Министерство образования, науки и молодежной политики
Краснодарского края
Государственное бюджетное профессиональное образовательное учреждение
Краснодарского края «Ейский полипрофильный колледж»

ПОДДЕРЖКА И ТЕСТИРОВАНИЕ ПРОГРАММНЫХ МОДУЛЕЙ

ПРАКТИЧЕСКАЯ РАБОТА №4

по теме:

Обработка исключений в Task

Выполнил:
студент ЕПК, группа
ФИО

Проверил:
преподаватель дисциплины
«Поддержка и тестирование
программных модулей»
Фомин А. Т.

Ейск 2024

1 Теоретические сведения

Необработанные исключения, которые создаются пользовательским кодом при выполнении в задаче, передаются обратно в вызывающий поток. Исключения возвращаются при вызове одного из статических методов или экземпляра Task.Wait, заключённого в инструкцию try/catch. Если задача является родительской для присоединенных дочерних задач или несколько задач находятся в ожидании, то может быть создано несколько исключений.

Чтобы вернуть все исключения обратно в вызывающий поток, инфраструктура задач заключает их в экземпляр AggregateException. Исключение AggregateException имеет свойство InnerExceptions, которое получает коллекцию для проверки всех созданных исходных исключений и обработки каждого исключения по отдельности. Обрабатывать исходные исключения можно с помощью метода AggregateException.Handle. Даже если возникает только одно исключение, оно по-прежнему заключается в исключение AggregateException.

Следующая программа иллюстрирует, как обрабатывать исключения в асинхронной операции, выполняемой задачей.

Листинг 1.

```
decimal Divide(decimal a, decimal b)
{
    Thread.Sleep(1000);
    return a / b;
}

try
{
    var task = Task.Run(() => Divide(10, 0));
    var result = task.Result;
}
catch (AggregateException ae)
{
    ae.Flatten().Handle(e =>
    {
        if (e is DivideByZeroException)
        {
```

```
            Console.WriteLine(e.Message);
            return true;
        }
        else
        {
            throw e;
        }
    });
}
```

Метод Divide() принимает два десятичных числа и возвращает результат их деления с задержкой в одну секунду:

```
decimal Divide(decimal a, decimal b)
{
    Thread.Sleep(1000);

    return a / b;
}
```

Метод Task.Run() асинхронно запускает Divide() в отдельном потоке.

```
var task = Task.Run(() => Divide(10, 0));
```

Метод Divide() возбуждает исключение DivideByZeroException. Поскольку асинхронная операция вызывает исключение AggregateException, то его можно перехватить и обработать в блоке catch.

В обработчике исключений используется метод Handle() класса AggregateException для обработки каждого внутреннего исключения.

Если исключение является DivideByZeroException, программа выводит сообщение об ошибке в консоль. В противном случае, она генерирует то же самое исключение.

Когда запускается программа, можно увидеть следующий результат:

```
Attempted to divide by zero.
```

Для обработки исключений, которые перехватываются AggregateException, использует цикл foreach для обработки каждого исключения в передаваемой коллекции InnerExceptions.

Листинг 2.

```
public static partial class Program
{
    public static void HandleThree()
    {
        var task = Task.Run(
            () => throw new CustomException("This exception is expected!"));

        try
        {
            task.Wait();
        }
        catch (AggregateException ae)
        {
            foreach (var ex in ae.InnerExceptions)
            {
                // Handle the custom exception.
                if (ex is CustomException)
                {
                    Console.WriteLine(ex.Message);
                }
                // Rethrow any other exception.
                else
                {
                    throw ex;
                }
            }
        }
    }
}
```

Можно использовать метод Handle для обработки каждого исключения, и повторно генерировать только те исключения, которые не являются экземплярами того исключения, которое возбуждается при выполнении задачи. Эту реализацию можно посмотреть в программе ниже.

Листинг 3.

```
using System;
using System.Threading.Tasks;

public class Example
{
   public static void Main()
   {
      var task1 = Task.Run( () => { throw new
CustomException("This exception is expected!"); } );

      try {
          task1.Wait();
      }
      catch (AggregateException ae)
      {
         // Call the Handle method to handle the custom
exception,
         // otherwise rethrow the exception.
         ae.Handle(ex => { if (ex is CustomException)
                              Console.WriteLine(ex.Message);
                           return ex is CustomException;
                         });
      }
   }
}

public class CustomException : Exception
{
   public CustomException(String message) : base(message)
   {}
}
```

2 Постановка задачи

Выполнить задания по вариантам. Решение реализовать с помощью Task. Использование обработчика необходимо показать только в контексте решаемой задачи в Task. Для прочих задач программы обработчик не использовать!
Результат работы — файл исходника с шапкой.

Варианты заданий:

1. Дан текстовый файл. Подсчитать количество строк в нем.
2. Дан текстовый файл. Подсчитать количество символов в нем.
3. Дан текстовый файл. Подсчитать количество символов в каждой строке.
4. Имеется текстовый файл. Удалить из него третью строку. Результат записать в другой файл.
5. Удалить из текстового файла его последнюю строку. Результат записать в другой файл.
6. Имеется текстовый файл. Удалить из него первую строку, в конце которой стоит вопросительный знак. Результат записать в другой файл.

   Имеется текстовый файл. Напечатать:
7. его первую строку;
8. его пятую строку;
9. его первые 5 строк;
10. его строки с s1-й по s2-ю;
11. Имеется текстовый файл, содержащий 20 строк. Переписать каждую из его строк в массив в том же порядке.
12. все его строки, начинающиеся с буквы Т;
13. все его строки, содержащие более 30 символов;
14. все его строки, в которых имеется более трех пробелов;
15. все его строки, содержащие в качестве фрагмента заданный текст.

    Имеется текстовый файл. Найти:
16. количество строк, начинающихся с букв А или а;
17. в которых имеется ровно 5 букв и.

    Имеется текстовый файл.
18. Найти длину самой длинной строки.
19. Найти номер самой длинной строки. Если таких строк несколько, то найти номер одной из них.
20. Напечатать самую длинную строку. Если таких строк несколько, то напечатать первую из них.

21. Имеется текстовый файл. Выяснить, имеется ли в нем строка, начинающаяся с буквы Т. Если да, то определить номер первой из таких строк.

Имеется текстовый файл. Напечатать:

22. первый символ первой строки;
23. пятый символ первой строки;
24. первые 10 символов первой строки;
25. символы с s1-го по s2-й в первой строке;
26. k-й символ n-й строки.