**Danh Quach – 7830615**

**Assignment#1**

**Feb 01, 2018**

Copy of Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

/// <summary>

/// Assigment01 by Danh Quach Thanh - 7830615

/// </summary>

namespace Assigment01

{

class Program

{

static void Main(string[] args)

{

Program p = new Program();

p.Go();

Console.ReadKey();

}

private void Go()

{

Circle c = new Circle();

string radiusString = string.Empty;

double circumference = 0;

double area = 0;

do

{

Console.Write("Enter radius (>=0): ");

radiusString = Console.ReadLine();

} while (!double.TryParse(radiusString, out c.radius) || c.radius < 0);

string selectionString = string.Empty;

int selection = 0;

do

{

Console.WriteLine("1. Add to Circle Radius");

Console.WriteLine("2. Substract from Circle Radius");

Console.WriteLine("3. Calculate Circle Circumference");

Console.WriteLine("4. Calculate Circle Area");

Console.WriteLine("5. Exit");

Console.Write("Enter choice: ");

selectionString = Console.ReadLine();

} while (!int.TryParse(selectionString, out selection) || selection < 1 || selection > 5);

double calculateNumber = 0;

switch(selection)

{

case 1:

calculateNumber = ValidateNumberInput();

c.AddToRadius(calculateNumber);

Console.Write("New Radius: {0}", c.radius);

break;

case 2:

calculateNumber = ValidateNumberInput();

c.SubsctractFromRadius(calculateNumber);

Console.Write("New Radius: {0}", c.radius);

break;

case 3:

circumference = c.GetCircumference();

Console.Write("Cirlce Circumference: {0}", circumference);

break;

case 4:

area = c.GetArea();

Console.Write("Circle Area: {0}", area);

break;

case 5:

Environment.Exit(0);

break;

default:

break;

}

}

private double ValidateNumberInput()

{

string numberString = string.Empty;

double number = 0;

do

{

Console.Write("Enter a number: ");

numberString = Console.ReadLine();

} while (!double.TryParse(numberString, out number));

return number;

}

}

}

Copy of Circle.cs file

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Assigment01

{

public class Circle

{

public double radius;

public Circle()

{

}

public Circle(double radius)

{

this.radius = radius;

}

public void AddToRadius(double num)

{

double total = radius + num;

if (total < 0)

{

Console.WriteLine("Radius cannot goes below zero");

}

else

radius = total;

}

public void SubsctractFromRadius(double num)

{

double total = radius - num;

if (total < 0)

{

Console.WriteLine("Radius cannot goes below zero");

}

else

radius = total;

}

public double GetCircumference()

{

if (radius < 0) return 0;

else

{

var c = 2 \* radius \* Math.PI;

return c;

}

}

public double GetArea()

{

if (radius < 0) return 0;

else

{

var a = radius \* radius \* Math.PI;

return a;

}

}

}

}

Copy of CircleTest.cs file

using NUnit.Framework;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using Assigment01;

namespace UnitTest\_Assignment01

{

[TestFixture]

public class CircleTest

{

Circle c;

[SetUp]

public void Init()

{

c = new Circle();

}

[Test]

public void InstantiateWithRadius()

{

c = new Circle(5.2);

Assert.AreEqual(5.2, c.radius);

}

[Test]

public void AddToRadiusWithIntegerNumber()

{

c.radius = 5.4;

int number = 6;

double total = 5.4 + 6;

c.AddToRadius(number);

Assert.AreEqual(total, c.radius);

}

[Test]

public void AddToRadiusWithNegativeDoubleNumberSmallerThanRadius()

{

c.radius = 6.5;

double number = -5.56;

double total = 6.5 - 5.56;

c.AddToRadius(number);

Assert.AreEqual(total, c.radius);

}

[Test]

public void SubstractFromRadiusWithIntNumberSmallerThanRadius()

{

c.radius = 12.64;

int number = 6;

double total = 12.64 - 6;

c.SubsctractFromRadius(number);

Assert.AreEqual(total, c.radius);

}

[Test]

public void SubstractFromRadiusWithDoubleNumberLargerThanRadius()

{

c.radius = 7.64;

double number = 8.5;

double total = 7.64 - 8.5;

c.SubsctractFromRadius(number);

Assert.AreNotEqual(total, c.radius);

}

[Test]

public void CalculateCircumferenceWithPositiveRadius()

{

c.radius = 5.7;

double c1 = 2\* 5.7 \* Math.PI;

double c2 = c.GetCircumference();

Assert.AreEqual(c1, c2);

}

[Test]

public void CalculateCircumferenceWithNegativeRadius()

{

c.radius = -5.7;

double c1 = 2 \* -5.7 \* Math.PI;

double c2 = c.GetCircumference();

Assert.AreNotEqual(c1, c2);

}

[Test]

public void CalculateAreaWithPositiveRadius()

{

c.radius = 5.7;

double c1 = 5.7 \* 5.7 \* Math.PI;

double c2 = c.GetArea();

Assert.AreEqual(c1, c2);

}

[Test]

public void CalculateAreaWithInvalidRadius()

{

c.radius = -5.7;

double c1 = -5.7 \* -5.7 \* Math.PI;

double c2 = c.GetArea();

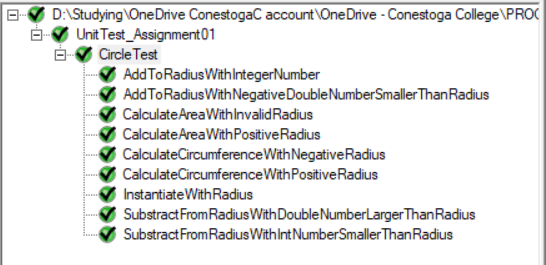
Assert.AreNotEqual(c1, c2);

}

}

}

Test Result:



Explanation for chosen Unit Testcases:

1. InstatiateWithRadius: this case check if the instantiated radius is same as expected
2. AddToRadiusWithIntegerNumber: this case will check the main flow of the method, verify the method work correctly in normal scenario