Лабораторна робота №10.

Тема : Виняткові ситуації. Події.

Мета роботи:

Познайомитися з механізмами обробки виняткових ситуацій та механізмом подійно-орієнтованого програмування мови C# (події).

Теоретичні відомості

Виняткові ситуації

Виняток це помилка, що відбувається під час виконання програми. За допомогою підсистем обробки винятків для **C#** можна обробляти такі помилки, не викликаючи краху програми. Обробка винятків у **C#** виконується застосуванням ключових слів: **try**, **catch**, **throw** і **finally**. Ці ключові слова утворюють взаємозалежну підсистему, у якій використання одного із ключових слів спричиняє використання інших. Основа обробки винятку заснована на використанні блоків **try**, **catch** і **finally**.

```
try {
Блок _ коду _ для _ якого _ виконується _ моніторинг _ помилок
      catch (Exceptypel ex) {
      Оброблювач _ винятків _ Exceptypel }
      catch (ExcepType2 ex) {
      Оброблювач _ винятків _ ExcepType2 }
      catch  {
      Оброблювач _ винятків _ всіх типів }
      finally {
      Код блоку finally. }
```

У **.NET Framework** передбачена розвинена система обробки помилок. Механізм обробки помилок C# дозволяє закодувати призначену для користувача обробку для кожного типу помилкових умов, а також відокремити код, потенційно породжує помилки, від коду, який займається обробкою їх. Бібліотека базових класів **.NET** міститься багато класів, які успадковуються від **System.Exception.** Наприклад, в просторі імен **System** визначені :

ArgumentOutOfRangeException,

IndexOutOfRangeException,

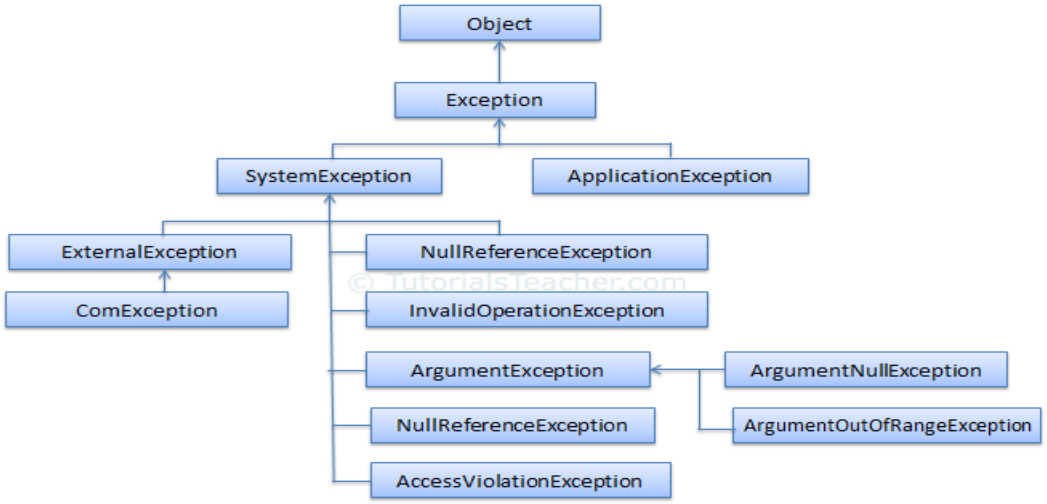
StackOverflowException і т.д.

В інших просторах імен є винятки, що відображають їхню поведінку. Наприклад, в просторі імен :

System.Drawing.Printing містяться виключення, що виникають при друку,

System.IO - виключення, що виникають під час введення-виведення,

System.Data - виключення, пов'язані з базами даних, і т.д.).



Основні системні винятків наведені в наступній таблиці:

Винятки	Значення
`ArrayTypeMismatchException`	Тип збереженого значення несумісний з типом масиву.
`DivideByZeroException`	Почата спроба розподілу на нуль.
`IndexOutOfRangeException`	Індекс масиву виходить за межі діапазону.
`InvalidCastException`	Некоректне перетворення в процесі виконання.
`OutOfMemoryException`	Виклик new був невдалим через недостатності пам'яті.
`OverflowException`	Переповнення при виконанні арифметичної операції.
`StackOverflowException`	Переповнення стека.

Тип винятків в операторові `catch` повинен відповідати типу винятків, що перехоплюється. Не перехоплене винятків неодмінне приводить до дострокового припинення виконання програми. Для виконання перехоплення винятку незалежно від їхнього типу (перехоплення всіх винятку) можливе використання `catch` без параметрів. Тому що оператор `catch` не викликається із програми, то після виконання блоку `catch` керування **не передається** назад операторові програми, при виконанні якого виник виняток. Виконання програми продовжується з операторів, що перебувають після блоку `catch`. З метою запобігання цієї ситуації можлива вказівка блоку коду, який викликається після виходу із блоку `try/catch`, за допомогою блоку `finally` наприкінці послідовності `try/catch`. Блок `finally` буде викликатися незалежно від того, з'явиться винятків чи ні, і незалежно від причин виникнення такого.

Винятків автоматично генеруються системою. Однак винятків може бути згенеровано за допомогою оператора `throw`.

```
throw exceptob;
```

Винятків, перехоплене одним оператором `catch`, може генеруватися повторно, завдяки чому воно може перехоплюватися зовнішнім оператором `catch`. Для цього вказується ключове слово `throw` без імені винятків.

Можна створювати замовлені винятків, що виконують обробку помилок у користувацькому коді. Генерування винятку не представляє особливих складностей. Просто визначите клас, успадкований з класу `Exception`. У якості загального правила необхідно керуватись тим, що задані користувачем винятки успадковуються із класу `ApplicationException`, тому що вони є ієрархію зарезервованих винятку, пов'язаних з додатками. Успадковані класи не мають потреби у фактичній реалізації в якому-небудь виді, оскільки саме їхнє існування в системі типів даних дозволяє скористатися ними в якості винятку. Створювані користувачем класи винятку автоматично одержують доступні для них властивості й методи, заданих в класі `Exception`.

Винятки рівня системи

Винятки, які генеруються самою платформою .NET, називаються винятками рівня системи. Ці виключення вважаються невід'ємними фатальними помилками. Вони успадковуються прямо від базового класу System.SystemException:

```
public class SystemException: Exception
{ // Різні конструктори.
}
```

Через те, що в System.SystemException ніякої додаткової функціональності крім набору спеціальних конструкторів більше не пропонується, це дає можливість зрозуміти, що сутністю, яка згенерувала виняток, є виконуючого середовища .NET, а не кодова база функціонуючого додатки.

Всі винятки .NET мають основні класи, допускається створювати власні винятки, призначені для конкретного додатка. Базовий клас System.SystemException представляє винятки, які генеруються CLR-середовищем, можна власні винятки успадковуватися від System.Exception, але рекомендується успадковувати їх не від System.Exception, а від System.ApplicationException:

```
public class ApplicationException: Exception
{
// Різні конструктори.
}
```

Якщо планується створити спеціальний клас виключення, необхідно подбати про те, щоб він відповідав рекомендаціям .NET. Зокрема це означає, що він повинен:

- успадковуватися від ApplicationException;
- супроводжуватися атрибутом [System.Serializable];
- мати конструктор за замовчуванням;
- мати конструктор, який встановлює значення успадкованого властивості Message;
- мати конструктор для обробки "внутрішніх винятків";
- мати конструктор для обробки серіалізації типу.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;

namespace ConsoleApplication1
{
    // Спеціальний виняток
    [Serializable]
    public class MyException: ApplicationException
    {
        public MyException () {}
        public MyException (string message): base (message) {}
        public MyException (string message, Exception ex): base (message) {}
        // Конструктор для обробки серіалізації типу
        protected MyException (System.Runtime.Serialization.SerializationInfo info,
            System.Runtime.Serialization.StreamingContext contex)
            : base (info, contex) {}
    }
class Program
    {
```

```
        static int MyDel (int x, int y)
        {   if (y == 0)
            {
        MyException exc = new MyException ("ПОМИЛКА: ділення на 0");
            exc.HelpLink = "http:\\mysite.org";
                throw exc;
            }
            return x / y;
        }
        static void Main ()
        {  try
            {
                MyDel (5, 0);
            }        // Обробляємо загальне виключення
            catch (MyException)
            {
                Console.WriteLine ("Виникла помилка");
            }
        Console.ReadLine ();
        }
    }
}
```

Обробка подій

Подія — це автоматичне повідомлення про яке-небудь що відбувся дії. Події є членами класу й оголошуються з використанням ключового слова **event**. Механізм подій заснований на використанні делегатів.

Синтаксис:

event *ім'я _ делегата ім'я _ об'єкта*;

Широкомовні події

Події можуть активізувати декілька оброблювачів, у тому числі ті, що визначені в інших об'єктах. Такі події називаються широкомовними. Широкомовні події створюються на основі багатоадресних делегатів.

Оголошення делегата, на основі якого буде

```
// визначена подія.
delegate void MyEventHandler () ;
// Оголошення класу, у якім ініціюється подія.
class MyEvent
  {
      public event MyEventHandler activate;
            // У цьому методі ініціюється подія.
      public void fire()
                  { if (activate != null) activate();
                  }
  }

class X
{
      public void XHandler()
      {
            Console.Writeline("Подія отримана об'єктом класу X.");
      }
}
class Y
{
```

```
        public void YHandler()
        {
        Console.Writeline("Подія отримана об'єктом класу Y.");
        }
}

class EventDemo
{
static void handler()
        {
        Console.Writeline("Подія отримана об'єктом класу EventDemo.")
}
public static void Main()
{
        Myevent evt = new MyEvent();
        X xob = new X();
        Y yob = new Y();
// Додавання методів handler (), Xhandler()
// і Yhandler() у ланцюжок оброблювачів події.
evt.activate += new Myeventhandler(handler);
evt.activate += new Myeventhandler(xob.Xhandler);
evt.activate += new Myeventhandler(yob.Yhandler);
evt.fire();
Console.Writeline();
evt.activate -= new Myeventhandler(xob.Xhandler);
evt.fire();
}
}
```

Приклад проекту "Місто і його служби при пожежі".

```
using System;

namespace EventsMyCSharp
{
    public delegate void FireEventHandler(object sender, FireEventArgs e);

    /// <summary>
    /// Модель міста з подіями  що й стежать за ними службами міста
    /// </summary>
    public class NewTown
    {
        //властивості
        string townname;    //назва міста
        int buildings;       //число будинків у місті
        int days;          //число днів спостереження
                           //міські служби
        Police policeman;
        Ambulance ambulanceman;
        FireDetect fireman;
        //події в місті
        public event FireEventHandler Fire;
        string[] resultservice;      //результати дій служб
                                     //моделювання випадкових подій
        private Random rnd = new Random();
        //імовірність пожежі в будинку в поточний день
        double fireprobability;
        /// <summary>
        /// Конструктор міста
        /// Створює служби й включає спостереження
        /// за подіями
        /// </summary>
        /// <param name="name">назва міста</param>
        /// <param name="buildings">число будинків</param>
        /// <param name="days">число днів спостереження</param>
        public NewTown(string name, int buildings, int days)
        {
            townname = name;
```

```
            this.buildings = buildings;
            this.days = days;
            fireprobability = 1e-3;
            //Створення служб
            policeman = new Police(this);
            ambulanceman = new Ambulance(this);
            fireman = new FireDetect(this);
            //Підключення до спостереження за подіями
            policeman.On();
            ambulanceman.On();
            fireman.On();
        }

        /// <summary>
        /// Запалюється подія.
        /// По черзі викликаються оброблювачі події
        /// </summary>
        /// <param name="e">
        /// вхідні й вихідні аргументи події
        /// </param>
        protected virtual void OnFire(FireEventArgs e)
        {
            const string MESSAGE_FIRE =
                        "У місті {0} пожежа! Будинок {1}. День {2}-й";
            Console.WriteLine(string.Format(MESSAGE_FIRE, townname,
                e.Building, e.Day));
            if (Fire != null)
            {
                Delegate[] eventhandlers =
                    Fire.GetInvocationList();
                resultservice = new string[eventhandlers.Length];
                int k = 0;
                foreach (FireEventHandler evhandler in
                    eventhandlers)
                {
                    evhandler(this, e);
                    resultservice[k++] = e.Result;
                }
            }
        }
        /// <summary>
        /// Моделювання життя міста
        /// </summary>
        public void LifeOurTown()
        {
            const string OK =
               "У місті {0} усі спокійно! Пожеж не було.";
            bool wasfire = false;
            for (int day = 1; day <= days; day++)
                for (int building = 1; building <= buildings; building++)
                {
                    if (rnd.NextDouble() < fireprobability)
                    {
                        FireEventArgs e = new FireEventArgs(building, day);
                        OnFire(e);
                        wasfire = true;
                        for (int i = 0; i < resultservice.Length; i++)
                            Console.WriteLine(resultservice[i]);
                    }
                }
            if (!wasfire)
                Console.WriteLine(string.Format(OK, townname));
        }


    }
    public abstract class Receiver
    {
```

```
    protected NewTown town;
    protected Random rnd = new Random();
    public Receiver(NewTown town)
    { this.town = town; }

    public void On()
    {
        town.Fire += new FireEventHandler(It_is_Fire);
    }
    public void Off()
    {
        town.Fire -= new  FireEventHandler(It_is_Fire);
    }
    public abstract void It_is_Fire(object sender, FireEventArgs e);
}//class Receiver

public class Police : Receiver
{
    public Police(NewTown town) : base(town) { }
    public override void It_is_Fire(object sender, FireEventArgs e)
    {
        const string OK =
            "Міліція знайшла винних!";
        const string NOK =
            "Міліція не знайшла винних! Наслідок триває.";
        if (rnd.Next(0, 10) > 6)
            e.Result = OK;
        else e.Result = NOK;
    }
}// class Police
public class FireDetect : Receiver
{
    public FireDetect(NewTown town) : base(town) { }
    public override void It_is_Fire(object sender, FireEventArgs e)
    {
        const string OK =
             "Пожежні згасили пожежу!";
        const string NOK =
             "Пожежа триває! Потрібна допомога.";
        if (rnd.Next(0, 10) > 4)
            e.Result = OK;
        else e.Result = NOK;
    }
}// class Firedetect
public class Ambulance : Receiver
{
    public Ambulance(NewTown town) : base(town) { }
    public override void It_is_Fire(object sender, FireEventArgs e)
    {
        const string OK =
            "Швидка надала допомогу!";
        const string NOK =
            " Є постраждалі! Потрібні ліки.";
        if (rnd.Next(0, 10) > 2)
            e.Result = OK;
        else e.Result = NOK;
    }
}// class Ambulance

/// <summary>
/// Клас, що задає вхідні й вихідні аргументи події
/// </summary>
public class FireEventArgs : EventArgs
{
    int building;
    int day;
    string result;
    //Доступ до вхідних і вихідних аргументів
```

```
        public int Building
        { get { return building; } }
        public int Day
        { get { return day; } }
        public string Result
        {
            get { return result; }
            set { result = value; }
        }
        public FireEventArgs(int building, int day)
        {
            this.building = building; this.day = day;
        }
    }//class Fireeventargs

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            NewTown sometown = new NewTown("Канск", 20, 100);
            sometown.LifeOurTown();

        }
    }
}
```

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Клонувати репозиторій : https://classroom.github.com/a/bomjk7gQ .
2) Написати C# програми згідно з варіантом завдання
3) Підготувати звіт в електронному виді надіслати мудл(https://moodle.chnu.edu.ua/course/view.php?id=3371) .

```
//  За бажанням студента для задач можна створювати консольний проект або WinForm
// Бажано для задач лаб. робіт створити окремі класи
// Виконання  виконати в стилі багатозаданості :
//   Lab10T  lab10task2 = new Lab10T; lab10task2.Run();
// При бажанні можна створити багатозадачний режим виконання задач.
```

Задача 1.Обробка винятків

Реалізувати обробку помилок для однієї із задач з попередніх **лабораторних робот**, при цьому перевизначивши за допомогою спадкування класу **`Exception`** власні класи винятків, також реалізувати обробку стандартних винятків:

1) **`ArrayTypeMismatchException`**
2) **`DivideByZeroException`**
3) **`IndexOutOfRangeException`**
4) **`InvalidCastException`**
5) **`OutOfMemoryException`**
6) **`OverflowException`**
7) **`StackOverflowException`**

Задача 2. Моделювання подій.

1) Створіть проект "Життя міста", у якім відбуваються різні події.
2) Створіть проект "Життя факультету" з подією "День факультету".
3) Створіть проект "Життя факультету", у якім відбуваються різні події.
4) Створіть проект "Життя коня".

5) Створіть проект "Життя автомобіля".
6) Створіть проект "Життя пароплава".
7) Створіть проект "Життя студента".
8) Створіть проект "Робота конвеєра".
9) Створіть проект " Воєнні дії". Класи, що задають супротивників, взаємно обробляють події один одного. Наприклад, об'єкт одного класу запалює подія "атака", оброблювач цієї події в іншому класі у відповідь запалює подія "контратака".
10) Створіть проект "Бики й Ведмеді", де об'єкти класу "Бики" відіграють на біржі на підвищення, а "Ведмеді" - на зниження.
Досліджуйте можливість створення класу, у якім одні об'єкти цього класу створюють події, а інші об'єкти цього ж класу обробляють ці події.

Контрольні питання

1) Що розуміється під терміном «подія»?
2) чи є події членами класів?
3) Яке ключове слово мови C# використовується для опису подій?
4) На якому механізмі мови C# заснована підтримка подій?
5) Приведіть синтаксис опису події в загальному виді.
Проілюструйте його фрагментом програми мовою C#.
6) Що розуміється під терміном «широкомовна подія»?
7) На основі якого механізму мови C# будуються широкомовні події?
8) Приведіть синтаксис опису широкомовної події в загальному виді. Проілюструйте його фрагментом програми мовою C#.
9) Що розуміється під терміном «виняткова ситуація (винятків)»?
10) У чому полягає значення механізму винятку у мові C#?
11) Які оператори мови C# використовуються для обробки винятку?
12) Які оператори мови C# є найважливішими для обробки винятку?
13) Приведіть синтаксис блоку try…catch у загальному виді.
Проілюструйте його фрагментом програми мовою C#.
14) Приведіть п'ять видів основних системних винятку.
15) чи Необхідно забезпечувати відповідність типів винятків в операторові catch типу винятків, що перехоплюється?
16) Що відбувається у випадку невдалого перехоплення винятків?
17) У якому випадку можливе використання оператора мови C# catch без параметрів?
18) Яким образом здійснюється повернення в програму після обробки виняткової ситуації?
19) Який оператор мови C# використовується для забезпечення повернення

у програму після обробки винятків?
20) Приведіть синтаксис блоку finally (у складі оператора try…catch) у загальному виді. Проілюструйте його фрагментом програми мовою C#.
21) чи Залежить виклик блоку finally від наявності винятків?
22) Які способи генерації винятку Вам відомі?
23) Що є джерелом автоматично генеруемих (неявних) винятку?
24) Яким образом можливо здійснити явну генерацію винятку?
25) Який оператор мови C# використовується для явної генерації винятку?
26) Приведіть синтаксис оператора throw у загальному виді. Проілюструйте його фрагментом програми мовою C#.
27) Яким образом здійснюється повторне перехоплення винятку у мові C#?
28) чи Можливо створювати спеціалізовані винятків для обробки помилок у коді користувача?
29) Який системний клас є базовим для створення винятку?
30) На основі якого системного класу здійснюється генерація користувацьких винятку?
31) чи Необхідна явна реалізація класів, наслідуваних від системних винятку?
32) Яким образом забезпечується звертання до властивостей і методам системних винятку?