МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «Гомельский государственный технический

университет имени П.О. Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

Отчёт по лабораторной работе №9

По дисциплине «Объектно-ориентированное программирование»

**«Обработка исключительных ситуаций»**

Выполнил: студент

группы ИТИ-21

Говядкова П. Ю.

Принял: преподаватель

Карабчикова Е. А.

Гомель 2020

**Цель работы:** изучить основы синтаксиса объектно-ориентированного языка программирования, обработка исключительных ситуаций.

**Задание:**

1. Необходимо дополнить решение задачи из лабораторной работы №2, (Рисунок 1) обработкой всевозможных исключительных ситуаций и обеспечить надёжное функционирование классов.
2. При создании классов руководствоваться Code Convention.
3. Весь код должен содержать элементы документирования.
4. При обработке исключений использовать собственные – классы исключений, которые необходимо разместить в отдельной библиотеке
5. классов.
6. Разработать модульные тесты для верификации созданного класса.
7. Класс должен быть размещён в библиотеке классов.
8. Модульные тесты – в отдельном проекте.
9. Дополнить задачу графическим интерфейсом: в отдельном проекте реализовать для нечётных вариантов – WFP, для чётных – WFА.

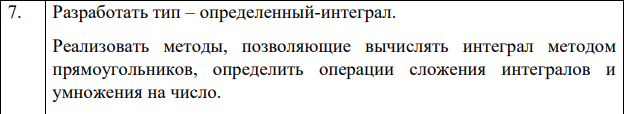


Рисунок 1 – Вариант задания

На рисунке 2 изображена структура решения.

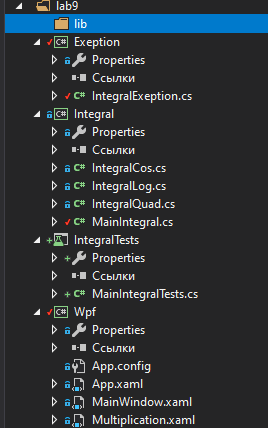


Рисунок 2 – Структура решения

На Рисунке 3 изображён вывод MainWindow.

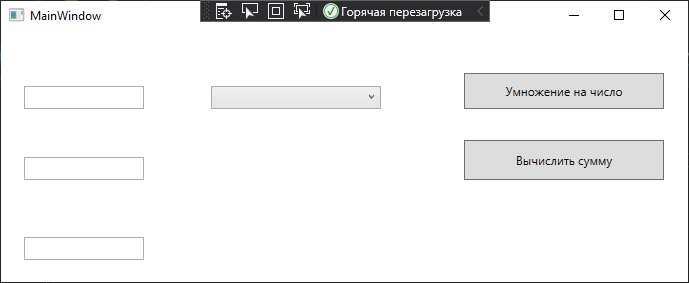


Рисунок 3 – Начало выполнения программы, вывод основной формы

На Рисунке 4 изображено выполнение пункта “Умножение интеграла на число”.

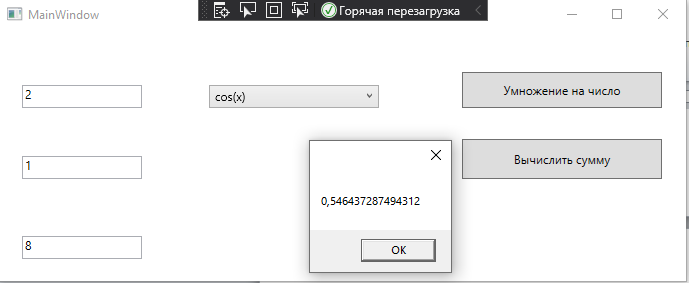


Рисунок 4 – Выполнение пункта “Умножение интеграла на число”

На Рисунке 5 изображено выполнение пункта “Умножение интеграла на число”, где вводим некорректные параметры.

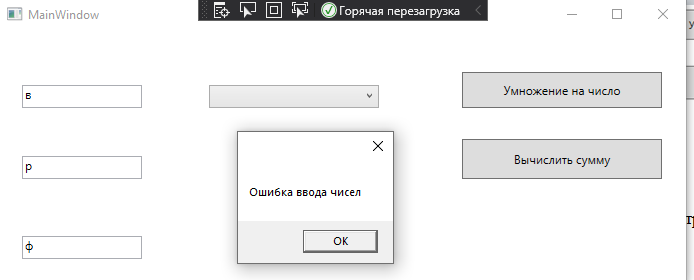


Рисунок 5 – Ввод некорректных параметров

На Рисунке 6 изображено выполнение пункта “Умножение интеграла на число”, где вводим верхний предел меньше нижнего.

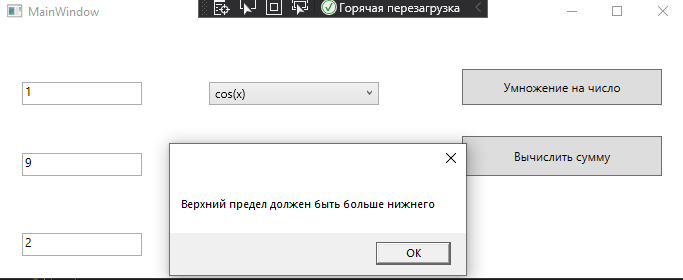


Рисунок 6 – Ввод некорректных параметров

На Рисунке 7 изображен вариант, если не выбрана функция.

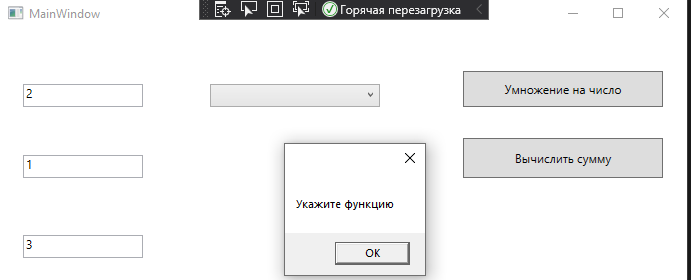


Рисунок 7 – Ввод некорректных параметров

На Рисунке 8 изображено выполнение пункта “Сложение интегралов”.

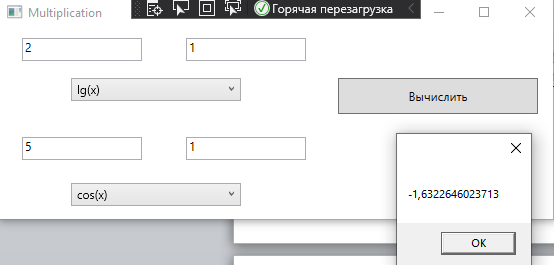


Рисунок 8 – Выполнение пункта “Сложение интегралов”.

На Рисунке 9 изображен вариант, если не указаны некорректные параметры.

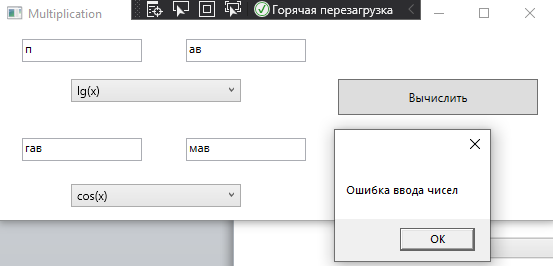


Рисунок 9 – Выполнение пункта “ Количество определённых покупок”

На Рисунке 10 изображен вариант, если не выбрана функция.

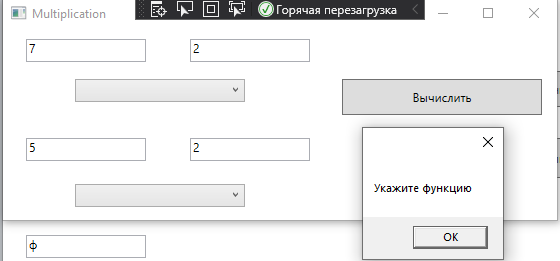


Рисунок 10 – Ввод некорректных параметров

На Рисунке 11 изображен вариант, если вводим верхний предел меньше нижнего.

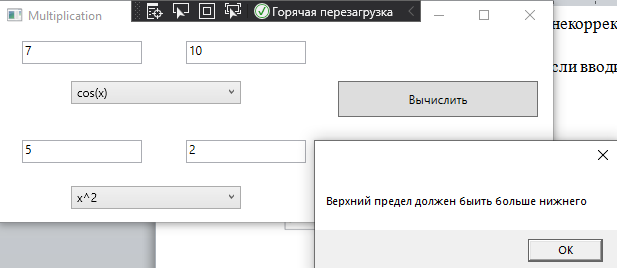


Рисунок 11 – Ввод некорректных параметров

На Рисунке 12 изображено выполнение тестов

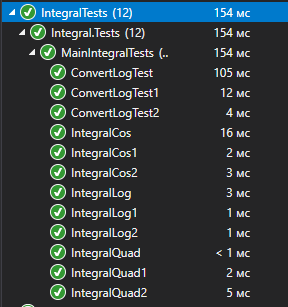


Рисунок 12 – Прохождение модульных тестов

На Рисунке 13 изображены созданные xml файлы

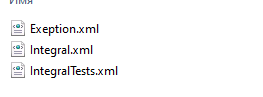


Рисунок 13 – Хml файлы

**Вывод:** в ходе лабораторной работы были изучены основы синтаксиса объектно-ориентированного языка программирования, обработка исключительных ситуаций.

**ПРИЛОЖЕНИЕ А**

**Листинг программы**

IntegralExeption

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Exeption

{

/// <summary>

/// IntegralExeption

/// </summary>

/// <seealso cref="System.Exception" />

public class IntegralExeption : Exception

{

/// <summary>

/// Initializes a new instance of the <see cref="IntegralExeption"/> class.

/// </summary>

/// <param name="text">The text.</param>

public IntegralExeption(string text) : base(text)

{

}

}

}

IntegralCos

using System;

namespace Integral

{

/// <summary>

///

/// </summary>

/// <seealso cref="Integral.MainIntegral" />

public class IntegralCos : MainIntegral

{

/// <summary>

/// Initializes a new instance of the <see cref="IntegralCos"/> class.

/// </summary>

/// <param name="lower\_limit">The lower limit.</param>

/// <param name="upper\_limit">The upper limit.</param>

public IntegralCos(double lower\_limit, double upper\_limit) : base(lower\_limit, upper\_limit)

{

}

/// <summary>

/// Functions the specified x.

/// </summary>

/// <param name="x">The x.</param>

/// <returns>

/// Returns a cos function

/// </returns>

public override double Func(double x)

{

return Math.Cos(x);

}

}

}

IntegralLog

using System;

namespace Integral

{

/// <summary>

///

/// </summary>

/// <seealso cref="Integral.MainIntegral" />

public class IntegralLog : MainIntegral

{

/// <summary>

/// Initializes a new instance of the <see cref="IntegralLog"/> class.

/// </summary>

/// <param name="lower\_limit">The lower limit.</param>

/// <param name="upper\_limit">The upper limit.</param>

public IntegralLog(double lower\_limit, double upper\_limit) : base(lower\_limit, upper\_limit)

{

}

/// <summary>

/// Functions the specified x.

/// </summary>

/// <param name="x">The x.</param>

/// <returns>

/// Returns a log function

/// </returns>

public override double Func(double x)

{

return Math.Log10(x);

}

}

}

IntegralQuad

using System;

namespace Integral

{

/// <summary>

///

/// </summary>

/// <seealso cref="Integral.MainIntegral" />

public class IntegralQuad : MainIntegral

{

/// <summary>

/// Initializes a new instance of the <see cref="IntegralQuad"/> class.

/// </summary>

/// <param name="lower\_limit">The lower limit.</param>

/// <param name="upper\_limit">The upper limit.</param>

public IntegralQuad(double lower\_limit, double upper\_limit) : base(lower\_limit, upper\_limit)

{

}

/// <summary>

/// Functions the specified x.

/// </summary>

/// <param name="x">The x.</param>

/// <returns>Returns a quadratic function</returns>

public override double Func(double x)

{

return Math.Pow(x, 2);

}

}

}

MainIntegral

using Exeption;

using System;

namespace Integral

{

/// <summary>

/// MainIntegral

/// </summary>

public class MainIntegral

{

/// <summary>

/// The lower limit

/// </summary>

private double \_lower\_limit;

/// <summary>

/// The upper limit

/// </summary>

private double \_upper\_limit;

/// <summary>

/// Initializes a new instance of the <see cref="MainIntegral"/> class.

/// </summary>

/// <param name="lower\_limit">The lower limit.</param>

/// <param name="upper\_limit">The upper limit.</param>

public MainIntegral(double lower\_limit, double upper\_limit)

{

\_lower\_limit = lower\_limit;

\_upper\_limit = upper\_limit;

}

/// <summary>

/// Converts the cos.

/// </summary>

/// <param name="new\_lower\_limit">The new lower limit.</param>

/// <param name="new\_upper\_limit">The new upper limit.</param>

/// <returns></returns>

/// <exception cref="IntegralExeption">Ошибка</exception>

public static MainIntegral ConvertCos(string new\_lower\_limit, string new\_upper\_limit)

{

double up;

double low;

try

{

low = Convert.ToDouble(new\_lower\_limit);

up = Convert.ToDouble(new\_upper\_limit);

}

catch

{

throw new IntegralExeption("Ошибка");

}

MainIntegral result = new IntegralCos(low, up);

return result;

}

/// <summary>

/// Converts the log.

/// </summary>

/// <param name="new\_lower\_limit">The new lower limit.</param>

/// <param name="new\_upper\_limit">The new upper limit.</param>

/// <returns></returns>

/// <exception cref="IntegralExeption">Ошибка</exception>

public static MainIntegral ConvertLog(string new\_lower\_limit, string new\_upper\_limit)

{

double up;

double low;

try

{

low = Convert.ToDouble(new\_lower\_limit);

up = Convert.ToDouble(new\_upper\_limit);

}

catch

{

throw new IntegralExeption("Ошибка");

}

MainIntegral result = new IntegralLog(low, up);

return result;

}

/// <summary>

/// Converts the quad.

/// </summary>

/// <param name="new\_lower\_limit">The new lower limit.</param>

/// <param name="new\_upper\_limit">The new upper limit.</param>

/// <returns></returns>

/// <exception cref="IntegralExeption">Ошибка</exception>

public static MainIntegral ConvertQuad(string new\_lower\_limit, string new\_upper\_limit)

{

double up;

double low;

try

{

low = Convert.ToDouble(new\_lower\_limit);

up = Convert.ToDouble(new\_upper\_limit);

}

catch

{

throw new IntegralExeption("Ошибка");

}

MainIntegral result = new IntegralQuad(low, up);

return result;

}

/// <summary>

/// Functions the specified x.

/// </summary>

/// <param name="x">The x.</param>

/// <returns></returns>

public virtual double Func(double x)

{

return x;

}

/// <summary>

/// Solves this instance.

/// </summary>

/// <returns></returns>

public double Solve()

{

double sum = 0;

double h = (\_upper\_limit - \_lower\_limit) / 1000;

for (int i = 0; i < 1000; ++i)

{

double x = \_lower\_limit + i \* h;

sum += Func(x);

}

double result = h \* sum;

return result;

}

/// <summary>

/// Implements the operator +.

/// </summary>

/// <param name="integral">The integral.</param>

/// <param name="integral1">The integral1.</param>

/// <returns>

/// The result of the operator.

/// </returns>

public static double operator +(MainIntegral integral, MainIntegral integral1)

{

return integral.Solve() + integral1.Solve();

}

/// <summary>

/// Implements the operator \*.

/// </summary>

/// <param name="number">The number.</param>

/// <param name="integral">The integral.</param>

/// <returns>

/// The result of the operator.

/// </returns>

public static double operator \*(double number, MainIntegral integral)

{

return number \* integral.Solve();

}

}

}

using Exeption;

using Integral;

using System;

using System.Windows;

using System.Windows.Controls;

namespace Wpf

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

/// <seealso cref="System.Windows.Window" />

/// <seealso cref="System.Windows.Markup.IComponentConnector" />

public partial class MainWindow : Window

{

/// <summary>

/// Initializes a new instance of the <see cref="MainWindow"/> class.

/// </summary>

public MainWindow()

{

InitializeComponent();

}

/// <summary>

/// Handles the TextChanged event of the TextBox control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the event data.</param>

private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

}

/// <summary>

/// Handles the 1 event of the TextBox\_TextChanged control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the event data.</param>

private void TextBox\_TextChanged\_1(object sender, TextChangedEventArgs e)

{

}

/// <summary>

/// Handles the SelectionChanged event of the ComboBox control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="SelectionChangedEventArgs"/> instance containing the event data.</param>

private void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

}

/// <summary>

/// Handles the Click event of the Button control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="RoutedEventArgs"/> instance containing the event data.</param>

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

string upper = Upper.Text;

string lower = Lower.Text;

if (upper == "" && lower == "") throw new IntegralExeption("Введите данные");

else

{

if (Convert.ToDouble(Lower.Text) > Convert.ToDouble(Upper.Text))

{

Upper.Text = "";

Lower.Text = "";

throw new IntegralExeption("Верхний предел должен быть больше нижнего");

}

else

{

string integral = List.Text;

double number = Convert.ToDouble(Number.Text);

if (integral == "lg(x)")

{

MainIntegral integralLog = MainIntegral.ConvertLog(lower, upper);

MessageBox.Show(Convert.ToString(number \* integralLog));

}

else if (integral == "cos(x)")

{

MainIntegral integralCos = MainIntegral.ConvertCos(lower, upper);

MessageBox.Show(Convert.ToString(number \* integralCos));

}

else if (integral == "x^2")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower, upper);

MessageBox.Show(Convert.ToString(number \* integralQuad));

}

else throw new IntegralExeption("Укажите функцию");

}

}

}

catch (IntegralExeption ex) { MessageBox.Show(ex.Message); }

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Handles the 1 event of the Button\_Click control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="RoutedEventArgs"/> instance containing the event data.</param>

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

Multiplication multiplication = new Multiplication();

multiplication.Show();

}

}

}

using Exeption;

using Integral;

using System;

using System.Windows;

using System.Windows.Controls;

namespace Wpf

{

/// <summary>

/// Логика взаимодействия для Multiplication.xaml

/// </summary>

/// <seealso cref="System.Windows.Window" />

/// <seealso cref="System.Windows.Markup.IComponentConnector" />

public partial class Multiplication : Window

{

/// <summary>

/// Initializes a new instance of the <see cref="Multiplication"/> class.

/// </summary>

public Multiplication()

{

InitializeComponent();

}

/// <summary>

/// Handles the Click event of the Button control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="RoutedEventArgs"/> instance containing the event data.</param>

private void Button\_Click(object sender, RoutedEventArgs e)

{

try

{

string upper = Upper.Text;

string lower = Lower.Text;

string upper1 = Upper1.Text;

string lower1 = Lower1.Text;

if (upper == "" && lower == "" && upper1 == "" && lower1 == "") throw new IntegralExeption("Введите данные");

else

{

if (Convert.ToDouble(lower) > Convert.ToDouble(upper) || (Convert.ToDouble(lower1) > Convert.ToDouble(upper1)))

{

Upper.Text = "";

Lower.Text = "";

Upper1.Text = "";

Lower1.Text = "";

throw new IntegralExeption("Верхний предел должен быть больше нижнего");

}

else

{

string integral = List.Text;

string integral1 = List1.Text;

if (integral == "lg(x)" && integral1 == "cos(x)")

{

MainIntegral integralLog = MainIntegral.ConvertLog(lower, upper);

MainIntegral integralCos = MainIntegral.ConvertCos(lower1, upper1);

MessageBox.Show(Convert.ToString(integralCos + integralLog));

}

else if (integral == "cos(x)" && integral1 == "lg(x)")

{

MainIntegral integralLog = MainIntegral.ConvertLog(lower1, upper1);

MainIntegral integralCos = MainIntegral.ConvertCos(lower, upper);

MessageBox.Show(Convert.ToString(integralCos + integralLog));

}

else if (integral == "x^2" && integral1 == "cos(x)")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower, upper);

MainIntegral integralCos = MainIntegral.ConvertCos(lower1, upper1);

MessageBox.Show(Convert.ToString(integralCos + integralQuad));

}

else if (integral == "cos(x)" && integral1 == "x^2")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower1, upper1);

MainIntegral integralCos = MainIntegral.ConvertCos(lower, upper);

MessageBox.Show(Convert.ToString(integralCos + integralQuad));

}

else if (integral == "cos(x)" && integral1 == "cos(x)")

{

MainIntegral integralCos = MainIntegral.ConvertCos(lower, upper);

MainIntegral integralCos1 = MainIntegral.ConvertCos(lower1, upper1);

MessageBox.Show(Convert.ToString(integralCos + integralCos1));

}

else if (integral == "x^2" && integral1 == "x^2")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower, upper);

MainIntegral integralQuad1 = MainIntegral.ConvertQuad(lower1, upper1);

MessageBox.Show(Convert.ToString(integralQuad1 + integralQuad));

}

else if (integral == "lg(x)" && integral1 == "x^2")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower1, upper1);

MainIntegral integralLog = MainIntegral.ConvertLog(lower, upper);

MessageBox.Show(Convert.ToString(integralLog + integralQuad));

}

else if (integral == "x^2" && integral1 == "lg(x)")

{

MainIntegral integralQuad = MainIntegral.ConvertQuad(lower, upper);

MainIntegral integralLog = MainIntegral.ConvertLog(lower1, upper1);

MessageBox.Show(Convert.ToString(integralQuad + integralLog));

}

else if (integral == "lg(x)" && integral1 == "lg(x)")

{

MainIntegral integralLog = MainIntegral.ConvertLog(lower, upper);

MainIntegral integralLog1 = MainIntegral.ConvertLog(lower1, upper1);

MessageBox.Show(Convert.ToString(integralLog1 + integralLog));

}

else throw new IntegralExeption("Укажите функцию");

}

}

}

catch (IntegralExeption ex) { MessageBox.Show(ex.Message); }

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

/// <summary>

/// Handles the SelectionChanged event of the ComboBox control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="SelectionChangedEventArgs"/> instance containing the event data.</param>

private void ComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

}

/// <summary>

/// Handles the 1 event of the TextBox\_TextChanged control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the event data.</param>

private void TextBox\_TextChanged\_1(object sender, TextChangedEventArgs e)

{

}

/// <summary>

/// Handles the TextChanged event of the TextBox control.

/// </summary>

/// <param name="sender">The source of the event.</param>

/// <param name="e">The <see cref="TextChangedEventArgs"/> instance containing the event data.</param>

private void TextBox\_TextChanged(object sender, TextChangedEventArgs e)

{

}

}

}