Лабораторна робота №3
Обробка винятків у C#

План
1 . Поняття винятків.
2 . Приклади обробки винятків.
3. Створення користувальницького класу винятків.
4 . Завдання.

1. Під **винятком** розуміється деяка подія (помилка), яка призвела до збою в роботі програми. Внаслідок виникнення винятку програма не може коректно продовжити своє виконання. Якщо виняток певним чином не обробити, програма буде аварійно завершена операційною системою (ОС).

Існують два основні типи винятків:

1) **апаратні** – генеруються процесором або іншими пристроями при різних збоях обладнання (наприклад, при розриві мережевого кабелю або падінню напруги в мережі);

2) **програмні** – генеруються ОС чи прикладними програмами при виникненні різних збоїв, таких, наприклад, як введення некоректних даних, виході за межі масиву і т. п.

У класичних мовах програмування таких, наприклад, як C обробка винятків виконувалася наступним чином. Після виклику кожної функції, при виконанні якої могла виникнути помилка, програміст повинен був перевіряти стан деякої глобальної змінної (errno у C), яка містила код останньої помилки, що виникла. Очевидно, що такий підхід до обробки помилок не завжди є зручним, оскільки велика кількість перевірок на помилки істотно збільшує розмір програмного коду і робить його менш читабельним і швидким.

Для зручної обробки винятків у мові C# передбачено спеціальний блочний оператор **try...catch...finally**:

```
try
{
        // Небезпечний код
}
catch (Exception_1 E1)
{
        // Обробка виключення Exception_1
}
catch (Exception_2 E2)
{
        // Обробка виключення Exception_2
}
...
catch (Exception_n En )
{
```

```
        // Обробка виключення Exception_n
}
finally
{
        // Код фіналізації (виконується в будь-якому випадку:
        // як за наявності помилок, так і при їх відсутності)
}
```

Тут E1, ..., En – об'єкти спеціальних класів, що описують різні типи винятків. Ці класи мають бути похідними від базового класу **System.Exception**, який описує загальне поняття помилки, що відбувається під час виконання програми.

2. Обробка можливої помилки конвертації рядка в число може виглядати, наприклад, таким чином:

```
using System;

namespace ExceptionTest
{
   internal class Program
   {
      public static void Main(string[] args)
      {
         double x;

         Console.WriteLine("Enter x");
         try
         {
            x = Convert.ToDouble(Console.ReadLine());
            Console.WriteLine($"Entered double value {x}");
         }
         catch (FormatException e)
         {
            // Обробка помилки перетворення рядка в число
            Console.WriteLine("Conversion error: {0}", e.Message);
         }
         catch (Exception e)
         {
            // Обробка інших помилок
            Console.WriteLine(e);
         }
      }
   }
}
```

Результат роботи такої програми може виглядати так:

```
/usr/bin/mono-sgen /home/serg/work/rider/test3/ExceptionTest/ExceptionTest/bin/Debug/ExceptionTest.exe
Enter x
asd
Conversion error: Input string was not in a correct format.

Process finished with exit code 0.
```

Або так (при коректному введенні даних):

```
/usr/bin/mono-sgen /home/serg/work/rider/test3/ExceptionTest/ExceptionTest/bin/Debug/ExceptionTest.exe
Enter x
-3,14159
Entered double value -3,14159

Process finished with exit code 0.
```

У наведеному прикладі реалізовано два обробники винятків. Один з них **FormatException** описує помилки формату даних, а інший – **Exception** – решту можливих помилок (необроблений виняток призводить до аварійного завершення програми).

Крім обробки винятків програма може їх і генерувати (іноді в таких випадках кажуть «кидати виняток»). Це робиться за допомогою спеціального оператора **throw** [e], де e – об'єкт деякого класу, що описує поточний виняток і є похідним від класу System.Exception.

Наприклад:

```
using System;

namespace ExceptionTest
{
   internal class Program
   {
      public static void Main(string[] args)
      {
         double x, y;

         Console.WriteLine("Enter x and y");
         try
         {
            x = Convert.ToDouble(Console.ReadLine());
            y = Convert.ToDouble(Console.ReadLine());
            if (y == 0)
               throw new ArithmeticException("Dividing by zero");
            Console.WriteLine($"{x}/{y} = {x/y}");
```

```
            }
            catch (FormatException e)
            {
                // Обробка помилки перетворення рядка в число
                Console.WriteLine("Conversion error: {0}", e.Message);
            }
            catch (ArithmeticException e)
            {
                // Обробка математичних помилок
                Console.WriteLine("Arithmetic error: {0}", e.Message);
            }
            catch (Exception e)
            {
                // Обробка інших помилок
                Console.WriteLine(e);
            }
        }
    }
}
```

Тоді результат роботи такої програми може мати такий вигляд:

```
/usr/bin/mono-sgen /home/serg/work/rider/test3/ExceptionTest/ExceptionTest/bin/Debug/ExceptionTest.exe
Enter x and y
1
0
Arithmetic error: Dividing by zero

Process finished with exit code 0.
```

Або при правильно введених даних (відповідно до логіки програми) так:

```
/usr/bin/mono-sgen /home/serg/work/rider/test3/ExceptionTest/ExceptionTest/bin/Debug/ExceptionTest.exe
Enter x and y
6,28
3,14159
6,28/3,14159 = 1,99898777370694

Process finished with exit code 0.
```

3. За потреби програміст може створювати власні класи, що описують винятки. Наприклад:

```
using System;

namespace ExceptionTest
{
    public class MyException : Exception
```

```
    {
        public string DetailedMessage;

        public MyException(string msg1, string msg2) : base(msg2)
        {
            DetailedMessage = msg1;
        }
    }

internal class Program
    {
        public static void Main(string[] args)
        {
            double x, y;

            Console.WriteLine("Enter x and y");
            try
            {
                x = Convert.ToDouble(Console.ReadLine());
                y = Convert.ToDouble(Console.ReadLine());
                if (x < 0)
                    throw new MyException("Negative number", "Sqrt");
                if (y == 0)
                    throw new MyException("Fatal error", "Dividing by zero");
                Console.WriteLine("sqrt({0})/{1} = {2}", x, y, Math.Sqrt(x) / y);
            }
            catch (FormatException e)
            {
                // Обробка помилки перетворення рядка в число
                Console.WriteLine("Conversion error: {0}", e.Message);
            }
            catch (MyException e)
            {
                // Обробка користувальницьких помилок
                Console.WriteLine("{0}: {1}", e.DetailedMessage, e.Message);
            }
            catch (Exception e)
            {
                // Обробка ынших помилок
                Console.WriteLine(e);
            }
        }
    }
```

Приклад роботи такої програми наведено нижче:

```
/usr/bin/mono-sgen /home/serg/work/rider/test3/ExceptionTest/ExceptionTest/bin/Debug/ExceptionTest.exe
Enter x and y
-1
2
Negative number: Sqrt

Process finished with exit code 0.
```

4. Переробити лабораторну роботу № 2 таким чином, щоб у ній оброблялися можливі винятки, пов'язані з некоректним введенням вихідних даних та/або математичними помилками. З цією метою створити власний клас, який описує ці види винятків.

Передбачити виведення на екран ПІБ та номери групи студента, який здає роботу.