МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №1 по дисциплине «Тестирование программного обеспечения»

**Разработка через тестирование**

|  |  |
| --- | --- |
| Выполнил: студент группы ФИб-4302-51-00 | / Д.А. Савин / |
| Проверил: ст. преподаватель каф. ПМиИ | / И.Н. Фищева / |

Киров 2022

Задание

Используя методологию TDD разработать класс, который вычисляет расстояние и азимут между двумя точками на Земном шаре. Последовательность цикла "red/green/refactor" не удаляем, а помещаем в комментарий. Это понадобится для отчета.

Азимут – это угол, между направлением на географический северный полюс (не магнитный) и направлением из точки 1 в точку 2.

Входными данными являются долгота и широта 1 и 2 точки.

Координаты (широта от −90° до +90°, долгота от −180° до +180°) записываются в ° градусах в виде десятичной дроби.

Положительные знаки координат – это северная широта и восточная долгота. Отрицательные знаки координат представляются либо знаком «−», южная широта, западная долгота.

Выходными данными будет азимут в ° градусах в виде десятичной дроби и расстояние в километрах.

RED/GREEN/REFACTOR

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Изображение выглядит как текст

Автоматически созданное описание

Комментарии

1. Пусть наш класс выдает допустимые значения: от 0(включая) до 360(не включая), none - для недопустимого азимута (завести перечисление, в котором будет элемент none = -1), any - для любого азимута (в имеющееся перечисление, добавить any = -2).

* Класс не может найти недопустимый азимут.

1. Для точек, лежащих на прямой, проходящей через центр Земли азимут = any

|  |
| --- |
| [TestMethod]  public void TestMethod1() //Для точек, лежащих на прямой, проходящей через центр Земли азимут = any  {  Point point\_1 = new Point(90, 0);  Point point\_2 = new Point(-90, 0);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("any", az);  } |

1. Для совпадающих точек азимут = none

|  |
| --- |
| [TestMethod]  public void TestMethod2() //Для совпадающих точек азимут = none  {  Point point\_1 = new Point(80, 40);  Point point\_2 = new Point(80, 40);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("none", az);  } |

1. Для одной точки на северном полюсе азимут = 180

|  |
| --- |
| [TestMethod]  public void TestMethod3() //Для одной точки на северном полюсе азимут = 180  {  Point point\_1 = new Point(90, 0);  Point point\_2 = new Point(80, 40);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("180", az);  } |

1. Для двух точек на одном полюсе азимут = none

|  |
| --- |
| [TestMethod]  public void TestMethod4() //Для двух точек на одном полюсе азимут = none  {  Point point\_1 = new Point(90, 0);  Point point\_2 = new Point(90, 0);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("none", az);  } |

1. Для точек на противоположных полюса (см 2) азимут = any

|  |
| --- |
| [TestMethod]  public void TestMethod5() //Для точек на противоположных полюса (см 2) азимут = any  {  Point point\_1 = new Point(90, 0);  Point point\_2 = new Point(90, 0);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("none", az);  } |

1. Для одной точки на южном полюсе азимут = 180

|  |
| --- |
| [TestMethod]  public void TestMethod6() //Для одной точки на южном полюсе азимут = 180  {  Point point\_1 = new Point(-90, 0);  Point point\_2 = new Point(15, 0);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("180", az);  } |

1. Для точек на экваторе 90 или 270, смотря куда ближе

|  |
| --- |
| public void TestMethod7() //Для точек на экваторе 90 или 270, смотря куда ближе  {  Point point\_1 = new Point(0, 120);  Point point\_2 = new Point(0, 50);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("270", az);  } |

1. При расчете расстояния между 2 произвольными точками расстояние должно быть меньше Pi\*r, где r - радиус Земли. Здесь хорошо бы проверить точки слева и справа от 180-го меридиана

|  |
| --- |
| [TestMethod]  public void TestMethod8() //При расчете расстояния между 2 произвольными точками расстояние должно быть меньше Pi\*r,  //где r - радиус Земли. Здесь хорошо бы проверить точки слева и справа от 180-го меридиана  {  Point point\_1 = new Point(0, 120);  Point point\_2 = new Point(0, -160);  var dis = point\_1.Distance(point\_2);  //Assert.AreEqual("270", az);  } |

1. Для точек на одном меридиане 0("идем на север") или 180("идем на юг")

|  |
| --- |
| [TestMethod]  public void TestMethod9() //Для точек на одном меридиане 0("идем на север") или 180("идем на юг")  {  Point point\_1 = new Point(23, 50);  Point point\_2 = new Point(15, 50);  string az = point\_1.Azimuth(point\_2);  Assert.AreEqual("0 (идем на север)", az);  } |

Изображение выглядит как текст, электроника

Автоматически созданное описание

**Вывод**

В ходе лабораторной работы разработка класса Point велась по методологии TDD, в процессе разработки выяснилось, что тесты нужно писать параллельно с написанием класса, преимуществом данного метода является отсутствие необходимости тестировать класс после его реализации.

# Приложение

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using ClassLibrary;

namespace zadanie\_1

{

[TestClass]

public class UnitTest1

{

[TestMethod]

public void TestMethod1() //Для точек, лежащих на прямой, проходящей через центр Земли азимут = any

{

Point point\_1 = new Point(90, 0);

Point point\_2 = new Point(-90, 0);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("any", az);

}

[TestMethod]

public void TestMethod2() //Для совпадающих точек азимут = none

{

Point point\_1 = new Point(80, 40);

Point point\_2 = new Point(80, 40);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("none", az);

}

[TestMethod]

public void TestMethod3() //Для одной точки на северном полюсе азимут = 180

{

Point point\_1 = new Point(90, 0);

Point point\_2 = new Point(80, 40);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("180", az);

}

[TestMethod]

public void TestMethod4() //Для двух точек на одном полюсе азимут = none

{

Point point\_1 = new Point(90, 0);

Point point\_2 = new Point(90, 0);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("none", az);

}

[TestMethod]

public void TestMethod5() //Для точек на противоположных полюса (см 2) азимут = any

{

Point point\_1 = new Point(90, 0);

Point point\_2 = new Point(90, 0);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("none", az);

}

[TestMethod]

public void TestMethod6() //Для одной точки на южном полюсе азимут = 180

{

Point point\_1 = new Point(-90, 0);

Point point\_2 = new Point(15, 0);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("180", az);

}

[TestMethod]

public void TestMethod7() //Для точек на экваторе 90 или 270, смотря куда ближе

{

Point point\_1 = new Point(0, 120);

Point point\_2 = new Point(0, 50);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("270", az);

}

[TestMethod]

public void TestMethod8() //При расчете расстояния между 2 произвольными точками расстояние должно быть меньше Pi\*r,

//где r - радиус Земли. Здесь хорошо бы проверить точки слева и справа от 180-го меридиана

{

Point point\_1 = new Point(0, 120);

Point point\_2 = new Point(0, -160);

var dis = point\_1.Distance(point\_2);

//Assert.AreEqual("270", az);

}

[TestMethod]

public void TestMethod9() //Для точек на одном меридиане 0("идем на север") или 180("идем на юг")

{

Point point\_1 = new Point(23, 50);

Point point\_2 = new Point(15, 50);

string az = point\_1.Azimuth(point\_2);

Assert.AreEqual("0 (идем на север)", az);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace ClassLibrary

{

public class Point

{

double \_latitude = -1, \_longitude = -1;

public double Latitude //wirota

{

get => \_latitude;

set

{

if (value >= -180 && value <= 180)

\_longitude = value;

}

}

public double Longitude //dolgota

{

get => \_longitude;

set

{

if (value >= -90 && value <= 90)

\_longitude = value;

}

}

public double Distance(Point point)

{

double distance;

try

{

double R = 6371; double rad = Math.PI / 180;

double delta\_PHI = point.Latitude - \_latitude; double delta\_Lambda = point.Longitude - \_longitude;

double a = Math.Pow(Math.Sin(delta\_PHI / 2.0 \* rad), 2) + Math.Cos(\_latitude \* rad) \* Math.Cos(point.Latitude \* rad) \* Math.Pow(Math.Sin(delta\_Lambda / 2.0 \* rad), 2);

double c = 2 \* Math.Atan2(Math.Sqrt(a), Math.Sqrt(1 - a));

distance = R \* c;

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return 0;

}

return distance;

}

public string Azimuth(Point point)

{

double azimuth;

try

{

if (point.Longitude == \_longitude && \_latitude == point.Latitude)

{

return "none";

}

if (((\_latitude + point.Latitude == 0) && (\_longitude + point.Longitude == 180)) || (\_latitude == -1 \* point.Latitude && \_latitude != 0))

{

return "any";

}

if ((\_latitude == 90) || (point.\_latitude == 90) || (\_latitude == -90) || (point.\_latitude == -90))

{

return 180.ToString();

}

if (\_latitude == 0 && point.Latitude == 0)

{

if (\_longitude > point.Longitude)

{

return 270.ToString();

}

else

{

return 90.ToString();

}

}

if (\_longitude == point.Longitude)

{

if (\_latitude > point.Latitude)

{

return 0.ToString() + " (идем на север)";

}

else

{

return 180.ToString() + " (идем на юг)";

}

}

double rad = Math.PI / 180;

double delta\_Lambda = point.Longitude - \_longitude;

azimuth = Math.Atan2(Math.Sin(delta\_Lambda \* rad) \* Math.Cos(point.Latitude \* rad),

Math.Cos(\_latitude \* rad) \* Math.Sin(point.Latitude \* rad) - Math.Sin(\_latitude \* rad) \* Math.Cos(point.Latitude \* rad) \* Math.Cos(delta\_Lambda \* rad));

if (azimuth < 0 || azimuth > 360)

{

return "none";

}

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

return 0.ToString();

}

return azimuth.ToString();

}

public Point(double latitude, double longitude)

{

try

{

if (latitude >= -90 && latitude <= 90) this.\_latitude = latitude;

else

{

throw new ArgumentException(latitude + " - Incorrect value.");

}

if (longitude >= -180 && longitude <= 180) this.\_longitude = longitude;

else

{

throw new ArgumentException(longitude + " - Incorrect value.");

}

}

catch (Exception ex)

{

throw new Exception(ex.Message);

}

}

}

}