|  |
| --- |
| Day-18 Morning Assignment  By  U.Joshna  [16-2-2022] |

|  |
| --- |
| 1. What is the use of XML |
| * XML is a self describing language and it gives the data as well as the rules to identify what information it contains.Like HTML, XML is a subset of SGML - Standard Generalized Markup Language. The following links give you more information about XML Files and its operations through C# Programming Language. |

|  |
| --- |
| 2. Write the points discussed about xml in the class |
| * XML stands for extensible Markup Language. * XML has user defined tags. * It is a Case Sensitive. * XML Has only one Root tag as an Entry. * It’s not a platform dependant. |

|  |
| --- |
| 3. Create a simple xml to illustrate: a. Tag based xml with 10 products b. Attribute based xml |
| a.Tag based xml with 10 products: |
| |  | | --- | |  | | b.Attribute based xml: | |

|  |
| --- |
| 4. Convert the above xml to JSON and display the JSON data |
| XML to JSON.json: |
|  |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
| 5. Research and write the benefits of JSON over XML ( 2 or 3 points ) |
| * JSON is always preferable in terms of the processing the client browser has to do for parsing the data. * Also, JSON is light weight data exchange format. XML parsing always consumes lot of browser resources and should be avoided as much as we can unless otherwise required. * JSON is easy and faster to parse. |

|  |
| --- |
| 6. For the below requirement, create a layered architecture project with seperate class library for Business logic. create console application create windows(or desktop) application Business Requirement: FIND FACTORIAL OF A NUMBER: 0 = 1 positive number (upto 7) = factorial answer > 7 = -999 (as answer) < 0 = -9999 (as answer) put the screen shots of the output and project (solution explorer) screen shot |
| JoshnaLibrary: |
| using System;  namespace JoshnaLibrary  {  public class Algebra  {  public static int Factorial(int n)  {  if (n == 0)  return 1;  else if (n < 0)  return -9999;  else if (n > 7)  return -999;  else  {  int fact = 1;  for (int i = 1; i <= n; i++)  {  fact \*= i;  }  return fact;  }  }  }  } |
| MyWindowsForms: |
| using System;  using System.Windows.Forms;  using JoshnaLibrary;    namespace MyWindowsForms1  {  public partial class Form1 : Form  {  public Form1()  {  InitializeComponent();  }    private void Form1\_Load(object sender, EventArgs e)  {  int n = int.Parse(textBox1.Text);  int result = Algebra.Factorial(n);  textBox2.Text = result.ToString();  Console.ReadLine();  }  }  } |
| ConsoleApp: |
| using System;  using JoshnaLibrary;      namespace Day18Project6  {  internal class Program  {  static void Main(string[] args)  {  Console.Write("Enter Any Number : ");  int n = int.Parse(Console.ReadLine());  Console.WriteLine("\n\n Factorial Result is : {0}", Algebra.Factorial(n));  Console.ReadLine();  }  }  } |
| |  | | --- | | Output: | | ConsoleApp: | |  | |  | |  |  |  | | --- | |  | | MyWindowsForms1: | | Output: | |  | |  | |  | |  | |  | |

|  |
| --- |
| 7. For the above method, Implement TDD and write 4 test cases and put the code in word document. put the screen shot of all test cases failing. make the test cases pass. put the screen shot |
| Code: |
| using Microsoft.VisualStudio.TestTools.UnitTesting;  using JoshnaLibrary2;  namespace JoshnaLibrary2.Tests  {  [TestClass()]  public class AlgebraTests  {  [TestMethod()]    public void FactorialTest\_Zero\_Input()  {  // Arrange  int n = 0;  int expected = 1;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Negative\_Input()  {  // Arrange  int n = -3;  int expected = -9999;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Greater\_than\_seven\_Input()  {  // Arrange  int n = 9;  int expected = -999;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Input()  {  // Arrange  int n = 5;  int expected = 120;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  }  } |
| |  | | --- | | All Test Cases are Failed: | |
|  |
| All Test Cases are Successed: |
|  |
|  |

|  |
| --- |
| 8. Add one more method to check if the number is palindrome or not in the above Algebra class and write test case for the same. |
| Algebra Library: |
| Code: |
| using Microsoft.VisualStudio.TestTools.UnitTesting;  using JoshnaLibrary2;  namespace JoshnaLibrary2.Tests  {  [TestClass()]  public class AlgebraTests  {  [TestMethod()]    public void FactorialTest\_Zero\_Input()  {  // Arrange  int n = 0;  int expected = 1;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Negative\_Input()  {  // Arrange  int n = -3;  int expected = -9999;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Greater\_than\_seven\_Input()  {  // Arrange  int n = 9;  int expected = -999;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }  [TestMethod()]  public void FactorialTest\_Input()  {  // Arrange  int n = 5;  int expected = 120;  // Act  int actual = Algebra.Factorial(n);  // Assert  Assert.AreEqual(expected, actual);  }    [TestMethod()]  public void PalindromeTest\_Input()  {  // Arrange  int n = 131;  bool expected = true;  // Act  bool actual = Algebra.IsPalindrome(n);  // Assert  Assert.AreEqual(expected, actual);  }  }  } |
| Output: |
|  |
|  |
|  |