Обробка виняткових ситуацій

Клакович Л.М.

Винятки

- Переваги винятків
- Схема використання винятків
- Особливості catch-блоків
- Особливості finaly
- class Exception
- Конструювання об'єктів Exception
- Властивості класу Exception
- Винятки FCL
- Приклади підходів

Переваги винятків

- Аргумент винятку точно описує ситуацію
- Відокремлення коду обробки
- Виняток не можна проігнорувати
- **unhandled exception** (у стеку викликів CLR не знайшов відповідний catch) – негайне знищення потоку

Схема використання винятків

```
void SomeMethod() {
    try {
    //Код, що може згенерувати винятки
    }
    catch (InvalidOperationException) {
     //обробка винятків InvalidOperationException типу  та похідних
    }
    catch (IOException) {
      // обробка винятків  IOException типу та  похідних
    }
    catch (Exception e) {
      // обробка винятків будь-якого  CLS exception типу
     // як правило, піднімають такий виняток вверх
      throw;
    }
    catch {
    // будь-які інші винятки
    // як правило, піднімають такий виняток вверх.
      throw;
    }
    finally {
    // Код, що очищає все, що було ініціалізовано в  try блоці.
    // Цей код виконається завжди, незалежно чи був згенерований виняток чи ні
    }
    // Код після блоку finally виконається тільки тоді, коли  в блоці try не згенерується виняток
    // або якщо блок catch перехопить виняток і не буде піднімати його вверх (rethrow ).
}
```

Handling Exceptions – Example

```
static void Main()
{
    string s = Console.ReadLine();
    try
    {
        Int32.Parse(s);
        Console.WriteLine(
            "You entered valid Int32 number {0}.", s);
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid integer number!");
    }
    catch (OverflowException)
    {
        Console.WriteLine(
            "The number is too big to fit in Int32!");
    }
}
```

Особливості catch-блоків

- розміщення catch в порядку поглиблення спеціалізації
- catch { } :
  - System.Exception і будь-який похідний
  - будь-який (несумісний з CLR)

Find the Mistake!

```
static void Main()
{
    string s = Console.ReadLine();
    try
    {
        Int32.Parse(s);
    }
    catch (Exception)
    {
        Console.WriteLine("Can not parse the number!");
    }
    catch (FormatException)
    {
        Console.WriteLine("Invalid integer number!");
    }
    catch (OverflowException)
    {
        Console.WriteLine(
         "The number is too big to fit in Int32!");
    }
}
```

This should be last

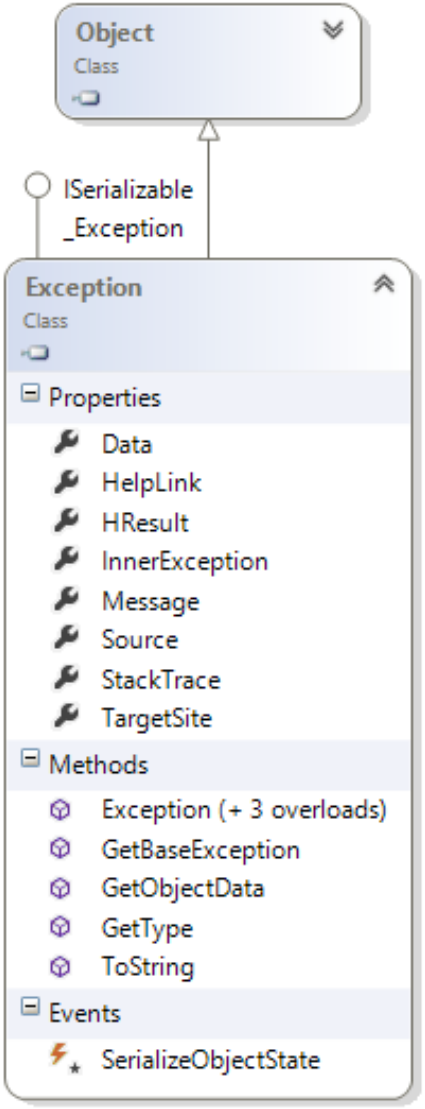
Unreachable code

Unreachable code

Особливості finaly

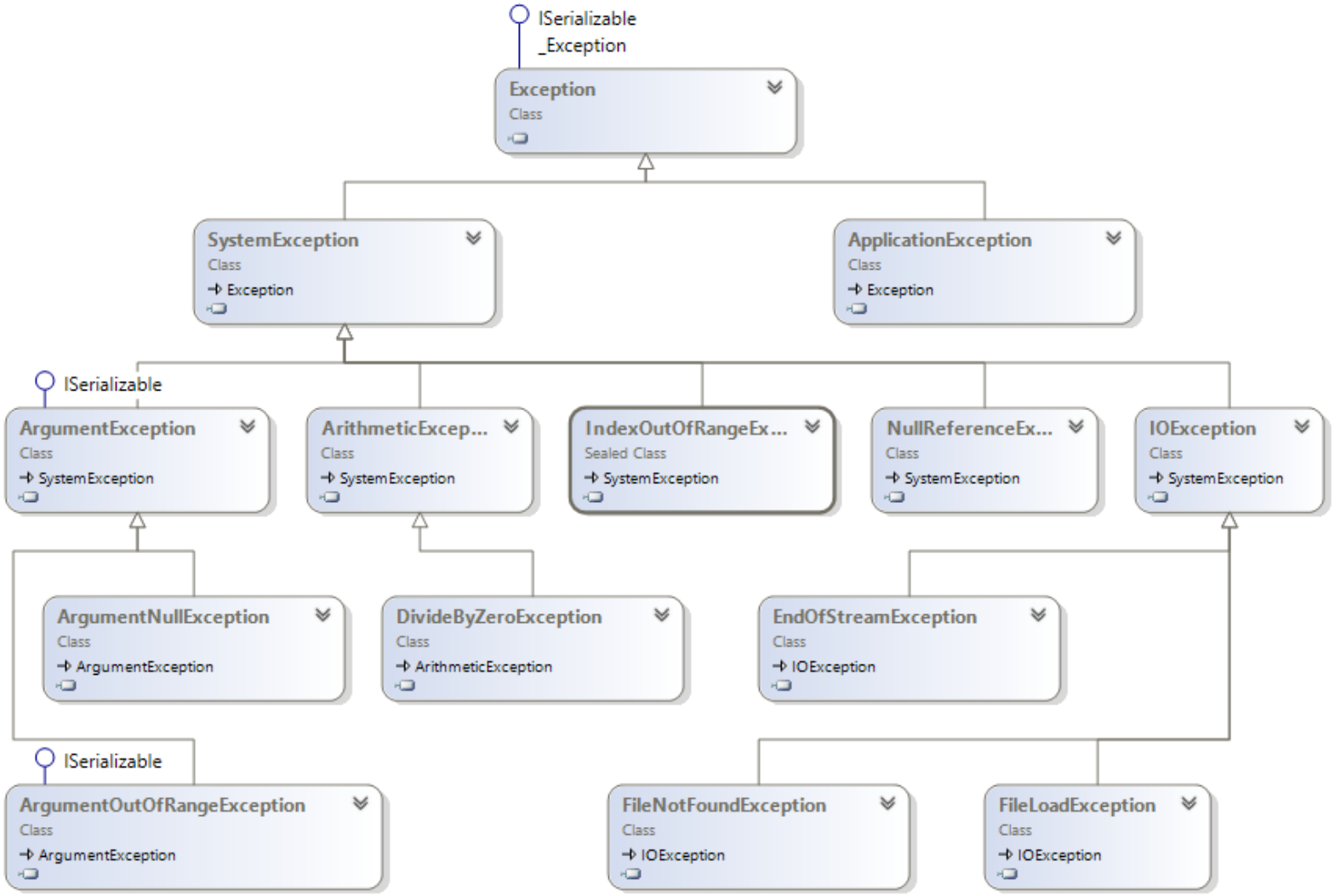
- try може мати не більше одного finaly
- finaly не завжди буде виконано?
- у finaly варто уникати коду, який може генерувати винятки
- якщо в finaly згенерується виняток:
  - CLR продовжить виконання всіх операторів блоку finaly
  - попередній виняток ігнорується
  - після виконання операторів finaly почнеться обробка нового винятку

class Exception



Name	Description
Exception ()	Ініціалізує об’єкт **Exception**.
Exception (String)	Ініціалізує об’єкт **Exception** вказаним error повідомлення
Exception (SerializationInfo, StreamingContext)	Ініціалізує об’єкт **Exception** із сериалізованою датою
Exception (String, Exception)	Ініціалізує об’єкт **Exception** вказаним errorповідомленням і внутрішнім винятком, що спричинив даний виняток

Exception Hierarchy



Властивості класу Exception

Property	Access	Type	Description
Message	Read-only	**String**	Містить корисну інформацію, яка вказує чому відбувся виняток.
Data	Read-only	**IDictionary**	Надає колекцію ключ/значення пар, яка містить додаткову інформацію про виняток.
Source	Read/write	**String**	Містить ім'я збірки, в якій згенеровано виняток
StackTrace	Read-only	**String**	Містить імена і сигнатуру методів, що були викликані перед генерацією винятку Ця властивість корисна для дебагу.
TargetSite	Read-only	**MethodBase**	Містить метод, який згенерував виняток
InnerException	Read-only	**Exception**	Містить внутрішній виняток, що був причиною даного винятку

Exception властивості

```
class ExceptionsExample
{
  public static void CauseFormatException()
  {
    string s = "an invalid number";
    Int32.Parse(s);
  }
  static void Main()
  {
    try
    {
      CauseFormatException();
    }
    catch (FormatException fe)
    {
      Console.Error.WriteLine("Exception: {0}\n{1}",
        fe.Message, fe.StackTrace);
    }
  }
}
```

Exception властивості

- Властивість Message дає короткий опис проблеми
- StackTrace – розгортає стек

```
Exception caught: Input string was not in a correct
format.
   at System.Number.ParseInt32(String s, NumberStyles
style, NumberFormatInfo info)
   at System.Int32.Parse(String s)
   at ExceptionsTest.CauseFormatException() in
c:\consoleapplication1\exceptionstest.cs:line 8
   at ExceptionsTest.Main(String[] args) in
c:\consoleapplication1\exceptionstest.cs:line 15
```

Приклади підходів

коректне поновлення виконання

```
public String CalculateSpreadsheetCell(Int32 row, Int32 column)
{
    String result;
    try {
        result = /* Code to calculate value of a spreadsheet's
        cell*/
    }
    catch (DivideByZeroException) {
        result = "Can't show value: Divide by zero";
    }
    return result;
}
```

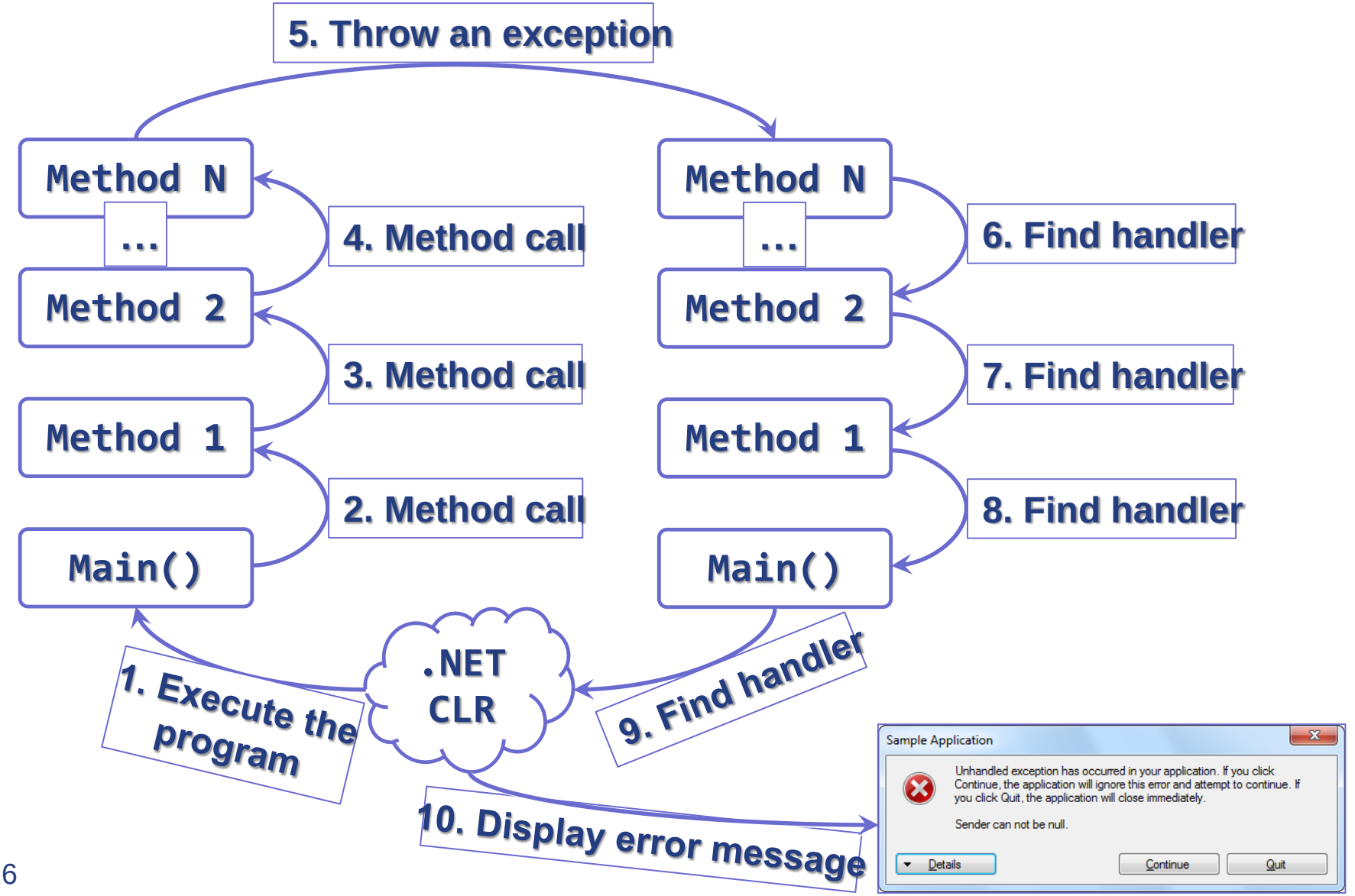
приховування деталей реалізації

```
public Int32 SomeMethod(Int32 x){
    try {
        return 100 / x;
    }
    catch (DivideByZeroException e) {
     throw new ArgumentOutOfRangeException( "x can't be 0", e);
    }
}
```

Throwing Exceptions

- Винятки генеруються через `throw` оператор
  - Використовується для повідомлення викликаючого коду про помилку чи проблему
- Якщо виняток згенеровано:
  - Зупиняється виконання програми
  - Виняток подорожує в стеку доти, доки відповідний блок `catch` не буде знайдено і виняток перехопиться
- Неперехоплені винятки відображають повідомлення про помилку

Генерація винятків



Using `throw` Keyword

- Генерація винятку з повідомленням про помилку

```
throw new ArgumentException("Invalid amount!");
```

- Винятки також можуть містити ще причину – внцтрішній виняток

```
try
{
  Int32.Parse(str);
}
catch (FormatException fe)
{
  throw new ArgumentException("Invalid number", fe);
}
```

Підняття винятку вверх

- Перехоплений виняток може бути перегенерований знову – піднятий вгору

```
try
{
  Int32.Parse(str);
}
catch (FormatException fe)
{
  Console.WriteLine("Parse failed!");
  throw fe; // Re-throw перехоплений виняток
}
```

```
catch (FormatException)
{
  throw; // Re-throw останні згенерований виняток
}
```

Підняття винятку вверх

```
public static double Sqrt(double value)
{
    if (value < 0)
        throw new System.ArgumentOutOfRangeException(
            "Sqrt for negative numbers is undefined!");
    return Math.Sqrt(value);
}

static void Main()
{
    try
    {
        Sqrt(-1);
    }
    catch (ArgumentOutOfRangeException ex)
    {
        Console.Error.WriteLine("Error: " + ex.Message);
        throw;
    }
}
```