Семинарно упражнение № 4

Обработка на изключения в C#

I. Теоретична част

1. Изключителните събития (изключения) в обектно-ориентираното програмиране

Създавайки програмен код, програмистът разчита на неговото нормално изпълнение и в повечето случаи приложението следва нормалния ход на изпълнението на алгоритъма. Възникват обаче и ситуации, които са изключение от този нормален ход. Например, ако от потребителя се очаква да въведе число, а вместо това, той въведе текст; също – ако се направи опит да се отвори файл, който не съществува. Такива и други подобни ситуации водят до грешки по време на изпълнението на програмите, който се дължат на тези т.нар. **изключителни събития** или **изключения** (exceptions)

При разработката на Windows приложения в течение на дълго време, обработката на подобни грешки се превръща в сложна система, смесваща различни начини за обработка. Възниква и ситуацията, от група програмисти всеки по свой собствен начин да реализира логика за обработка на грешки в рамките на едно единствено приложение (проект).

Основен проблем при тези обработки на грешки е, че са различни. Всяка е свързана с конкретна технология, с конкретен проект или с конкретен език за програмиране. За да се избягнат тези различия, технологията .NET предлага единна техника за намиране на грешки по време на изпълнението, предаването на съответните съобщения и обработката на събитието. Това е т.нар. структурирана обработка на изключения. Предимството на този метод е, че се създава единен и добре премислен подход към обработката на грешки. Този подход е общ за всички езици в платформата .NET (C#, C++, Visual Basic и др.).

Съгласно подхода, е създаден клас System.Exeption и всички системни и потребителски изключения са наследници на този клас. От своя страна, клас Exception наследява клас Object. Клас Exeption има множество членове, от които могат да бъдат споменати следните:

- **HelpLink** – свойство, което връща URL файл със справки и описание на грешката;
- **Message** – свойство само за четене, което връща текстово съобщение с описанието на грешката. Самото съобщение за грешката се задава от програмиста като параметър на конструктора на клас Exception;
- **Source** – свойство, което връща името на обекта, генерирал грешка;
- **StackTrace** – свойство само за четене, което връща поредица от извиквания, довели до възникван на грешката;
- **InterExepton** – свойство, което се използва за съхраняване на сведения за грешката между серии изключения.

2. Конструкции за обработка на изключителни събития

2.1. Блок try-catch

Обектно ориентираното програмиране предлага структурно решение за обработката на грешки чрез използването на конструкция ***try-catch***. Идеята е физически да се раздели програмният код на две: съществена част от алгоритъма, реализирана със съответните оператори и конструкции, които отговарят за неговото нормалното изпълнение; и на програмен код, който обслужва обработката на грешки. Секцията от кода, който би предизвикал изключение се поставя в блок ***try***, а кодът, който обработва възникнала грешка е отделен в блок ***catch***.

Общият вид на констркукцията ***try-catch*** е следния:

```
try
{
        // тяло на блок try
}
catch(Тип идентификатор)
{
        // тяло на блок catch
}
```

В случая, **Тип** е име на клас изключение, което е или Exception, или някой от неговите производни класове. **Идентификатор** е име на променлива, която определя самото изключение и се използва в блок ***catch***.

Програмният код в тялото на блок ***try*** е този, който евентуално би предизвикал изключително събитие. Разбира се, възможно е той да бъде изпълнен нормално, без да се предизвика такова събитие. В тялото на блок catch е програмният код, който ще се изпълни ако настъпи изключително събитие от типа, който е посочен в скобите.

Пример:

```
try
{
 Console.Write("Enter a number:");
 int k = Int32.Parse(Console.ReadLine());
}
catch(OverflowException caught)
{
 Console.WriteLine(caught.Message);
}
```

В примера, в блок try е оформен програмен код, с който се въвежда целочислена стойност. Ако въведената стойност е по-голяма от обхвата на тип int, тогава ще настъпи изключителното събитие `OverflowException`. След блока try е предвидена клаузата catch, която ще прихване точно това събитие и ще изведе съотвеното съобщение.

Конструкцията try-catch работи по следния начин: Блок try обхваща частта от програмата, където би могло да възникне изключително събитие. Ако възникне такова, средата за изпълнение спира потока на нормалното

изпълнение и търси блок catch, който може да прихване възникналото изключение. Първоначално се търси след блок try и ако средата не намери подходящ catch блок непосредствено, тогава тя се обръща към стека и търси извикващия метод. Ако в изивикващия метод отново не се намери подходящ catch блок, търсенето продължава докато не бъде достигнат метод Main. Това е последният метод, в който се търси подходящ catch блок и в случай че не бъде намерен, средата прекратява аварийно изпълнението на програмния код. Ако в някой от методите бъде намерен подходящ catch блок, изпълнението на програмния код продължава като започне кода във въпросния catch блок.

2.2. Съвместно използване на повече от един catch блока

Възможно е в блок ***try*** да възникнат различни изключителни събития. Това зависи от програмния код. Още повече, че дори само един програмен ред може да е причина за възникване на повече от едно изключения. Затова един блок ***try*** може да бъде свързан с няколко блока ***catch***. Съответно, общият вид на конструкцията може да се представи по следния начин:

```
try
{
        // тяло на блок try
}
catch(КласИзключение идентификатор)
{
        // тяло на блок catch
}
catch(КласИзключение идентификатор)
{
        // тяло на блок catch
}
...
```

В случая, в класовете изключения в различните catch клаузи трябва да са различни.

Пример:

```
try
{
 Console.Write("Enter first number:");
 int first = Int32.Parse(Console.ReadLine());
 Console.Write("Enter second number:");
 int second = Int32.Parse(Console.ReadLine());
 int k = first/second;
}
catch(OverflowException caught)
{
 Console.WriteLine(caught.Message);
}
catch(DivideByZeroException caught)
{
 Console.WriteLine(caught.Message);
}
```

Ако в блок try възникне изключение поради препълване, тогава управлението ще се поеме от първия блок catch. Ако възникне изключение

поради деление на нула, тогава ще се изпълни втория catch блок. След това се изпълняват действията, които следват catch блоковете.

2.3. Блок catch с общо предназначение

Всеки блок try може да има и общ блок catch. Клауза catch с общо предназначение може да прихване всяко изключение, независимо от неговия тип. Тя често се използва за всяко непредвидено изключение, което би могло да възникне.

Синтаксисът на общата catch клауза е следния:

```
catch
{
        // блок за обработка на изключение
}
```

Един блок try може да има само една единствена catch клауза с общо предназначение. Тя задължително трябва да е последна от всички catch клаузи.

2.4. Блок finally

Език C# поддържа клауза finally за да обедини множество действия, които трябва да бъдат изпълнени независимо от хода на управляващия поток. Ето защо, ако управлението напусне блок try, в резултат от нормалното изпълнение при достигане на края на блока, действията в блок finally ще се изпълнят. Блок finally се изпълнява и когато блок try е напуснат или поради възникване на изключение, или при наличие на конструкции goto, break, continue.

Блок finally е полезен за предодвратяване дублирането на едни и същи действия.

3. Предизивикване на изключения

Освен за обработката на системни грешки, блок try-catch може да се използва и за обработка на изключения, които са предизвикани от програмиста. Например, ако стойностите на дадена променлива трябва да са в определен интервал, програмистът може да предизвика изключение, ако въведената стойност за нея излиза извън интервала.

Предизвикването на изключение е чрез конструкция throw. Когато в кода се срещне throw, незабавно се преустановява нормалното изпълнение на програмата и управлението се прехвърля към подходящия catch блок, който да обработи възникналото изключение.

Общият вид на конструкцията е:

throw Изключение;

Пример:

```
try
{
        Console.Write("Enter a minute:");
        int minute = Int32.Parse(Console.ReadLIne());
        if(minute<1 || minute>60)
        {
              string fault= minute+ " is not a valid minute";
```

```
                throw new Exception(fault);
        }
}
Catch(Exception e)
{
        Console.WriteLine(e.Message);
}
```

В посочения пример се предизвиква изключение, ако въведената стойност за минутата не е в интервала [1,60]. Обикновено изключенията очакват смислено същобщение (низ), което се предава като параметър при тяхното създаване. Съобщението може да бъде разпечатано или включено при възникване на изключението. Добра практика е да се към подходящ клас изключение.

II. Задачи за самостоятелна работа:

1. Да се въведат две цели числа от клавиатурата и да се намери резултатът от делението им. Да се обработят изключенията, които евентуално биха възникнали при изпълнението на програмния код. Да се изведе текст „God bye!" след завършване изпълнението на програмата.

2. Да се въведе число от клавиатурата и да се изведе корен квадратен от въведената стойност. Да се предизвика изключение, ако стойността е по-малка или равна на нула като се изведе съобщение „Invalid number". Да се изведе низ „End of program" след приключване изпълнението на програмния код.
3. Да се напише метод, който извежда име на месец при въвеждане на номера на съответния месец. Кодът да генерира изключение, ако въведената стойност не отговаря на месец.
4. Да се декларира клас, описващ време с час, минута и секунда. В класа да се съдържа метод, който въвежда стойности за полетата. Кодът на метода да генерира изключение при въвеждане на невалидни стойности за полетата.