**Калькулятор**

**(Лидовской и Меринов)**

**Цель:**

1. Научиться пользоваться исключениями
2. Применять исключения в коде
3. Получить хорошую оценку
4. Получить знание о тестах

**Задачи:**

* Написать калькулятор на котором мы будем учиться тестировать код.
* Научиться правильно проверять свою работу

**Теория:**

* Знание языка (в данном случае C#)
* Знать, как работают исключения
* Умение проводить тест
  + - использованы оператор if, методы Any, Assert, Intersect, Char.Upper, Char.Lower, Char.Digit

CalcChecker.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalculatorMerinovLidovskoi

{

public class CalcChecker

{

public static int summ(int nums, int num2)

{

return nums+num2;

}

public static int razn(int nums3, int num4)

{

return nums3 - num4;

}

public static int proizv(int num11, int num22)

{

return num11 \* num22;

}

public static float delenie(float num12, float num23)

{

string result = "";

try

{

return num12 / num23;

}

catch(DivideByZeroException)

{

result = "Divide by zero";

}

return num12 / num23;

}

public static double sqrt(double a)

{

return Math.Sqrt(a);

}

public static double square(double a)

{

return Math.Pow(a, 2.0);

}

public static double factorial(double a)

{

double f = 1;

for (int i = 1; i <= a; i++)

f \*= (double)i;

return f;

}

}

}

CalcCheckerTests

using Microsoft.VisualStudio.TestTools.UnitTesting;

using CalculatorMerinovLidovskoi;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace CalculatorMerinovLidovskoi.Tests

{

[TestClass()]

public class CalcCheckerTests

{

[TestMethod()]

public void Check\_Summ\_ReturnsTrue()

{

int nums = 6;

int num1 = 2;

int expected = 8;

//Act

int actual = CalcChecker.summ(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Summ\_ReturnsFalse()

{

int nums = 4;

int num1 = 2;

int expected = 6;

//Act

int actual = CalcChecker.summ(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Razn\_ReturnsTrue()

{

int nums = 2;

int num1 = 2;

int expected = 0;

//Act

int actual = CalcChecker.razn(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Razn\_ReturnsFalse()

{

int nums = 2;

int num1 = 3;

int expected = -1;

//Act

int actual = CalcChecker.razn(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_proizv\_ReturnsTrue()

{

int nums = 2;

int num1 = 2;

int expected = 4;

//Act

int actual = CalcChecker.proizv(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_proizv\_ReturnsFalse()

{

int nums = 2;

int num1 = 0;

int expected = 0;

//Act

int actual = CalcChecker.proizv(nums, num1);

//Asset

Assert.AreEqual(expected,actual);

}

[TestMethod()]

public void Check\_Divide\_ReturnsTrue()

{

float nums = 4;

float num1 = 2;

float expected = 2;

//Act

float actual = CalcChecker.delenie(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Divide\_ReturnsFalse()

{

float nums = 4;

float num1 = 0;

float expected = float.PositiveInfinity;

//Act

float actual = CalcChecker.delenie(nums, num1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_KvKoren\_ReturnsTrue()

{

double sqrt1 = 144;

double expected = 12;

//Act

double actual = CalcChecker.sqrt(sqrt1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_KvKoren\_ReturnsFalse()

{

double sqrt1 = 144;

double expected = 12;

//Act

double actual = CalcChecker.sqrt(sqrt1);

//Asset

Assert.AreEqual(expected,actual);

}

[TestMethod()]

public void Check\_Kvadrat\_ReturnsTrue()

{

double sqrt1 = 2;

double expected = 4;

//Act

double actual = CalcChecker.square(sqrt1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Kvadrat\_ReturnsFalse()

{

double sqrt1 = 2;

double expected = 4;

//Act

double actual = CalcChecker.square(sqrt1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Factorial\_ReturnsTrue()

{

double sqrt1 = 2;

double expected = 2.0;

//Act

double actual = CalcChecker.factorial(sqrt1);

//Asset

Assert.AreEqual(expected, actual);

}

[TestMethod()]

public void Check\_Factorial\_ReturnsFalse()

{

double sqrt1 = 2;

double expected = 2.0;

//Act

double actual = CalcChecker.factorial(sqrt1);

//Asset

Assert.AreEqual(expected, actual);

}

}

}