Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»
(СибГУТИ)

Кафедра прикладной математики и кибернетики

Современные Технологии Программирования 2

Лабораторная работа

«Приложение Конвертер p1_p2»

Выполнил:
Студент IV курса ИВТ,
группы ИП-713
Михеев Никита Алексеевич

Работу проверил:
Ассистент кафедры ПМиК
Агалаков Антон Александрович

Новосибирск 2020 г.

Оглавление

1. Цель

Объектно-ориентированный анализ, проектирование и реализация приложения «Конвертор p1_p2» под Windows для преобразования действительных чисел, представленных в системе счисления с основанием p1 в действительные числа представленные в системе счисления с основанием p2. В процессе выполнения работы студенты изучают: отношения между классами: ассоциация, агрегация, зависимость, их реализацию средствами языка программирования высокого уровня; этапы разработки приложений в технологии ООП; элементы технологии визуального программирования; диаграммы языка UML для документирования разработки.

2. Задание

Приложение должно обеспечивать пользователю: преобразование действительного числа представленного в системе счисления с основанием p1 в число, представленное в системе счисления с основанием p2;
основания систем счисления p1, p2 для исходного числа и результата преобразования выбираются пользователем из диапазона от 2..16;
возможность ввода и редактирования действительного числа представленного в системе счисления с основанием p2 с помощью командных кнопок и мыши, а также с помощью клавиатуры;
контекстную помощь по элементам интерфейса и справку о назначении приложения; просмотр истории сеанса (журнала) работы пользователя с приложением – исходные данные, результат преобразования и основания систем счисления, в которых они представлены;
дополнительные повышенные требования: автоматический расчёт необходимой точности представления результата.

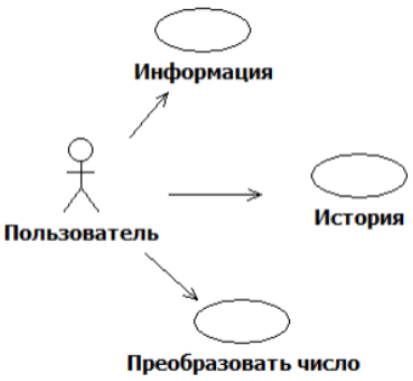


Рис.1 – Диаграмма прецедентов

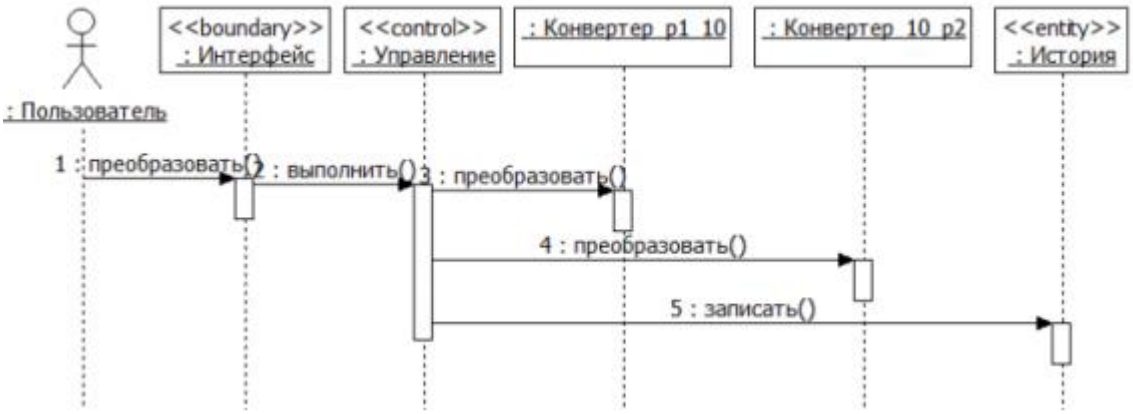


Рис.2 – поток событий для прецедента «Преобразовать»

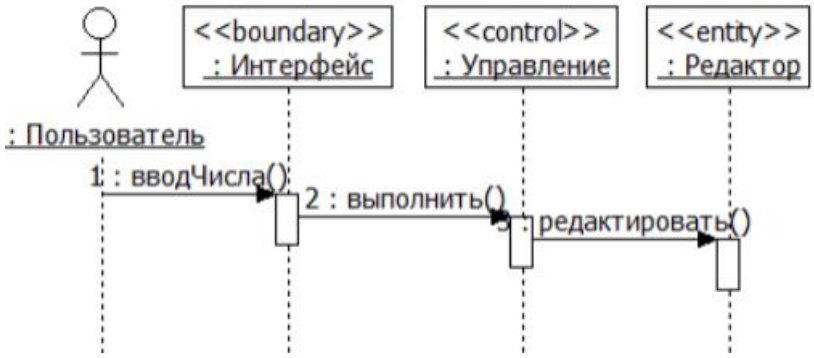


Рис.3 – поток событий для прецедента «Ввести число»

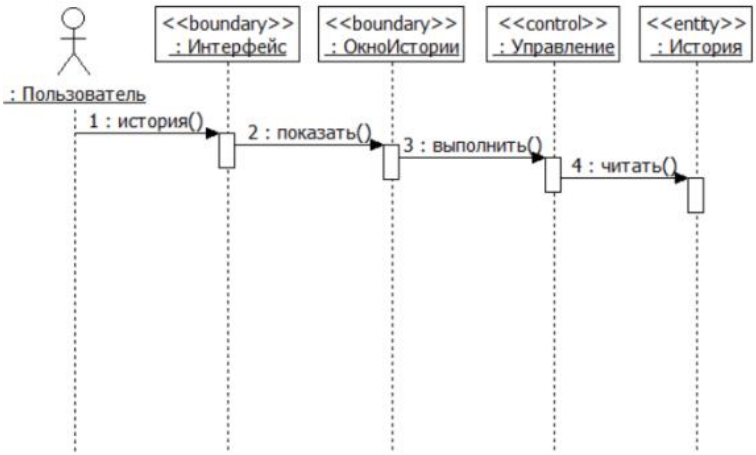


Рис.4 – поток событий для прецедента «История»

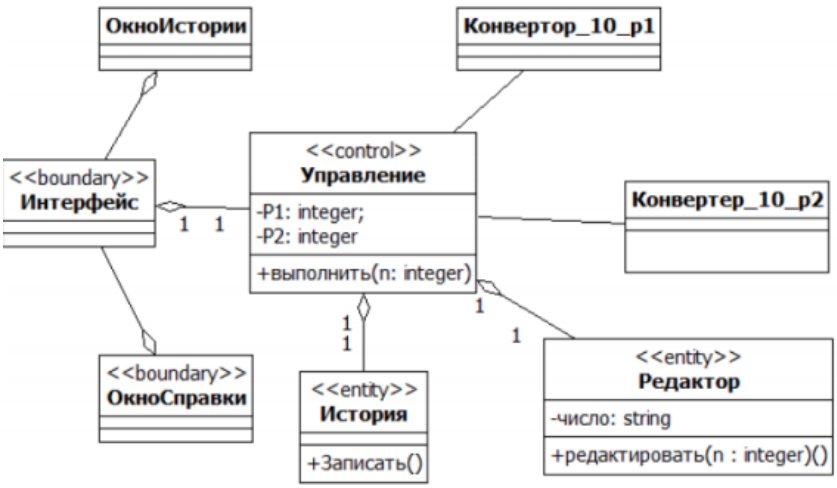


Рис.5 – диаграмма классов

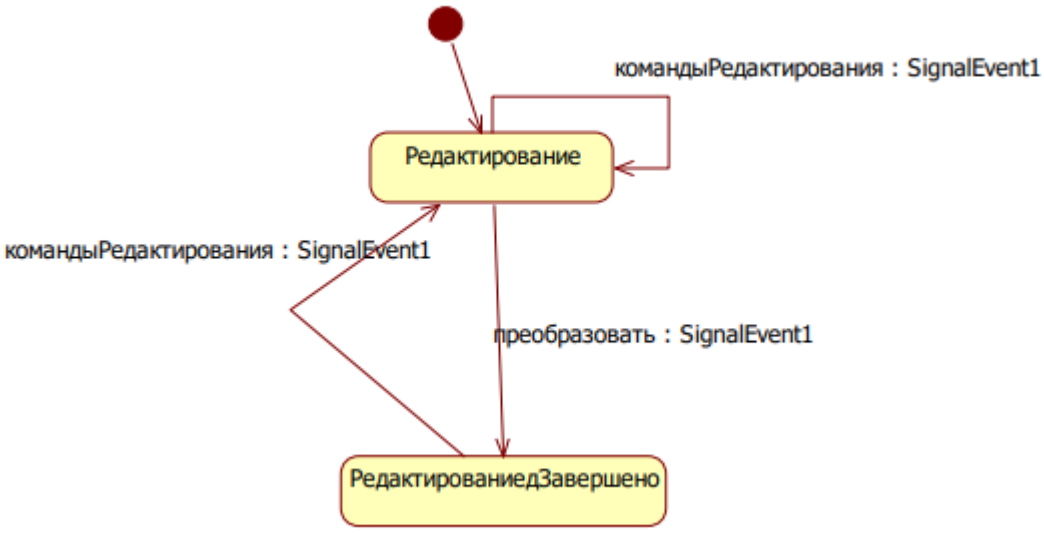


Рис.6 – диаграмма состояний класс Управление

3. Листинг

ADT_Convert_10_p.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Converter
{
    public class ADT_Convert_10_p
    {
        //Преобразовать целое в символ.
        public static char int_to_Char(int d)
        {
            if (d < 0 || d > 15)
                throw new IndexOutOfRangeException();

            string allSymbols = "0123456789ABCDEF";
            return allSymbols.ElementAt(d);
        }

        //Преобразовать десятичное целое в с.сч. с основанием p.
        public static string int_to_p(long n, long p)
        {
            if (p < 2 || p > 16)
                throw new IndexOutOfRangeException();
            if (n == 0)
                return "0";
            if (p == 10)
                return n.ToString();

            bool isNegative = false;
            if (n < 0)
            {
                isNegative = true;
                n *= -1;
            }
```

```
            string pNumber = String.Empty;

            while(n > 0)
            {
                pNumber += int_to_Char((int)(n % p));
                n /= p;
            }

            if (isNegative)
                pNumber += "-";

            char[] tmp = pNumber.ToCharArray();
            Array.Reverse(tmp);
            return new string(tmp);
        }

        //Преобразовать десятичную дробь в с.сч. с основанием p.
        public static string flt_to_p(double n, int p, int c)
        {
            if (p < 2 || p > 16)
                throw new IndexOutOfRangeException();
            if (c < 0 || c > 10)
                throw new IndexOutOfRangeException();

            string pNumber = String.Empty;
            for(int i = 0; i < c; i++)
            {
                pNumber += int_to_Char((int)(n * p));
                n = n * p - (int)(n * p);
            }
            return pNumber;
        }

        //Преобразовать десятичное
        //действительное число в с.сч. с осн. p.
        public static string Do(double n, int p, int c)
        {
            if (p < 2 || p > 16)
                throw new IndexOutOfRangeException();
            if (c < 0 || c > 10)
                throw new IndexOutOfRangeException();

            long leftSide = (long)n;

            double rightSide = n - leftSide;
            if (rightSide < 0)
                rightSide *= -1;

            string leftSideString = int_to_p(leftSide, p);
            string rightSideString = flt_to_p(rightSide, p, c);

            return leftSideString + (rightSideString == String.Empty ? "" : ".") +
rightSideString;
        }
    }
}
```

ADT_Convert_p_10.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
```

```
namespace Converter
{
    public class ADT_Convert_p_10
    {
        //Преобразовать цифру в число.
        private static int char_to_num(char ch)
        {
            string allNums = "0123456789ABCDEF";
            if (!allNums.Contains(ch))
                throw new IndexOutOfRangeException();
            return allNums.IndexOf(ch);
        }

        //Преобразовать строку в число
        private static double convert(string pNumber, int p, double weight)
        {
            if (weight % p != 0)
                throw new Exception();

            long degree = (long)Math.Log(weight, p) - 1;
            double result = 0.0f;

            for(int i = 0; i < pNumber.Length; i++, degree--)
                result += char_to_num(pNumber.ElementAt(i)) * Math.Pow(p, degree);
            return result;
        }

        //Преобразовать из с.сч. с основанием p
        //в с.сч. с основанием 10.
        public static double dval(string pNumber, int p)
        {
            if (p < 2 || p > 16)
                throw new IndexOutOfRangeException();
            foreach (char ch in pNumber)
            {
                if (ch == '.')
                    continue;
                if (char_to_num(ch) > p)
                    throw new Exception();
            }

            Regex LeftRight = new Regex("^[0-9A-F]+\\.[0-9A-F]+$");
            Regex Right = new Regex("^0\\.[0-9A-F]+$");
            Regex Left = new Regex("^[0-9A-F]+$");

            double Number;
            if (LeftRight.IsMatch(pNumber))
                Number = convert(pNumber.Remove(pNumber.IndexOf('.'), 1), p, Math.Pow(p,
pNumber.IndexOf('.')));
            else if (Left.IsMatch(pNumber))
                Number = convert(pNumber, p, Math.Pow(p, pNumber.Length));
            else if (Right.IsMatch(pNumber))
                Number = convert(pNumber.Remove(pNumber.IndexOf('.'), 1), p, 0);
            else
                throw new Exception();

            return Number;
        }
    }
}
```

ADT_Control_.cs:

```
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Converter
{
    public class ADT_Control_
    {
        //Основание системы сч. исходного числа.
        int pin = 10;
        //Основание системы сч. результата.
        int pout = 16;
        //Число разрядов в дробной части результата.
        const int accuracy = 10;
        public History history = new History();
        public enum State { Edit, Converted }
        private State state;
        //Свойство для чтения и записи состояние Конвертера.
        internal State St { get => state; set => state = value; }
        //Свойство для чтения и записи основание системы сч. p1.
        public int Pin { get => pin; set => pin = value; }
        //Свойство для чтения и записи основание системы сч. p2.
        public int Pout { get => pout; set => pout = value; }
        //Конструктор.
        public ADT_Control_()
        {
            St = State.Edit;
            Pin = pin;
            Pout = pout;
        }
        //объект редактор
        public Editor editor = new Editor();
        //Выполнить команду конвертера.
        public string doCmnd(Editor.Commands j)
        {
            if(j == Editor.Commands.exec)
            {
                double r = ADT_Convert_p_10.dval(editor.Number, Pin);
                string res = ADT_Convert_10_p.Do(r, Pout, acc());
                St = State.Converted;
                history.addRecord(Pin, Pout, editor.Number, res);
                return res;
            }
            else
            {
                St = State.Edit;
                return editor.doEdit(j);
            }
        }

        //Точность представления результата.
        private int acc()
        {
            return (int)Math.Round(editor.acc() * Math.Log(Pin) / Math.Log(Pout) + 0.5);
        }
    }
}
```

Editor.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
using System.Threading.Tasks;

namespace Converter
{
    public class Editor
    {
        public enum Commands
        {
            add0, add1, add2, add3,
            add4, add5, add6, add7,
            add8, add9, addA, addB,
            addC, addD, addE, addF,
            addDot, BS, CLEAR, exec
        };
        //Поле для хранения редактируемого числа.
        string number = zero;
        //Разделитель целой и дробной частей.
        const string delim = ".";
        //Ноль.
        const string zero = "0";
        //Свойствое для чтения редактируемого числа.
        public string Number
        { get => number; set => number = value; }
        //Добавить цифру.
        public string addDigit(int n)
        {
            if (n < 0 || n > 16)
                throw new IndexOutOfRangeException();

            if (number == zero)
                number = ADT_Convert_10_p.int_to_Char(n).ToString();
            else
                number += ADT_Convert_10_p.int_to_Char(n);
            return Number;
        }
        //Точность представления результата.
        public int acc()
        {
            return number.Contains(delim) ? number.Length - number.IndexOf(delim) - 1 :
0;
        }
        //Добавить ноль.
        public string addZero()
        {
            return addDigit(0);
        }
        //Добавить разделитель.
        public string addDelim()
        {
            if (number.Length > 0 && !number.Contains(delim))
                number += delim;
            return Number;
        }
        //Удалить символ справа.
        public string bs()
        {
            if (number.Length > 1)
                number = number.Remove(number.Length - 1);
            else
                number = zero;
            return Number;
        }
        //Очистить редактируемое число.
        public string clear()
```

```
{
    number = zero;
    return Number;
}
//Выполнить команду редактирования.
public string doEdit(Commands j)
{
    switch(j)
    {
        case Commands.add0:
            addZero();
            break;
        case Commands.add1:
            addDigit(1);
            break;
        case Commands.add2:
            addDigit(2);
            break;
        case Commands.add3:
            addDigit(3);
            break;
        case Commands.add4:
            addDigit(4);
            break;
        case Commands.add5:
            addDigit(5);
            break;
        case Commands.add6:
            addDigit(6);
            break;
        case Commands.add7:
            addDigit(7);
            break;
        case Commands.add8:
            addDigit(8);
            break;
        case Commands.add9:
            addDigit(9);
            break;
        case Commands.addA:
            addDigit(10);
            break;
        case Commands.addB:
            addDigit(11);
            break;
        case Commands.addC:
            addDigit(12);
            break;
        case Commands.addD:
            addDigit(13);
            break;
        case Commands.addE:
            addDigit(14);
            break;
        case Commands.addF:
            addDigit(15);
            break;
        case Commands.addDot:
            addDelim();
            break;
        case Commands.BS:
            bs();
            break;
        case Commands.CLEAR:
```

```
                    clear();
                    break;
                default:
                    return number;
            }
            return number;
        }
    }
}
```

History.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Converter
{
    public class History
    {
        public struct Record
        {
            private int p1;
            private int p2;
            private string number1;
            private string number2;
            public Record(int p1, int p2, string n1, string n2)
            {
                this.p1 = p1;
                this.p2 = p2;
                number1 = n1;
                number2 = n2;
            }
            public List<string> toList()
            {
                return new List<string> { p1.ToString(), number1, p2.ToString(), number2
};
            }
        }

        List<Record> L;
        public History()
        {
            L = new List<Record>();
        }
        public void addRecord(int p1, int p2, string n1, string n2)
        {
            Record record = new Record(p1, p2, n1, n2);
            L.Add(record);
        }

        public Record this[int i]
        {
            get
            {
                if (i < 0 || i >= L.Count)
                    throw new IndexOutOfRangeException();
                return L[i];
            }
            set
            {
                if (i < 0 || i >= L.Count)
                    throw new IndexOutOfRangeException();
```

```
                L[i] = value;
            }
        }

        public void clear()
        {
            L.Clear();
        }

        public int Count()
        {
            return L.Count();
        }
    }
}
```

Form1.cs:

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace Converter
{
    public partial class Form1 : Form
    {
        ADT_Control_ control_ = new ADT_Control_();
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            label1.Text = control_.editor.Number;
            trackBar1.Value = control_.Pin;
            trackBar2.Value = control_.Pout;
            label2.Text = "0";
            UpdateButtons();
        }

        private void doCmnd(Editor.Commands j)
        {
            if(j == Editor.Commands.exec)
                label2.Text = control_.doCmnd(j);
            else
            {
                if (control_.St == ADT_Control_.State.Converted)
                    label1.Text = control_.doCmnd(Editor.Commands.CLEAR);
                label1.Text = control_.doCmnd(j);
                label2.Text = "0";
            }
        }

        private void UpdateButtons()
        {
            foreach(Control i in Controls)
            {
                if(i is Button)
                {
                    int j = Convert.ToInt16(i.Tag.ToString());
                    if (j < trackBar1.Value)
                        i.Enabled = true;
                    if ((j >= trackBar1.Value) && (j <= 15))
                        i.Enabled = false;
```

```
            }
        }
    }

    private void trackbar1_Scroll(object sender, EventArgs e)
    {
        numericUpDown1.Value = trackBar1.Value;
        UpdateP1();
    }

    private void numericUpDown1_ValueChanged(object sender, EventArgs e)
    {
        trackBar1.Value = Convert.ToByte(numericUpDown1.Value);
        UpdateP1();
    }

    private void UpdateP1()
    {
        control_.Pin = trackBar1.Value;
        UpdateButtons();
        label1.Text = control_.doCmnd(Editor.Commands.CLEAR);
        label2.Text = "0";
    }

    private void trackBar2_Scroll(object sender, EventArgs e)
    {
        numericUpDown2.ValueChanged -= numericUpDown2_ValueChanged;
        numericUpDown2.Value = trackBar2.Value;
        numericUpDown2.ValueChanged += numericUpDown2_ValueChanged;
        UpdateP2();
    }

    private void numericUpDown2_ValueChanged(object sender, EventArgs e)
    {
        trackBar2.Value = Convert.ToByte(numericUpDown2.Value);
        UpdateP2();
    }

    private void UpdateP2()
    {
        control_.Pout = trackBar2.Value;
        label2.Text = control_.doCmnd(Editor.Commands.exec);
    }

    private void выходToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Close();
    }

    private void историяToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Form2 history = new Form2();
        history.Show();
        if (control_.history.Count() == 0)
        {
            MessageBox.Show("История пуста", "Внимание", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            return;
        }
        for (int i = 0; i < control_.history.Count(); i++)
        {
            List<string> currentRecord = control_.history[i].toList();
            history.dataGridView1.Rows.Add(currentRecord[0], currentRecord[1],
currentRecord[2], currentRecord[3]);
```

```
            }
        }

        private void справкаToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Ковертер p1_p2.\n" +
                "Авторы: Михеев Никита\n" +
                "Устенко Дмитрий\n" +
                "Группа: ИП-713.", "Справка", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        private void Form1_KeyPress(object sender, KeyPressEventArgs e)
        {
            int i = -1;
            if (e.KeyChar >= 'A' && e.KeyChar <= 'F')
                i = e.KeyChar - 'A' + 10;
            if (e.KeyChar >= 'a' && e.KeyChar <= 'f')
                i = e.KeyChar - 'a' + 10;
            if (e.KeyChar >= '0' && e.KeyChar <= '9')
                i = e.KeyChar - '0';
            if (e.KeyChar == '.')
                i = 16;
            if (e.KeyChar == 8)
                i = 17;
            if (e.KeyChar == 13)
                i = 19;
            if (i == -1)
                return;
            doCmnd((Editor.Commands)Enum.GetValues(typeof(Editor.Commands)).GetValue(i));
        }

        private void Form1_KeyDown(object sender, KeyEventArgs e)
        {
            if (e.KeyCode == Keys.Delete)
                doCmnd(Editor.Commands.BS);
            if (e.KeyCode == Keys.Execute)
                doCmnd(Editor.Commands.exec);
            if (e.KeyCode == Keys.Decimal)
                doCmnd(Editor.Commands.addDot);
        }

        private void button_Click(object sender, EventArgs e)
        {
            Button but = (Button)sender;
            int j = Convert.ToInt16(but.Tag.ToString());
            doCmnd((Editor.Commands)Enum.GetValues(typeof(Editor.Commands)).GetValue(j));
        }
    }
}
```

UnitTest1.cs:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System;

namespace ConverterTests
{
    [TestClass]
    public class Test_ADT_Convert_10_p
    {
        [TestMethod]
        public void TestDo1()
        {
            double n = 123.123;
```

```
    int p = 12;
    int c = 3;
    string Expect = "A3.158";
    string Actual = Converter.ADT_Convert_10_p.Do(n, p, c);
    Assert.AreEqual(Expect, Actual);
}

[TestMethod]
public void TestDo2()
{
    double n = -144.523;
    int p = 3;
    int c = 8;
    string Expect = "-12100.11201002";
    string Actual = Converter.ADT_Convert_10_p.Do(n, p, c);
    Assert.AreEqual(Expect, Actual);
}

[TestMethod]
[ExpectedException(typeof(IndexOutOfRangeException))]
public void TestDo3()
{
    double n = -12312.1231;
    int p = 3;
    int c = -8;
    Converter.ADT_Convert_10_p.Do(n, p, c);
}

[TestMethod]
[ExpectedException(typeof(IndexOutOfRangeException))]
public void TestDo4()
{
    double n = -12312.1231;
    int p = -3;
    int c = 8;
    Converter.ADT_Convert_10_p.Do(n, p, c);
}

[TestMethod]
public void TestIntToChar1()
{
    int n = 12;
    char ExpectedChar = 'C';
    char ActualChar = Converter.ADT_Convert_10_p.int_to_Char(n);
    Assert.AreEqual(ExpectedChar, ActualChar);
}

[TestMethod]
public void TestIntToChar2()
{
    int n = 3;
    char ExpectedChar = '3';
    char ActualChar = Converter.ADT_Convert_10_p.int_to_Char(n);
    Assert.AreEqual(ExpectedChar, ActualChar);
}

[TestMethod]
[ExpectedException(typeof(System.IndexOutOfRangeException))]
public void TestIntToChar3()
{
    int n = -12;
    Converter.ADT_Convert_10_p.int_to_Char(n);
}
```

```
    [TestMethod]
    public void TestIntToP1()
    {
        int n = 123;
        int p = 12;
        string ExpectedString = "A3";
        string ActualString = Converter.ADT_Convert_10_p.int_to_p(n, p);
        Assert.AreEqual(ExpectedString, ActualString);
    }

    [TestMethod]
    public void TestIntToP2()
    {
        int n = -234567;
        int p = 9;
        string ExpectedString = "-386680";
        string ActualString = Converter.ADT_Convert_10_p.int_to_p(n, p);
        Assert.AreEqual(ExpectedString, ActualString);
    }

    [TestMethod]
    [ExpectedException(typeof(System.IndexOutOfRangeException))]
    public void TestIntToP3()
    {
        int n = 123;
        int p = -24;
        Converter.ADT_Convert_10_p.int_to_p(n, p);
    }

    [TestMethod]
    public void TestFltToP1()
    {
        double n = 0.123;
        int p = 12;
        int c = 3;
        string ExpectedString = "158";
        string ActualString = Converter.ADT_Convert_10_p.flt_to_p(n, p, c);
        Assert.AreEqual(ExpectedString, ActualString);
    }

    [TestMethod]
    public void TestFltToP2()
    {
        double n = 0.417;
        int p = 9;
        int c = 5;
        string ExpectedString = "36688";
        string ActualString = Converter.ADT_Convert_10_p.flt_to_p(n, p, c);
        Assert.AreEqual(ExpectedString, ActualString);
    }


    [TestMethod]
    [ExpectedException(typeof(System.IndexOutOfRangeException))]
    public void TestFltToP3()
    {
        double n = 1.5;
        int p = 12;
        int c = 3;
        Converter.ADT_Convert_10_p.flt_to_p(n, p, c);
    }
}
```

```
[TestClass]
public class Test_ADT_Convert_p_10
{
    [TestMethod]
    public void TestDval1()
    {
        string Number = "123.321";
        int P = 4;
        double ExpectedValue = 27.890625;
        double ActualValue = Converter.ADT_Convert_p_10.dval(Number, P);
        Assert.AreEqual(ExpectedValue, ActualValue, 0.00001);
    }

    [TestMethod]
    public void TestDval2()
    {
        string Number = "37.53";
        int P = 8;
        double ExpectedValue = 31.671875;
        double ActualValue = Converter.ADT_Convert_p_10.dval(Number, P);
        Assert.AreEqual(ExpectedValue, ActualValue, 0.00001);
    }

    [TestMethod]
    public void TestDval3()
    {
        string Number = "A8F.9C9";
        int P = 16;
        double ExpectedValue = 2703.611572265625;
        double ActualValue = Converter.ADT_Convert_p_10.dval(Number, P);
        Assert.AreEqual(ExpectedValue, ActualValue, 0.00001);
    }

    [TestMethod]
    public void TestDval4()
    {
        string Number = "0.23A5";
        int P = 13;
        double ExpectedValue = 0.17632435839081264662;
        double ActualValue = Converter.ADT_Convert_p_10.dval(Number, P);
        Assert.AreEqual(ExpectedValue, ActualValue, 0.00001);
    }

    [TestMethod]
    public void TestDval5()
    {
        string Number = "9876";
        int P = 11;
        double ExpectedValue = 13030;
        double ActualValue = Converter.ADT_Convert_p_10.dval(Number, P);
        Assert.AreEqual(ExpectedValue, ActualValue, 0.00001);
    }

    [TestMethod]
    [ExpectedException(typeof(System.Exception))]
    public void TestDval6()
    {
        string Number = ".A";
        int P = 11;
        Converter.ADT_Convert_p_10.dval(Number, P);
    }
```

```
    [TestMethod]
    [ExpectedException(typeof(System.IndexOutOfRangeException))]
    public void TestDval7()
    {
        string Number = "AA";
        int P = 77;
        Converter.ADT_Convert_p_10.dval(Number, P);
    }

    [TestMethod]
    [ExpectedException(typeof(System.Exception))]
    public void TestDval8()
    {
        string Number = "FFF";
        int P = 2;
        Converter.ADT_Convert_p_10.dval(Number, P);
    }
}

[TestClass]
public class Test_Editor
{
    [TestMethod]
    public void TestAddDigit1()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(0);
        string ExpectedValue = "0";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestAddDigit2()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(0);
        editor.addDigit(0);
        editor.addDigit(0);
        editor.addDigit(0);
        editor.addDigit(0);
        string ExpectedValue = "0";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestAddDigit3()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(0);
        editor.addDelim();
        editor.addDigit(0);
        editor.addDigit(0);
        editor.addDigit(0);
        editor.addDigit(0);
        string ExpectedValue = "0.0000";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestAddDigit4()
```

```
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(15);
    editor.addDigit(12);
    editor.addDigit(1);
    editor.addDelim();
    editor.addDigit(1);
    editor.addDigit(9);
    string ExpectedValue = "FC1.19";
    string ActualValue = editor.Number;
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
[ExpectedException(typeof(System.IndexOutOfRangeException))]
public void TestAddDigit5()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(17);
}

[TestMethod]
[ExpectedException(typeof(System.IndexOutOfRangeException))]
public void TestAddDigit6()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(-12);
}

[TestMethod]
public void TestAcc1()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(15);
    editor.addDigit(12);
    editor.addDigit(1);
    editor.addDelim();
    editor.addDigit(1);
    editor.addDigit(9);
    int ExpectedValue = 2;
    int ActualValue = editor.acc();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAcc2()
{
    Converter.Editor editor = new Converter.Editor();
    int ExpectedValue = 0;
    int ActualValue = editor.acc();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAcc3()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDelim();
    editor.addDigit(1);
    editor.addDigit(9);
    editor.addDigit(9);
    editor.addDigit(9);
    editor.addDigit(9);
```

```
    int ExpectedValue = 5;
    int ActualValue = editor.acc();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDelim1()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(15);
    editor.addDigit(15);
    editor.addDigit(15);
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    editor.addDigit(15);
    editor.addDigit(15);
    editor.addDigit(15);
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    string ExpectedValue = "FFF.FFF";
    string ActualValue = editor.Number;
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDelim2()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(0);
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    editor.addDigit(0);
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    string ExpectedValue = "0.0";
    string ActualValue = editor.Number;
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDelim3()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    string ExpectedValue = "0.";
    string ActualValue = editor.Number;
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestBs1()
{
    Converter.Editor editor = new Converter.Editor();
    editor.bs();
    editor.bs();
    editor.bs();
    editor.bs();
```

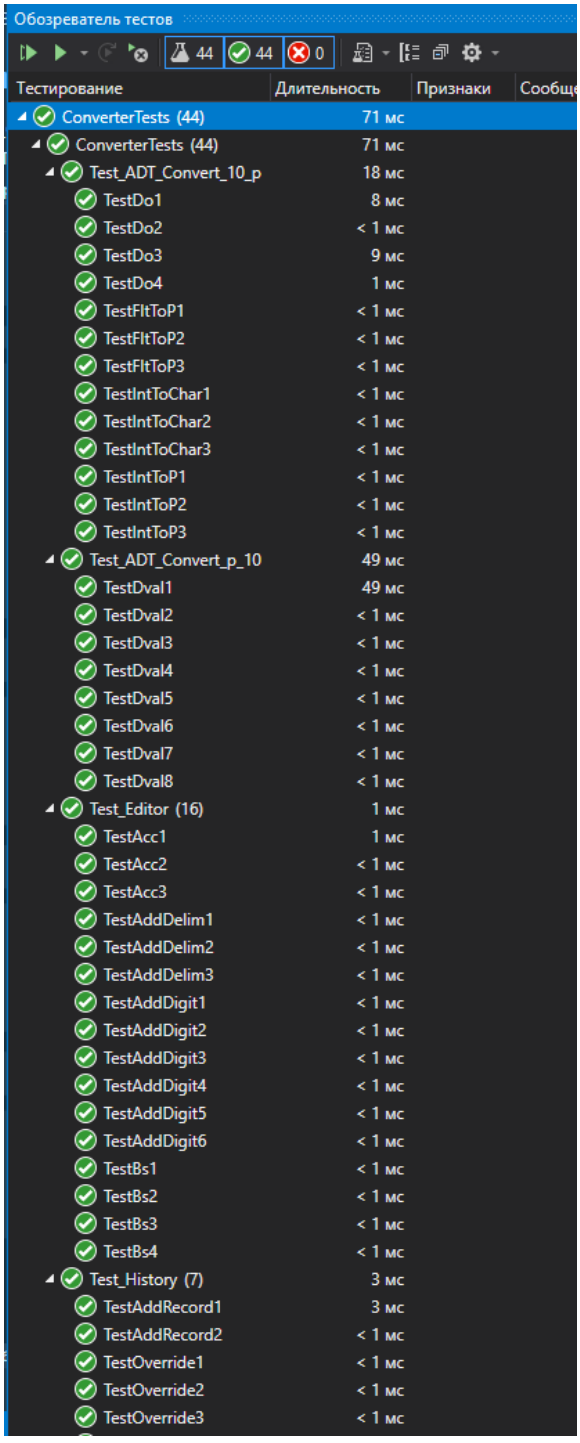
```
        editor.bs();
        string ExpectedValue = "0";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs2()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.bs();
        editor.bs();
        editor.bs();
        editor.bs();
        editor.bs();
        string ExpectedValue = "0";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs3()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(3);
        editor.addDigit(3);
        editor.addDigit(3);
        editor.addDelim();
        editor.bs();
        string ExpectedValue = "333";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs4()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(3);
        editor.addDigit(3);
        editor.addDigit(3);
        editor.addDelim();
        editor.addDigit(3);
        editor.addDigit(3);
        editor.addDigit(3);
        editor.bs();
        editor.bs();
        editor.bs();
        string ExpectedValue = "333.";
        string ActualValue = editor.Number;
        Assert.AreEqual(ExpectedValue, ActualValue);
    }
}

[TestClass]
public class Test_History
{
    [TestMethod]
    public void TestAddRecord1()
    {
        Converter.History history = new Converter.History();
        history.addRecord(12, 4, "23.42", "52.42");
```

```
            Converter.History.Record ExpectedValue = new Converter.History.Record(12, 4,
"23.42", "52.42");
            Converter.History.Record ActualValue = history[0];
            Assert.AreEqual(ExpectedValue, ActualValue);
        }

        [TestMethod]
        public void TestAddRecord2()
        {
            Converter.History history = new Converter.History();
            history.addRecord(3, 7, "11.11", "11.11");
            Converter.History.Record ExpectedValue = new Converter.History.Record(3, 7,
"11.11", "11.11");
            Converter.History.Record ActualValue = history[0];
            Assert.AreEqual(ExpectedValue, ActualValue);
        }

        [TestMethod]
        public void TestOverride1()
        {
            Converter.History history = new Converter.History();
            history.addRecord(12, 4, "23.42", "52.42");
            history.addRecord(12, 4, "23.42", "52.42");
            history.addRecord(12, 4, "11", "11");
            Converter.History.Record ExpectedValue = new Converter.History.Record(12, 4, "11",
"11");
            Converter.History.Record ActualValue = history[2];
            Assert.AreEqual(ExpectedValue, ActualValue);
        }

        [TestMethod]
        public void TestOverride2()
        {
            Converter.History history = new Converter.History();
            history.addRecord(12, 4, "23.42", "52.42");
            history.addRecord(12, 4, "23.42", "52.42");
            history.addRecord(12, 4, "11", "11");
            Converter.History.Record ToOverride = new Converter.History.Record(1, 1, "1",
"1");
            history[1] = ToOverride;
            Converter.History.Record ExpectedValue = new Converter.History.Record(1, 1, "1",
"1");
            Converter.History.Record ActualValue = history[1];
            Assert.AreEqual(ExpectedValue, ActualValue);
        }

        [TestMethod]
        [ExpectedException(typeof(System.IndexOutOfRangeException))]
        public void TestOverride3()
        {
            Converter.History history = new Converter.History();
            history.addRecord(3, 7, "11.11", "11.11");
            Converter.History.Record Value = history[-1];
        }

        [TestMethod]
        [ExpectedException(typeof(System.IndexOutOfRangeException))]
        public void TestOverride4()
        {
            Converter.History history = new Converter.History();
            history.addRecord(3, 7, "11.11", "11.11");
            Converter.History.Record Value = history[1];
        }
```

```
        [TestMethod]
        [ExpectedException(typeof(System.IndexOutOfRangeException))]
        public void TestOverride5()
        {
            Converter.History history = new Converter.History();
            Converter.History.Record Value = new Converter.History.Record(12, 4, "11", "11");
            history[0] = Value;
        }
    }
}
```

4. Результаты тестирования

Сначала были успешно выполнены все юнит-тесты классов:



Затем программа была проверена с интерфейсом: