();  }  }  } |
| **Output:** |

|  |
| --- |
| **4.Out, ref – Parameters**  **Research on these two types of parameters. Write C# program to illustrate the same.** |
| Ref and out keywords in C# are used to pass arguments within a method or function. Both indicate that an parameter is passed by reference. By default parameters are passed to a method by value. By using these keywords (ref and out) we can pass a parameter by reference. |
| **Write a C# code using Ref keyword:**  The ref keyword passes arguments by reference. It means any changes made to this argument in the method will be reflected in that variable when control returns to the calling method. |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Author: JEEVITHA  //Purpose: C# code using ref parameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  namespace project\_4  {  internal class Program  {  public static string GetNextName(ref int id)  {  string returnText = "Next-"+ id.ToString();  id += 1;  return returnText;  }  static void Main(string[] args)  {  int i = 7;  Console.WriteLine("Previous value of integer i:" + i.ToString());  string test = GetNextName(ref i);  Console.WriteLine("Current value of intger i:"+i.ToString());  Console.ReadLine();  }  }  } |
| **Output:** |
| **Write a C# code using out keyword:**  The out keyword passes arguments by reference. This is very similar to the ref keyword. |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  //Author: JEEVITHA  //Purpose: C# code using out parameter  //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  namespace project\_5  {  internal class Program  {  public static string GetNextNameByOut(out int id)  {  id = 1;  string returnText = "Next-" + id.ToString();  return returnText;  }  static void Main(string[] args)  {  int i =0;  Console.WriteLine("Previous value of integer i:" + i.ToString());  string test = GetNextNameByOut(out i);  Console.WriteLine("Current value of intger i:" + i.ToString());  Console.ReadLine();  }  }  } |
| **Output:** |