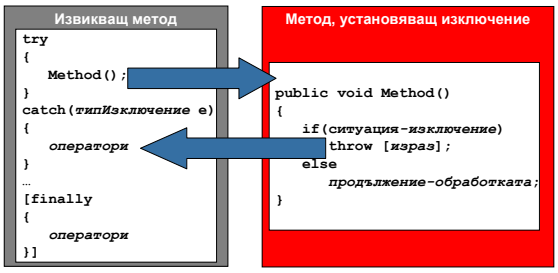
Обработка на изключения

Изключения – обектно-ориентирани решения за обработка на грешки при нарушаване на нормалната последователност на кода. Съществува разлика между изключение и очаквано събитие – напр. достигане край на файл.

Обработка – чрез ключовите думи try, catch, throw и finally.

Когато даден метод установи ситуация на изключение, той хвърля изключението към извикващия метод чрез ключовата дума throw. Извикващият метод приема изключението чрез ключовата дума catch и решава какво действие да предприеме.

try, catch, throw, finally



Клас System.Exception

Представя изключенията.

Конструктори:

а) подразбиращ се конструктор;

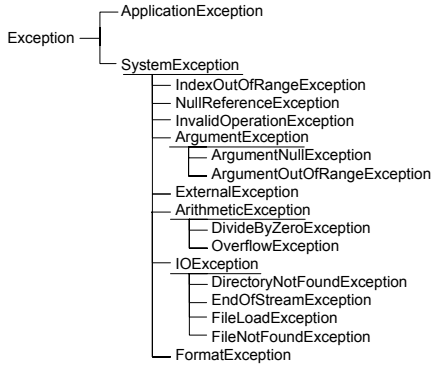
public Exception();

б) с параметър string message **– определя съобщението за грешка (извлича се чрез свойството** System.Exception.Message**);**

public Exception (string message);

в) с два параметъра: string message **– определя съобщението за грешка и** Exception innerException **– причината за това изключение.**

public Exception (string message, Exception innerException);



1. Хвърляне на изключение

throw [израз];

израз – **незадължителен;**
обект от тип System.Exception **(или негов наследник).**

```
public void <име_на_метод>()
{
   if (<ситуация на изключение>)
      throw new Exception();
   else
      <продължение на обработката>;
}
```

2. Хващане на изключение

```
try
{
    // оператори
}
catch (<тип_на_изключение> <изключение>)
{
    // оператори
}
…
[finally
{
    // оператори
}]
```

3. Хвърляне на изключение от конструктори – сигнализира за грешки при конструиране на типа.

Пример: Класът InRange **въвежда число в определени граници – хвърля изключение, което се хваща в метода** Main **чрез оператор** try-catch**.**

```
using System;
class InRange
{
   private int number;
   public int Number
   {
      get
      {
        return number;
      }
   }
   public InRange (int min, int max)
   {
      Console.Write ("Въведи число в интервала [{0},{1}]: ", min, max);
      number = int.Parse (Console.ReadLine());
      if (number < min || number > max)
          throw new Exception ("Числото е извън областта");
   }
}
```

```
class Test
{
   static void Main (string[] args)
   {
      int n;
      InRange r;
      try
      {
          r = new InRange(-10, 10);
          n = r.Number;
      }
      catch (Exception e)
      {
        Console.WriteLine ("Обработено е изключение - '{0}'", e.Message);
      }
   }
}
```

Резултати:

```
Въведи число в интервала [-10,10]: 20
Обработено е изключение - 'Числото е извън областта'
```

Методът Parse **хвърля изключението** System.FormatException **при въвеждане на нечислови данни – хваща се от** Main**.**

```
Въведи число в интервала [-10,10]: абв
Обработено е изключение - 'Input string was not in the correct format'
```

Получава се съобщение за грешка, ако метод хвърли изключение, но извикващият клас не го хваща.

4. Прехвърляне на изключение

Може даден метод да хване изключение, да го обработи според контекста му и да го прехвърли обратно в стека чрез ключовата дума throw**, така че по-нататък отново може да стане грешка.**

Пример: В конструктора на класа InRange **се извиква методът** Parse**, който може да хвърля три изключения (**ArgumentNullException**,** FormatException**,** OverflowException**), които се хващат от конструктора. При опит да се въведат нечислови символи изключението се предава обратно в стека и така извикващият метод знае, че не е въведено правилно число.**

```
using System;
class InRange
{
   private int number;
   public int Number
   {  get  {  return number;  }  }
   public InRange (int min, int max)
   {
      Console.Write ("Въведи число в интервала [{0},{1}]: ", min, max);
      try
      {
         number = int.Parse (Console.ReadLine());
      }
      catch (Exception e)
      {  Console.WriteLine("Обработено е изключение от тип - '{0}'",
                                e.GetType());
         throw;
      }
      if (number < min || number > max)
          throw new Exception ("Числото е извън областта");
   }
}
```

```
class Test
{
   static void Main (string[] args)
   {
      int n;
      InRange r;
      try
      {
           r = new InRange(-10,10);
           n = r.Number;
      }
      catch (Exception e)
      {
           Console.WriteLine ("Обработено е изключение - '{0}'", e.Message);
      }
   }
}
```

Резултати:

```
Въведи число в интервала [-10,10]: абв
Обработено е изключение от тип - 'System.FormatException'
Обработено е изключение - 'Input string was not in the correct format'
```

5. Изчистване с finally

Кодът в блока finally се изпълнява винаги.

Пример:

```
using System;
class InRange
{
   private int number;
   public int Number
   {  get  {  return number;  }  }
   public InRange (int min, int max)
   {  Console.Write ("Въведи число в интервала [{0},{1}]: ", min, max);
      try
      {
         number = int.Parse (Console.ReadLine());
      }
      catch (Exception e)
      {
         Console.WriteLine("Обработено е изключение от тип - '{0}'",
                           e.GetType());
         throw;
      }
      if (number < min || number > max)
         throw new Exception ("Числото е извън областта");
   }
}
```

```
class Test
{  static void Main (string[] args)
   {  int n;
      InRange r = null;
      try
      {    r = new InRange(-10,10);
           n = r.Number;
      }
      catch (Exception e)
      {
        Console.WriteLine ("Обработено е изключение - '{0}'", e.Message);
      }
      finally
      {  if (r != null) Console.WriteLine ("Успешно конструиране на обекта!");
         else Console.WriteLine ("Неуспешно конструиране на обекта!");
      }
   }
}
```

Резултати:

```
Въведи число в интервала [-10,10]: 20
Обработено е изключение - 'Числото е извън областта'
Неуспешно конструиране на обекта!
```

6. Хващане на много изключения

Базовият клас трябва да се обработи последен – ако неговият блок catch е първи, останалите блокове catch ще бъдат недостижими и останалите изключения няма да се обработят.

Пример:

```
using System;
class MultipleExceptions
{
   static void Main (string[] args)
   {
      int [] A = new int[] {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
      int i, k;
      i = 11;
      k = 3;
      try
      {  int temp = A[i] / k;
         A[i+1] = int.Parse ("10" + temp);
      }
      catch (System.IndexOutOfRangeException e)
      {  Console.WriteLine ("Грешка при индексиране");
      }
      catch (System.DivideByZeroException e)
      {  Console.WriteLine ("Грешка при деление на нула");
      }
      catch (System.FormatException e)
      {  Console.WriteLine("Грешка при преобразуване на низ в цяло число");
      }
      catch (System.Exception e)
      {   Console.WriteLine ("Грешка: {0}", e.ToString());
      }
   }
}
```

Резултати:

```
Грешка при индексиране
```

7. Създаване на собствено изключение

Дефинира се клас, наследник на класа Exception; необходимият код се поставя в конструктора му.

Правила:

а) добра практика е името на наследника да завършва с Exception;

б) реализират се конструкторите на класа Exception.

Пример:

```
using System;
public class MonthException : Exception
{  public MonthException () : base ()
   {
      LogException();
   }
   public MonthException (String message) : base (message)
   {
      LogException();
   }
   public MonthException (String message, Exception innerException) :
                                                    base (message, innerException)
   {
     LogException();
   }
   protected void LogException()
   {
      Console.WriteLine ("Грешка: {0}: {1}.", this.GetType(), this.Message);
   }
}
```

```
class DerivedExceptionApp
{
   public static string MonthName (int month)
   {
      switch (month)
      {
         case 1:  return "януари";
         case 2:  return "февруари";
         case 3:  return "март";
         case 4:  return "април";
         case 5:  return "май";
         case 6:  return "юни";
         case 7:  return "юли";
         case 8:  return "август";
         case 9:  return "септември";
         case 10: return "октомври";
         case 11: return "ноември";
         case 12: return "декември";
         default:  throw new MonthException ("Грешен месец");
      }
   }
```

```
   static void Main()
   {  string month = null;
      try
      {
         month = MonthName(13);
      }
      catch (Exception e)
      {
         Console.WriteLine ("Грешка! {0}", e.ToString());
      }
      finally
      {
         if (month != null) Console.WriteLine ("Месецът е {0}.", month);
      }
   }
}
```

Резултати:

```
Грешка: MonthException: Грешен месец.
Грешка! MonthException: Грешен месец
   at DerivedExceptionApp.MonthName(Int32 month) in
   f:\...monthexception\class1cs: line 35
   at DerivedExceptionApp.Main() in f:\...monthexception\class1cs: line 43
```

8. Сравняване на техниките за обработка на грешки

а) връщане код на грешка – стандартен начин за управление на грешките; недостатък: няма гаранция, че извикващият метод ще провери върнатия код на грешка.

б) предимства на обработка на изключения пред връщане на код:

1) гаранция за обработка на изключението – управлението се предава обратно в стека и извикващият метод е принуден да го обработи;

2) управление на грешките в коректния контекст – осигурява разширяемост – най-значимото предимство;

3) подобряване читаемостта на кода.

Пример: Използване на върнати кодове.

```
class AccessDatabase
{  public bool CreatePhysicalDatabase () { }
   public bool CreateTables () { }
   public bool CreateIndexes () { }
   public bool GenerateDatabase()
   {  if (CreatePhysicalDatabase())
      {  if (CreateTables())
         {  if (CreateIndexes()) return true;
            else {  // Обработва грешката
           return false;
         }
         }
         else {  // Обработва грешката
         return false;
         }
      }
      else {  // Обработва грешката
         return false;
      }
   }
}
```

Пример: Използване на обработка на изключенията.

```
public void GenerateDatabase ()
{
    CreatePhysicalDatabase ();
    CreateTables ();
    CreateIndexes ();
}
// Извикващ код
try
{
        AccessDatabase accessDb = new AccessDatabase();
        accessDb.GenerateDatabase ();
}
catch (Exception e)
{
        // Разглеждане на хванатото изключение
}
```