|  |
| --- |
| Day 20 Assignment  (18-02-2022)  -By  Ram Charan |

|  |
| --- |
| **1. Research and understand scope of variables in C#** |
| Scope of Variable: |
| * The part of the program where a particular variable is accessible is termed as Scope of the variable. * A variable can be defined in a class, method , loop. * So C# scope rules of variables can be divided into three categories as follows:   Class Level Scope  Method level scope  Block level scope |
| Example:  namespace Day20Project1  {  //Author: Rc  //Purpose:Scope of variables  class Scope  {  int a; //class variable  public void Print()  {  int b=9; //Method variable  for(int i=0;i<=a;i++) // loop variable  {  if(a>b)  {  int n = 1; //this is only accessible for this if block only  }  }  }    } |

|  |
| --- |
| **2. What are delegates in C# Write the points dicussed about delegates in the class Write C# code to illustrate the usage of delegates.** |
| DELEGATES: |
| * Delegate is like a function pointer. * Using delegates, we can call or point to one or more methods. * When we declare a delegate, returntype and parameters must be same as method,you want to point using delegate. * Benefit of using delegate is, using single call from delegate ,all your methods pointing to delegate will be called. |
| Two types of Delegates: |
| 1.Single cast Delegate -  It points to only one method.  2.Multi-cast Delegate -  It points to more than one method. |
| Code: |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace Day20Project1  {  //Author: Rc  //Purpose: Delegates    class Program  {  public delegate void MyCaller(int a, int b);  public static void Add(int a, int b)  {  Console.WriteLine(a + b);  }  public static void Sub(int a, int b)  {  Console.WriteLine(a - b);  }  public static void Mul(int a, int b)  {  Console.WriteLine(a \* b);  }  static void Main(string[] args)  {  Console.WriteLine("Output1:");  MyCaller mc = new MyCaller(Add);  mc += Mul;  mc(5, 6);  Console.WriteLine("Output2:");  mc += Sub;  mc(4, 4);  Console.WriteLine("Output3:");  mc -= Mul;  mc(5, 2);  Console.ReadLine();    }  }  } |
| Output: |

|  |
| --- |
| **3. What are nullable types in C# WACP to illustrate nullable types Write some properties of nullable types (like HasValue)** |
| Nullable Types: |
| We use this, to initialize null to int or primitive datatypes. |
| Code: |
| namespace Day20Project2  {  //Author:Rc  //Purpose:Nullable types  class Program  {  static void Main(string[] args)  {  byte? i= null;  if (i.HasValue)  Console.WriteLine(i \* i);  else  Console.WriteLine("No value");  Console.ReadLine();  }  }  } |
| Output: |
|  |
| Properties :  1.HasValue  2.Value |

|  |
| --- |
| **4. out, ref - parameters please research on these two types of parameters write a C# program to illustrate the same.** |
| Out Parameter: |
| The out is a keyword in C# which is used for passing arguments to methods as a reference type.  It is generally used when a method returns multiple values. |
| Code:  using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace Day20Project3  {  //Author:Rc  /\*Purpose:Out parameters (Call By Value)\*/    class Program  {  public static void Add(out int p, out int q)  {  p = 30;  q = 40;  //p += q;  //q += q;  }  static void Main(string[] args)  {  int i, j;  Add(out i, out j);  Console.WriteLine(i);  Console.WriteLine(j);  Console.ReadLine();    }  }  } |
| Output: |
| Ref parameter:  The ref keyword in C# is used for passing or returning references of values to or from methods.  Basically, it means that any change made to a value that is passed by reference will reflect this change since you are modifying the value at the address and not just the value. |
| Code:  namespace Day20Project4  {  //Author : Rc  //Purpose : ref parameter  class Program  {  public static void Multi(ref int x)  {  x \*= x;  Console.WriteLine("Inside method:"+x);  }  static void Main(string[] args)  {  int a = 10;  Console.WriteLine("Before:"+a);  Multi(ref a);  Console.WriteLine("After:"+a);  Console.ReadLine();  }  }  } |
| Output: |

|  |
| --- |
| END OF THE DAY |