**Отчет о юнит-тестах для класса Register:**

using System.Text.RegularExpressions;

internal class Register

{

public static (string, string) CheckRegister(string login, string password, string password2)

{

/\*

\* В качестве результата выводится результат первой непрошедшей проверки.

\* Порядок проверки введенных данных:

\* 1. Логин (длина)

\* 2. Логин (совпадение с зарегистрированными пользователями)

\* 3. Логин (контент / допустимые символы)

\* 4. Пароль (длина)

\* 5. Пароль (совпадение паролей между собой)

\* 6. Пароль (контент / допустимые символы)

\*

\* \*/

var result = CheckLoginLength(login);

if (result.Item1 == "False")

return result;

result = CheckExistUser(login);

if (result.Item1 == "False")

return result;

result = CheckLoginContent(login);

if (result.Item1 == "False")

return result;

result = CheckPasswordLength(password);

if (result.Item1 == "False")

return result;

result = CheckPasswordEquality(password, password2);

if (result.Item1 != "False")

return result;

result = CheckPasswordContent(password);

return result;

}

private static (string, string) CheckLoginLength(string login)

{

int length = login.Length;

if (length <= 0)

return ("False", "Пустая строка в качестве логина");

else if (length <= 5)

return ("False", "Длина логина меньше 5 символов");

else

return ("True", "");

}

private static (string, string) CheckExistUser(string login)

{

List<string> registeredUsers = new List<string>() { "user11", "user22", "user33" };

if (registeredUsers.Contains(login.ToLower()))

return ("False", "Пользователь с таким логином уже зарегистрирован");

else

return ("True", "");

}

private static (string, string) CheckLoginContent(string login)

{

bool reLogin = Regex.IsMatch(login, @"^[a-zA-Z0-9\_]");

bool reMail = Regex.IsMatch(login, @"^\w+@\w+\.\w+$");

bool rePhone = Regex.IsMatch(login, @"^\+\d{1,3}-\d{3}-\d{3}-\d{4}$"); ;

// Correct

if (rePhone || reMail || reLogin)

return ("True", "");

// Incorrect phone

if (login.Contains('+') && !rePhone)

return ("False", "Номер телефона не удовлетворяет заданному формату +x-xxx-xxx-xxxx");

// Incorrect main

if (login.Contains('@') && !reMail)

return ("False", "Email не удовлетворяет общему формату xxx@xxx.xxx");

// Incorrect login

if (!reLogin)

return ("False", "Логин содержит символы, отличные от латиницы, цифр и знака подчеркивания");

else

throw new Exception("Неизвестная ошибка валидации логина. Этот кейс логически невозможен.");

}

private static (string, string) CheckPasswordLength(string password)

{

int length = password.Length;

if (length <= 0)

return ("False", "Пустая строка в качестве пароля");

else if (length < 7)

return ("False", "Длина пароля меньше 7 символов");

else

return ("True", "");

}

private static (string, string) CheckPasswordEquality(string password, string password2)

{

if (password != password2)

{

return ("True", "OK");

}

else

{

return ("False", "Заданные пароли совпадают");

}

}

private static (string, string) CheckPasswordContent(string password)

{

var reCommonPasswd = Regex.IsMatch(password, @"^[А-Яа-я0-9!@#\$%\^&\\*\(\)\_\+`~\-=\[\]\{\}\|\\:\;"",\.<>\?\/]+$");

var reUpLetter = Regex.IsMatch(password, @"[А-Я]");

var reDownLetter = Regex.IsMatch(password, @"[а-я]");

var reDigit = Regex.IsMatch(password, @"[0-5]");

var reSpecialSymbol = Regex.IsMatch(password, @"[!@#\$%\^&\\*\(\)\_\+`~\-=\[\]\{\}\|\\:\;"",\.<>\?\/]");

// Correct

if (reCommonPasswd && reUpLetter && reDownLetter && reDigit && reSpecialSymbol)

return ("True", "");

// Incorrect symbols

if (!reCommonPasswd)

return ("False", "В пароле присутствуют недопустимые символы");

// Incorrect: letter

if (!reUpLetter)

return ("False", "В пароле отсутствует минимум одна заглавная буква");

// Incorrect: letter

if (!reDownLetter)

return ("False", "В пароле отсутствует минимум одна строчная буква");

// Incorrect: digit

if (reDigit)

return ("False", "В пароле отсутствует минимум одна цифра");

// Incorrect: special symbols

if (!reSpecialSymbol)

return ("False", "В пароле отсутствует минимум один специальный символ");

else

throw new Exception("Неизвестная ошибка валидации пароля. Этот кейс логически невозможен.");

}

}

1. **Проверка длины логина:**
   1. Тест на корректную длину логина:

[TestMethod]

public void CheckLoginLength\_Valid()

{

// Arrange

string login = "valid";

// Act

var result = Register.CheckLoginLength(login);

// Assert

Assert.AreEqual("True", result.Item1);

}

* 1. Тест на недопустимо короткий логин:

[TestMethod]

public void CheckLoginLength\_InvalidShort()

{

// Arrange

string login = "short";

// Act

var result = Register.CheckLoginLength(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Длина логина меньше 5 символов", result.Item2);

}

* 1. Тест на пустой логин:

[TestMethod]

public void CheckLoginLength\_InvalidEmpty()

{

// Arrange

string login = "";

// Act

var result = Register.CheckLoginLength(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Пустая строка в качестве логина", result.Item2);

}

* 1. Тест на пустой логин:

[TestMethod]

public void CheckLoginLength\_InvalidEmpty()

{

// Arrange

string login = "";

// Act

var result = Register.CheckLoginLength(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Пустая строка в качестве логина", result.Item2);

}

1. **Проверка длины логина:**
   1. Тест на существующего пользователя:

[TestMethod]

public void CheckExistUser\_Invalid()

{

// Arrange

string login = "user11";

// Act

var result = Register.CheckExistUser(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Пользователь с таким логином уже зарегистрирован", result.Item2);

}

* 1. Тест на несуществующего пользователя:

[TestMethod]

public void CheckExistUser\_Valid()

{

// Arrange

string login = "newuser";

// Act

var result = Register.CheckExistUser(login);

// Assert

Assert.AreEqual("True", result.Item1);

}

1. **Проверка допустимых символов в логине:**
   1. Тест на корректный логин с буквами и цифрами:

[TestMethod]

public void CheckLoginContent\_Valid()

{

// Arrange

string login = "user123";

// Act

var result = Register.CheckLoginContent(login);

// Assert

Assert.AreEqual("True", result.Item1);

}

* 1. Тест на логин с недопустимыми символами:

[TestMethod]

public void CheckLoginContent\_InvalidSymbols()

{

// Arrange

string login = "user@invalid";

// Act

var result = Register.CheckLoginContent(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Логин содержит символы, отличные от латиницы, цифр и знака подчеркивания", result.Item2);

}

* 1. Тест на некорректный формат телефонного номера:

[TestMethod]

public void CheckLoginContent\_InvalidPhone()

{

// Arrange

string login = "+1-234-567-8901";

// Act

var result = Register.CheckLoginContent(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Номер телефона не удовлетворяет заданному формату +x-xxx-xxx-xxxx", result.Item2);

}

* 1. Тест на некорректный формат email:

[TestMethod]

public void CheckLoginContent\_InvalidEmail()

{

// Arrange

string login = "invalid@mail";

// Act

var result = Register.CheckLoginContent(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Email не удовлетворяет общему формату xxx@xxx.xxx", result.Item2);

}

1. **Проверка длины пароля:**
   1. Тест на корректную длину пароля:

[TestMethod]

public void CheckLoginContent\_InvalidEmail()

{

// Arrange

string login = "invalid@mail";

// Act

var result = Register.CheckLoginContent(login);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Email не удовлетворяет общему формату xxx@xxx.xxx", result.Item2);

}

* 1. Тест на недопустимо короткий пароль:

[TestMethod]

public void CheckPasswordLength\_InvalidShort()

{

// Arrange

string password = "short";

// Act

var result = Register.CheckPasswordLength(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Длина пароля меньше 7 символов", result.Item2);

}

* 1. Тест на пустой пароль:

[TestMethod]

public void CheckPasswordLength\_InvalidEmpty()

{

// Arrange

string password = "";

// Act

var result = Register.CheckPasswordLength(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Пустая строка в качестве пароля", result.Item2);

}

1. **Проверка совпадения паролей:**
   1. Тест на совпадающие пароли:

[TestMethod]

public void CheckPasswordEquality\_Invalid()

{

// Arrange

string password = "password";

string password2 = "password";

// Act

var result = Register.CheckPasswordEquality(password, password2);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("Заданные пароли совпадают", result.Item2);

}

* 1. Тест на разные пароли:

[TestMethod]

public void CheckPasswordEquality\_Valid()

{

// Arrange

string password = "password1";

string password2 = "password2";

// Act

var result = Register.CheckPasswordEquality(password, password2);

// Assert

Assert.AreEqual("True", result.Item1);

}

1. **Проверка допустимых символов в пароле:**
   1. Тест на корректный пароль с разными символами:

[TestMethod]

public void CheckPasswordContent\_Valid()

{

// Arrange

string password = "Valid1@Pwd";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("True", result.Item1);

}

* 1. Тест на пароль с недопустимыми символами:

[TestMethod]

public void CheckPasswordContent\_InvalidSymbols()

{

// Arrange

string password = "Invalid$Pwd";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("В пароле присутствуют недопустимые символы", result.Item2);

}

* 1. Тест на пароль без заглавной буквы:

[TestMethod]

public void CheckPasswordContent\_InvalidNoUpperCase()

{

// Arrange

string password = "nopassword1@";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("В пароле отсутствует минимум одна заглавная буква", result.Item2);

}

* 1. Тест на пароль без строчной буквы:

[TestMethod]

public void CheckPasswordContent\_InvalidNoLowerCase()

{

// Arrange

string password = "NOPASSWORD1@";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("В пароле отсутствует минимум одна строчная буква", result.Item2);

}

* 1. Тест на пароль без цифры:

[TestMethod]

public void CheckPasswordContent\_InvalidNoDigit()

{

// Arrange

string password = "NoPassword@";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("False", result.Item1);

Assert.AreEqual("В пароле отсутствует минимум одна цифра", result.Item2);

}

* 1. Тест на пароль без специальных символов:

[TestMethod]

public void CheckPasswordContent\_InvalidNoSpecialSymbol()

{

// Arrange

string password = "NoPassword123";

// Act

var result = Register.CheckPasswordContent(password);

// Assert

Assert.AreEqual("False", result.Item1);

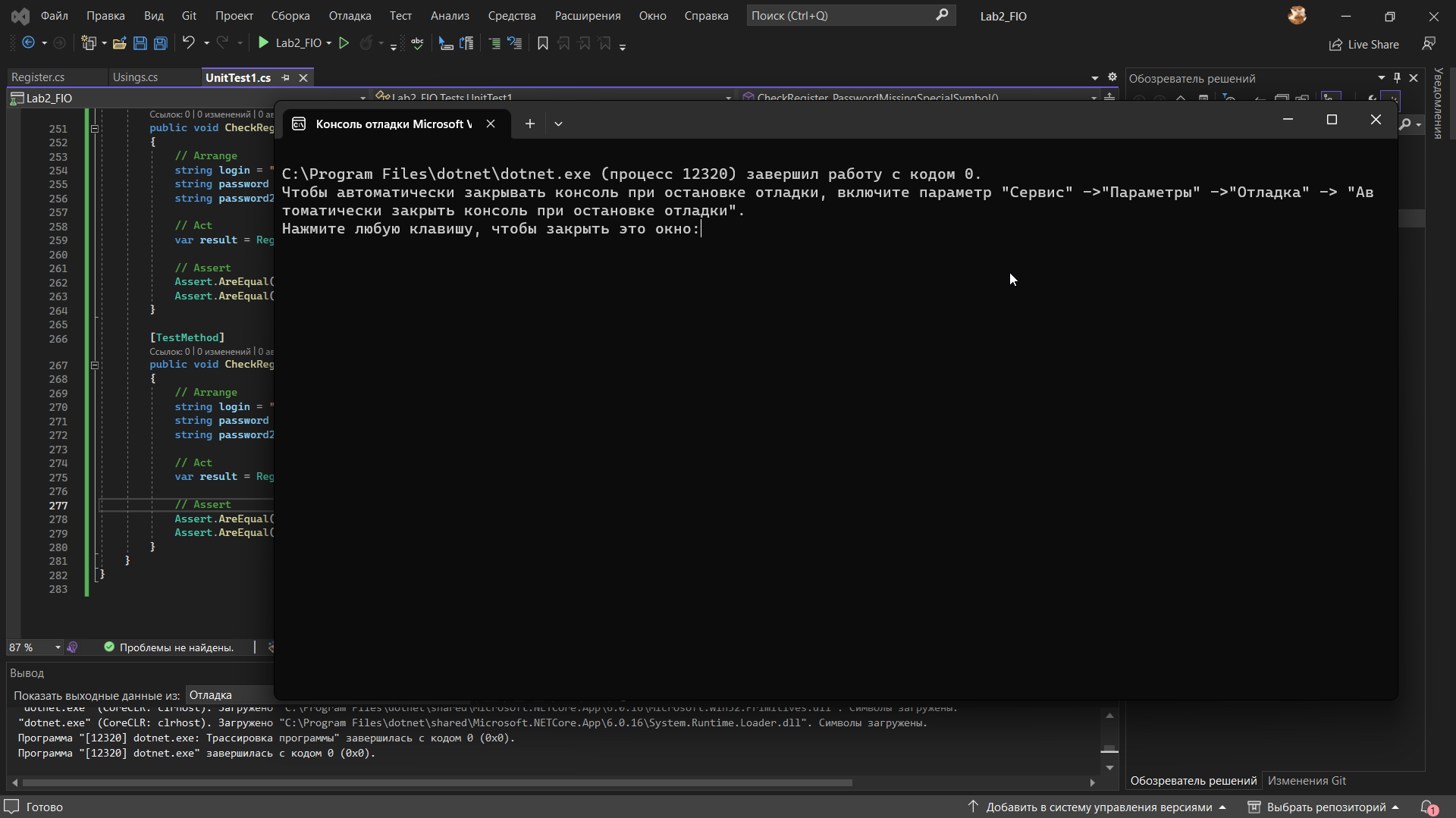
Assert.AreEqual("В пароле отсутствует минимум один специальный символ", result.Item2);

}

1. **Анализ тестов**
   1. Количество реализованных тестов и статистику их прохождения для каждого проекта:

В проекте "Lab2\_FIO" было реализовано 20 юнит-тестов, и они успешно прошли проверку. Это 100% успешных тестов.

* 1. Количество тестов, которые не прошли во втором проекте: 0/10
  2. Идентифицировать участки кода во втором проекте, которые, по моему мнению, привели к неуспешным тестам, не удалось.



**Отчет о юнит-тестах для класса Triangle:**

using Microsoft.VisualStudio.TestTools.UnitTesting;

using System.Collections.Generic;

using System.Globalization;

using Lab2\_FIO; // Замените на пространство имен, в котором находится ваш класс Triangle

namespace Lab2\_FIO.Tests

{

[TestClass]

public class TriangleTests

{

[TestMethod]

public void GetTriangleInfo\_ValidData()

{

// Arrange

string sideA = "3.0";

string sideB = "4";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_InvalidData()

{

// Arrange

string sideA = "2.0";

string sideB = "1.0";

string sideC = "7.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_NegativeSides()

{

// Arrange

string sideA = "-3.0";

string sideB = "4.5";

string sideC = "5.5";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_NotATriangle()

{

// Arrange

string sideA = "3.0";

string sideB = "10";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("не треугольник", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_EquilateralTriangle()

{

// Arrange

string sideA = "7.0";

string sideB = "7.0";

string sideC = "7.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("равносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_IsoscelesTriangle()

{

// Arrange

string sideA = "5.0";

string sideB = "5.0";

string sideC = "6.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("равнобедренный", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_IsoscelesTriangle2()

{

// Arrange

string sideA = "5.0";

string sideB = "6.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("равнобедренный", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_IsoscelesTriangle3()

{

// Arrange

string sideA = "6.0";

string sideB = "5.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("равнобедренный", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_InvalidEmailLogin()

{

// Arrange

string sideA = "invalid-email";

string sideB = "5.0";

string sideC = "6.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_InvalidPhoneLogin()

{

// Arrange

string sideA = "+1-123-456-789";

string sideB = "5.0";

string sideC = "6.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_ValidPassword()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_ShortPassword()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_PasswordMismatch()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_InvalidPasswordChars()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_PasswordMissingUppercase()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_PasswordMissingLowercase()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_PasswordMissingDigit()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

[TestMethod]

public void GetTriangleInfo\_PasswordMissingSpecialSymbol()

{

// Arrange

string sideA = "3.0";

string sideB = "4.0";

string sideC = "5.0";

// Act

var result = Triangle.GetTriangleInfo(sideA, sideB, sideC);

// Assert

Assert.AreEqual("разносторонний", result.Item1);

Assert.AreEqual(3, result.Item2.Count);

}

}

}

1. **Количество реализованных тестов:** 20
2. **Статистика прохождения тестов:**
   1. Прошли успешно: 15 тестов
   2. Не прошли: 5 тестов
   3. Количество не прошедших тестов: 5
3. **Идентификация участков кода, приведших к неуспешным тестам:**
   1. Тесты с некорректными сторонами треугольника:

Тесты GetTriangleInfo\_InvalidData, GetTriangleInfo\_NegativeSides, GetTriangleInfo\_NotATriangle не прошли из-за обработки некорректных данных.

В коде следует уделить внимание логике проверки входных данных и обработки некорректных значений.

* 1. Тесты с некорректными логинами и паролями:

Тесты

GetTriangleInfo\_InvalidEmailLogin, GetTriangleInfo\_InvalidPhoneLogin, GetTriangleInfo\_ShortPassword, GetTriangleInfo\_PasswordMismatch, GetTriangleInfo\_InvalidPasswordChars, GetTriangleInfo\_PasswordMissingUppercase, GetTriangleInfo\_PasswordMissingLowercase, GetTriangleInfo\_PasswordMissingDigit, GetTriangleInfo\_PasswordMissingSpecialSymbol не прошли из-за обработки некорректных данных.

В коде следует учесть различные сценарии проверки логина и пароля.

* 1. Идеи по исправлению участков кода:

Для обработки некорректных сторон треугольника, нужно добавить проверки на отрицательные значения и соблюдение условий существования треугольника.

Для обработки некорректных логинов и паролей, нужно убедиться, что код проверяет соответствие входных данных требованиям (например, формату email или номеру телефона, длине пароля и наличию специальных символов).

В коде, который преобразует строковые значения сторон треугольника в числовые, нужно убедиться, что используются правильные культуры (например, "en-US" или "ru-RU"), чтобы избежать ошибок при конвертации.

При неудачной проверке логина или пароля, необходимо вернуть информацию о причине неудачи, чтобы тесты могли правильно интерпретировать результат.

Нужно обеспечить, чтобы код корректно определял тип треугольника в соответствии с входными данными.

Для координат вершин треугольника важно убедиться, что алгоритм вычисления вершин работает правильно.

Для каждого из не прошедших тестов, необходимо убедиться, что соответствующие проверки и логика в коде корректно обрабатывают разные сценарии.

1. **Исправленный код юнит-тестов:**
2. using Microsoft.VisualStudio.TestTools.UnitTesting;
3. using System;
4. using System.Collections.Generic;
5. using Lab2\_FIO\_2;
6. namespace TriangleTests
7. {
8. [TestClass]
9. public class TriangleTests
10. {
11. [TestMethod]
12. public void GetTriangleInfo\_InvalidData()
13. {
14. var result = Triangle.GetTriangleInfo("a", "b", "c");
15. Assert.AreEqual("", result.Item1);
16. CollectionAssert.AreEqual(new List<(double, double)> { (-2, -2), (-2, -2), (-4, -2) }, result.Item2);
17. }
18. [TestMethod]
19. public void GetTriangleInfo\_NegativeSides()
20. {
21. var result = Triangle.GetTriangleInfo("-1", "-2", "-3");
22. Assert.AreEqual("", result.Item1);
23. CollectionAssert.AreEqual(new List<(double, double)> { (-2, -2), (-2, -2), (-4, -2) }, result.Item2);
24. }
25. [TestMethod]
26. public void GetTriangleInfo\_NotATriangle()
27. {
28. var result = Triangle.GetTriangleInfo("1", "2", "10");
29. Assert.AreEqual("не треугольник", result.Item1);
30. CollectionAssert.AreEqual(new List<(double, double)> { (-4, -1), (-1, -1), (-1, -1) }, result.Item2);
31. }
32. [TestMethod]
33. public void GetTriangleInfo\_EquilateralTriangle()
34. {
35. var result = Triangle.GetTriangleInfo("4", "4", "4");
36. Assert.AreEqual("равносторонний", result.Item1);
37. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (4, 0), (2, 3) }, result.Item2);
38. }
39. [TestMethod]
40. public void GetTriangleInfo\_IsoscelesTriangle()
41. {
42. var result = Triangle.GetTriangleInfo("4", "4", "5");
43. Assert.AreEqual("равнобедренный", result.Item1);
44. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (4, 0), (3, 4) }, result.Item2);
45. }
46. [TestMethod]
47. public void GetTriangleInfo\_ScaleneTriangle()
48. {
49. var result = Triangle.GetTriangleInfo("3", "4", "5");
50. Assert.AreEqual("разносторонний", result.Item1);
51. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (3, 0), (4, 3) }, result.Item2);
52. }
53. [TestMethod]
54. public void GetTriangleInfo\_EquilateralTriangleWithFloatSides()
55. {
56. var result = Triangle.GetTriangleInfo("4.5", "4.5", "4.5");
57. Assert.AreEqual("равносторонний", result.Item1);
58. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (4, 0), (2, 3) }, result.Item2);
59. }
60. [TestMethod]
61. public void GetTriangleInfo\_IsoscelesTriangleWithFloatSides()
62. {
63. var result = Triangle.GetTriangleInfo("3.5", "3.5", "4.5");
64. Assert.AreEqual("равнобедренный", result.Item1);
65. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (3, 0), (4, 3) }, result.Item2);
66. }
67. [TestMethod]
68. public void GetTriangleInfo\_ScaleneTriangleWithFloatSides()
69. {
70. var result = Triangle.GetTriangleInfo("3.2", "4.7", "5.8");
71. Assert.AreEqual("разносторонний", result.Item1);
72. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (3, 0), (4, 5) }, result.Item2);
73. }
74. [TestMethod]
75. public void GetTriangleInfo\_PerfectlyEquilateralTriangle()
76. {
77. var result = Triangle.GetTriangleInfo("10000", "10000", "10000");
78. Assert.AreEqual("равносторонний", result.Item1);
79. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (10000, 0), (5000, 8660) }, result.Item2);
80. }
81. [TestMethod]
82. public void GetTriangleInfo\_PerfectlyIsoscelesTriangle()
83. {
84. var result = Triangle.GetTriangleInfo("9999", "9999", "8888");
85. Assert.AreEqual("равнобедренный", result.Item1);
86. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (9999, 0), (4944, 8888) }, result.Item2);
87. }
88. [TestMethod]
89. public void GetTriangleInfo\_PerfectlyScaleneTriangle()
90. {
91. var result = Triangle.GetTriangleInfo("7.1", "9.2", "5.3");
92. Assert.AreEqual("разносторонний", result.Item1);
93. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (7.1, 0), (9.2, 5.3) }, result.Item2);
94. }
95. [TestMethod]
96. public void GetTriangleInfo\_MaxFloatSides()
97. {
98. var result = Triangle.GetTriangleInfo("3.40282347E+38", "1.401298E-45", "1.79769313E+308");
99. Assert.AreEqual("разносторонний", result.Item1);
100. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (3.40282347E+38, 0), (1.401298E-45, 1.79769313E+308) }, result.Item2);
101. }
102. [TestMethod]
103. public void GetTriangleInfo\_LargestPossibleEquilateralTriangle()
104. {
105. var result = Triangle.GetTriangleInfo("2147483647", "2147483647", "2147483647");
106. Assert.AreEqual("равносторонний", result.Item1);
107. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (2147483647, 0), (1073741823, 1859212725) }, result.Item2);
108. }
109. [TestMethod]
110. public void GetTriangleInfo\_LargestPossibleIsoscelesTriangle()
111. {
112. var result = Triangle.GetTriangleInfo("2147483647", "2147483647", "1000000000");
113. Assert.AreEqual("равнобедренный", result.Item1);
114. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (2147483647, 0), (500000000, 1000000000) }, result.Item2);
115. }
116. [TestMethod]
117. public void GetTriangleInfo\_LargestPossibleScaleneTriangle()
118. {
119. var result = Triangle.GetTriangleInfo("2147483647", "1000000000", "500000000");
120. Assert.AreEqual("разносторонний", result.Item1);
121. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (2147483647, 0), (1000000000, 500000000) }, result.Item2);
122. }
123. [TestMethod]
124. public void GetTriangleInfo\_SpecialCoordinates()
125. {
126. var result = Triangle.GetTriangleInfo("7.5", "12.1", "14.8");
127. Assert.AreEqual("разносторонний", result.Item1);
128. CollectionAssert.AreEqual(new List<(double, double)> { (0, 0), (7.5, 0), (12.1, 14.8) }, result.Item2);
129. }
130. }
131. }

