Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Институт информатики и вычислительной техники
09.03.01 "Информатика и вычислительная техника"
профиль "Программное обеспечение средств
вычислительной техники и автоматизированных систем"

Кафедра прикладной математики и кибернетики

Современные технологии программирования

Практическая работа
«Приложение Конвертор p1_p2»

Выполнил:
Студент группы ИП-911
Мироненко К.А.
Работу проверил:
доцент кафедры ПМиК
Зайцев М.Г.

г. Новосибирск 2022-2023 уч. года

Оглавление

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.

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.

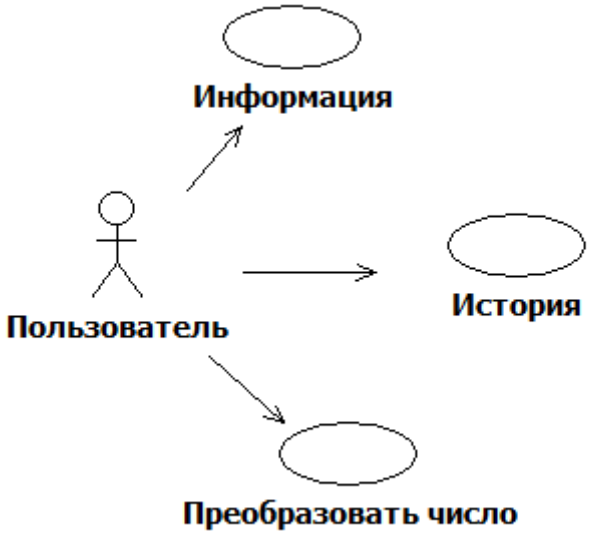


Рис 1. Use-case диаграмма

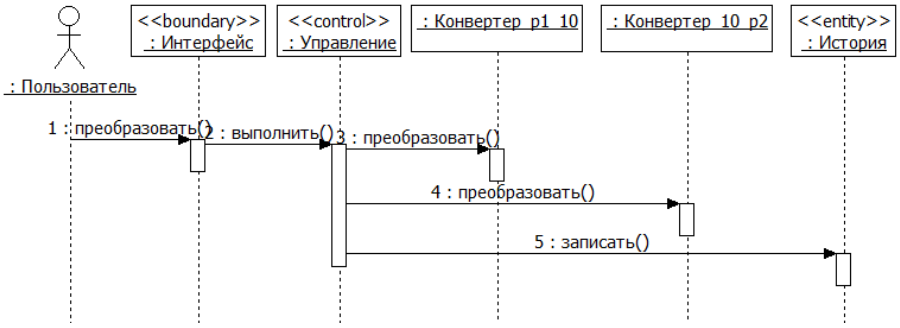


Рис 2. Поток событий для прецендента «Преобразователь»

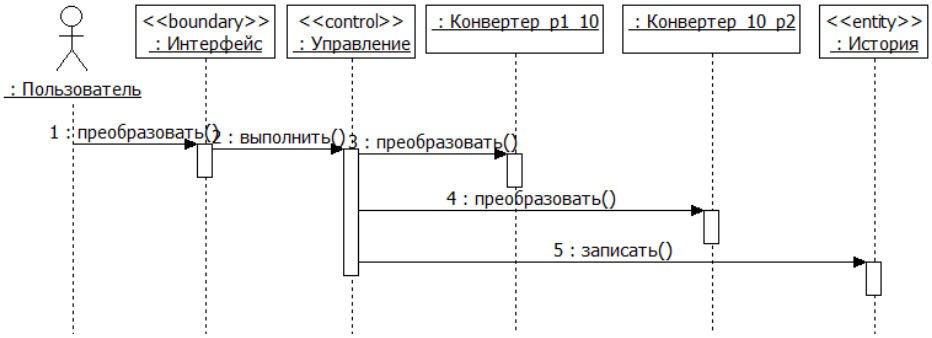


Рис 3. Поток событий для прецендента «Ввести число»

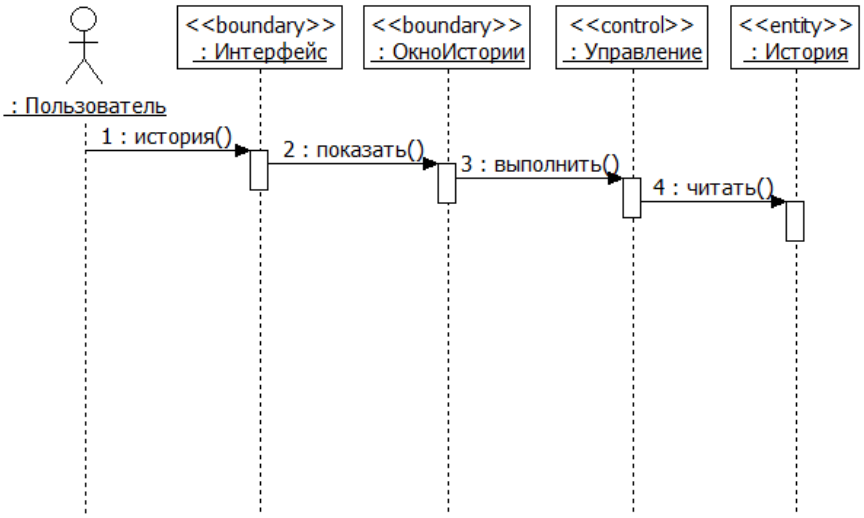


Рис 4. Поток событий для прецендента «История»

Рис 5. Диаграмма классов

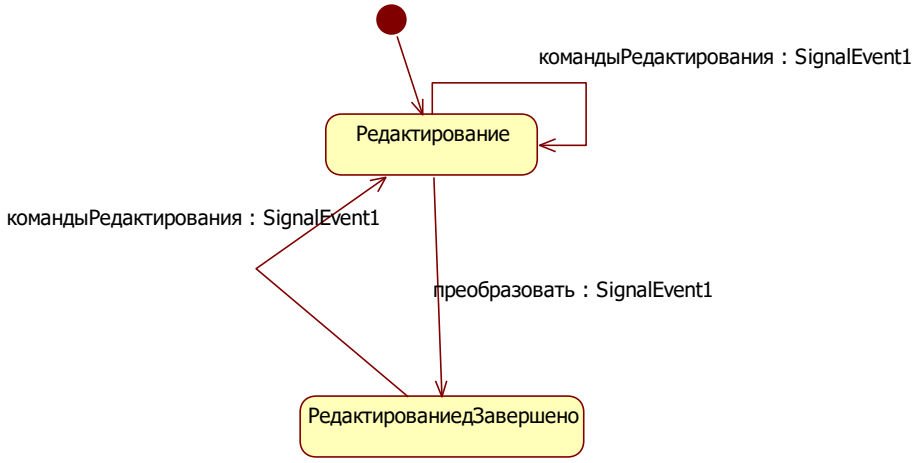
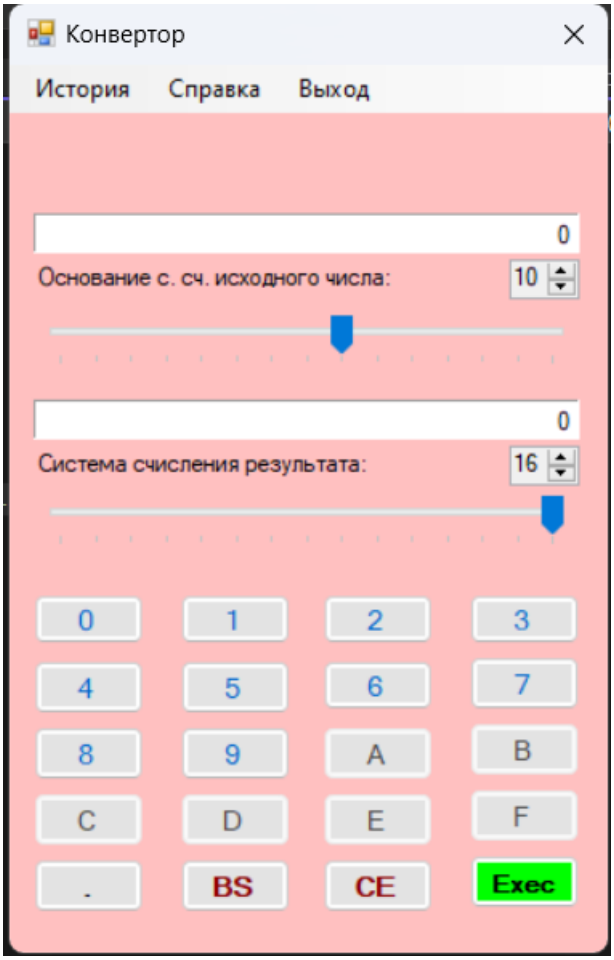


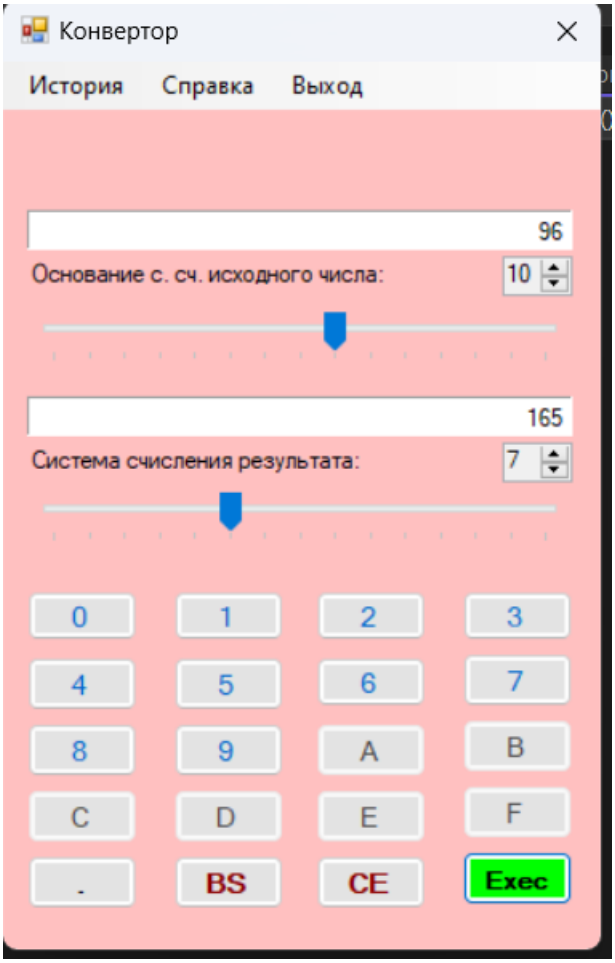
Рис 6. Диаграмма состояния класса «Управление»

Результаты тестирования программы

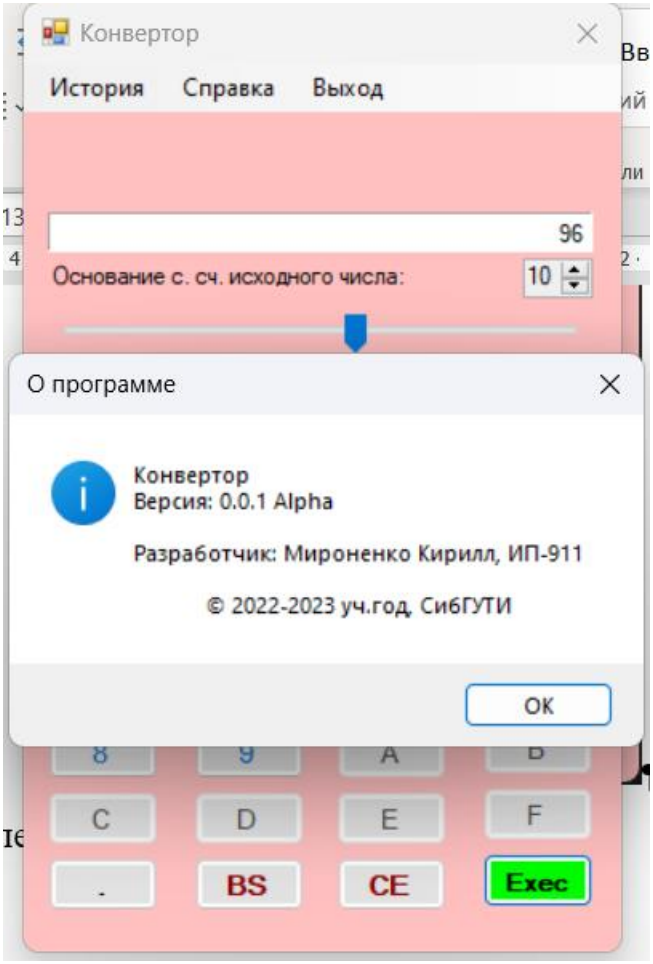
Начало работы:



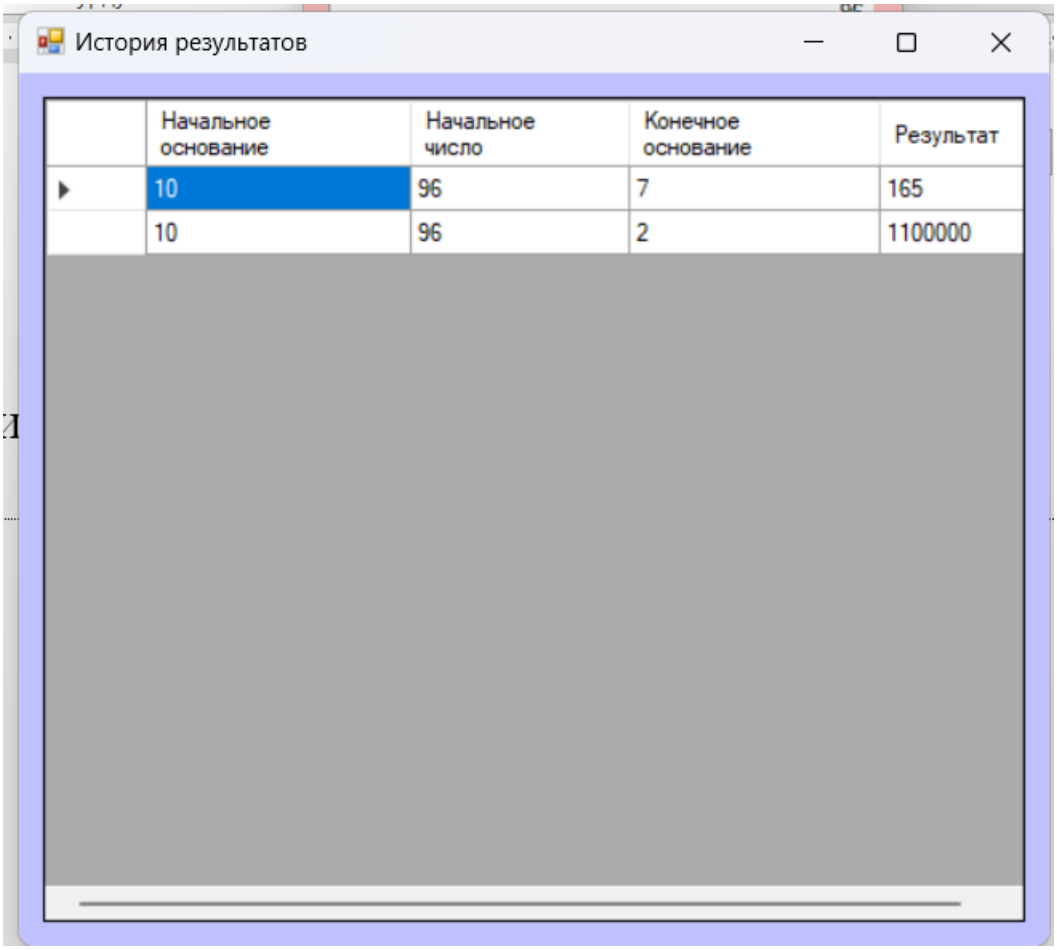
Ввод числа и его перевод в другую СС:



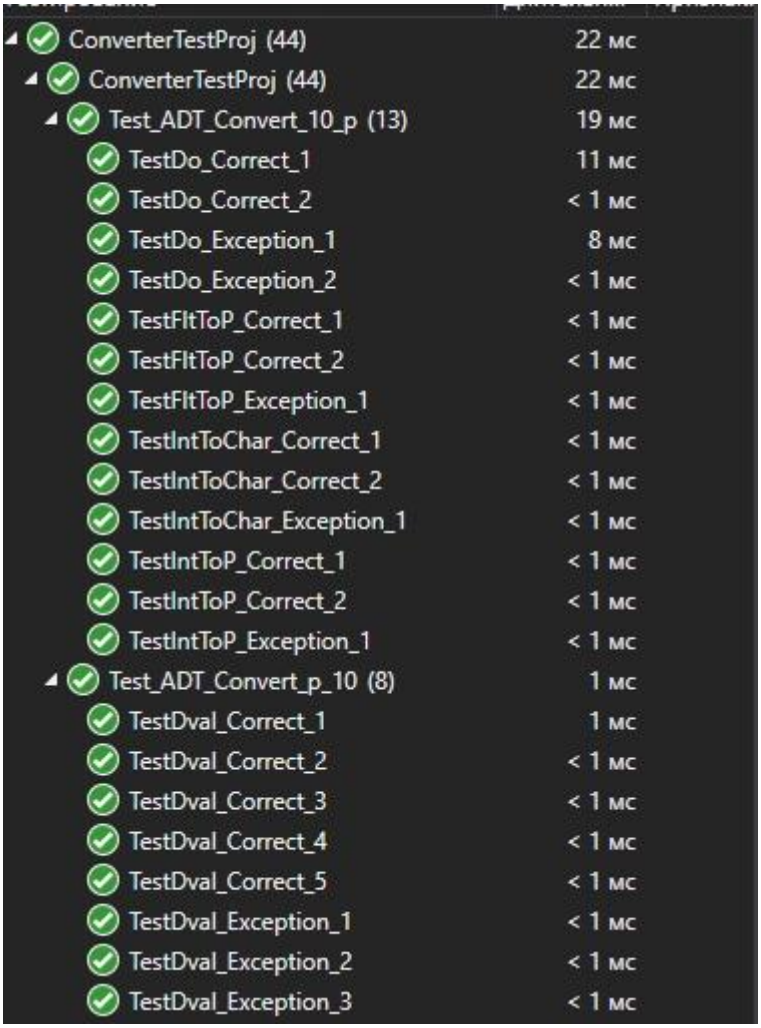
Окно справки:

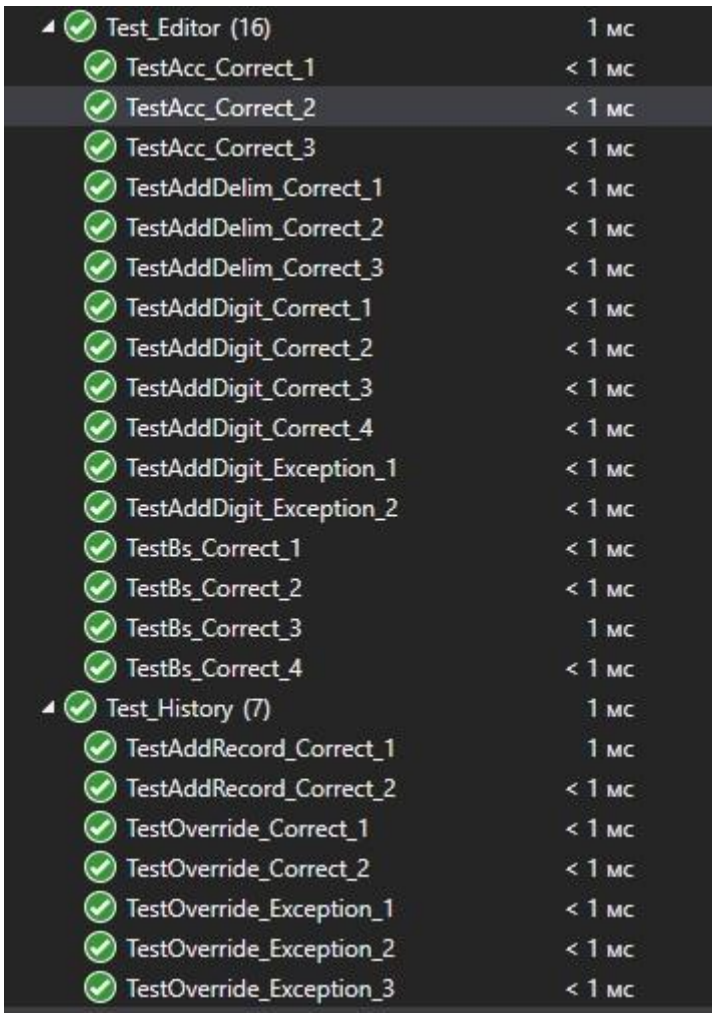

Конвертор
История
Справка
Выход
96
Основание с. сч. исходного числа:
10
О программе
Конвертор
Версия: 0.0.1 Alpha
Разработчик: Мироненко Кирилл, ИП-911
© 2022-2023 уч.год, СибГУТИ
OK
C
D
E
F
.
BS
CE
Exec

История:


История результатов
Начальное основание
Начальное число
Конечное основание
Результат
10
96
7
165
10
96
2
1100000

Результат работы тестов




Test_Editor (16) 1 мс
TestAcc_Correct_1 < 1 мс
TestAcc_Correct_2 < 1 мс
TestAcc_Correct_3 < 1 мс
TestAddDelim_Correct_1 < 1 мс
TestAddDelim_Correct_2 < 1 мс
TestAddDelim_Correct_3 < 1 мс
TestAddDigit_Correct_1 < 1 мс
TestAddDigit_Correct_2 < 1 мс
TestAddDigit_Correct_3 < 1 мс
TestAddDigit_Correct_4 < 1 мс
TestAddDigit_Exception_1 < 1 мс
TestAddDigit_Exception_2 < 1 мс
TestBs_Correct_1 < 1 мс
TestBs_Correct_2 < 1 мс
TestBs_Correct_3 1 мс
TestBs_Correct_4 < 1 мс
Test_History (7) 1 мс
TestAddRecord_Correct_1 1 мс
TestAddRecord_Correct_2 < 1 мс
TestOverride_Correct_1 < 1 мс
TestOverride_Correct_2 < 1 мс
TestOverride_Exception_1 < 1 мс
TestOverride_Exception_2 < 1 мс
TestOverride_Exception_3 < 1 мс

Вывод

По итогам проектной работы были проведены объектно-ориентированный анализ, проектирование и реализация приложения «Конвертор p1_p2» для преобразования действительных чисел из одной системы счисления в другую. В процессе работы было изучено: отношения между классами: ассоциация, агрегация, зависимость, их реализация средствами языка программирования высокого уровня; этапы разработки приложений в технологии ООП; элементы технологии визуального программирования; диаграммы языка UML для документирования разработки. Также были закреплены знания по разработке тестов для методов класса в проекте.

Листинг

1. Исходный код программы

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Converter
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
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.getNumber();
            trackBar1.Value = control_.Pin;
            trackBar2.Value = control_.Pout;
            label2.Text = "0";
            UpdateButtons();
        }

        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(18);
            label2.Text = "0";
        }

        private void trackBar2_Scroll(object sender, EventArgs e)
        {
            numericUpDown2.Value = trackBar2.Value;
            this.updateP2();
        }
        private void numericUpDown2_ValueChanged(object sender, EventArgs e)
        {
            trackBar2.Value = Convert.ToByte(numericUpDown2.Value);
            this.updateP2();
        }
        private void updateP2()
        {
            control_.Pout = trackBar2.Value;
            //label2.Text = control_.doCmnd(19);
        }

        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("Конвертор\nВерсия: 0.0.1 Alpha\n\nРазработчик: Мироненко
Кирилл, ИП-911\n\n            © 2022-2023 уч.год, СибГУТИ", "О программе",
MessageBoxButtons.OK, MessageBoxIcon.Information);

        }

        private void doCmnd(int j)
        {
            if (j == 19)
                label2.Text = control_.doCmnd(j);
            else
            {
                if (control_.St == ADT_Control_.State.Converted)
                    label1.Text = control_.doCmnd(18);
                label1.Text = control_.doCmnd(j);
                label2.Text = "0";
            }
        }


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

Form2.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Converter
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }
    }
}
```

Editor.cs

```
using System;
using System.Collections.Generic;
```

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

namespace Converter
{
    public class Editor
    {
        string number = "0";
        const string zero = "0";
        const string delim = ".";

        public string getNumber() { return number; }

        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;
        }

        public int acc()
        {
            if (number.Contains(delim))
            {
                string[] chs = number.Split('.');
                return chs[1].Length;
            }
            return 0;
        }
        public string addZero()
        {
            number += zero;
            return number;
        }
        public string addDelim()
        {
            if (number.Length == 0)
            {
                addZero();
            }
            if (number.Length > 0 && !number.Contains(delim))
                number += delim;
            return number;
        }
        public string bs()
        {
            if (number.Length > 1)
                number = number.Remove(number.Length - 1);
            else
                number = zero;
            return number;
        }
        public string clear()
        {
            number = "0";
            return number;
        }
        public string doEdit(int j)
        {
            if (j < 16)
```

```
            {
                addDigit(j);
            }
            switch (j)
            {
                case 16:
                    addDelim();
                    break;
                case 17:
                    bs();
                    break;
                case 18:
                    clear();
                    break;
                case 19:
                    break;
            }
            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
        {
            int p1, p2;
            string number1, number2;
            public Record(int p1, int p2, string number1, string number2)
            {
                this.p1 = p1;
                this.p2 = p2;
                this.number1 = number1;
                this.number2 = number2;
            }
            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 number1, string number2)
        {
            L.Add(new Record(p1, p2, number1, number2));
        }
```

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

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

        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;
            }
        }
    }
}
```

ADT_Control_.cs

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

namespace Converter
{
    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(int j)
```

```
        {
            if (j == 19)
            {
                double r = ADT_Convert_p_10.Dval(editor.getNumber(), (Int16)Pin);
                string res = ADT_Convert_10_p.Do(r, (Int32)Pout, Acc());
                St = State.Converted;
                history.addRecord(Pin, Pout, editor.getNumber(), 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);
        }
    }
}
```

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 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;
        }

        //Преобразовать целое в символ.
        public static char Int_to_char(int d)
        {
            if (d > 15 || d < 0)
            {
                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, int 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 buf = "";
            while (n > 0)
            {
                buf += Int_to_char((int)n % p);
                n /= p;
            }

            if (isNegative)
                buf += "-";

            char[] chs = buf.ToCharArray();
            Array.Reverse(chs);
            return new string(chs);
        }

        //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;
        }
    }
}
```

ADT_Convert_p_10.cs

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

```
namespace Converter
{
    public class ADT_Convert_p_10
    {
        //Преобразовать из с.сч. с основанием p
        //в с.сч. с основанием 10
        public static double Dval(string p_num, int p)
        {
            if (p < 2 || p > 16)
                throw new IndexOutOfRangeException();

            double buf = 0d;
            if (p_num.Contains("."))
            {
                string[] lr = p_num.Split('.');
                if (lr[0].Length == 0)
                    throw new Exception();
                char[] chs = lr[0].ToCharArray();
                Array.Reverse(chs);
                for (int i = 0; i < chs.Length; i++)
                {
                    if(Char_to_num(chs[i]) > p)
                        throw new Exception();
                    buf += Char_to_num(chs[i]) * Math.Pow(p, i);
                }
                char[] chsr = lr[1].ToCharArray();
                for (int i = 0; i < chsr.Length; i++)
                {
                    if (Char_to_num(chsr[i]) > p)
                        throw new Exception();
                    buf += Char_to_num(chsr[i]) * Math.Pow(p, -(i + 1));
                }
            }
            else
            {
                char[] chs = p_num.ToCharArray();
                Array.Reverse(chs);
                for (int i = 0; i < chs.Length; i++)
                {
                    if (Char_to_num(chs[i]) > p)
                        throw new Exception();
                    buf += Char_to_num(chs[i]) * Math.Pow(p, i);
                }
            }
            return buf;
        }

        //Преобразовать цифру в число
        public static double Char_to_num(char ch)
        {
            string allNums = "0123456789ABCDEF";
            if (!allNums.Contains(ch))
                throw new IndexOutOfRangeException();
            return allNums.IndexOf(ch);

        }

        //Преобразовать строку в число
        public static double Convert(string p_num, int p, double weight)
        {
            return 0d;
        }
    }
}
```

2. Исходный код тестов

UnitTest.cs

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

namespace ConverterTests
{
    [TestClass]
    public class Test_ADT_Convert_10_p
    {
        [TestMethod]
        public void TestDo_Correct_1()
        {
            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 TestDo_Correct_2()
        {
            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(System.IndexOutOfRangeException))]
        public void TestDo_Exception_1()
        {
            double n = -12312.1231;
            int p = -3;
            int c = 8;
            string Actual = Converter.ADT_Convert_10_p.Do(n, p, c);
        }

        [TestMethod]
        [ExpectedException(typeof(System.IndexOutOfRangeException))]
        public void TestDo_Exception_2()
        {
            double n = -12312.1231;
            int p = -3;
            int c = 8;
            string Actual = Converter.ADT_Convert_10_p.Do(n, p, c);
        }

        [TestMethod]
        public void TestIntToChar_Correct_1()
        {
            int n = 12;
            char ExpectedChar = 'C';
            char ActualChar = Converter.ADT_Convert_10_p.Int_to_char(n);
            Assert.AreEqual(ExpectedChar, ActualChar);
        }
```

```
[TestMethod]
public void TestIntToChar_Correct_2()
{
    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 TestIntToChar_Exception_1()
{
    int n = -12;
    Converter.ADT_Convert_10_p.Int_to_char(n);
}

[TestMethod]
public void TestIntToP_Correct_1()
{
    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 TestIntToP_Correct_2()
{
    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 TestIntToP_Exception_1()
{
    int n = 123;
    int p = -24;
    string Actual = Converter.ADT_Convert_10_p.Int_to_p(n, p);
}

[TestMethod]
public void TestFltToP_Correct_1()
{
    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 TestFltToP_Correct_2()
{
    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 TestFltToP_Exception_1()
    {
        double n = 1.5;
        int p = 12;
        int c = 3;
        string Actual = Converter.ADT_Convert_10_p.Flt_to_p(n, p, c);
    }
}


[TestClass]
public class Test_ADT_Convert_p_10
{
    [TestMethod]
    public void TestDval_Correct_1()
    {
        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 TestDval_Correct_2()
    {
        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 TestDval_Correct_3()
    {
        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 TestDval_Correct_4()
    {
        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 TestDval_Correct_5()
    {
        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 TestDval_Exception_1()
    {
        string Number = ".A";
        int P = 11;
        Converter.ADT_Convert_p_10.Dval(Number, P);
    }

    [TestMethod]
    [ExpectedException(typeof(System.IndexOutOfRangeException))]
    public void TestDval_Exception_2()
    {
        string Number = "AA";
        int P = 77;
        Converter.ADT_Convert_p_10.Dval(Number, P);
    }

    [TestMethod]
    [ExpectedException(typeof(System.Exception))]
    public void TestDval_Exception_3()
    {
        string Number = "FFF";
        int P = 2;
        Converter.ADT_Convert_p_10.Dval(Number, P);
    }
}

[TestClass]
public class Test_Editor
{
    [TestMethod]
    public void TestAddDigit_Correct_1()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.addDigit(0);
        string ExpectedValue = "0";
        string ActualValue = editor.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestAddDigit_Correct_2()
    {
        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.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestAddDigit_Correct_3()
    {
        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.getNumber();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDigit_Correct_4()
{
    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.getNumber();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
[ExpectedException(typeof(System.IndexOutOfRangeException))]
public void TestAddDigit_Exception_1()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(17);
}

[TestMethod]
[ExpectedException(typeof(System.IndexOutOfRangeException))]
public void TestAddDigit_Exception_2()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDigit(-12);
}

[TestMethod]
public void TestAcc_Correct_1()
{
    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 TestAcc_Correct_2()
{
    Converter.Editor editor = new Converter.Editor();
    int ExpectedValue = 0;
    int ActualValue = editor.acc();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
```

```
public void TestAcc_Correct_3()
{
    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 TestAddDelim_Correct_1()
{
    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.getNumber();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDelim_Correct_2()
{
    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.getNumber();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
public void TestAddDelim_Correct_3()
{
    Converter.Editor editor = new Converter.Editor();
    editor.addDelim();
    editor.addDelim();
    editor.addDelim();
    string ExpectedValue = "0.";
    string ActualValue = editor.getNumber();
    Assert.AreEqual(ExpectedValue, ActualValue);
}

[TestMethod]
```

```
    public void TestBs_Correct_1()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.bs();
        editor.bs();
        string ExpectedValue = "0";
        string ActualValue = editor.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs_Correct_2()
    {
        Converter.Editor editor = new Converter.Editor();
        editor.bs();
        editor.addDigit(1);
        editor.addDigit(2);
        editor.bs();
        string ExpectedValue = "1";
        string ActualValue = editor.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs_Correct_3()
    {
        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.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }

    [TestMethod]
    public void TestBs_Correct_4()
    {
        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.getNumber();
        Assert.AreEqual(ExpectedValue, ActualValue);
    }
}

[TestClass]
public class Test_History
{
    [TestMethod]
    public void TestAddRecord_Correct_1()
    {
        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 TestAddRecord_Correct_2()
        {
            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 TestOverride_Correct_1()
        {
            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 TestOverride_Correct_2()
        {
            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 TestOverride_Exception_1()
        {
            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 TestOverride_Exception_2()
        {
            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 TestOverride_Exception_3()
        {
            Converter.History history = new Converter.History();
            Converter.History.Record Value = new Converter.History.Record(12, 4, "11",
"11");
            history[0] = Value;
        }
    }
}
```