**Problem Statement 1 : Spell the Number**

Please write code which takes a whole number and spells it out in words. For instance if 13456 is input it should be spelled as “thirteen thousand four hundred and fifty six”. The developer could choose to spell it out using https://en.wikipedia.org/wiki/Indian\_numbering\_system or international numbering

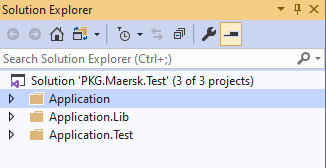
system <https://www.math-only-math.com/international-numbering-system.html>

**Solution**

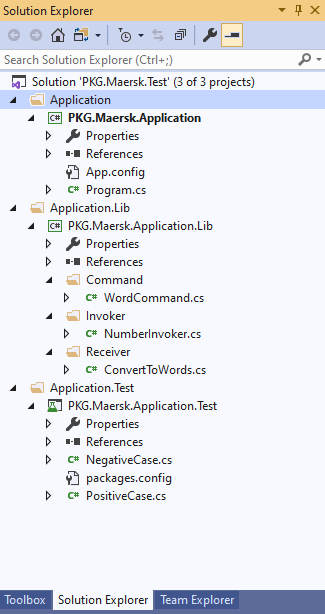
Test has written with three different project listing

* PKG.Maersk.Application– Console Application
* PKG.Maersk.Application.Lib – Class Library
* PKG.Maersk.Application.Test – Test Project

Please review below screen shot and project creation in Visual Studio



Maintain below folder structure



Class Library Project : **PKG.Maersk.Application.Lib – Class Library**

For This Project I have Create and follow with **Command Design Patter** because here as i know in development experience need to convert **one datatype (Integer)** to **another datatype(double).**  So I just give one command to specific input and it will reflect with expected output

Basically, Command Design Patter we have following node which need to maintain

* Command
* Invoker
* Receiver

**For Command review my below File.**

using PKG.Maersk.Application.Lib.Receiver;

namespace PKG.Maersk.Application.Lib.Command

{

#region Declare Interface

/// <summary>

/// ICommand Interface To Declare Executate Method

/// </summary>

public interface ICommand

{

/// <summary>

/// Define Execuate Method For Convert Number to Word

/// </summary>

/// <param name="numberInString">numberInString as string variable</param>

/// <returns>return string in Word</returns>

string Execute(double numberInString);

}

#endregion Declare Interface

#region Command Class Declaraction

/// <summary>

/// Decalre Word Command Class To Convert Number To Word

/// </summary>

public class WordCommand : ICommand

{

/// <summary>

/// ConvertToWord Receiver Object

/// </summary>

private ConvertToWords convertToWords;

/// <summary>

/// Constructor

/// </summary>

/// <param name="\_convertToWords">\_convertToWords as ConvertToWord Receiver Object</param>

public WordCommand(ConvertToWords \_convertToWords)

{

convertToWords = \_convertToWords;

}

/// <summary>

/// Executate Public Method and Get OutPut

/// </summary>

/// <param name="numberInString">numberInString as string variable</param>

/// <returns>return in Word</returns>

public string Execute(double numberInString)

{

return convertToWords.ConvertToWordsInformation(numberInString);

}

}

#endregion Command Class Declaraction

}

**For Invoker review my below File.**

using PKG.Maersk.Application.Lib.Command;

namespace PKG.Maersk.Application.Lib.Invoker

{

#region Invoker Class Declare

/// <summary>

/// Number Invoker Class Declaraction

/// </summary>

public class NumberInvoker

{

/// <summary>

/// Declare wordCommand class object using interface

/// </summary>

private ICommand wordCommand;

/// <summary>

/// Constructor For Invoker Class

/// </summary>

/// <param name="\_iCommand">\_iCommand as Interface reference</param>

public NumberInvoker(ICommand \_iCommand)

{

this.wordCommand = \_iCommand;

}

/// <summary>

/// Return string value

/// </summary>

/// <param name="numberInString">numberInString as input number</param>

/// <returns>return full string with Word</returns>

public string ReturnWord(double numberInString)

{

return wordCommand.Execute(numberInString);

}

}

#endregion Invoker Class Declare

}

**For Receiver review my below File.**

using System;

namespace PKG.Maersk.Application.Lib.Receiver

{

#region Receiver Class Declaraction

/// <summary>

/// Receiver Class Implementation

/// </summary>

public class ConvertToWords

{

#region Public Method Declaraction

public string ConvertToWordsInformation(double inputNumber)

{

string isNegative = string.Empty;

string returnWords = string.Empty;

try

{

string strinputNumber = Convert.ToString(inputNumber);

if (strinputNumber.Contains("-"))

{

isNegative = $"MINUS ";

inputNumber = Convert.ToDouble(strinputNumber.Substring(1, strinputNumber.Length - 1));

}

if (inputNumber == 0)

returnWords = $"ZERO ONLY";

else

returnWords = $"{isNegative}{ActualNumberString(inputNumber)}";

}

catch { }

return returnWords;

}

#endregion Public Method Declaraction

#region Private Method Declaraction

/// <summary>

/// Representation Actual String if it is Less than zero

/// </summary>

/// <param name="inputNumber"></param>

/// <returns></returns>

private string ActualNumberString(double inputNumber)

{

string stringOutPut = string.Empty;

stringOutPut += NumToWords((int)(inputNumber / 10000000), "CRORE ");

stringOutPut += NumToWords((int)((inputNumber / 100000) % 100), "LAKH ");

stringOutPut += NumToWords((int)((inputNumber / 1000) % 100), "THOUSAND ");

stringOutPut += NumToWords((int)((inputNumber / 100) % 10), "HUNDRED ");

if (inputNumber > 100 && inputNumber % 100 > 0) { stringOutPut += "AND "; }

stringOutPut += NumToWords((int)(inputNumber % 100), " ONLY");

return stringOutPut.Trim();

}

/// <summary>

/// Return Tens Number Value

/// </summary>

/// <param name="number">number as string variable</param>

/// <returns>return Tens Number value</returns>

private string Tens(string number)

{

int intNumber = Convert.ToInt32(number);

string returnName = string.Empty;

switch (intNumber)

{

case 10:

returnName = "TEN";

break;

case 11:

returnName = "ELEVEN";

break;

case 12:

returnName = "TWELVE";

break;

case 13:

returnName = "THIRTEEN";

break;

case 14:

returnName = "FOURTEEN";

break;

case 15:

returnName = "FIFTEEN";

break;

case 16:

returnName = "SIXTEEN";

break;

case 17:

returnName = "SEVENTEEN";

break;

case 18:

returnName = "EIGHTEEN";

break;

case 19:

returnName = "NINETEEN";

break;

case 2:

returnName = "TWENTY";

break;

case 3:

returnName = "THIRTY";

break;

case 4:

returnName = "FOURTY";

break;

case 5:

returnName = "FIFTY";

break;

case 6:

returnName = "SIXTY";

break;

case 7:

returnName = "SEVENTY";

break;

case 8:

returnName = "EIGHTY";

break;

case 9:

returnName = "NINETY";

break;

}

return returnName;

}

/// <summary>

/// return One To Nine Number Value

/// </summary>

/// <param name="number">number as string variable</param>

/// <returns>return one to nine number text</returns>

private string Ones(string number)

{

var intNumber = Convert.ToInt32(number);

string returnName = string.Empty;

switch (intNumber)

{

case 1:

returnName = "ONE"; break;

case 2:

returnName = "TWO"; break;

case 3:

returnName = "THREE"; break;

case 4:

returnName = "FOUR"; break;

case 5:

returnName = "FIVE"; break;

case 6:

returnName = "SIX"; break;

case 7:

returnName = "SEVEN"; break;

case 8:

returnName = "EIGHT"; break;

case 9:

returnName = "NINE"; break;

}

return returnName;

}

/// <summary>

/// Representation World in Data

/// </summary>

/// <param name="inputNumber">inputNumber as integer</param>

/// <param name="\_outputString">\_outputString as string</param>

/// <returns></returns>

private string NumToWords(int inputNumber, string \_outputString)

{

string outputString = string.Empty;

if (inputNumber > 19)

outputString += $"{Tens(Convert.ToString(inputNumber / 10))} {Ones(Convert.ToString(inputNumber % 10))} ";

else

outputString += $"{Ones(Convert.ToString(inputNumber))} ";

if (inputNumber != 0)

outputString += $"{ \_outputString} ";

return outputString;

}

#endregion Private Method Declaraction

}

#endregion Receiver Class Declaraction

}

Console Project : **PKG.Maersk.Application– Console Application**

For This Project I have add reference with Class Library Project **PKG.Maersk.Application** and create simple object of command to introduce **Constructor Dependency Injection**

**Here I have testing multiple output with application please review it as below code**

using PKG.Maersk.Application.Lib.Command;

using PKG.Maersk.Application.Lib.Receiver;

using System;

namespace PKG.Maersk.Application

{

#region Start Up Program

/// <summary>

/// Class Program To Start Executation

/// </summary>

class Program

{

#region Static Main Method

/// <summary>

/// Start Application From Here

/// </summary>

/// <param name="args">args as string array value</param>

static void Main(string[] args)

{

try

{ // Declare Command Class with Constructure Injuction

WordCommand wordCommand = new WordCommand(new ConvertToWords());

Console.WriteLine("-------------------Integer Value Conversion-------------------");

// Greater Than Zero Converstion Printing

Console.WriteLine(wordCommand.Execute(963753));

// Zero Value Printing

Console.WriteLine(wordCommand.Execute(0));

// Less Than Zero Converstion Printing

Console.WriteLine(wordCommand.Execute(-852741));

Console.WriteLine("-----------------End Integer Value Conversion-----------------");

}

catch (Exception e)

{

Console.WriteLine("Not a valid number");

}

Console.ReadLine();

}

#endregion Static Main Method

}

#endregion Start Up Program

}

Test Case Project : **PKG.Maersk.Application.Test– Testing Application**

For This Project I have add reference with Class Library Project **PKG.Maersk.Application** and create Positive and Negative both the test case.

**For Positive Test Case Please Review Below Screen**

using System;

using System.Linq;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PKG.Maersk.Application.Lib.Command;

using PKG.Maersk.Application.Lib.Receiver;

namespace PKG.Maersk.Application.Test

{

[TestClass]

public class PositiveCase

{

#region Default Declaraction

int? inputNumber = null;

string expected = string.Empty;

#endregion Default Declaraction

#region Test Initialize

/// <summary>

/// Test Initialize Default Value

/// </summary>

[TestInitialize]

public void PositiveCaseInit()

{

inputNumber = -852741;

expected = "MINUS NINE LAKH FIFTY TWO THOUSAND SEVEN HUNDRED AND FOURTY ONE ONLY";

}

#endregion Test Initalize

#region Test Cleanup

/// <summary>

/// Test Cleanup

/// </summary>

[TestCleanup]

public void PositiveCaseClean()

{

inputNumber = null;

expected = string.Empty;

}

#endregion Test Cleanup

#region Test Method

/// <summary>

/// Define test and executate

/// </summary>

[TestMethod]

public void PositiveCaseTest()

{

// Act

WordCommand wordCommand = new WordCommand(new ConvertToWords());

string actual = wordCommand.Execute(Convert.ToDouble(inputNumber));

// Assert

Assert.IsTrue(expected.ToUpper().SequenceEqual(actual.ToUpper()));

}

#endregion Test Method

}

}

**For Negative Test Case Please Review Below Screen**

using System;

using System.Linq;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using PKG.Maersk.Application.Lib.Command;

using PKG.Maersk.Application.Lib.Receiver;

namespace PKG.Maersk.Application.Test

{

[TestClass]

public class NegativeCase

{

#region Default Declaraction

int? inputNumber = null;

string expected = string.Empty;

#endregion Default Declaraction

#region Test Initialize

/// <summary>

/// Test Initialize Default Value

/// </summary>

[TestInitialize]

public void NegativeCaseInit()

{

inputNumber = 963753;

expected = "NINE LAKH SIXTY THREE THOUSAND SEVEN HUNDRED AND FIFTY THREE ONLY";

}

#endregion Test Initalize

#region Test Cleanup

/// <summary>

/// Test Cleanup

/// </summary>

[TestCleanup]

public void NegativeCaseClean()

{

inputNumber = null;

expected = string.Empty;

}

#endregion Test Cleanup

#region Test Method

/// <summary>

/// Define test and executate

/// </summary>

[TestMethod]

public void NegativeCaseTest()

{

// Act

WordCommand wordCommand = new WordCommand(new ConvertToWords());

string actual = wordCommand.Execute(Convert.ToDouble(inputNumber));

// Assert

Assert.IsTrue(expected.ToUpper().SequenceEqual(actual.ToUpper()));

}

#endregion Test Method

}

}