Лекція 4

ОБРОБКА ВИНЯТКІВ

Лекція 4. Обробка винятків

План

1. Загальне поняття про винятки.
2. Обробка винятків у C#.
3. Стандартні винятки.
4. Створення винятків користувача.

1. Загальне поняття про винятки

Виняткова ситуація (ВС) або **виняток** – це певна подія, що призвела до збою у роботі програми. Внаслідок виникнення ВС програма не може коректно продовжити своє виконання. Тобто виняток – це **порушення природного ходу виконання алгоритму**.

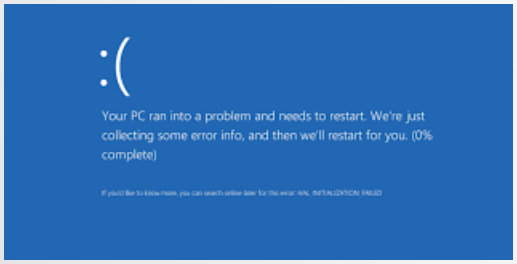
Існують два основні типи ВС:

- **апаратні** (генеруються процесором чи іншими пристроями);
- **програмні** (генеруються операційною системою або прикладними програмами).

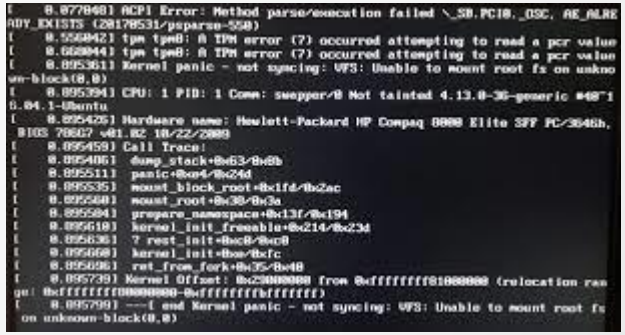
1. Загальне поняття про винятки

Апаратні винятки виникають, як правило, при різних збоях обладнання, таких, наприклад, як:

- переповнення;
- звернення до неправильної адреси пам'яті;
- розрив мережного з'єднання;
- відключення живлення;
- ...



BSOD – «синій екран смерті» у Windows

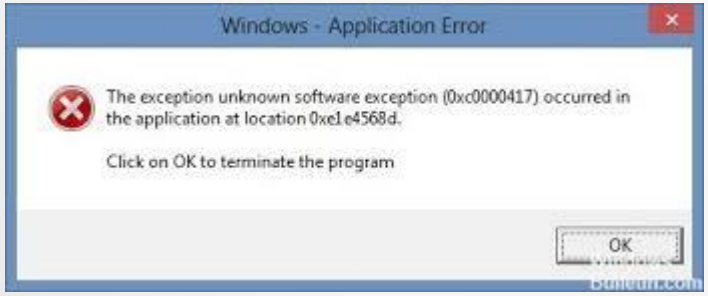


Kernel panic у Linux

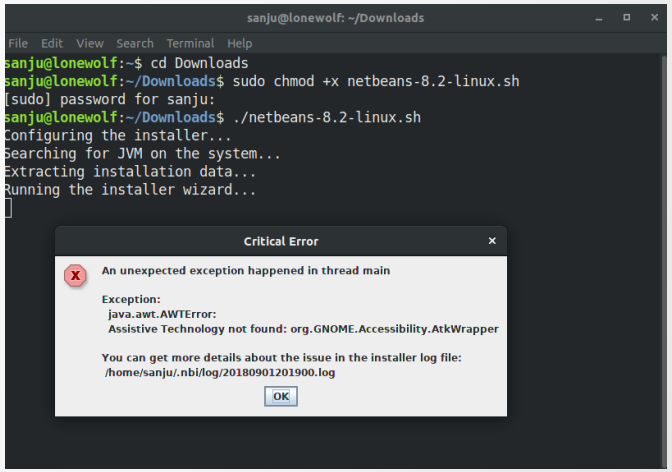
1. Загальне поняття про винятки

Програмні винятки явно **генеруються** операційною системою або прикладною програмою, коли вони виявляють аномальну ситуацію, що виникла в процесі їхньої роботи. Наприклад:

- нестача оперативної пам'яті;
- вихід за межі масиву;
- спроба обчислення кореня з від'ємного числа;
- некоректні вихідні дані і т.п.



ВС, що «кидається» Windows при некоректній роботі програми з пам'яттю



ВС, що генерується Linux при помилці встановлення програми

1. Загальне поняття про винятки

Розглянемо наступний приклад.

```
// Функція, що реалізує ділення двох чисел
double div(double top, double bot)
{
    return top/bot;
}
```

Ясно, що в такому вигляді функція div() є небезпечною, тому що при нульовому значенні параметра bot відбудеться ділення на нуль, що може викликати відповідне апаратне переривання з аварійним завершенням **програми**.

Програміст, який, наприклад, реалізує математичну бібліотеку, знає про «вузькі місця», де можливі виникнення помилок, але поняття не має, як їх обробляти (це повинен робити користувач бібліотеки).

Примітка. Наявність у програмі необроблених винятків є не просто поганим тоном, а й погано характеризує розробника в очах замовника!

1. Загальне поняття про винятки

Стандартним підходом до вирішення цієї проблеми є використання деякої глобальної змінної, якій присвоюється код помилки у разі її виникнення. Наприклад.

```
using System;

namespace ConsoleApplication2
{
    // Коди помилок
    enum ErrCode: int {NoErr = 0, DivZero = 1};
    internal class Program
    {
        // Код останньої помилки
        static ErrCode _errno = ErrCode.NoErr;
        static double Div(double top, double bot)
        {
            _errno = ErrCode.NoErr;
            if (bot == 0)
            {
                _errno = ErrCode.DivZero;
                return 0;
            }
            return top/bot;
        }
        public static void Main(string[] args)
        {
            double a = 10, b = 0, res = Div (a, b);

            if (_errno == ErrCode.DivZero)
              Console.WriteLine("Divide by zero!");
            else
              Console.WriteLine("Result: {0}", res);
            }
        }
    }
```

1. Загальне поняття про винятки

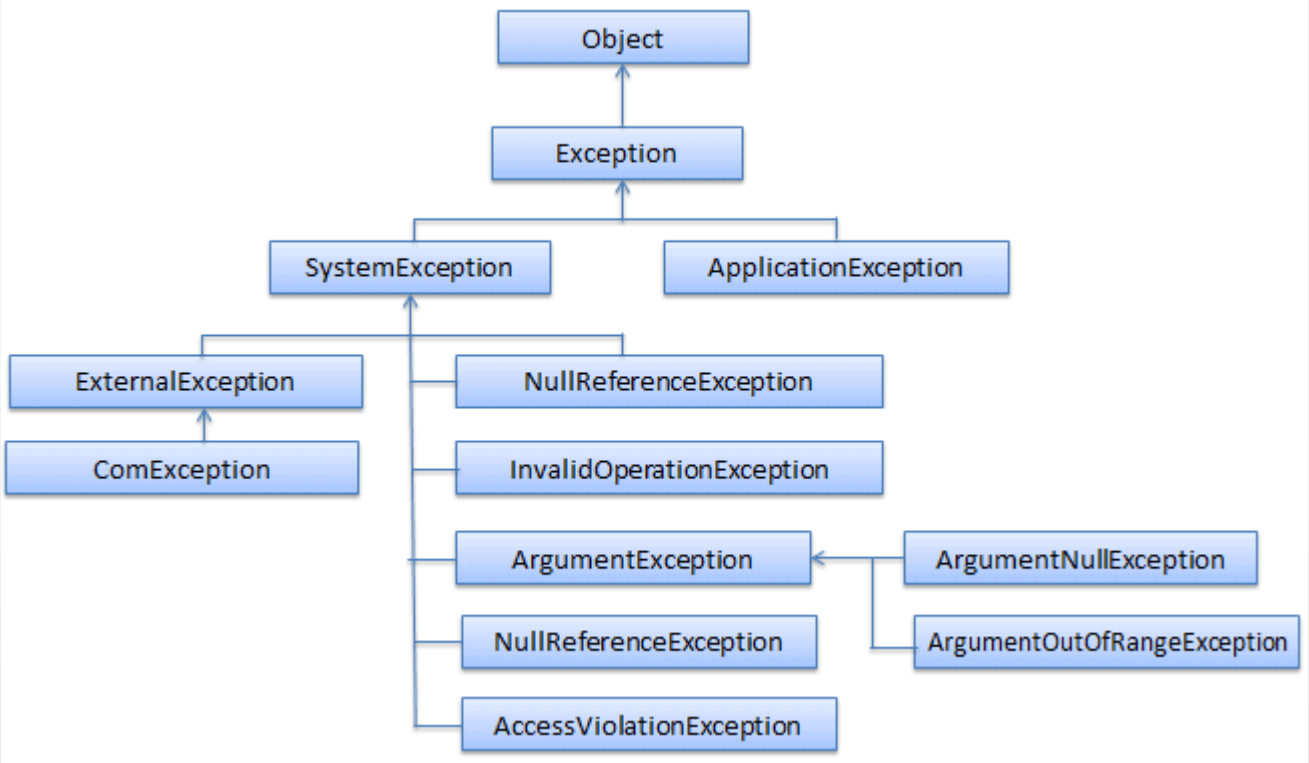
Такий підхід до обробки помилок, очевидно, є дуже незручним, тому що велика кількість перевірок на помилки істотно збільшує розмір програмного коду і робить його менш читабельним і швидким.

У сучасних мовах програмування для обробки винятків використовуються спеціальні мовні конструкції. У C# для цього використовується блочний оператор **try...catch...finaly**:

```
try
{
    // Небезпечний код
}
catch (Exception e)
{
    // Обробка виключення
}
finally
{
    // Код фіналізації
}
```

2. Обробка винятків у C#

Виняток у C# – це спеціальна змінна (об’єкт спеціалізованого класу), яка описує конкретну ВС. У C# реалізовані спеціалізовані класи для обробки різних типів винятків (похідні від класу **System.Exception**).



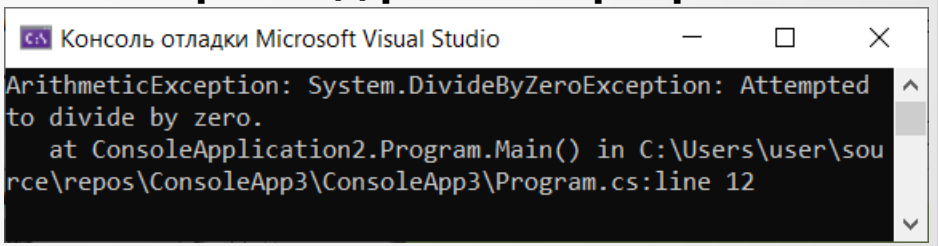
2. Обробка винятків у C#

Для **перехоплення та обробки** винятків у C# використовується конструкція **try ... catch**. Наприклад:

```
using System;

namespace ConsoleApplication2
{
   internal class Program
   {
      public static void Main()
      {
         int x = 0;
         try
         {
            int y = 100/x;
         }
         catch (ArithmeticException e)
         {
            Console.WriteLine("ArithmeticException: {0}", e);
         }
         catch (Exception e)
         {
            Console.WriteLine("Generic Exception: {0}", e);
         }
      }
   }
}
```

Приклад роботи програми



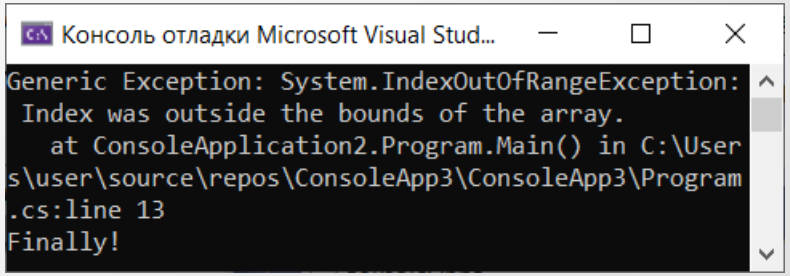
Примітка: Конструкція try...catch() у C# допускає використання декількох блоків catch() для обробки різних типів винятків.

2. Обробка винятків у C#

Для обробки ВС також може використовуватися спеціальний блок **finally**. Управління **обов'язково** передається до нього перед завершенням блоку **try...catch...finally**, навіть якщо у блоці **try{}** не виникло жодних винятків.

```
using System;

namespace ConsoleApplication2
{
   internal class Program
   {
      public static void Main()
      {
         int[] x = new int[5];
         try
         {
            for (int i = 0; i < 10; i++)
               x[i] = i;
         }
         catch (Exception e)
         {
            Console.WriteLine("Exception: {0}", e);
         }
         finally
         {
            Console.WriteLine("Finally!");
         }
      }
   }
}
```



2. Обробка винятків у C#

Для **генерації** винятку вручну у C# використовується оператор **throw**. Він має таку форму:

throw [e];

Тут необов'язковий параметр **e** – це об'єкт похідного від **System.Exception** класу, що ідентифікує виняток. В результаті виконання throw e «кидається» ВС, тип якої визначається значенням параметра **e** і який має бути оброблено у відповідному блоці catch.

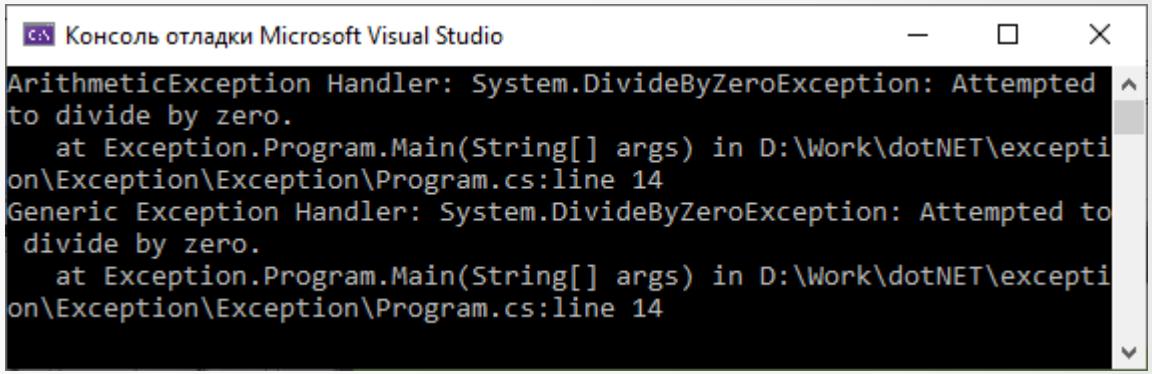
```
// …
try
{
	// …
	throw new IndexOutOfRangeException(); // ІВ «Вихід за межі діапазону»
}
catch (IndexOutOfRangeException e)
{
	Console.WriteLine("{0}: is outside the bounds of the array", e.GetType().Name);
	// Обробка...
}
```

2. Обробка винятків у C#

Виклик **throw** без параметра можливий лише всередині блоку catch та призначений для обробки ВС зовнішнім блоком catch. Наприклад:

```
using System;
internal class Program
{
    public static void Main()
    {
        int x = 0;
        try
        {
            try
            {
                int y = 100/x;
            }
            catch (ArithmeticException e)
            {
                Console.WriteLine($"ArithmeticException Handler: {e}");
                throw;
            }
        }
        catch (Exception e)
        {
            Console.WriteLine($"Generic Exception Handler: {e}");
        }
    }
}
```

Результати роботи програми



3. Стандартні винятки

У мові C# реалізовано безліч класів обробки різних типів ИС. Як зазначалося, всі вони є похідними від класу **System.Exception** .

Клас Exception містить ряд властивостей, що містять інформацію про помилку:

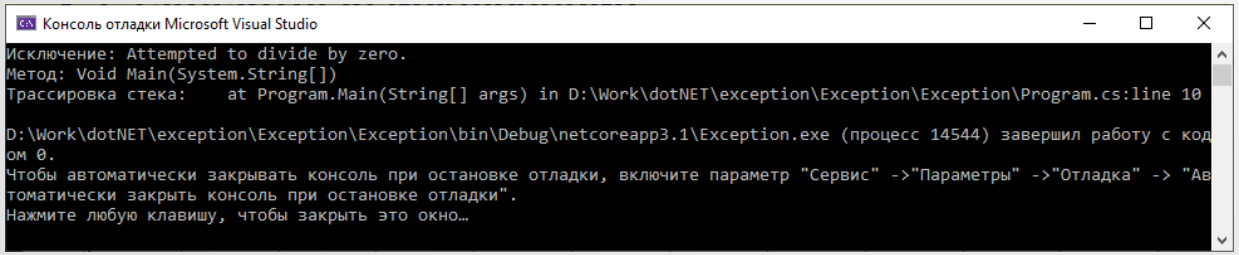
- **InnerException** – зберігає інформацію про виключення, що спричинило поточну помилку;
- **Message** - зберігає текстове повідомлення про виключення;
- **Source** - зберігає ім'я об'єкта або складання, яке викликало виняток;
- **StackTrace** – повертає рядкове представлення стека викликів, які призвели до виключення;
- **TargetSite** – повертає метод, у якому було викликано виняток.

3. Стандартні винятки

Наприклад, обробка винятку типу Exception

```
using System;
class Program
{
    static void Main(string[] args)
    {
        try
        {
            int x = 5, y = x/0;
            Console.WriteLine("Результат: {0}", y);
        }
        catch (Exception ex)
        {
            Console.WriteLine("Виключення: {0}", ex.Message);
            Console.WriteLine("Метод: {0}", ex.TargetSite);
            Console.WriteLine("Трасування стека: {0}", ex.StackTrace);
        }
    }
}
```

Приведе до такого результату:



3. Стандартні винятки

Спадкоємцями Exception є безліч класів, серед яких можна виділити такі:

- **DivideByZeroException** – виняток, що генерується при діленні на нуль;
- **ArgumentOutOfRangeException** – генерується, якщо значення аргументу знаходиться поза діапазоном допустимих значень;
- **ArgumentException** – генерується, якщо в якості параметра передається некоректне значення;
- **IndexOutOfRangeException** – генерується, якщо індекс елемента масиву чи колекції знаходиться поза діапазоном допустимих значень;
- **InvalidCastException** – генерується при спробі зробити неприпустимі перетворення типів;
- **NullReferenceException** – генерується при спробі звернення до невизначеного об'єкта;

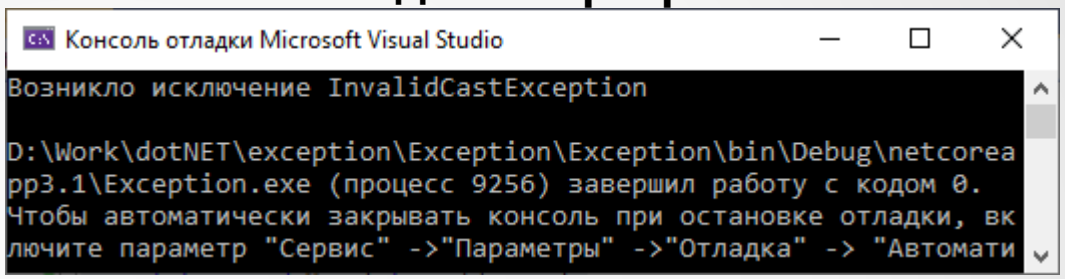
3. Стандартні винятки

Приклад обробки некоректного перетворення типів даних.

```
using System;

class Program
{
   static void Main(string[] args)
   {
       try
       {
          object obj = "you";
          int num = (int) obj; // Неправильне перетворення
          Console.WriteLine($"Результат: {num}");
       }
       catch (DivideByZeroException)
       {
          Console.WriteLine("Виник виняток DivideByZeroException");
       }
       catch (InvalidCastException)
       {
          Console.WriteLine("Виник виняток InvalidCastException");
       }
   }
}
```

Виведення програми



4. Створення винятків користувача

При необхідності програміст може створювати свої власні класи для обробки необхідних йому типів винятків користувача.

Для цього необхідно визначити клас-спадкоємець від System.Exctption.

Наприклад:

```
class MyException : System.Exception
{
  // Потрібні програмісту поля та методи, що розширюють функціональність
  // базового класу Exception
}
```

4. Створення винятків користувача

Розглянемо приклад створення класу для обробки винятків користувача:

```
using System;
class MyException : System.Exception
{
    public string Prefix;
    public MyException(string pfx, string msg) : base(msg)
    {
        Prefix = pfx;
    }
}
class Program
{
    static void Main(string[] args)
    {
        int x = 0;
        try
        {
            if (x == 0)
                throw new MyException("Fatal error", "Dividing by zero!");
            int y = 100/x;
        }
        catch (MyException e)
        {
            Console.WriteLine("{0}: {1}", e.Prefix, e.Message);
        }
    }
}
```

Результати роботи програми