7.Управление на изключения

Съдържание

- Exceptions & OOP
- Exceptions & .NET
  - Механизъм за прихващане на изключения
  - Йерархия на изключенията
  - Видове изключения
- Синтаксис
- Custom Exceptions

Съдържание (2)

- Други начина за обработка на грешки
- Exceptions vs. Return codes
- Добри практики за използване не изключения

Exceptions & OOP

- Механизъм за обработка на грешки по време на изпълнение на програмата
- Дава се възможност за централизирана обработка на грешки
- Заместват процедурно-ориентирания подход, при който всяка функция връща код за грешка
- Улесняват писането и поддръжката на надежден програмен код

Exception-а е нещо изключително

Не всяка грешка е изключение ☺

Контекст на изключенията

- За да обработите изключението трябва да знаете контекста

Контекст на изключенията

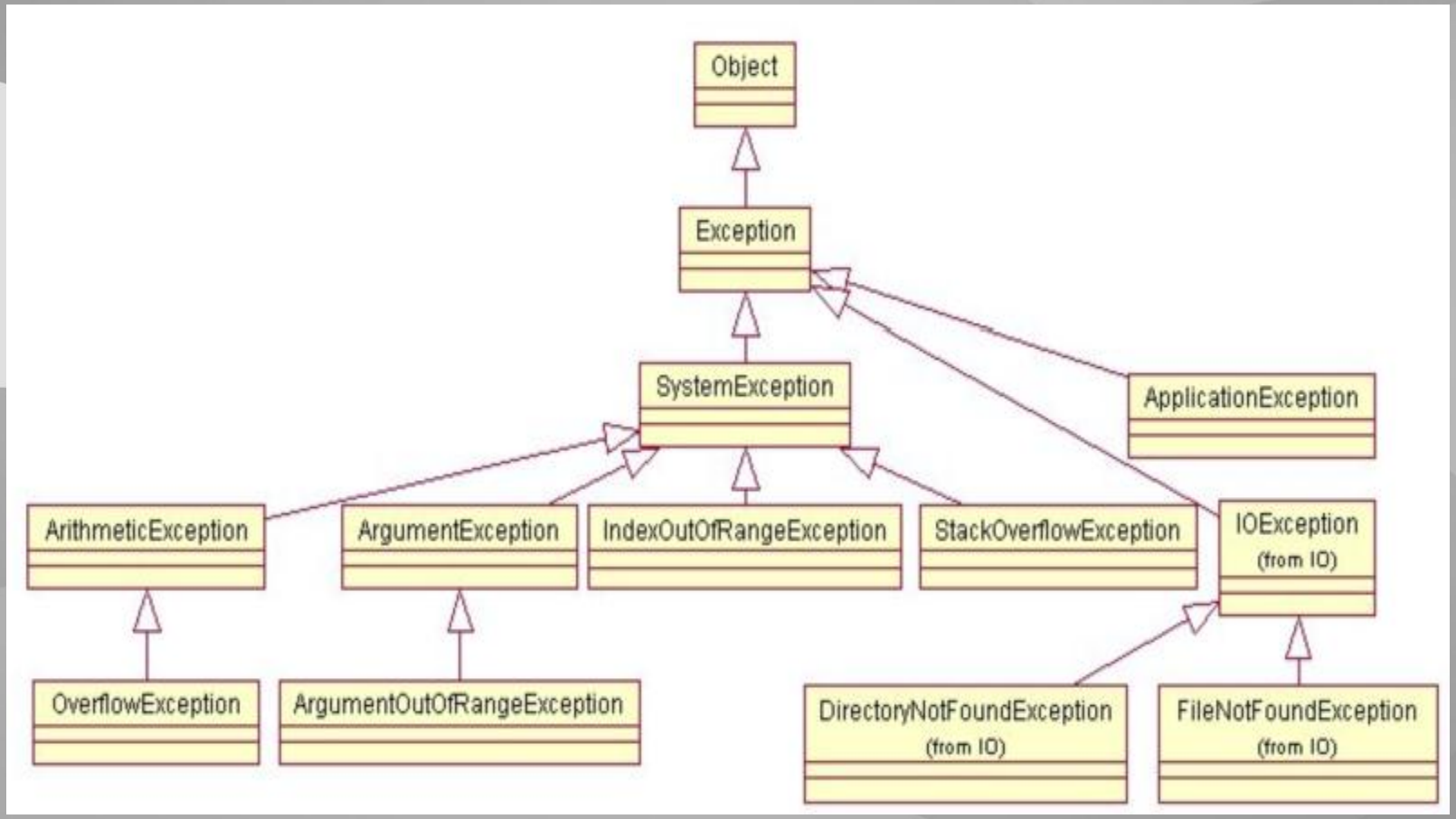
```
public void Foo()
{
    File file = this.OpenFile(this.filename);
    while(!file.IsEOF())
    {
        String record = file.ReadRecord();
    }
    CloseFile();
}

public void OpenFile(String filename)
{
    //Attempt to open file ...
}
```

Аз не зная как да обработя exception, ако възникне

Exceptions & .NET

- Изключенията в .NET са стандартна имплементация на изключенията в ООП-то
- Предоставят мощен механизъм за централизирана обработка на грешки и необичайни ситуации
- Позволяват проблемните ситуации да се обработват на много нива
- Изключенията са ...обекти

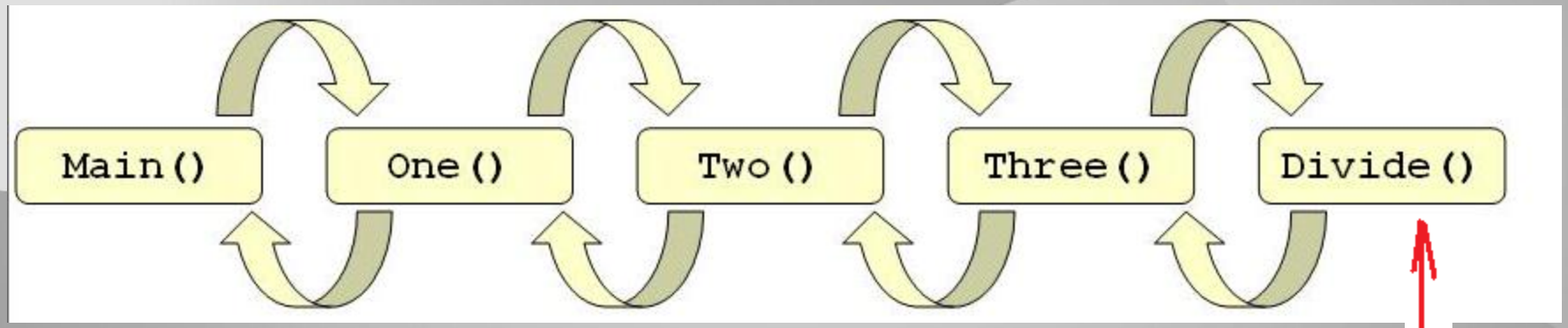
Йерархия на изключенията



Видове изключения

Exception Class	Description
SystemException	The base class for all exceptions
AccessException	Unable to access a method, a field or some other member.
ArgumentException	An argument value passed to a method was invalid, wrong type etc.
ArgumentNullException	A null argument (parameter) was passed to a method which requires a value, the argument is not missing, it just has a value of Null
ArgumentOutOfRangeException	The value passed in or returned by a method is out of the acceptable range
ArrayTypeMismatchException	One or more of the objects in array are of different types
ArithmeticException	Arithmetic overflow / underflow occured
BadImageFormatException	Image format is unrecognised.
Core Exception.	Runtimes Base Class for Exceptions.
DivideByZeroException	Zero was used as a denominator
FormatException	Argument format is not allowed
IndexOutofRangeException	An arrays index(position) is out of bounds for the array.
InvalidCastException	An object was cast to an unknown type.
InavalidOperatorException	Method called at an unrecognised time.

Механизъм за прихващане на изключенията



Възниква Exception в Devide().
Стека започва да се развива докато не се намери exception handler

Exceptions + .NET = …

…

- **Try** - загражда блока, в който може да възникне изключение
- **Catch** - хваща изключението; може да се задават филтри за типа изключения, които да прихваща
- **Finally** – изпълнява се винаги, независимо какво се е случило в try блока
- **Throw** - хвърля изключение за по-нататъшна обработка

Simple catch

```
public static void SimpleCatch()
{
    try
    {
        int data = new int[10];
        data[10] =17;

        //Още код ...
    }
    catch(IndexOutOfRangeException ex)
    {
        Console.WriteLine("Error has occured" + ex.Message);
    }
```

Check all statement

```
public static void Catch_All()
{
    // numer е повече от denom.
    int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
    int[] denom = { 2, 0, 4, 4, 0, 8 };

    for (int i = 0; i < numer.Length; i++)
    {
        try
        {
            Console.WriteLine(numer[i] + " / " +
                denom[i] + " is " +
                numer[i] / denom[i]);
        }
        catch
        {
            Console.WriteLine("Some exception occurred.");
        }
    }
}
```

Demo Time – Видове изключения

Нека да почистим-finally

```
FileStream f = new FileStream("data.txt", FileMode.Open);
try
{
    StreamReader t = new StreamReader(f);
    string line;
    while ((line = t.ReadLine()) != null)
    {
        int count = Convert.ToInt32(line);
        line = t.ReadLine();
        int sum = Convert.ToInt32(line);
        CalculateAverage(count, sum);
    }
}
// Винаги се изпълнява
finally
{
    f.Close();
```

Кога се изпълнява finally

- Ако не възникне изключение , finally се изпълнява веднага след try
- Ако възникне изключение и не се прихване в метода, CLR започва да търси обработчик
  - Ако намери: Изпълнява се finally и **след това** се извиква обработчика
  - Ако не намери: CLR обработва изключението (дава съобщение за грешка) и **след това** се изпълнява finally

Demo Time – finally

Хвърляне на изключения

throw exception;

```
static void Main(string[] args)
{
    // Проверка дали са подадени аргументи
    if (args.Length == 0)
    {
        throw new ArgumentException(
                "Parameter is required.");
    }

    Console.WriteLine("{0} argument(s) provided", args.Length);
}
```

Кога да използваме throw

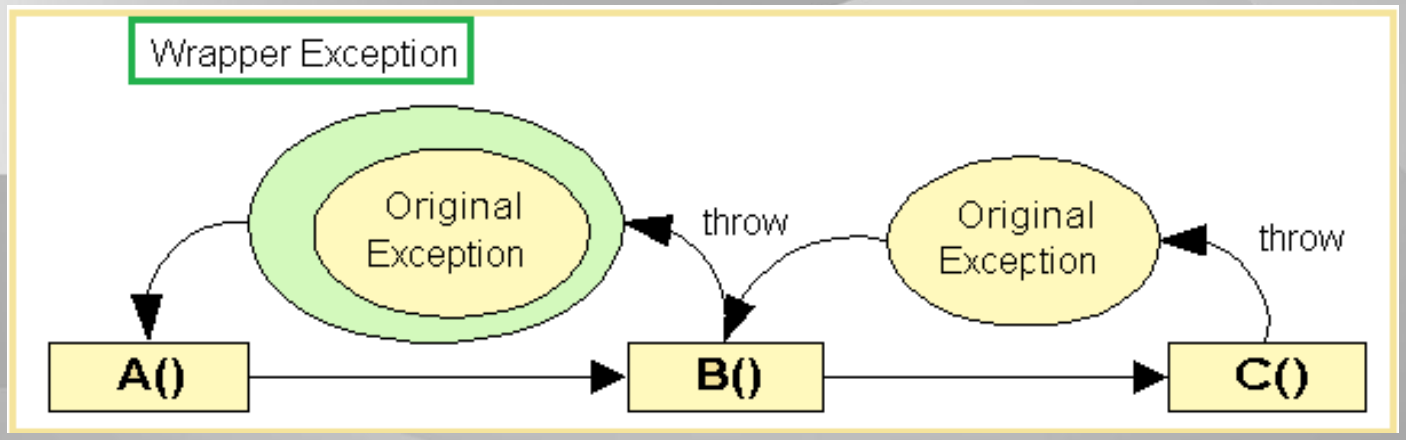
- Метод не може да изпълни предназначението си, защото
  - Подадени са невалидни параметри (*ArgumentException*)
  - Необходими са предварителни действия (*InvalidOperationException*)

Къде е проблемът?

```
Catch(Exception ex)
{
    throw ex;
}
```

Re-throw an Exception

throw;



Обработка на изключения

Table 1. Exception Propagation Summary

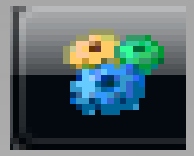
Means of Propagation	Allows you to react to the exception	Allows you to add relevancy
Let the exception propagate automatically	No	No
Catch and rethrow the exception	Yes	No
Catch, wrap, and throw the wrapped exception	Yes	Yes

Обработка на изключенията

```
try
{
    // Some code that could throw an exception.
}

catch (TypeAException e)
{
    // Code to do any processing needed.
    // Rethrow the exception
    throw;
}

catch (TypeBException e)
{
    // Code to do any processing needed.

    // Wrap the current exception in a more relevant
    // outer exception and rethrow the new exception.
    throw (new TypeCException(strMessage, e));
}

finally
{
    // Code that gets executed regardless of whether
    // an exception was thrown.
}
```

DemoTime – Обработка на изключенията

Класът System.Exception



```
public class Exception : ISerializable, _Exception
```

- Конструктори:

```
public Exception();
public Exception(string message);
protected Exception(SerializationInfo info, StreamingContext context);
public Exception(string message, Exception innerException);
```

- По –важни свойства:

```
// Properties
public Exception InnerException { get; }
public virtual string Message { get; }
public virtual string Source { get; set; }
public virtual string StackTrace { get; }
```

Класът System.Exception

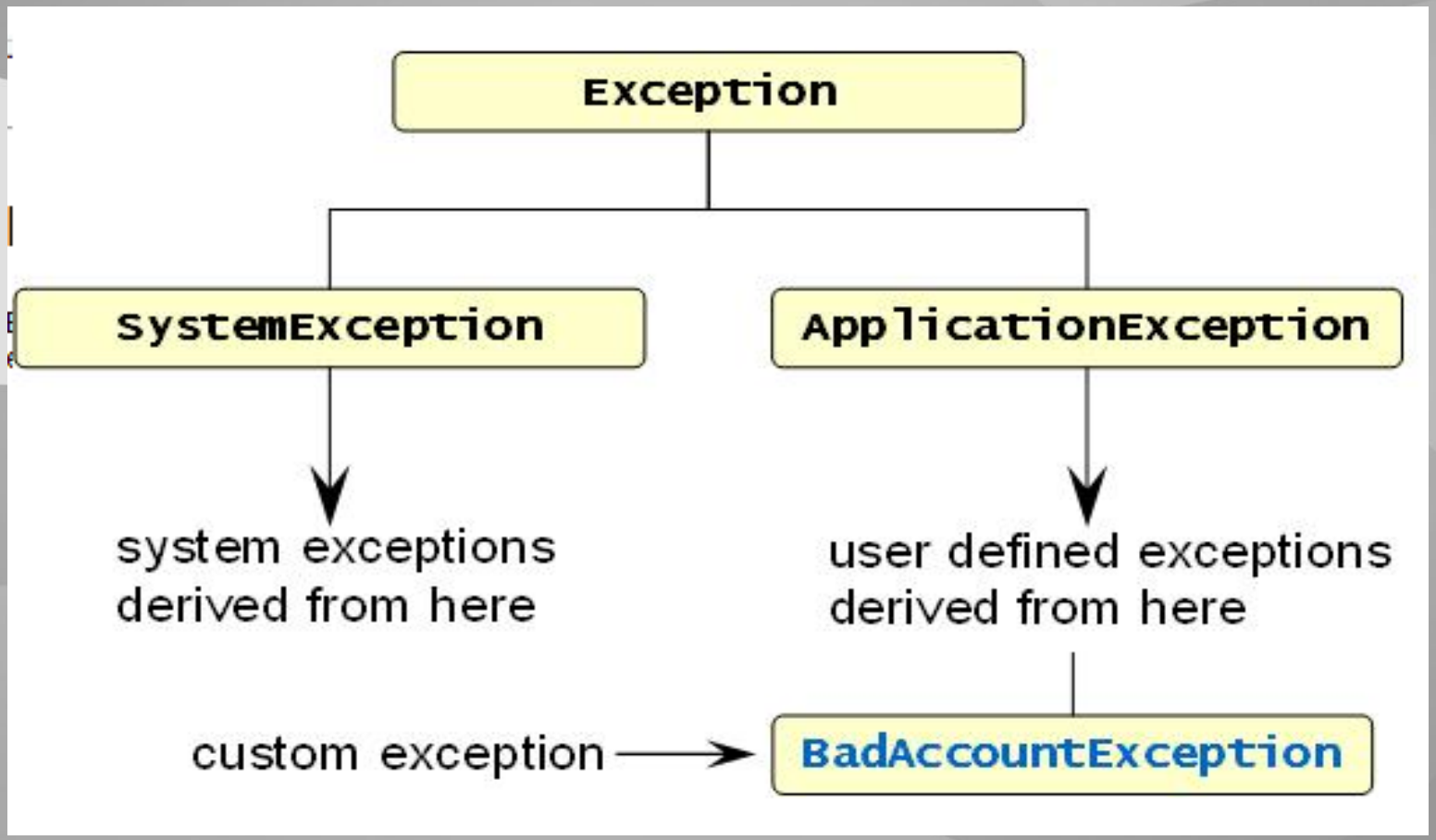
- **Message** – текстово описание на грешката
- **InnerException** -изключението, причинило текущото изключение
- **StackTrace** – състоянието на стека в момента на възникване на изключението
- **Source** – обекта, предизвикал изключението

Класът System.Exception

```
try
{
    int[] myArray = new int[2];
    Console.WriteLine("Attempting to access an invalid array element");
    myArray[2] = 1;
}
catch (DivideByZeroException e)
{
    Console.WriteLine("Handling a System.DivideByZeroException object");
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace = " + e.StackTrace);
}
catch (IndexOutOfRangeException e)
{
    Console.WriteLine("Handling a System.IndexOutOfRangeException object");
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace = " + e.StackTrace);
}
catch (Exception e)
{
    Console.WriteLine("Handling a System.Exception object");
    Console.WriteLine("Message = " + e.Message);
    Console.WriteLine("StackTrace = " + e.StackTrace);
}
```

DemoTime – System.Exception

System vs. Application Exceptions



SystemException

"ApplicationException is thrown by a user program, not by the common language runtime. If you are designing an application that needs to create its own exceptions, derive from the ApplicationException class."

VS.

"We added ApplicationException thinking it would add value by grouping exceptions declared outside of the .NET Framework, but there is no scenario for catching ApplicationException and it only adds unnecessary depth to the hierarchy. [...] You should not define new exception classes derived from ApplicationException; use Exception instead. In addition, you should not write code that catches ApplicationException."

ApplicationException

CustomExceptions - практики

- Наследявайте Exception
- Не правете дълбоки йерархии
- Класовете трябва да завършват на Exception
- Задавайте допълнителни свойства,които да дават информация за проблема
- Не давайте инфомация, свързана със сигурността

Custom Exceptions

```
public class LoginFailedException : System.Exception
{
    public LoginFailedException()
    {
    }

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

DemoTime:
Custom Exceptions

Алтернативата

```
public int CreateDatabase()
{
    if (!CreatePhysicalDatabase()) return 1;
    if (!CreateTables()) return 2;
    if (!CreateViews()) return 3;
    if (!CreateIndexes()) return 4;
    if (!PopulateData()) return 5;

    return 0;
}
```

Exceptions vs. Return Codes

- Изключенията се прихващат в подходящия контекст и могат да се обработват на нива
- Кодовете за грешка не носят пълна информация и не са говорящи
- Изключенията могат да се обработват централизирано

Бавни ли са изключенията?

- Да!

  НО…

- Не самата конструкция try- catch е бавна, а създаването на изключението

Code ? Try {Code}

(<,>,=)

Добри практики

- Хвърляйте изключения само при ситуации, които наистина са изключителни и трябва да се обработят
- При създаване на инстанция на изключение винаги й подавайте в конструктора подходящо съобщение
- `Catch` блоковете трябва да са подредени така, че да започват от изключенията най-ниско в йерархията и да продължават с по-общите

Добри практики

- Не е правилно да прихващате всички изключения, без да се интересувате от типа. Избягвайте конструкциите `catch (Exception) {…}` и просто `catch {`
- Хвърлянето на изключение не трябва да има странични ефекти
- Имайте предвид, че някои изключения могат да възникват по всяко време без да ги очаквате (например `System.OutOfMemoryException`)

Кога да използваме изключенията?

Samurai principal:

"Return a meaningful & usable result or throw an exception"

Упражнение ;-)

```
try
{
HOME:
    do
    {
        Play("World of Warcraft");
    }
    while (!asleep);

    Thread.Sleep(12 * 60 * 60 * 1000);

    WakeUp(coffee);

    if (you_still_give_a_shit())
        goto WORK;
    else
        goto OUT;
```

```
WORK:
    do
    {
        if (got_something_to_do())
        {
            LookAtTheMonitor();
            Press_Some_Keys(new string[] { "Ctrl+C", "Ctrl+V" });
        }

        Browse("vbox7.com"); Browse("topsport.bg"); Browse("youtube.com");

        Have_a_Break();
        Have_a_Kitkat();
    }
    while (DateTime.Now.Hour < 5);

    if (DateTime.Now.Day == 1)
    {
        // at least
        GetSomeCash(3000);
    }
```

Упражнение ;-)

```
OUT:
    switch (mood)
    {
      case Mood.Horny:
          ChaseChicks("hot!");
          break;
      case Mood.Dull:
          SmokeSomeStuff(new Stuff[] { "Grass", "Serious Stuff" });
          break;
      default:
          DrinkBeer(5);
          break;
     }
     goto HOME;
    }
```



```
catch (HealthException x)
{
    SeeTheDoctor(x);
}
catch (NoMoneyException)
{
    ShitHappens();
}
```

Въпроси?