ФГАОУ ВО «КФУ им. В.И. Вернадского»
Физико-технический институт (структурное подразделение)

Кафедра компьютерной инженерии и моделирования

Скибинский Дмитрий Константинович

отчет по практической работе №3
по дисциплине **«ОБЪЕКТНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ»**

Направление подготовки:
09.03.04 "Программная инженерия"

Оценка 95

Симферополь, 2024

Практическая работа №3
Тема: Обработка исключительных ситуаций

Цель работы: Научиться на практике обрабатывать исключения, генерировать собственные исключения, отлавливать исключения различных типов, проверять числовые данные на выход за границы значений.

Описание ключевых понятий:

Exception (Исключение) — это ошибка, которая возникает во время выполнения программы. В C# исключения представлены классами, наследуемыми от базового класса System.Exception. Примеры встроенных
исключений: NullReferenceException, DivideByZeroException, ArgumentException,

Try/Catch, Finally:

try: Блок кода, в котором может возникнуть исключение.
catch: Блок кода, который выполняется, если в блоке try возникает исключение. Можно указать конкретный тип исключения для обработки.
finally: Блок кода, который выполняется всегда, независимо от того, было ли выброшено исключение. Обычно используется для освобождения ресурсов.

Checked и Unchecked:

checked: Контекст, в котором арифметические операции проверяются на переполнение. Если переполнение происходит, выбрасывается исключение OverflowException.
unchecked: Контекст, в котором арифметические операции не проверяются на переполнение. Переполнение игнорируется.

Throw - используется для явного выброса исключения. Можно использовать для повторного выброса исключения или создания нового исключения.

Перед выполнением лабораторной работы изучена следующая литература:

1. Изучены презентацию лектора курса: «Обработка исключений в C# » (материалы доступны в "облаке" на Mail.ru и в Moodle КФУ).
2. Сайт Metanit.com
3. Справочник по C#. Корпорация Microsoft. http://msdn.microsoft.com/ru-ru/library/618ayhy6.aspx
4. Биллиг В.А. Основы программирования на C#. Интернет-университет информационных технологий. http://www.intuit.ru/studies/courses/2247/18/info
5. Павловская Т. Программирование на языке высокого уровня C#. http://www.intuit.ru/studies/courses/629/485/info
6. Руководство по программированию на C#. Корпорация Microsoft. http://msdn.microsoft.com/ru-ru/library/67ef8sbd.aspx
7. Корпорация Microsoft. C#. Спецификация языка.

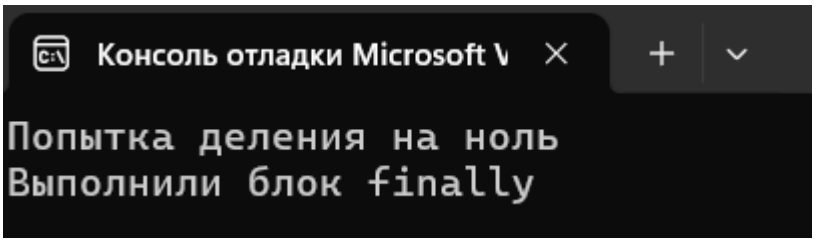
Выполнены 4 задания, описанных в методических указания к выполнению лабораторных работ.

Тренировочное задание.

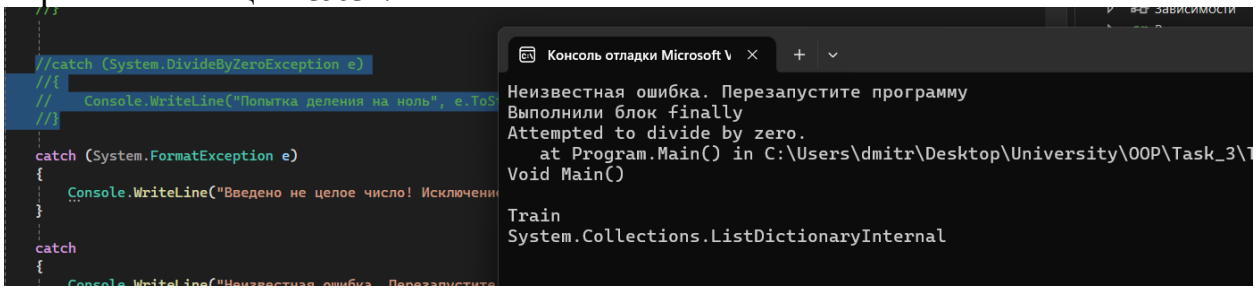
Для начала возьмем и переместим код в программную среду и добавить try catch для дополнения структуры

```
Ссылок: 0
static void Main()
{
    try
    {
        int i = 0;
        int x;
        float y;

        try
        {
            x = 5;
            y = x / i;
            Console.WriteLine($"x={0}, y= {1}", x, y);
        }

        //catch
        //{
        //    Console.WriteLine("Неизвестная ошибка. Перезапустите программу");
        //    throw;
        //}


        catch (System.DivideByZeroException e)
        {
            Console.WriteLine("Попытка деления на ноль", e.ToString());
        }

        catch (System.FormatException e)
        {
            Console.WriteLine("Введено не целое число! Исключение", e.ToString());
        }
```

1. Для начала поставим значение I = 0 и убедимся, что отрабатывают Exceptions.



2. Закомментируем блок обработки ошибки деления на 0. В итоге отработает общий catch.



3. Поставим на первое место общий catch и просмотрим как поведет программа. В итоге среда программирования предупредит, что это является ошибкой, тк универсальная конструкция закроет все остальные обработки исключений

```
try
{
    x = 5;
    y = x / i;
    Console.WriteLine($"x={0}, y= {1}", x, y);
}

catch
{
    Console.WriteLine("Неизвестная ошибка. Перезапустите программу");
    throw;
}


catch (System.DivideByZeroException e)
{
```

CS1017: Конструкции catch не могут использоваться после универсальной конструкции catch оператора try

Показать возможные решения (Alt+ВВОДилиCtrl+ю)

4. Уберем обработку ошибок. Visual Studio сообщит нам о том, что try не может существовать без catch или finally

```
try
{
    int i = 0;
    int x;
    float y;

    try
    {
        x = 5;
        y = x / i;
        Console.WriteLine($"x={0}, y= {1}", x, y);
    }
```

```
try
{
```

CS1524: Требуется catch или finally.

Показать возможные решения (Alt+ВВОДилиCtrl+ю)

```
//catch (System.FormatException e)
//{
```

5. Если исключение блок Try – Catch не обработал, это приведет к краху программы. Программа никогда не закончиться аварийнно если вся программа написана в блоке Try и присутствует общий Catch.

Задание 1. Написать с использованием конструкций Switch, Try, Catch метод анализа опасных состояний оборудования компьютера.

Приложение, которое имитирует процесс безопасного выключения ядерного реактора. Приложение состоит из нескольких классов и методов, которые выполняют различные задачи, связанные с мониторингом и управлением системы реактора.

Для генерации исключений будем использовать rand.Next и сравнивая с числом будет выдавать исключения

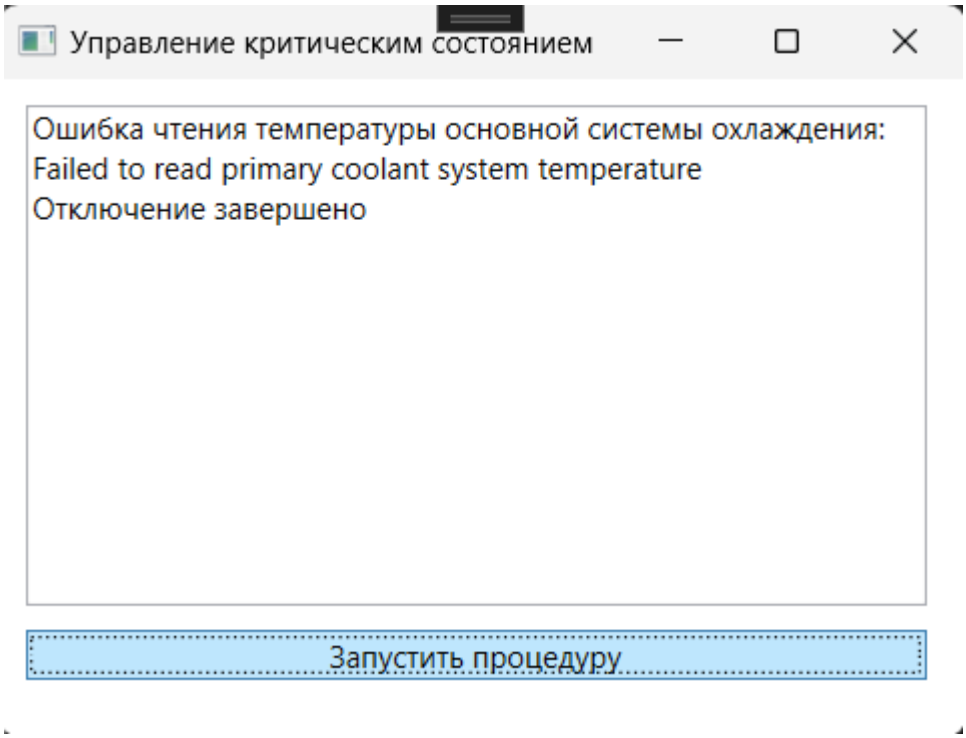
```
public SuccessFailureResult DisconnectPowerGenerator()
{
    SuccessFailureResult r = SuccessFailureResult.Fail;
    if (rand.Next(1, 10) > 2) r = SuccessFailureResult.Success;
    if (rand.Next(1, 20) > 18) throw new PowerGeneratorCommsException("Net
    return r;
}
```

```
public CoolantSystemStatus VerifyPrimaryCoolantSystem()
{
    CoolantSystemStatus c = CoolantSystemStatus.Fail;
    int r = rand.Next(1, 10);
    if (r > 5)
    {
        c = CoolantSystemStatus.OK;
    }
    else if (r > 2)
    {
        c = CoolantSystemStatus.Check;
    }
    if (rand.Next(1, 20) > 18) throw new CoolantTemperatureReadException("Failed to
    if (rand.Next(1, 20) > 18) throw new CoolantPressureReadException("Failed to rea
    return c;
}
```

```
public CoolantSystemStatus VerifyBackupCoolantSystem()
{
    CoolantSystemStatus c = CoolantSystemStatus.Fail;
    int r = rand.Next(1, 10);
    if (r > 5)
    {
        c = CoolantSystemStatus.OK;
    }
    else if (r > 2)
    {
        c = CoolantSystemStatus.Check;
    }
    if (rand.Next(1, 20) > 19) throw new CoolantTemperatureReadException("Fa
    if (rand.Next(1, 20) > 19) throw new CoolantPressureReadException("Faile
    return c;
}
```

1. **DisconnectPowerGenerator**:
    - Имитирует отключение от внешней системы генерации энергии.
    - Может выбросить исключение PowerGeneratorCommsException.
2. **VerifyPrimaryCoolantSystem**:
    - Проверяет статус основной системы охлаждения.
    - Может выбросить исключения CoolantTemperatureReadException и CoolantPressureReadException.
3. **VerifyBackupCoolantSystem**:
    - Проверяет статус резервной системы охлаждения.
    - Может выбросить исключения CoolantTemperatureReadException и CoolantPressureReadException.
4. **GetCoreTemperature**:
    - Считывает температуру из ядра реактора.
    - Может выбросить исключение CoreTemperatureReadException.
5. **InsertRodCluster**:
    - Вставляет управляющие стержни для выключения реактора.
    - Может выбросить исключение RodClusterReleaseException.
6. **GetRadiationLevel**:
    - Считывает уровень радиации из ядра реактора.
    - Может выбросить исключение CoreRadiationLevelReadException.
7. **SignalShutdownComplete**:
    - Отправляет сообщение о завершении выключения.
    - Может выбросить исключение SignallingException.

Пример работы программы:

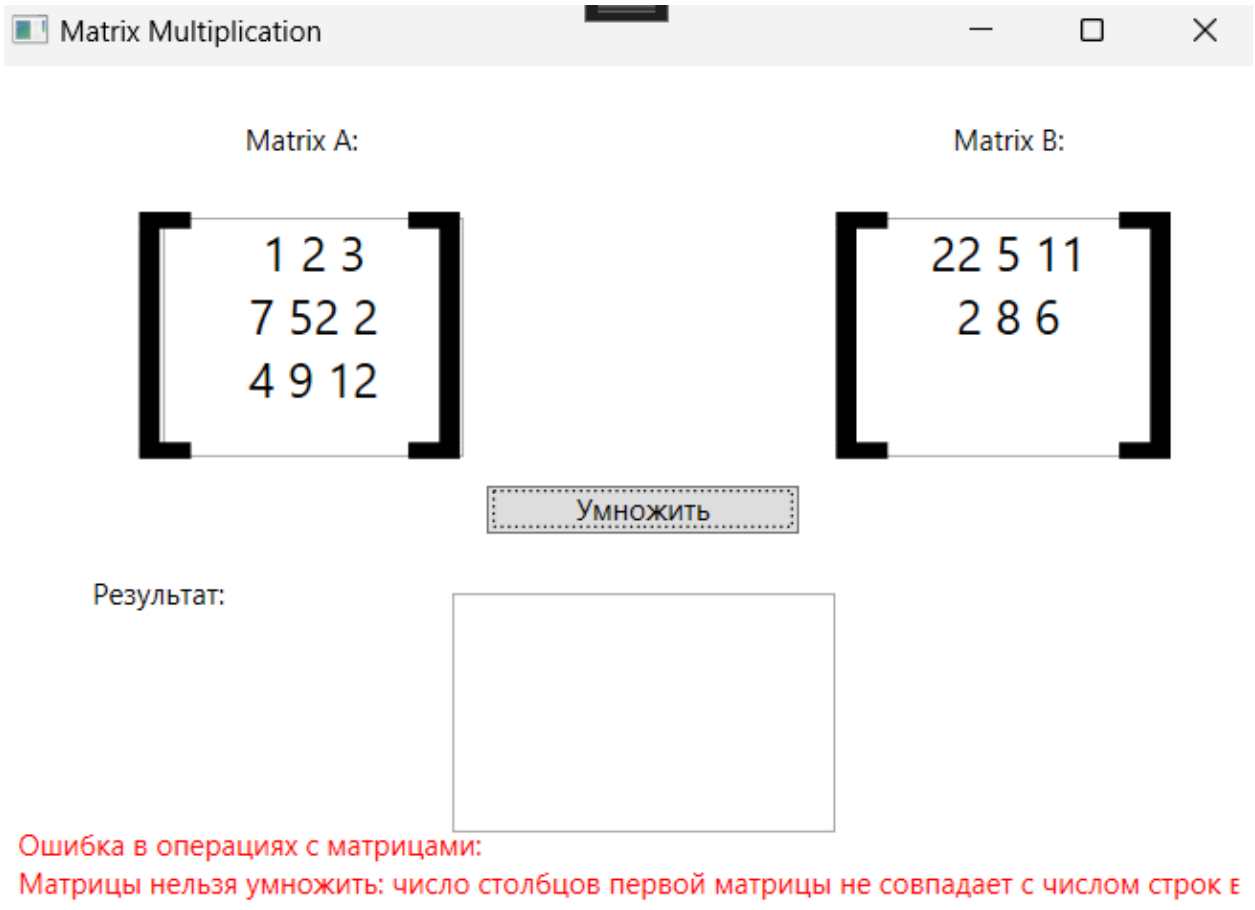


Задание 2. Обработка исключительных состояний при вычислении произведения матриц

Для решения этого задания воспользуемся кодом из второй лабораторной работы и добавим обработку всех исключений

```
        // Проверка возможности умножения матриц
        if (matrixA[0].Length != matrixB.Length)
        {
            throw new InvalidOperationException("Матрицы нельзя умножить: число столбцов первой матрицы не совпадает с числом строк второй матрицы.");
        }

        // Выполнение умножения матриц
        var result = Multiply(matrixA, matrixB);
        Result.Text = MatrixToString(result);
    }
    catch (FormatException ex)
    {
        // Ошибка при преобразовании строк в числа
        ErrorTextBlock.Text = $"Ошибка формата данных: \n{ex.Message}\nПроверьте, что все элементы матрицы — это числа.";
    }
    catch (InvalidOperationException ex)
    {
        // Ошибка при неправильной размерности матриц
        ErrorTextBlock.Text = $"Ошибка в операциях с матрицами: \n{ex.Message}";
    }
    catch (IndexOutOfRangeException ex)
    {
        // Ошибка доступа к элементам массивов (например, если одна из строк пустая)
        ErrorTextBlock.Text = $"Ошибка доступа к элементам матрицы: \n{ex.Message}\nПроверьте структуру введенных данных.";
    }
    catch (Exception ex)
    {
        // Ловим любые другие необработанные исключения
        ErrorTextBlock.Text = $"Произошла неизвестная ошибка: \n{ex.Message}";
    }
}
```

Исключение неправильного ввода матриц оставим. Добавим исключения FormatException, InvalideOperationException, IndexOutOfRange, Exception.



1. FormatException. Это исключение возникает, когда метод, который выполняет преобразование строки в число (например, double.Parse), не может распознать входную строку как допустимое число.

2. InvalidOperationException. Это исключение возникает, когда операция не является допустимой в текущем состоянии объекта. В вашем случае, это происходит, если размерности матриц не совпадают для выполнения умножения.

3. IndexOutOfRangeException. Это исключение возникает, когда вы пытаетесь получить доступ к элементу массива или коллекции по индексу, который находится за пределами допустимого диапазона.

4. Exception. Это базовый класс для всех исключений в .NET. Если возникает исключение, которое не было обработано предыдущими блоками catch, оно будет перехвачено этим блоком.

Задание 3. Использование Checked AnChecked для обработки переполнения целых чисел.

Для решения этой задачи будем принимать два числа int и будем проверять переполнения с помощью checked

```
Ссылок: 0
public MainWindow()
{
    InitializeComponent();
}

Ссылок: 0
private void MultiplyButton_Click(object sender, RoutedEventArgs e)
{
    try
    {
        int number1 = int.Parse(Number1TextBox.Text);
        int number2 = int.Parse(Number2TextBox.Text);

        checked
        {
            int result = number1 * number2;
            ResultTextBlock.Text = result.ToString();
        }
    }
    catch (OverflowException ex)
    {
        ResultTextBlock.Text = "Переполнение!";
        ResultTextBlock.Foreground = System.Windows.Media.Brushes.Red;
    }
    catch (FormatException ex)
    {
        ResultTextBlock.Text = "Неверный формат числа!";
        ResultTextBlock.Foreground = System.Windows.Media.Brushes.Red;
    }
    catch (Exception ex)
    {
        ResultTextBlock.Text = "Произошла ошибка: " + ex.Message;
        ResultTextBlock.Foreground = System.Windows.Media.Brushes.Red;
    }
}
```

Напишем обработку трех исключений: для переполнения, для неверного формата и общее

Задание 4. Подпишите отчет к работе №3 хэшем строки, состоящей из вашей фамилии, имени и отчества, написанных через запятые

Для выполнения задания сделаем функцию которая будет принимать строку и метод хэширования и возвращать преобразованную строку.

```
Ссылок: 2
static string ComputeHash(string input, HashAlgorithm hashAlgorithm)
{
    // Преобразуем строку в массив байтов
    byte[] inputBytes = Encoding.UTF8.GetBytes(input);

    // Получаем хэш в виде массива байтов
    byte[] hashBytes = hashAlgorithm.ComputeHash(inputBytes);

    // Преобразуем хэш из массива в строку шестнадцатеричных символов
    return Convert.ToHexString(hashBytes).ToLowerInvariant();
}
}
```

Используя библиотеку System.Security.Cryptography нам становятся доступны методы хэширования такие как SHA256 и MD5. В методе Main создадим строку, которую мы дальше и передадим алгоритму

```
Ссылок: 0
static void Main()
{

    string fullName = "Скибинский, Дмитрий, Константинович";

    using (SHA256 sha256 = SHA256.Create())
    {
        string hash = ComputeHash(fullName, sha256);
        Console.WriteLine($"SHA256 хэш: {hash}");
    }

    using (MD5 md5 = MD5.Create())
    {
        string hash = ComputeHash(fullName, md5);
        Console.WriteLine($"MD5 хэш: {hash}");
    }
}
```

Ответы на вопросы:

1. Какие виды исключений вы знаете?

В C# существует множество видов исключений, которые могут возникать в различных ситуациях. Некоторые из наиболее распространенных исключений включают:

- **System.Exception**: Базовый класс для всех исключений в .NET.

- **System.ArgumentException**: Возникает, когда метод получает недопустимый аргумент.
- **System.ArgumentNullException**: Специализация ArgumentException, возникающая, когда аргумент равен null.
- **System.InvalidOperationException**: Возникает, когда метод вызывается в недопустимом состоянии объекта.
- **System.IndexOutOfRangeException**: Возникает при попытке обратиться к элементу массива или коллекции с индексом, выходящим за его границы.
- **System.DivideByZeroException**: Возникает при попытке деления на ноль.
- **System.FormatException**: Возникает, когда строка имеет неправильный формат для преобразования в другой тип.
- **System.NotImplementedException**: Возникает, когда метод или операция не реализованы.
- **System.NotSupportedException**: Возникает, когда метод или операция не поддерживаются.

2. Как написать класс - потомок Exeption?

Для создания собственного исключения в C# нужно создать класс, который наследуется от System.Exception или от любого другого класса исключений.

```
public class MyCustomException : Exception
{
  public MyCustomException() : base() { }

  public MyCustomException(string message) : base(message) { }

  public MyCustomException(string message, Exception innerException) :
base(message, innerException) { }
```

3. С помощью какого служебного слова происходит выброс собственного исключения?

Для выброса собственного исключения используется служебное слово throw

4. Что происходит если в методе не будет обработано исключение?

Если исключение не будет обработано в методе, оно будет передано вызывающему методу. Если исключение не будет обработано ни в одном из вызывающих методов, оно достигнет точки входа в приложение (например, Main метода), и приложение завершит свою работу с ошибкой.

5. Как можно отследить переполнение арифметического типа?

Использование ключевых слов checked и unchecked поможет отследить переполнение арифметического типа. Checked: вызывает

исключение OverflowException, если происходит переполнение. Unchecked: Игнорирует переполнение и просто обрезает результат.

6. Класс Exeption и его методы.

Класс Exception является базовым классом для всех исключений в .NET. Он предоставляет множество свойств и методов, которые могут быть полезны при обработке исключений. Вот некоторые из них:

Свойства:

- **Message**: Возвращает сообщение, описывающее текущее исключение.
- **InnerException**: Возвращает экземпляр исключения, которое вызвало текущее исключение.
- **Source**: Возвращает или задает имя приложения или объекта, вызвавшего ошибку.
- **TargetSite**: Возвращает метод, создавший текущее исключение.

Методы:

- **ToString()**: Возвращает строку, представляющую текущее исключение.
- **GetBaseException()**: Возвращает исключение, которое является корневой причиной одного или нескольких последующих исключений.

7. Стратегия использования специализированных и общих исключений.

При обработке исключений рекомендуется использовать специализированные исключения, когда это возможно, так как они предоставляют более точную информацию о проблеме. Общие исключения, такие как Exception, следует использовать только в крайних случаях, когда тип исключения неизвестен или когда необходимо обработать все возможные исключения.

8. Что произойдет если сначала идет блок catch с общим исключением, а затем со специфическим?

Если сначала идет блок catch с общим исключением (например, catch (Exception ex)), а затем со специфическим (например, catch (ArgumentNullException ex)), то специфический блок catch никогда не будет выполнен. Это происходит потому, что общий блок catch будет перехватывать все исключения, включая специфические. Современная версия Visual Studio не позволяет писать общее исключение выше, чем специфичные

9. Можно убрать все внутренние блоки catch?

Да, можно убрать все внутренние блоки catch, если вам не нужно обрабатывать исключения в текущем методе. В этом случае исключение будет передано вызывающему методу. Однако, если исключение не будет обработано нигде в стеке вызовов, оно достигнет точки входа в приложение и приведет к его завершению с ошибкой.

10. Можно ли оставить только блок Try?

Нет, блок try должен быть сопряжен с блоком catch или finally (или обоими). Если вы оставите только блок try, компилятор выдаст ошибку.

11. Если исключение в блоке Try – Catch не поймано, означает ли это крах программы.

Да, если исключение в блоке try-catch не поймано, оно будет передано вызывающему методу. Если исключение не будет обработано ни в одном из вызывающих методов, оно достигнет точки входа в приложение (например, метода Main), и приложение завершит свою работу с ошибкой. Это может привести к аварийному завершению программы.

12. Можно ли написать программу, которая никогда аварийно не завершается?

Теоретически, можно написать программу, которая никогда не завершается аварийно, если учесть все возможные исключительные ситуации и корректно их обработать. Вторым вариантом решения этой задачи является написания всей программы (большей ее части) в блоке try-catch.

13. Какой метод предпочтительней throw или throw(ex), где Exeption ex?

Предпочтительнее использовать просто throw, а не throw(ex). Разница между ними заключается в том, как они влияют на стек вызовов.

- **throw**: При повторном выбросе исключения с помощью throw без аргументов, стек вызовов остается неизменным. Это означает, что стек вызовов будет содержать информацию о том, где исключение было изначально выброшено.
- **throw(ex)**: При использовании throw(ex) стек вызовов сбрасывается до текущего метода. Это означает, что информация о том, где исключение было изначально выброшено, будет потеряна.

Представлены 4 проекта, реализованных в Visual Studio Community 2022. Проекты представлены преподавателю в электронной форме, продемонстрирована их работоспособность, разъяснены детали программного кода.