Лабораторна робота 6

ВИКОРИСТАННЯ ОБРОБНИТКІВ ВИНЯТКІВ

Мета роботи: опанувати написання програм з використанням блоків з винятками.

Короткі теоретичні відомості

Використання механізму обробки виняткових ситуацій є дуже важливою складовою частиною практики програмування на всіх сучасних об'єктноорієнтованих мовах. Об'єкти-винятки дозволяють програмісту відокремити точки виникнення помилок часу виконання від коду, де ці помилки повинні оброблятися. Виняткова ситуація являє собою подію, що виникає під час виконання програми та порушує нормальне виконання інструкцій коду. Для генерації виняткової ситуації використовується оператор *throw*. Після ключового слова *throw* повинен бути розташований об'єкт класу System.Exception чи класів, похідних від нього. Такі похідні класи відбивають специфіку конкретної програми.

```
class SpecificException : Exception
{
}
```

Клас System.Exception містить ряд властивостей, за допомогою яких можна одержати доступ до інформації про виняткову ситуацію, зокрема:

Message – текстовий опис помилки, що задається як параметр конструктора під час створення об'єкта-винятку, Source – ім'я об'єкта чи застосунку, що згенерувало помилку, StackTrace - послідовність викликів, що привели до виникнення помилки. У більшості випадків об'єкт-виняток створюється в місці генерації виняткової ситуації за допомогою оператора new, однак іноді об'єктвиняток створюється заздалегідь. Типове твердження throw може виглядати так:

```
void F()
{  . . .  if (/* помилка */)     throw new SpecificException();  . . .
```

}

У заголовку функції не специфікуються типи винятків, які генеруються цією функцією. У наступному прикладі функція Reciprocal() генерує виняткову ситуацію у випадку ділення на нуль.

```
class DivisionByZero : Exception
{ }  class Test  {   public double
Reciprocal(double x)
 {   if (x == 0)       throw new
DivisionByZero();   return 1 / x;
 } }
```

На відміну від C++, C# не допускає створення винятків примітивних типів. Дозволені тільки об’єкти класів, похідних від Exception. У блоці *try* розміщують код, що може генерувати виняткову ситуацію: double x, y;

```
. . . try  {   y = Reciprocal(x);
}
```

Після блоку *try* повинен йти один чи кілька оброблювачів (блоків catch). Кожен такий оброблювач відповідає визначеному типу винятку. Блок catch без дужок обробляє всі інші виняткові ситуації:

```
catch (DivisionByZero d)
{
 // обробка виняткової ситуації
} catch (SpecificException)
{
 // обробка виняткової ситуації
} catch  {
 // обробка виняткової ситуації
}
```

Як видно з прикладу, у заголовку блоку catch можна опускати ідентифікатор об’єкта-винятку, якщо важливий тільки тип.

Класи винятків утворять ієрархію. При зіставленні типів винятків, оброблювач базового типу сприймає також винятки всіх створених від нього типів. Звідси випливає, що оброблювачі похідних типів варто розміщувати до оброблювачів базових типів.

У деяких випадках оброблювач виняткових ситуацій не може цілком обробити виняток і повинен передати його зовнішньому оброблювачу. Це можна зробити за допомогою оператора throw: catch (SomeEx ex)

```
{
  // локальна обробка виняткової ситуації   throw (ex);  // повторна генерація
}
```

Якщо в заголовку оброблювача не визначений ідентифікатор, то можна використовувати *throw* без виразу:

```
catch (Exception)   {
  // локальна обробка виняткової ситуації   throw;
}
```

Після останнього блоку catch можна розмістити блок *finally*. Цей код завжди виконується незалежно від того, виникла виняткова ситуація чи ні.

```
try  {   openFile();   // інші дії
} catch (FileError f)
{
  // обробка виняткової ситуації  }
catch (Exception ex) {
  // обробка виняткової ситуації
} finally {   closeFile();
}
```

В .NET Framework визначені стандартні особливі ситуації - класи, що теж є нащадками Exception. Один із найчастіше виникаючих стандартних винятків - System.NullReferenceException, який генерується при спробі звертатися до елементів класу або структури через посилання, яке дорівнює null. Виняток

System.IndexOutOfRangeException генерується, коли відбувається вихід за межі масиву.

Внутрішні виняткові ситуації .NET сигналізують про серйозну проблему під час виконання програми і можуть виникнути при виконанні будьякого оператора. До них відносяться ExecutionEngineException (внутрішня помилка CLR), StackOverflowException (переповнення стеку), OutOfMemoryException (брак оперативної пам'яті). Зазвичай такі винятки не перехоплюються.

Реалізувати обробку винятків в трьох варіантах:

- використовувати відповідні стандартні винятки;
- використовувати – винятки-спадкоємці від стандартних винятків; – визначити власні – винятки.

Варіант 1. Клас *Vector3d* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 2. Клас *Money* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 3. Клас *Triangle* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 4. Клас *Angle* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 5. Клас *Data* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 6. Клас *Time* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 7. Клас *Account* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 8. Клас *Fraction* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 9. Клас *Goods* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 10. Клас *Payment* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 11. Клас *Vector3d* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 12. Клас *Money* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 13. Клас *Triangle* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 14. Клас *Angle* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 15. Клас *Data* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 16. Клас *Time* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 17. Клас *Account* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 18. Клас *Fraction* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 19. Клас *Goods* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 20. Клас *Payment* з конструкторами, перевантаженням операцій і обробкою винятків.

Варіант 10. Клас *Payment* з конструкторами, перевантаженням операцій і обробкою винятків.

Реалізація програми з використанням стандартного винятку:

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

namespace Learn.NET
{

    class Payment
    {
        private double amount;

        public Payment(double amount)
        {
            if (amount <= 0)
            {
                throw new ArgumentOutOfRangeException("Сума платежу не
може бути меншою або дорівнювати 0.");
            }
            this.amount = amount;
        }

        public static Payment operator +(Payment lhs, Payment rhs)
        {
            return new Payment(lhs.amount + rhs.amount);
        }

        public double GetAmount()
        {
            return amount;
        }
    }


    class lab7_defaultException
    {
        static void Main(string[] args)
        {
            try
            {
                Payment payment1 = new Payment(-100); // Отримання
вийнятку PaymentNegativeAmountException
            }
            catch (ArgumentOutOfRangeException ex)
            {
                Console.WriteLine("Помилка: " + ex.Message);
```

```
            }

            Payment payment2 = new Payment(200);
            Payment payment3 = new Payment(300);
            Payment totalPayment = payment2 + payment3;

            Console.WriteLine("Загальна сума платежу: " +
totalPayment.GetAmount());
            Console.ReadLine();
        }
    }
}
```

Результат роботи програми зображено на рисунку 6.1 нижче.

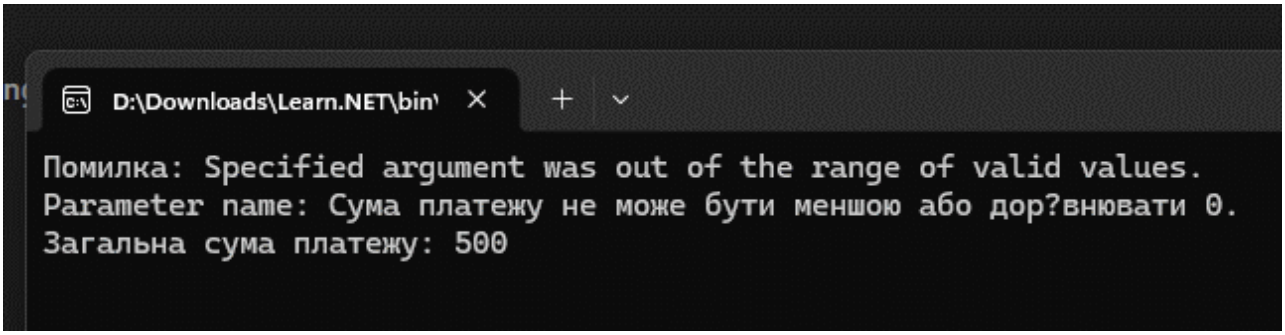

Рисунок 6.1 – Генерація стандартного виключення

Код програми із спадкуванням існуючого виключення:

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

namespace Learn.NET
{
    class PaymentNegativeAmountException : ArgumentOutOfRangeException
//Клас спакдоємець суперкласу стандартного вийнятку
    {
        public PaymentNegativeAmountException() : base("Сума платежу не
може бути меншою або дорівнювати 0.") { }
    }

    class Payment
    {
        private double amount;
```

```
        public Payment(double amount)
        {
            if (amount <= 0)
            {
                throw new PaymentNegativeAmountException();
            }
            this.amount = amount;
        }

        public static Payment operator +(Payment lhs, Payment rhs)
//Перенавнтаження - знаходження суми дво платежів
        {
            return new Payment(lhs.amount + rhs.amount);
        }

        public double GetAmount()
        {
            return amount;
        }
    }

    class lab7_inheritException
    {
        static void Main(string[] args)
        {
             try
        {
            Payment payment1 = new Payment(-100); // Отримання вийнятку
PaymentNegativeAmountException
        }
        catch (PaymentNegativeAmountException ex)
        {
            Console.WriteLine("Помилка: " + ex.Message);
        }

        Payment payment2 = new Payment(200);
        Payment payment3 = new Payment(300);
        Payment totalPayment = payment2 + payment3;

        Console.WriteLine("Загальна сума платежу: " +
totalPayment.GetAmount());
        Console.ReadLine();
        }
    }
}
```

Результат виконання програми зображено на рисунку 6.2.

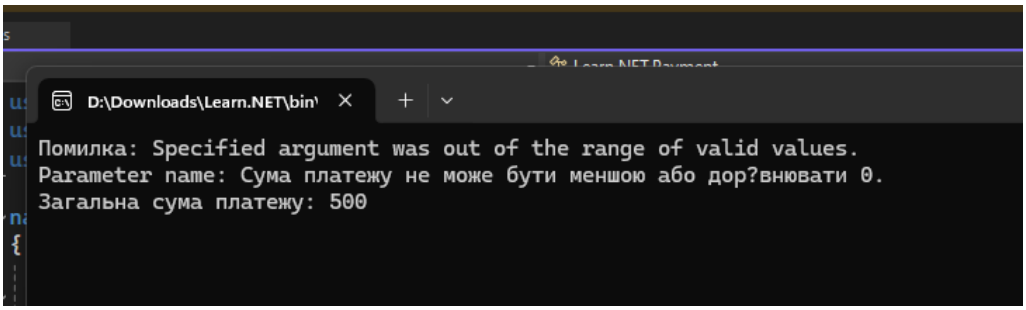

Рисунок 6.2 – Успадкування для створення власного винятку(результат)

Для створення власного виключення створюється новий клас, який спадковує суперклас Exception. Код програми:

```
using System;

namespace Learn.NET
{

    class PaymentNegativeAmountException : Exception
    {
        public PaymentNegativeAmountException() : base("Сума платежу не
може бути меншою або дорівнювати 0.") { }
    }

    class Payment
    {
        private double amount;

        public Payment(double amount)
        {
            if (amount <= 0)
            {
                throw new PaymentNegativeAmountException();
            }
            this.amount = amount;
        }

        public static Payment operator +(Payment lhs, Payment rhs)
        {
            return new Payment(lhs.amount + rhs.amount);
        }

        public double GetAmount()
        {
            return amount;
        }
    }
    class Program
    {
```

```
        public static void Main(string[] args)
        {
            try
            {
                Payment payment1 = new Payment(-100); // Гереація
винятку PaymentNegativeAmountException
            }
            catch (PaymentNegativeAmountException ex)
            {
                Console.WriteLine("Помилка: " + ex.Message);
            }

            Payment payment2 = new Payment(200);
            Payment payment3 = new Payment(300);
            Payment totalPayment = payment2 + payment3;

            Console.WriteLine("Загальна сума платежу: " +
totalPayment.GetAmount());
            Console.ReadLine();
        }
    }
}
```

Результат роботи програми зображено на рисунку 6.3.

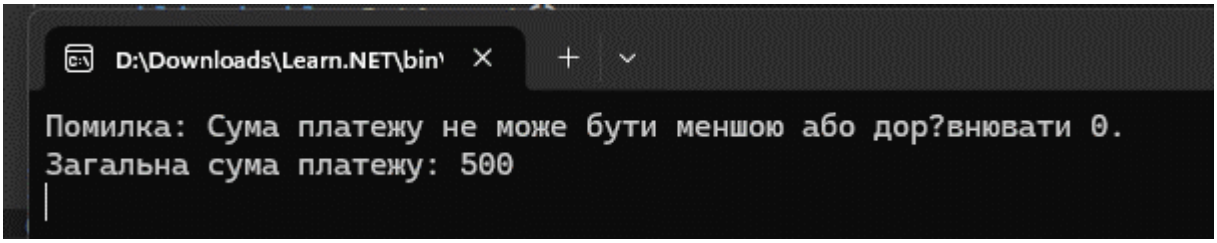

Рисунок 6.3 – Створення власного винятку(результат)