Lea Monica Belo

7638471

01/06/2017

Assignment 1

PROG8170: Software Quality Assurance Techniques

# Source Code

## Program.cs File

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PROG8170\_Assignment1

{

class Program

{

private static bool endProg = false;

private static double circRadius = 0;

private static Circle newCircle = new Circle();

static void Main(string[] args)

{

Console.Write("\nPlease enter value of Radius: ");

while (!double.TryParse(Console.ReadLine(), out circRadius) ||

!(circRadius >= 0))

{

Console.WriteLine("\nInvalid input. Radius must be a numerical value and greater than or equal to 0.");

Console.Write("\nPlease enter value of Radius: ");

}

newCircle.circleRadius = circRadius;

while (endProg != true)

{

RunMenu();

}

}

private static void RunMenu()

{

Console.Write("\nMenu:");

Console.Write("\n--------------------------------------------");

Console.Write("\n1 - Add to Circle Radius");

Console.Write("\n2 - Subtract from Circle Radius");

Console.Write("\n3 - Calculate Circle Circumference");

Console.Write("\n4 - Calculate Circle Area");

Console.Write("\n5 - Exit");

Console.Write("\n\nEnter Option: ");

int optionSelected = 0;

while (!int.TryParse(Console.ReadLine(), out optionSelected))

{

Console.Write("\nInvalid input.");

Console.Write("\n\nMenu:");

Console.Write("\n--------------------------------------------");

Console.Write("\n1 - Add to Circle Radius");

Console.Write("\n2 - Subtract from Circle Radius");

Console.Write("\n3 - Calculate Circle Circumference");

Console.Write("\n4 - Calculate Circle Area");

Console.Write("\n5 - Exit");

Console.Write("\n\nEnter Option: ");

}

RunOption(optionSelected);

Console.Write("\nPress any key to go back to main menu...");

Console.ReadKey();

Console.Clear();

}

private static void RunOption(int option)

{

switch(option)

{

case 1:

double addedValue = 0;

Console.Write("\nCurrent value of radius is: " + newCircle.circleRadius);

Console.Write("\nEnter value to be added to radius: ");

if (!double.TryParse(Console.ReadLine(), out addedValue) ||

!(addedValue >= 0))

{

Console.Write("\nInvalid input. Radius must be a numerical value and greater than 0.");

}

else

{

newCircle.AddToRadius(addedValue);

Console.Write("\nNew value of radius is: " + newCircle.circleRadius);

}

endProg = false;

break;

case 2:

double lessValue = 0;

Console.Write("\nCurrent value of radius is: " + newCircle.circleRadius);

Console.Write("\nEnter value to be deducted from radius: ");

if (!double.TryParse(Console.ReadLine(), out lessValue) ||

!(lessValue >= 0) || lessValue > newCircle.circleRadius)

{

if (lessValue > newCircle.circleRadius)

{

Console.Write("\nValue to be subtracted must not be greater than Radius value.");

}

else

{

Console.Write("\nInvalid input. Radius must be a numerical value and greater than 0.");

}

}

else

{

newCircle.SubtractFromRadius(lessValue);

Console.Write("\nNew value of radius is: " + newCircle.circleRadius);

}

endProg = false;

break;

case 3:

newCircle.GetCircumference();

Console.Write("\nCircumference of Circle is: " + newCircle.Circumference);

endProg = false;

break;

case 4:

newCircle.GetArea();

Console.Write("\nArea of Circle is " + newCircle.Area);

endProg = false;

break;

case 5: Environment.Exit(0);

endProg = true;

break;

default:

Console.Write("\nInvalid input.");

break;

}

}

}

}

## Circle.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace PROG8170\_Assignment1

{

public class Circle

{

public double circleRadius { get; set; }

public double Circumference { get; set; }

public double Area { get; set; }

public Circle()

{

}

public Circle(double circleRadius)

{

if (circleRadius < 0)

{

throw new System.ArgumentException("Value of radius must be greater than or equal to 0.");

}

else

{

this.circleRadius = circleRadius;

}

}

public void AddToRadius(double num)

{

if (num < 0)

{

throw new System.ArgumentException("Value to be added must be greater than or equal to 0.");

}

else

{

circleRadius += num;

}

}

public void SubtractFromRadius(double num)

{

if (circleRadius < num)

{

throw new System.ArgumentException("Value to be subtracted must not be greater than Radius value.");

}

else if (num < 0)

{

throw new System.ArgumentException("Value to be subtracted must be greater than or equal to 0.");

}

else

{

circleRadius -= num;

}

}

public void GetCircumference()

{

try

{

Circumference = 2 \* Math.PI \* circleRadius;

}

catch (Exception e)

{

throw new System.ArgumentException("Error on circumference computation.");

}

}

public void GetArea()

{

try

{

Area = Math.PI \* Math.Pow(circleRadius, 2);

}

catch (Exception e)

{

throw new System.ArgumentException("Error on area computation.");

}

}

}

}

## CircleTest.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assignment1\_Test

{

using NUnit.Framework;

using NUnit.Framework.Constraints;

using PROG8170\_Assignment1;

[TestFixture]

class CircleTest

{

[Test]

//Whole Number Test: Program should accept whole number value

[TestCase(5)]

//Zero Test: Program should accept 0 value

[TestCase(0)]

//Decimal Test: Program should accept long decimal value

[TestCase(0.000000000001)]

////Expected Failure Test: Program should reject negative value

//[TestCase(-1500)]

////Expected Failure Test: Program should reject string

//[TestCase("Test")]

public void Create\_Circle\_Test(double radiusvalue)

{

//Arrange & Act

PROG8170\_Assignment1.Circle Circle = new PROG8170\_Assignment1.Circle(radiusvalue);

//Assert

Assert.AreEqual(radiusvalue, Circle.circleRadius);

}

[Test]

//Whole Number Test: Program should accept whole number value

[TestCase(5)]

//Zero Test: Program should accept 0 value

[TestCase(0)]

//Decimal Test: Program should accept long decimal value

[TestCase(0.000000000001)]

////Expected Failure Test: Program should reject string

//[TestCase("Test")]

public void Create\_Circle\_Blank\_Constructor\_Test(double radiusvalue)

{

//Arrange & Act

PROG8170\_Assignment1.Circle Circle = new PROG8170\_Assignment1.Circle();

Circle.circleRadius = radiusvalue;

//Assert

Assert.AreEqual(radiusvalue, Circle.circleRadius);

}

#region AddToRadius

[Test]

//Radius set as 5.05 across all tests

//Normal Test: Radius to be added is set as 1.5 to check integrity of result by adding a double value

[TestCase(5.05, 1.5)]

//Long Decimal Value Test: Radius to be added is set as +0.000000000000000001 to check integrity of result emphasis on decimal

[TestCase(5.05, +0.000000000000000001)]

////Expected Failure Test: Radius to be added is set as negative to check if program returns ArgumentException error & not accept value

//[TestCase(5.05, -0.0000001)]

public void AddToRadius\_Positive\_Value\_Test(double initialRadius, double addRadius)

{

//Arrange

double expectedRadius = initialRadius + addRadius;

PROG8170\_Assignment1.Circle CircleTest1 = new PROG8170\_Assignment1.Circle(initialRadius);

//Act

CircleTest1.AddToRadius(addRadius);

//Assert

Assert.AreEqual(expectedRadius, CircleTest1.circleRadius);

}

[Test]

//Radius set as 5.05 across all tests

//Normal Test: Radius to be added is set as negative to check if program returns ArgumentException error & not accept value

[TestCase(5.05, -3.5)]

//Long Decimal Value Test: Radius to be added is set as -0.000000000000000001 to check even small negative value returns ArgumentException error & program does not accept the value

[TestCase(5.05, -0.000000000000000001)]

////Expected Failure Test: Radius to be added is set as -0 to check that program returns true even if given a negative sign before the value of 0

//[TestCase(5.05, -0)]

public void AddToRadius\_Negative\_Value\_Test\_ThrowsExcptn (double initialRadius, double addRadius)

{

//Arrange

double expectedRadius = initialRadius + addRadius;

PROG8170\_Assignment1.Circle CircleTest2 = new PROG8170\_Assignment1.Circle(initialRadius);

//Act & Assert

Assert.Throws<ArgumentException>(() => CircleTest2.AddToRadius(addRadius));

}

#endregion

#region SubtractFromRadius

[Test]

//Radius set as 5.05 across all tests

//Normal Test: Radius to be subtracted is set as 2.5 to check integrity of result by subtracting a double value

[TestCase(5.05, 2.5)]

//Zero Test: Radius to be subtracted is equal to zero to check if value of radius remains unaltered

[TestCase(5.05, 0)]

////Expected Failure Test: Radius to be subtracted is set as negative to check if program returns ArgumentException error & not accept value

//[TestCase(5.05, -1)]

public void SubtractFromRadius\_Positive\_Test(double initialRadius, double subtractRadius)

{

//Arrange

double expectedRadius = initialRadius - subtractRadius;

PROG8170\_Assignment1.Circle CircleTest3 = new PROG8170\_Assignment1.Circle(initialRadius);

//Act

CircleTest3.SubtractFromRadius(subtractRadius);

//Assert

Assert.AreEqual(expectedRadius, CircleTest3.circleRadius);

}

[Test]

//Radius set as 5.05 across all tests

//Normal Test: Radius to be subtracted is set as negative to check if program returns ArgumentException error & not accept value

[TestCase(5.05, -3.5)]

//Long Decimal Value Test: Radius to be subtracted is set as -0.0000000001 to check even small negative value returns ArgumentException error & program does not accept the value

[TestCase(5.05, -0.0000000001)]

////Expected Failure Test: Radius to be subtracted is set as -0 to check that program returns true even if given a negative sign before the value of 0

//[TestCase(5.05, -0)]

public void SubtractFromRadius\_Negative\_Value\_Test\_ThrowsExcptn(double initialRadius, double subtractRadius)

{

//Arrange

PROG8170\_Assignment1.Circle CircleTest4 = new PROG8170\_Assignment1.Circle(initialRadius);

//Act & Assert

Assert.Throws<ArgumentException>(() => CircleTest4.SubtractFromRadius(subtractRadius));

}

[Test]

//Radius set as 5.05 across all tests

//Normal Test: Radius to be subtracted is set as 21 (higher than default radius) to check if program returns ArgumentException error & not accept value

[TestCase(5.05, 21)]

//Long Decimal Value Test: Radius to be subtracted is set as 5.05000000001 (higher than default radius) to check even small value difference returns ArgumentException error & program does not accept the value

[TestCase(5.05, 5.05000000001)]

////Expected Failure Test: Radius to be subtracted is set as 4.04999999999 to check that program returns true even if value is smaller and only has small difference from radius

//[TestCase(5.05, 4.04999999999)]

public void SubtractFromRadius\_Greater\_Value\_Test\_ThrowsExcptn (double initialRadius, double subtractRadius)

{

//Arrange

PROG8170\_Assignment1.Circle CircleTest5 = new PROG8170\_Assignment1.Circle(initialRadius);

//Act & Assert

Assert.Throws<ArgumentException>(() => CircleTest5.SubtractFromRadius(subtractRadius));

}

#endregion

#region GetCircumference

[Test]

//Zero value for Radius Test: Program should accept 0

[TestCase(0)]

//Positive value for Radius Test: Program should accept value

[TestCase(1.25125)]

////Expected Failure Test: Negative value for Radius - Program should throw Exception of value must be equal or greater than 0

//[TestCase(-1.25)]

public void GetCircumference\_Test(double radius)

{

//Arrange

double expectedCircumference = 2 \* Math.PI \* radius;

PROG8170\_Assignment1.Circle Circle = new Circle(radius);

//Act

Circle.GetCircumference();

//Assert

Assert.AreEqual(expectedCircumference, Circle.Circumference);

}

[Test]

//Null value test: Program will proceed to set radius to 0 even after given null

[TestCase(null)]

public void GetCircumference\_Null\_Value\_Test(double radius)

{

//Arrange

double expectedCircumference = radius;

PROG8170\_Assignment1.Circle Circle = new Circle(radius);

//Act

Circle.GetCircumference();

//Assert

Assert.AreEqual(expectedCircumference, Circle.Circumference);

}

#endregion

#region GetArea

[Test]

//Zero value for Radius Test: Program should accept 0

[TestCase(0)]

//Positive value for Radius Test: Program should accept value

[TestCase(1.25125)]

////Expected Failure Test: Negative value for Radius - Program should throw Exception of value must be equal or greater than 0

//[TestCase(-1.25)]

public void GetArea\_Test(double radius)

{

//Arrange

double expectedArea = Math.PI \* Math.Pow(radius, 2);

PROG8170\_Assignment1.Circle Circle = new Circle(radius);

//Act

Circle.GetArea();

//Assert

Assert.AreEqual(expectedArea, Circle.Area);

}

[Test]

//Null value test: Program will proceed to set radius to 0 even after given null

[TestCase(null)]

public void GetArea\_Null\_Value\_Test(double radius)

{

//Arrange

double expectedArea = radius;

PROG8170\_Assignment1.Circle Circle = new Circle(radius);

//Act

Circle.GetArea();

//Assert

Assert.AreEqual(expectedArea, Circle.Area);

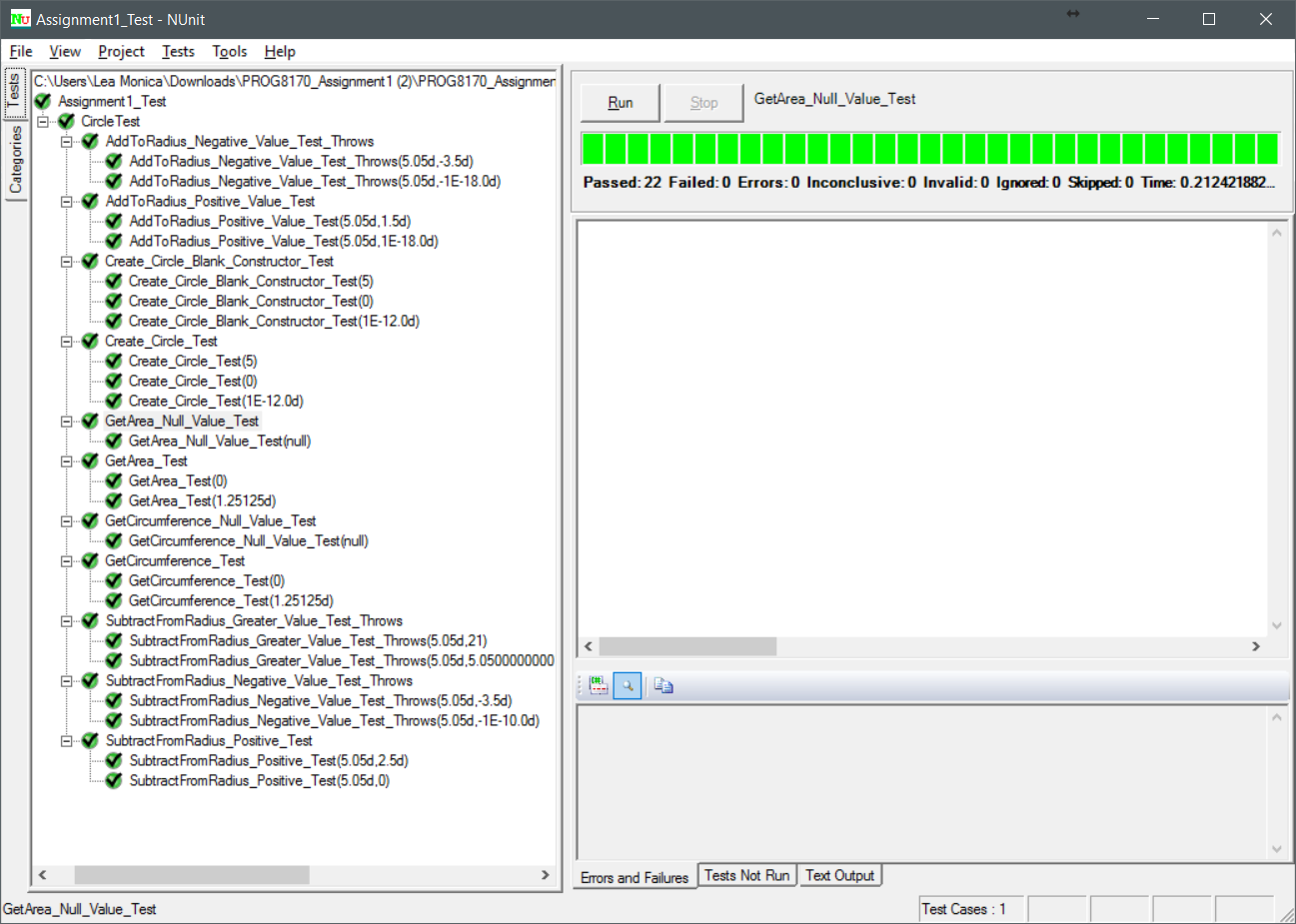
}

#endregion

}

}

# Unit Test



|  |  |  |
| --- | --- | --- |
| **Test Case** | | **Explanation** |
|  | | |
| Create Circle Test | | |
|  | [TestCase(5)] | Whole Number Test: to check that program accepts a whole number |
| [TestCase(0)] | Zero Test: to check that program accepts the value 0 |
| [TestCase(0.000000000001)] | Decimal Test: to check that program accepts a decimal value |
|  | | |
| Create Circle Test – Blank Constructor | | |
|  | [TestCase(5)] | Whole Number Test: to check that program accepts a whole number |
| [TestCase(0)] | Zero Test: to check that program accepts the value 0 |
| [TestCase(0.000000000001)] | Decimal Test: to check that program accepts a decimal value |
|  | | |
| Add to Radius: Positive Value Test | | |
|  | Radius: 5.05 | Radius set as 5.05 across all tests |
| [TestCase(5.05, 1.5)] | Normal Test: Radius to be added is set as 1.5 to check integrity of result by adding a double value |
| [TestCase(5.05, +0.000000000000000001)] | Long Decimal Value Test: Radius to be added is set as +0.000000000000000001 to check integrity of result emphasis on decimal |
|  | | |
| Add to Radius: Negative Value Test Throws Exception | | |
|  | Radius: 5.05 | Radius set as 5.05 across all tests |
| [TestCase(5.05, -3.5)] | Normal Test: Radius to be added is set as negative to check if program returns ArgumentException error & not accept value |
| [TestCase(5.05,  -0.000000000000000001)] | Long Decimal Value Test: Radius to be added is set as -0.000000000000000001 to check even small negative value returns ArgumentException error & program does not accept the value |
|  | | |
| Subtract from Radius: Positive Value Test | | |
|  | Radius: 5.05 | Radius set as 5.05 across all tests |
| [TestCase(5.05, 2.5)] | Normal Test: Radius to be subtracted is set as 2.5 to check integrity of result by subtracting a double value |
| [TestCase(5.05, 0)] | Zero Test: Radius to be subtracted is equal to zero to check if value of radius remains unaltered |
|  | | |
| Subtract from Radius: Negative Value Test Throws Exception | | |
|  | Radius: 5.05 | Radius set as 5.05 across all tests |
| [TestCase(5.05, -3.5)] | Normal Test: Radius to be subtracted is set as negative to check if program returns ArgumentException error & not accept value |
| [TestCase(5.05, -0.0000000001)] | Long Decimal Value Test: Radius to be subtracted is set as -0.0000000001 to check even small negative value returns ArgumentException error & program does not accept the value |
|  | | |
| Subtract from Radius: Greater Value Test Throws Exception | | |
|  | Radius: 5.05 | Radius set as 5.05 across all tests |
| [TestCase(5.05, 21)] | Normal Test: Radius to be subtracted is set as 21 (higher than default radius) to check if program returns ArgumentException error & not accept value |
| [TestCase(5.05, 5.05000000001)] | Long Decimal Value Test: Radius to be subtracted is set as 5.05000000001 (higher than default radius) to check even small value difference returns ArgumentException error & program does not accept the value |
|  | | |
| Get Circumference Test | | |
|  | [TestCase(0)] | Zero value for Radius Test: to check that program accepts 0 and returns calculated Circumference as 0 |
| [TestCase(1.25125)] | Positive value for Radius Test: to check that program accepts a double value and check integrity of calculated value vs Circle.Circumference |
|  | | |
| Get Circumference: Null Value Test | | |
|  | [TestCase(null)] | Null value test: to check that program will proceed to set radius to 0 even after given null |
|  | | |
| Get Area Test | | |
|  | [TestCase(0)] | Zero value for Radius Test: to check that program accepts 0 and returns calculated Area as 0 |
| [TestCase(1.25125)] | Positive value for Radius Test: to check that program accepts a double value and check integrity of calculated value vs Circle.Area |
|  | | |
| Get Area: Null Value Test | | |
|  | [TestCase(null)] | Null value test: to check that program will proceed to set radius to 0 even after given null |
|  | | |
|  |  |  |
|  |  |  |
|  |  |  |