МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ТЕХНОЛОГІЇ ПРОГРАМУВАННЯ ІВС
ПРАКТИКУМ

Навчальний посібник

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інформаційні вимірювальні технології»
спеціальності 152 Метрологія та інформаційно-вимірювальна техніка

Укладачі: М. В. Добролюбова, М. В. Філіппова, О. В. Стаценко

Електронне мережне навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2023

Рецензент *Тішков, Б. О.,* к.е.н., доцент,
в.о. завідувача кафедри, кафедра інформаційних систем в економіці, ІІТЕ, КНЕУ імені Вадима Гетьмана

Відповідальний редактор *Богомазов, С. А.,* к.т.н., доцент

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 5 від 23.02.2023 р.)
за поданням Вченої ради приладобудівного факультету
(протокол № 1/23 від 30.01.2023 р.)

Навчальний посібник забезпечує курс комп'ютерних практикумів з дисципліни «Технології програмування ІВС».
Навчальний посібник сприяє набуттю студентами повного розуміння синтаксису та семантики мови програмування C#, засвоєнню принципів об'єктно-орієнтованого програмування цією мовою, розумінню подієво-орієнтованого, структурного, функціонального та аспектно-орієнтованого підходів, що надає розширені можливості для створення програмних додатків інформаційних вимірювальних систем.

Реєстр. № НП 22/23-431. Обсяг 9 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Перемоги, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© КПІ ім. Ігоря Сікорського, 2023

ПЕРЕДМОВА

Навчальний посібник «Технології програмування ІВС. Практикум» складено авторами на основі багаторічного досвіду викладання програмування. Він відповідає Силабусу вибіркової дисципліни «Технології програмування ІВС», яка викладається для студентів третього курсу за спеціальністю 152 «Метрологія та інформаційно-вимірювальна техніка».

Навчальний посбіник орієнтований на організацію самостійної роботи студентів з вивчення і систематизації знань за темами, які стосуються базового та поглибленого рівня опанування мови програмування C# – основних понять та парадигм об'єктно-орієнтованого програмування мовою C#, а саме: поняття класів та об’єктів, наслідування та поліморфізм, абстрактні класи та інтерфейси, структури, делегати, універсальні шаблони, потоки, колекції, базові поняття технологій LINQ та архітектури .NET Framework.

Курс комп’ютерних практикумів містить теоретичний матеріал, велику кількість практичних прикладів, загальні і індивідуальні завдання, рекомендовані ресурси та контрольні питання з відповідних тем, необхідних для закріплення здобутих знань і навичок.

ЗМІСТ

Комп'ютерний практикум №1
КЛАСИ ТА ОБ'ЄКТИ В МОВІ C# ---------- 5

Комп'ютерний практикум №2
НАСЛІДУВАННЯ ТА ПОЛІМОРФІЗМ ---------- 32

Комп'ютерний практикум №3
АБСТРАКТНІ КЛАСИ ТА ІНТЕРФЕЙСИ ---------- 50

Комп'ютерний практикум №4
СТРУКТУРИ ---------- 71

Комп'ютерний практикум № 5
ДЕЛЕГАТИ ---------- 99

Комп'ютерний практикум №6
УНІВЕРСАЛЬНІ ШАБЛОНИ ---------- 113

Комп'ютерний практикум №7
ПОТОКИ---------- 135

Комп'ютерний практикум № 8
КОЛЕКЦІЇ ---------- 146

Комп'ютерний практикум №9
LINQ ---------- 163

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ ---------- 191

ДОДАТОК А---------- 196

Комп'ютерний практикум №1

КЛАСИ ТА ОБ'ЄКТИ В МОВІ C#

Мета роботи: розгляд поняття класів, екземплярів класів, основних членів класу, властивостей і конструкторів, часткових класів і часткових методів, основних парадигм ООП, діаграм класів UML, зв'язків відношень між класами (самоасоціації, асоціація) [1-15].

Теоретичні відомості

- **ООП** – об'єктно-орієнтоване програмування – парадигма програмування, в якій основними концепціями є поняття об'єктів і класів.
- **Клас** (лат. classis – група) – група предметів або явищ, що володіють загальними ознаками.
- **Клас** – це конструкція мови, що складається з ключового слова class, ідентифікатора і тіла. Клас може містити в своєму тілі поля і методи. Також класи можуть включати в своє тіло інші класи, але такий підхід не є поширеною технікою.
- **Об'єкт** – це деяка сутність, що володіє певним станом і поведінкою, має задані значення властивостей (полів) і операцій над ними (методів).
- Об'єкт складається з наступних частин:
  1) ім'я об'єкта;
  2) стан (змінні стану);
  3) методи (операції).
- **Властивість** – це спосіб доступу до внутрішнього стану об'єкта, що імітує змінну деякого типу. Звернення до властивості об'єкта виглядає так само, як і звернення до структурного поля (в структурному програмуванні), але, в дійсності, реалізоване через виклик функції. При спробі задати значення даної властивості викликається один метод, а при спробі отримати значення даної властивості – інший.
- **Екземпляр класу** (instance) – це опис конкретного об'єкта в пам'яті.
- **Інстанціювання** (instantiation) – створення екземпляра класу. На відміну від слова «створення», застосовується не до об'єкта, а до класу. Тобто кажуть: «створити екземпляр класу або інстанціювати клас».
- **Конструктор класу** (constructor, іноді скорочують ctor) – спеціальний блок інструкцій, що викликається при створенні об'єкта.
- Рекомендується завжди створювати явно конструктор за замовчуванням.
- Рекомендується використовувати ту ж саму назву для параметрів конструктора і поля або властивості, якщо параметри конструктора використовуються для того, щоб форматувати поле або встановити властивість.
- Рекомендується мінімізувати роботу конструктора.

- Конструктори бувають двох видів конструктори типу і конструктори екземпляра класу. Конструктори типу є статичними і виконуються середовищем CLR до використання типу. Конструктори екземпляра класу працюють тоді, коли створюється екземпляр класу.
- Конструктори типу не можуть приймати параметри.
- Конструктор не буває віртуальним (тобто virtual).
- Конструктори екземпляра можуть приймати параметри.
- Конструктори екземпляра класу, які не приймають параметрів, називають конструкторами за замовчуванням.
- Конструктори екземпляра класу, які приймають параметри, називають конструкторами користувача.
- **CTS** – Common Type System (загальна система типів) – це специфікація, яка визначає, як тип повинен бути визначений для правильного виконання середовищем .Net.
- Специфікація для CTS закріплена в стандарті Ecma 335 під назвою «Common Language Infrastructure (CLI) Partitions I to VI». CLI і CTS були розроблені корпорацією Microsoft, а .NET Framework – реалізація цього стандарту.
- В C# реалізована можливість розділити створення класу або методу (структури, інтерфейсу) між двома або більше вихідними файлами або модулями. Кожен вихідний файл містить визначення типу або методу і всі частини об'єднуються при компіляції програми.
- Існує кілька ситуацій, при яких бажане розділення створення класу:
  1) При роботі над великими проектами розподіл класу між різними файлами дозволяє декільком програмістам працювати з ним одночасно.
  2) При роботі з використанням автоматично створюваного джерела, код можна додавати в клас без повторного створення файлу джерела. Наприклад: при створенні форм Windows Forms.
- Щоб розділити клас на кілька частин, використовуйте ключове слово partial.
- Ключове слово partial вказує на те, що інші частини класу (структури або інтерфейсу) можуть бути створені в цьому просторі імен. Всі частини повинні використовувати ключове слово partial. Всі частини повинні мати однакові модифікатори доступу, наприклад public, private тощо.
- Частковий клас або структура можуть містити частковий метод.
- **Часткові методи** – це методи, де «прототип» або сигнатура методу визначена при створенні часткового класу, а реалізація виконується в будь-якій інший (тільки одній) частині цього класу.
- Перевага використання таких методів полягає в тому, що їх реалізацію можна не виконувати. В такому випадку, при компіляції коду даний метод навіть не компілюється і не потрапляє в IL-код. Тобто, як ніби цей метод взагалі не існував.

- Правила використання часткових методів:
  1) Часткові методи повинні бути визначені тільки в часткових класах.
  2) Часткові методи повинні бути позначені ключовим словом partial.
  3) Часткові методи є прихованими (private), але явне використання з ними модифікатора доступу призведе до помилки.
  4) Часткові методи повинні повертати void.
  5) Часткові методи можуть бути нереалізованими.
  6) Часткові методи можуть не мати аргументів.
- **Ключове слово** readonly – це модифікатор, який можна використовувати для полів.
- Якщо створене поле містить модифікатор readonly, присвоїти значення такому полю можна тільки безпосередньо в місці створення або в конструкторі того ж класу.
- Ключове слово readonly відрізняється від ключового слова const тим, що поле з модифікатором const може бути ініціалізоване тільки при створенні поля. Поле з модифікатором readonly може бути ініціалізоване при створенні або в конструкторі. Отже, поля з модифікатором readonly можуть мати різні значення в залежності від використаного конструктора.
- **Парадигма програмування** – це система ідей і понять, що визначають стиль написання комп'ютерних програм, а також образ мислення програміста.
- До основних парадигм ООП належать:
  1) **Інкапсуляція** – це властивість системи, що дозволяє об'єднати дані і методи, що працюють з ними, в класі і приховати деталі реалізації від користувача.
  2) **Наслідування** – це властивість системи, що дозволяє описати новий клас на основі вже існуючого.
  3) **Поліморфізм** – можливість об'єктів з однаковою специфікацією мати різну реалізацію.
  4) **Абстракція** – це надання об'єкту характеристик, які чітко визначають його концептуальні границі, відрізняючи від всіх інших об'єктів. Дозволяє працювати з об'єктами, не вдаючись у особливості їх реалізації.
  5) **Здійснення повідомлень** – це спосіб передачі управління об'єкту. Якщо об'єкт повинен «відповідати» на це повідомлення, то у нього має бути метод, який відповідає цьому повідомленню.
  6) **Повторне використання** – парадигма ООП в якій стверджується, що програми (комп'ютерна програма, програмний модуль) частково або повністю повинні складатися з двох частин, написаних раніше компонентів і / або частин іншої програми (системи). Це основна методологія, яка застосовується для скорочення трудовитрат при розробці складних систем.

- **UML** (англ. Unified Modeling Language – уніфікована мова моделювання) – мова графічного опису для об'єктного моделювання в області розробки програмного забезпечення.
- UML є мовою широкого профілю, це відкритий стандарт, який використовує графічні позначення для створення абстрактної моделі системи, званої UML-моделлю. UML була створена для визначення, візуалізації, проектування та документування в основному програмних систем. UML не є мовою програмування.
- Розроблена в 1994 р Граді Бучем, Джеймсом Рамбо і Іваром Якобсоном.
- **Діаграма класів** (class diagram) – статична структурна діаграма, що описує структуру системи, вона демонструє класи системи, їх атрибути, методи і залежності між класами.
- **Взаємозв'язок** – це особливий тип логічних відношень між сутностями, показаних на діаграмах класів і об'єктів.
- **Асоціація** показує, що об'єкти однієї сутності (класу) пов'язані з об'єктами іншої сутності.
- **Агрегація** – це різновид асоціації при відношенні між цілим і його частинами. (Відношення типу: «Я знаю про ... і без цього можу існувати»). Одне відношення агрегації не може включати більше двох класів (контейнер і вміст).
- **Композиція** – більш суворий варіант агрегації. Композиція має жорстку залежність часу існування екземплярів класу-контейнера і екземплярів вмістовних класів.
- Якщо контейнер буде знищений, то весь його вміст буде також знищено. (Відношення типу: «Я знаю про ... і без цього не можу існувати»).
- **Залежність** – це слабка форма відношення використання, при якому зміна в специфікації одного тягне за собою зміну іншого, причому зворотне не обов'язково.

Практичні приклади

Приклад 1 Classes

```
using System;

// Класи.

namespace Classes
{
    // Створюємо клас з ім'ям MyClass.
    // В тілі класу створюємо відкрите поле типу string з ім'ям field та метод з ім'ям
Method.

    class MyClass
    {
        public string field;

        public void Method()
        {
            Console.WriteLine(field);
        }
    }
```

```
    class Program
    {
        static void Main()
        {
            // 1. Створюємо екземпляр класу MyClass (за сильним посиланням).
            // 2. Створюємо екземпляр класу MyClass з ім'ям instance.
            // 3. Інстанцируємо клас MyClass.
            // 4. Створюємо змінну з ім'ям instance, типу MyClass та присвоюємо їй
адресу екземпляра на купі.
            // (instance - посилання на екземпляр класу MyClass, який побудований на
купі)

            MyClass instance = new MyClass();

            // Полю field, екземпляра instance, присвоюємо значення Hello world!

            instance.field = "Hello world!";

            // Виводимо на екран значення поля field екземпляра instance.

            Console.WriteLine(instance.field);

            // Виклик метода з ім'ям Method на екземплярі instance.
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Properties

```
using System;

// Класи.

// Використання методів доступу до закритих полів.

namespace Classes
{
    class MyClass
    {
        private string field = null;

        public void SetField(string value) // Метод-мутатор - mutator  (setter)
        {
            field = value;
        }

        public string GetField()           // Метод-аксесор -  accessor  (getter)
        {
            return field;
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass instance = new MyClass();

            instance.SetField("Hello world!");     // Метод-мутатор

            string @string = instance.GetField();  // Метод-аксесор

            Console.WriteLine(@string);

            // Delay.
```

```
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Properties

```
using System;

// Класи. (Використання властивостей, для доступу до закритих полів.)

// Властивість — інтерфейс доступу до поля об'єкта.
// Властивості в C# — поля з логічним блоком, в якому є ключові слова get та set,
// що, в свою чергу, є суррогатом для заміни методів доступу до поля.
// При зверненні до властивості викликається певний метод, який виконує певні операції
з обієктом.

namespace Classes
{
    class MyClass
    {
        private string field = null;

        public string Field
        {
            set  // void SetField(string value)    -    Метод-мутатор - mutator
(setter)
            {
                field = value;
            }

            get  // string GetField()              -    Метод-аксесор - accessor
(getter)
            {
                return field;
            }
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass instance = new MyClass();

            instance.Field = "Hello world!";    // Метод-мутатор

            Console.WriteLine(instance.Field);  // Метод-аксесор

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Properties

```
using System;

// Класи. (Використання властивостей для доступу до закритих полівм.)

namespace Classes
{
    class MyClass
    {
        private string field = null;

        public string Field
        {
            set
```

```
            {
                if (value == "Goodbye")
                    Console.WriteLine("Ви ввели неприпустиме значення. Повторіть
спробу.");
                else
                    field = value;
            }

            get
            {
                if (field == null)
                    return "В полі field відсутні дані.";
                else if (field == "hello world")
                    return field.ToUpper() + "!";
                else
                    return field;
            }
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass instance = new MyClass();

            instance.Field = "Goodbye";
            Console.WriteLine(instance.Field);

            Console.WriteLine(new string('-', 50));

            instance.Field = "hello world";
            Console.WriteLine(instance.Field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Properties

```
using System;

// Класи. Властивості з одним методом доступу.

namespace Classes
{
    class Constants
    {
        private double pi = 3.14D;
        private double e = 2.71D;

        // Властивість лише для запису.  - WriteOnly Property
        public double Pi
        {
            set { pi = value; }
        }

        // Властивість лише для читання.  -  ReadOnly Property
        public double E
        {
            get { return e; }
        }
    }

    class Program
    {
        static void Main()
        {
```

```
            Constants constants = new Constants();

            constants.Pi = 3.14159265D;
            //Console.WriteLine(constants.Pi); // Неприпустимо.

            //constants.E = 3.71D;              // Неприпустимо.
            Console.WriteLine("e = {0}", constants.E);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 Constructors

Program.cs

```
using System;

// Конструктори.

namespace Classes
{
    class Program
    {
        static void Main()
        {
            // Застосовуємо конструктор за замовчуванням.
            Point pointA = new Point();
            Console.WriteLine("pointA.X = {0} pointA.Y = {1}", pointA.X, pointA.Y);

            Console.WriteLine(new string('-', 30));

            // Застосовуємо конструктор з параметрами.
            Point pointB = new Point(100, 200);
            Console.WriteLine("pointB.X = {0} pointB.Y = {1}", pointB.X, pointB.Y);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Point.cs

```
using System;

namespace Classes
{
    class Point
    {
        // Поля.
        private int x, y;

        // Властивості.
        public int X
        {
            get { return x; }
        }

        public int Y
        {
            get { return y; }
        }

        // Конструктор за замовчуванням, ініціалізує поля значеннями за замовчуванням.
        public Point()
        {
            Console.WriteLine("Конструктор за замовчуванням!");
        }
```

```
        // Конструктор користувача, ініціалізує поля значеннями, які задані
користувачем.
        public Point(int x, int y)
        {
            Console.WriteLine("Конструктор користувача!");
            this.x = x;
            this.y = y;
        }
    }
}
```

Приклад 7 Constructors

Program.cs

```
using System;

// Конструктори.

// ВАЖЛИВО:
// Якщо ви створили конструктор користувача (що принимає аргументи),
// то конструктор за замовчуванням, автоматично створюватися НЕ БУДЕ, його необхідно
буде створити явно.

namespace Classes
{
    class Program
    {
        static void Main()
        {
            // Застосовуємо конструктор за замовчуванням.
            //Point pointA = new Point();
//  Зняти
            //Console.WriteLine("pointA.X = {0} pointA.Y = {1}", pointA.X, pointA.Y);
//  коментар

            Console.WriteLine(new string('-', 30));

            // Застосовуємо конструктор з параметрами.
            Point pointB = new Point(100, 200);
            Console.WriteLine("pointB.X = {0} pointB.Y = {1}", pointB.X, pointB.Y);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Point.cs

```
using System;
// ВАЖЛИВО:
// Якщо ви створили конструктор користувача (що принимає аргументи),
// то конструктор за замовчуванням, автоматично створюватися НЕ БУДЕ, його необхідно
буде створити явно.
namespace Classes
{
    class Point
    {
        // Поля.
        private int x, y;

        // Властивості.
        public int Y
        {
            get { return y; }
        }
        public int X
        {
            get { return x; }
        }
        // Конструктор за замовчуванням, ініціалізує поля значеннями за замовчуванням.
```

```
        //public Point()
        //{
        //    Console.WriteLine("Конструктор за замовчуванням!");
        //}
        // Конструктор користувача, ініціалізує поля значеннями, які задані
користувачем.
        public Point(int x, int y)
        {
            Console.WriteLine("Конструктор користувача!");
            this.x = x;
            this.y = y;
        }
    }
}
```

Приклад 8 Constructors

Program.cs

```
using System;

// Конструктори.

namespace Classes
{
    class Program
    {
        static void Main()
        {
            Point point = new Point("A");
            Console.WriteLine("{0}.X = {1}, {0}.Y = {2}", point.Name, point.X,
point.Y);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Point.cs

```
using System;

// Конструктор може викликати в тому ж самому об'єкті інший конструктор за допомогою
ключового слова this.

namespace Classes
{
    class Point
    {
        // Поля.

        private int x, y;
        private string name;

        // Властивості.
        public int X
        {
            get { return x; }
        }

        public int Y
        {
            get { return y; }
        }

        public string Name
        {
            get { return name; }
        }

        // Конструктори.
```

```
        // Використання ключового слова this в конструкторі з одним параметром з 43-го
рядка
        // призводить до виклика цього конструктора.
        public Point(int x, int y)
        {
            Console.WriteLine("Конструктор з двома параметрами.");
            this.x = x;
            this.y = y;
        }

        // Використання ключового слова this в конструкторі призводить до виклика
конструктора з двома параметрами з 35-го рядка.
        public Point(string name)
            : this(300, 400)
        {
            Console.WriteLine("Конструктор з одним параметром.");
            this.name = name;
        }
    }
}
```

Приклад 9 Classes

```
using System;

// Передача екземпляра класа в якості аргумента метода

namespace Classes
{
    class MyClass
    {
        public void Method()
        {
            Console.WriteLine("Викликано метод класу MyClass");
        }
    }

    class MyClass2
    {
        // На 18 рядку створюємо метод з ім'ям CallMethod, який приймає один аргумент
типу MyClass і нічого не повертає
        public void CallMethod(MyClass my)
        {
            // В тілі метода CallMethod, на 21 рядку викликаємо метод з ім'ям Method
аргумента my
            my.Method();
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass my = new MyClass();

            MyClass2 my2 = new MyClass2();

            my2.CallMethod(my);

            // Затримка
            Console.ReadKey();
        }
    }
}
```

Приклад 10 AutoProperties

```
using System;
```

```
// Автоматично реалізуємі властивості (Auto-Implemented properties).

// Автоматично реалізуємі властивості це більш лаконічна форма властивостей, їх є сенс використовувати,
// коли в методах доступа (get та set) не потрібна додаткова логіка.
// При створенні автоматично реалізуємих властивостей, компілятор створить закрите, анонімне резервне поле,
// яке буде доступно за допомогою методів get та set властивості.

namespace Classes
{
    class Program
    {
        public class Author
        {
            // Автоматично реалізуємі властивості.
            public string Name { get; set; }
            public string Book { get; set; }
        }

        static void Main()
        {
            Author author1 = new Author()
            {
                Name = "Jeffrey Richter",            // Блок ініціалізатора.
                Book = "CLR via C#"
            };

            Author author2 = new Author        // ()
            {
                Name = "Steve McConnell",            // Блок ініціалізатора.
                Book = "Code Complete"
            };

            Console.WriteLine("Name: {0}, Book: {1}", author1.Name, author1.Book);
            Console.WriteLine("Name: {0}, Book: {1}", author2.Name, author2.Book);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 11 Classes

```
using System;

// Створення екземпляра за слабким посиланням. (Анонімні об'єкти)

namespace Classes
{
    class MyClass
    {
        public void Method()
        {
            Console.WriteLine("Hello world!");
        }
    }

    class Program
    {
        static void Main()
        {
            new MyClass().Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 Document

Program.cs

```
using System;
namespace Classes
{
    class Program
    {
        static void Main()
        {
            Title title = new Title();
            title.Content = "Контракт";

            Body body = new Body();
            body.Content = "Тіло контракту...";

            Footer footer = new Footer();
            footer.Content = "Директор: Петренко П.П.";

            Document document = new Document(title, body, footer);
            document.Show();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Document.cs

```
namespace Classes
{
    class Document
    {
        // Поля
        Title title;
        Body body;
        Footer footer;

        public Document(Title title, Body body, Footer footer)
        {
            this.title = title;
            this.body = body;
            this.footer = footer;
        }

        public void Show()
        {
            this.title.Show();
            this.body.Show();
            this.footer.Show();
        }
    }
}
```

Parts

Body.cs

```
using System;
namespace Classes
{
    class Body
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Тіло документа відсутнє.";
```

```
            }
            set
            {
                content = value;
            }
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Footer.cs

```
using System;

namespace Classes
{
    class Footer
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Нижній колонтитул відсутній.";
            }
            set
            {
                content = value;
            }
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Title.cs

```
using System;

namespace Classes
{
    class Title
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Заголовок відсутній.";
            }
            set
            {
                content = value;
```

```
            }
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Приклад 13 Document

Program.cs

```
using System;

namespace Classes
{
    class Program
    {
        static void Main()
        {
            Document document = new Document("Контракт");
            document.Body = "Тіло контракта...";
            document.Footer = "Директор: Петренко П.П.";

            document.Show();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Document.cs

```
namespace Classes
{
    class Document
    {
        // Поля
        Title title = null;
        Body body = null;
        Footer footer = null;

        void InitializeDocument()
        {
            this.title = new Title();
            this.body = new Body();
            this.footer = new Footer();
        }

        public Document(string title)
        {
            InitializeDocument();
            this.title.Content = title;
        }

        public void Show()
        {
            this.title.Show();
            this.body.Show();
            this.footer.Show();
        }

        public string Body
        {
            set
            {
                this.body.Content = value;
```

```
            }
        }

        public string Footer
        {
            set
            {
                this.footer.Content = value;
            }
        }
    }
}
```

Parts

Body.cs

```
using System;

namespace Classes
{
    class Body
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Тіло документу відсутнє.";
            }
            set
            {
                content = value;
            }
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Blue;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Footer.cs

```
using System;

namespace Classes
{
    class Footer
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Нижній колонтитул відсутній.";
            }
            set
            {
                content = value;
            }
```

```
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Title.cs

```
using System;

namespace Classes
{
    class Title
    {
        string content;

        public string Content
        {
            private get
            {
                if (content != null)
                    return content;
                else
                    return "Заголовок відсутній.";
            }
            set
            {
                content = value;
            }
        }

        public void Show()
        {
            Console.ForegroundColor = ConsoleColor.Yellow;
            Console.WriteLine(Content);
            Console.ForegroundColor = ConsoleColor.Gray;
        }
    }
}
```

Приклад 14 PartialClasses

Program.cs

```
using System;

// Часткові класи (Partial classes).

namespace Classes
{
    class Program
    {
        static void Main()
        {
            PartialClass instance = new PartialClass();

            instance.MethodFromPart1(); // Метод з першої частини класу PartialClass
            instance.MethodFromPart2(); // Метод з другої частини класу PartialClass.

            // Delay.
            Console.ReadKey();
        }
    }
}
```

PartialClass_1.cs

```
using System;

namespace Classes
{
    // Перша частини класу.
    partial class PartialClass
    {
        public void MethodFromPart1()
        {
            Console.WriteLine("I Part");
        }
    }
}
```

PartialClass_2.cs

```
using System;

namespace Classes
{
    // Друга частини класу.
    partial class PartialClass
    {
        public void MethodFromPart2()
        {
            Console.WriteLine("II Part");
        }
    }
}
```

Приклад 15 PartialMethods

Program.cs

```
using System;

// Часткові методи (Partial methods).

namespace Classes
{
    class Program
    {
        static void Main()
        {
            PartialClass instance = new PartialClass();

            instance.CallPartialMethod();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

PartialClass_1.cs

```
using System;

namespace Classes
{
    // Перша частини класу.
    partial class PartialClass
    {
        // Оголошення часткового методу.
        partial void PartialMethod();

        partial void MyMethod();
    }
}
```

PartialClass_2.cs

```
using System;

namespace Classes
{
    // Друга частина класу.
    partial class PartialClass
    {
        // Визначення часткового методу.
        partial void PartialMethod()
        {
            Console.WriteLine("Викликано частковий метод - PartialMethod()");
        }

        // Виклик часткового методу через метод обгортку.
        public void CallPartialMethod()
        {
            PartialMethod();

            // При виклику нереалізованого часткового метода, компілятор його
проігнорує
            // і помилки рівня виконання не буде. Див. 26 рядок.
            MyMethod();
        }

        #region Реалізація часткових методів в інших частинах класу необов'язкова!

        //partial void MyMethod()
        //{
        //    Console.WriteLine("Реалізація часткових методів в інших частинах класу
PartialClass - необов'язкова!"); ;
        //}
        #endregion
    }
}
```

Приклад 16 Encapsulation

```
using System;

// Об'єкт не може приховувати (інкапсулювати) нічого від іншого об'єкта того ж класу.

namespace Classes
{
    class MyClass
    {
        MyClass my = null;

        private void Method()
        {
            Console.WriteLine("Hello");
        }

        public void CallMethod()
        {
            my = new MyClass();

            // private метод видимий на екземплярі!
            my.Method();
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass my = new MyClass();
            my.CallMethod();
            //my.Method(); // Неприпустимо.
```

```
            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 17 Readonly

```
using System;

// readonly - поля лише для читання.

namespace Classes
{
    class Program
    {
        // Поле лише для читання (встановлюється лише конструктором)!
        public readonly string field = "hello";

        // Конструктор.
        public Program()
        {
            field = "Поле лише для читання ";

            field += "!";
        }

        static void Main()
        {
            Program program = new Program();

            Console.WriteLine(program.field);

            // Помилка Компіляції.
            //program.field = "Спроба запису в поле лише для читання.";

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 18 SelfAssociation

```
using System;

// Самоасоціація (self-association).

namespace Classes
{
    class Program
    {
        public void Method()
        {
            Console.WriteLine("Hello world!");
        }

        static void Main()
        {
            Program program = new Program();

            program.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 19 SelfAssociation

```
using System;

// Самоасоціація (self-association).

namespace Classes
{
    class MyClass
    {
        MyClass instance;

        public MyClass()  // StackOverflowException
        {
            Console.WriteLine("Constructor MyClass");

            // Самоасоціація, після першої спроби інстанціювання призводить до
циклічного інстанціювання.

            instance = new MyClass();
        }
    }

    class Program
    {
        static void Main()
        {
            // Спроба інстанціювання класу MyClass.

            MyClass instance = new MyClass();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 20 Association

```
using System;

// Використання техніки фабричних методів.

namespace Classes
{
    class Product
    {
        public Product()
        {
            Console.WriteLine("Створено екземпляр класу Product");
        }
    }

    class Factory
    {
        public Product FactoryMethod()
        {
            return new Product();
        }
    }

    class Program
    {
        static void Main()
        {
            Factory factory = new Factory();

            Product product = factory.FactoryMethod();

            // Delay.
```

```
            Console.ReadKey();
        }
    }
}
```

Приклад 21 Association

```
using System;

namespace Classes
{
    class MyClass1
    {
        public MyClass1()
        {
            Console.WriteLine("Створено екземпляр класу MyClass1");
        }
    }

    class MyClass2
    {
        private MyClass1 myObj = null;

        // Даний метод не є фабричним.
        public void Method()
        {
            myObj = new MyClass1();
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass2 my = new MyClass2();

            my.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Вивчити основні конструкції та поняття, розглянуті на занятті.

Завдання 2

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Rectangle.
В телі класу створити два поля, які описують довжини сторін double side1, side2.
Створити конструктор користувача Rectangle(double side1, double side2), в тілі якого поля side1 та side2 ініціалізуються значеннями аргументів.
Створити два методи, які обчислюють площу прямокутника – double AreaCalculator() та периметр прямокутника – double PerymeterCalculator().

Створити дві властивості double Area і double Perymeter з одним методом доступу get.
Написати програму, яка приймає від користувача довжини двох сторін прямокутника і виводить на екран периметр та площу.

Завдання 3

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Book. Створити класи Title, Author та Content, кожен з яких повинен містити одне рядкове поле та метод void Show().
Реалізувати можливість додавання в книгу назви книги, імені автора та змісту.
Виведіть на екран різними кольорами за допомогою метода Show() назву книги, ім'я автора та зміст.

Завдання 4

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити класи Point та Figure.
Клас Point повинен містити два цілочисельні поля та одне рядкове поле.
Створити дві властивості з одним методом доступу get.
Створити конструктор користувача, в тілі якого проініціалізувати поля значеннями аргументів. Клас Figure повинен містити конструктори, які приймають від 3-х до 5-ти аргументів типу Point.
Створити два методи: double LengthSide(Point A, Point B), який розраховує довжину сторони багатокутника; void PerymeterCalculator(), який розраховує периметр багатокутника.
Написати програму, яка виводить на екран назву та периметр багатокутника.

Завдання 5

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Address.
В телі класу створити поля: index, country, city, street, house, apartment. Для кожного поля створити властивість з двома методами доступу.
Створити екземпляр класу Address.
В поля екземпляру записати інформацію про поштову адресу.
Вивести на екран значення полів, що описують адресу.

Загальні завдання (необов'язкові)

Завдання 1

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям User, який містить інформацію про користувача (логін, ім'я, прізвище, вік, дата заповнення анкети). Поле «дата заповнення анкети»

повинне бути проініціалізоване лише один раз (при створенні екземпляра даного класу) без можливості його подальшої зміни.
Реалізуйте виведення на екран інформації про користувача.

Завдання 2

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Converter.
В тілі класу створити конструктор користувача, який приймає три дійсні аргументи та ініціалізує поля, які відповідають курсу 3-х основних валют по відношенню до гривні – public Converter (double usd, double eur, double uan).
Написати програму, яка буде виконувати конвертацію з гривні в одну із вказаних валют, також програма має виконувати конвертацію з вказаних валют в гривню.

Завдання 3

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас Employee.
В тілі класу створити конструктор користувача, який приймає два рядкові аргументи та ініціалізує поля, що відповідають прізвищу та імені співробітника.
Створити метод, який розраховує зарплату співробітники (в залежності від посади та стажу) та податковий збір.
Написати програму, яка виводить на екран інформацію про співробітника (прізвище, ім'я, посада), зарплату та податковий збір.

Завдання 4

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Invoice.
В телі класу створити три поля: int account, string customer, string provider, які повинні бути проініціалізовані один раз (при створенні екземпляру даного класа) без можливості їх подальшої зміни.
В тілі класу створити два закриті поля string article, int quantity.
Створити метод розрахунку вартості замовлення з ПДВ та без ПДВ.
Написати програму, яка виводить на екран суму сплати замовленного товару з ПДВ або без ПДВ.

Індивідуальні завдання

Кожен клас, що розробляється повинен містити такі елементи: приховані поля, методи з параметрами і без параметрів. Методи повинні забезпечувати несуперечливий, повний, мінімальний і зручний інтерфейс класу. При виникненні помилок мають викидатися виключення.

Варіант 1

Описати клас, що реалізує шістнадцятковий лічильник, який може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення і зменшення. При виході за границі діапазону викидаються виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 2

Описати клас, що реалізує десятковий лічильник, який може збільшувати або зменшувати своє значення на одиницю в заданому діапазоні. Передбачити ініціалізацію лічильника значеннями за замовчуванням і довільними значеннями. Лічильник має два методи: збільшення і зменшення. При виході за границі діапазону викидаються виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 3

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, переміщення на площині, зміни розмірів і обертання на заданий кут. При неможливості побудови трикутника викидається виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 4

Побудувати опис класу, що містить інформацію про поштову адресу організації. Передбачити можливість роздільної зміни складових частин адреси та перевірки припустимості значень, що вводяться. У разі неприпустимих значень полів викидаються виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 5

Скласти опис класу для подання комплексних чисел. Забезпечити виконання операцій додавання, віднімання і множення комплексних чисел.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 6

Скласти опис класу для вектора, заданого координатами його кінців в тривимірному просторі. Забезпечити операції додавання і віднімання векторів з отриманням нового вектора (суми або різниці), обчислення скалярного добутку двох векторів, довжини вектора, косинуса кута між векторами.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 7

Скласти опис класу прямокутників зі сторонами, паралельними осям координат. Передбачити можливість переміщення прямокутників на площині, зміну розмірів, побудову найменшого прямокутника, що містить два заданих прямокутника, і прямокутника, що є спільною частиною (перетином) двох прямокутників.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 8

Скласти опис класу для представлення дати. Передбачити можливості встановлення дати та зміни її окремих полів (рік, місяць, день) з перевіркою припустимості значень, що вводяться. У разі неприпустимих значень полів викидаються виключення. Створити методи зміни дати на задану кількість днів, місяців і років.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 9

Описати клас «файл», що містить відомості про ім'я, дату створення і довжину файлу. Передбачити ініціалізацію з перевіркою припустимості значень полів. У разі неприпустимих значень полів викидаються виключення. Описати метод додавання інформації в кінець файлу і властивості для отримання стану файлу.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 10

Скласти опис класу багаточлена виду ax2 + bx + c. Передбачити методи, які реалізують:
• обчислення значення багаточлена для заданого аргументу;
• операцію додавання, віднімання і множення багаточленів з отриманням нового об'єкта-багаточлена;
• виведення на екран опису багаточлена.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 11

Описати клас, що представляє трикутник. Передбачити методи для створення об'єктів, обчислення площі, периметра і точки перетину медіан. При неможливості побудови трикутника викидається виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 12

Описати клас, що представляє коло. Передбачити методи для створення об'єктів, обчислення площі кола, довжини кола і перевірки попадання заданої точки всередину кола.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 13

Описати клас для роботи з рядком, що дозволяє зберігати тільки двійкове число і виконувати з ним арифметичні операції. Передбачити ініціалізацію з перевіркою припустимості значень. У разі неприпустимих значень викидаються виключення.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 14

Описати клас дробів – раціональних чисел, які є відношенням двох цілих чисел. Передбачити методи додавання, віднімання, множення і ділення дробів.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 15

Скласти опис класу для представлення часу. Передбачити можливості установки часу і зміни його окремих полів (година, хвилина, секунда) з перевіркою припустимості значень, що вводяться. У разі неприпустимих значень полів викидаються виключення. Створити методи зміни часу на задану кількість годин, хвилин і секунд.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 16

Описати клас «кімната», що містить відомості про метраж, висоту стель і кількість вікон. Передбачити ініціалізацію з перевіркою припустимості значень полів. У разі неприпустимих значень полів викидаються виключення. Описати методи обчислення площі та об'єму кімнати.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 17

Описати клас, що представляє нелінійне рівняння виду ax - cos (x) = 0. описати метод, що обчислює розв'язок цього рівняння на заданому інтервалі методом ділення навпіл і викидає виключення в разі відсутності кореня.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 18

Описати клас, що представляє квадратне рівняння виду ax2 + bx + c. Описати метод, що обчислює розв'язок цього рівняння і викидає виключення в разі відсутності коренів.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 19

Описати клас «процесор», що містить відомості про марку, тактову частоту, об'єм кеша і вартість. Передбачити ініціалізацію з перевіркою припустимості значень полів. У разі неприпустимих значень полів викидаються виключення.
Описати клас «материнська плата», що включає клас «процессор» і обсяг встановленої оперативної пам'яті. Передбачити ініціалізацію з перевіркою припустимості значень поля об'єму пам'яті. У разі неприпустимих значень поля викидається виключення. Написати програму, яка демонструвала б всі розроблені елементи класу.

Варіант 20

Описати клас «кольорова точка». Для точки задаються координати і колір. Колір описується за допомогою трьох складових (червоний, зелений, синій). Передбачити різні методи ініціалізації об'єкта з перевіркою припустимості значень. Допустимим діапазоном для кожної складової є [0, 255].
У разі неприпустимих значень полів викидаються виключення. Описати метод зміни кольору.
Написати програму, яка демонструвала б всі розроблені елементи класу.

Контрольні питання

1. Що таке клас, об'єкт, екземпляр класу, властивість, конструктор, partial клас, partial метод?
2. Що таке ООП? Назвіть основні парадигми ООП.
3. Які види конструкторів вам відомі?

Комп'ютерний практикум №2

НАСЛІДУВАННЯ ТА ПОЛІМОРФІЗМ

Мета роботи: ознайомлення із застосуванням модифікаторів доступу; розгляд поняття інкапсуляції, механізмів наслідування та поліморфізму [1-6, 8, 9 16-18].

Теоретичні відомості

- **ООП** – об'єктно-орієнтоване програмування – це парадигма програмування, в якій основними концепціями є поняття об'єктів і класів.
- **Наслідування** – це механізм об'єктно-орієнтованого програмування (разом з інкапсуляцією, поліморфізмом і абстракцією), що дозволяє описати новий клас на основі вже існуючого (батьківського), при цьому властивості і функціональність батьківського класу запозичуються новим класом.
- Недолік наслідування – крихкий базовий клас. Крихкий базовий клас – це фундаментальна проблема об'єктно-орієнтованого програмування. Проблема крихкого базового класу полягає в тому, що найменші виправлення в деталях реалізації базового класу можуть привнести помилку у похідні класи. У найгіршому випадку це призводить до того, що будь-яка успішна модифікація базового класу вимагає попереднього вивчення всього дерева наслідування, і часто неможлива (без створення помилки) навіть в цьому випадку.
- Рекомендується використовувати наступні пари:
  - Базовий клас – Похідний Клас;
  - Супер клас – Підклас;
  - Батьківський клас – Дочірній клас;
  - клас Батько – клас Нащадок.
- **Модифікатори доступу** – це ключові слова, що задають оголошену доступність члена або типу. За допомогою модифікаторів доступу можна задати наступні п'ять рівнів доступності:
  - public – доступ до типу або члена можливий з будь-якого іншого коду в тій самій збірці або іншій збірці, що посилається на неї;
  - protected – доступ до типу або елементу можна отримати тільки з коду в тому ж класі або в похідному класі;
  - internal – доступ до типу або члену можливий з будь-якого коду в тій самій збірці, але не з іншої збірки;
  - protected internal – доступ обмежений поточною збіркою або типами, які є похідними від класу, що містить;
  - private – доступ до типу або члена можна отримати тільки з коду в тому ж класі або структурі.

- **Поліморфізм** – це можливість об'єктів з однаковою специфікацією мати різну реалізацію.
- Поліморфізм надає підкласу спосіб визначення власної версії метода, визначеного в його базовому класі, з використанням процесу, який називається перевизначенням методу (method overiding).
- Базові класи можуть визначати і реалізовувати віртуальні методи, а похідні класи можуть перевизначати їх. Це означає що вони надають свої власні визначення та реалізацію.
- Під час виконання, коли клієнтський код викликає метод, середовище CLR шукає тип часу виконання об'єкта і викликає це перевизначення віртуального методу. Таким чином, у вихідному коді можна викликати метод у базовому класі та викликати виконання методу з версією похідного класу.
- Якщо похідний клас наслідує від базового класу, то він набуває всіх методів, полів, властивостей та подій базового класу. Проєктувальник похідного класу може вибирати з наступних можливостей:
  - перевизначити віртуальні члени в базовому класі;
  - успадкувати метод останнього базового класу без його перевизначення;
  - визначити нову не віртуальну реалізацію цих членів, яка розкриває реалізації базового класу.
- Поля не можуть бути віртуальними.
- Віртуальними можуть бути тільки методи, властивості, події і індексатори.
- Якщо у похідному класі віртуальний метод перевизначається, то цей член визивається навіть у тому випадку, якщо доступ до екземпляра цього класу здійснюється як до екземпляра базового класу.
- Віртуальні методи та властивості дають можливість похідним класам розширювати базовий клас без необхідності використання реалізації методу базового класу.
- Якщо необхідно, щоб похідний член мав те саме ім'я, що і член базового класу, але не потрібно, щоб він брав участь у віртуальному виклику, можна використовувати ключове слово new. Ключове слово new розміщується перед повертаємим типом члену класу, який заміщується.
- Оператор is – перевіряє сумісність об'єкта із заданим типом.
- Якщо наданий об'єкт може бути приведений до наданого типу, не викликаючи виключеної ситуації, вираз is приймає значення true.
- Оператор as використовується для виконання перетворень між сумісними типами-посиланнями.
- Оператор as подібний до оператора приведення. Однак, якщо перетворення неможливо, as повертає значення null, а не викликає виключення.

- В загальному вигляді логіка роботи оператора as є механізмом використання оператора is, тільки в скороченому вигляді.
- Ключове слово sealed – запобігає наслідування. Якщо клас позначений sealed (запечатаний), компілятор не дозволяє наслідувати від нього. Вважається, що клас герметизований або «запечатаний».

Практичні приклади

Приклад 1 Inheritance

Program.cs

```
using System;

// Наслідування.

// Наслідування — механізм об'єктно-орієнтованого програмування, який дозволяє описати новий (похідний) клас на основі вже існуючого (базового), при цьому властивості та функційність базового класу запозичуються новим похідним класом.

//        Базовий клас              -               Похідний клас
//        Супер клас                -               Подкласс или (сабкласс)
//        Батьківський клас         -               Дочерний клас
//        Батько                    -               Нащадок

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();

            Console.WriteLine(instance.publicField);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
// Наслідування.

namespace Inheritance
{
    class BaseClass
    {
        // Поля

        // Відкрите поле.
        public string publicField = "BaseClass.publicField";

        // Закрите поле.
        private string privateField = "BaseClass.privateField";

        // Захищене поле.
        protected string protectedField = "BaseClass.protectedField";
    }
}
```

DerivedClass.cs

```
// Наслідування.

namespace Inheritance
{
    class DerivedClass : BaseClass
```

```
    {
        // Конструктор.
        public DerivedClass()
        {
            // Змінюємо всі доступні поля, успадковані від базового класу.

            publicField = "DerivedClass.publicField";
            protectedField = "DerivedClass.protectedField";

        }
    }
}
```

Приклад 2 Inheritance

Program.cs

```
using System;

// Наслідування.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();

            Console.WriteLine(instance.publicField);

            instance.Show();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

// Наслідування.

namespace Inheritance
{
    class BaseClass
    {
        // Поля

        public string publicField = "BaseClass.publicField";
        private string privateField = "BaseClass.privateField";
        protected string protectedField = "BaseClass.protectedField";

        // Методи

        public void Show()
        {
            Console.WriteLine(privateField);
        }
    }
}
```

DerivedClass.cs

```
// Наслідування.

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
```

```
        // Конструктор.
        public DerivedClass()
        {
            publicField = "DerivedClass.publicField";
            protectedField = "DerivedClass.protectedField";
        }
    }
}
```

Приклад 3 Constructors

Program.cs

```
using System;

// Наслідування.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass(1, 2);

            Console.WriteLine(instance.baseNumber);
            Console.WriteLine(instance.derivedField);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
// Наслідування.

namespace Inheritance
{
    class BaseClass
    {
        public int baseNumber;

        // Конструктор за замовчуванням.
        public BaseClass()
        {
        }

        // Конструктор користувача.
        public BaseClass(int baseNumber)
        {
            this.baseNumber = baseNumber;
        }
    }
}
```

DerivedClass.cs

```
// Наслідування.

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        public int derivedField;

        // Конструктор за замовчуванням.
        //public DerivedClass()
        //{
        //}

        // Конструктор користувача.
```

```
        // При створенні об'єкта похідного класу, конструктор похідного класса
        // автоматично викликає конструктор за замовчуванням з базового класу.
        // Конструктор базового класу, присвоїть всім даним свої безпечні значення.
        // Після цього почне роботу конструктор похідного класу, який повторно
        // буде визначати значення для успадкованих членів. (ПОДВІЙНА РОБОТА)!
        public DerivedClass(int number1, int number2)
        {
            // Ініціалізуємо поле базового класу.
            baseNumber = number1;

            // Ініціалізуємо поле похідного (даного) класу.
            derivedField = number2;
        }
    }
}
```

Приклад 4 Constructors

Program.cs

```
using System;

// Наслідування.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass(1, 2);

            Console.WriteLine(instance.baseNumber);
            Console.WriteLine(instance.derivedField);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
// Наслідування.

namespace Inheritance
{
    class BaseClass
    {
        public int baseNumber;

        // Конструктор за замовчуванням.
        public BaseClass()
        {
        }

        // Конструктор користувача.
        public BaseClass(int number)
        {
            this.baseNumber = number;
        }
    }
}
```

DerivedClass.cs

```
// Наслідування.

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        public int derivedField;
```

```
        // Конструктор за замовчуванням.
        public DerivedClass()
        {
        }

        // Конструктор користувача.
        // Викликається конструктор користувача базового класу, при цьому не потрібно,
        // присвоювати значення, успадкованим членам в конструкторі похідного класу.
        public DerivedClass(int number1, int number2)
            : base(number1)
        {
            derivedField = number2;
        }
    }
}
```

Приклад 5 Inheritance

Program.cs

```
using System;

// Приведення до базового типу.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();

            instance.field1 = 1;
            instance.field2 = 2;
            instance.field3 = 3;

            instance.field4 = 4;
            instance.field5 = 5;

            // Приведення екземпляра класу DerivedClass до базового типу BaseClass.
            BaseClass newInstance = (BaseClass)instance; // Upcast

            Console.WriteLine(newInstance.field1);
            Console.WriteLine(newInstance.field2);
            Console.WriteLine(newInstance.field3);

            Console.WriteLine(newInstance.field4);
            Console.WriteLine(newInstance.field5);


            // Перевірка.
            Console.WriteLine("instance Id     {0}", instance.GetHashCode());
            Console.WriteLine("newInstance Id  {0}", newInstance.GetHashCode());


            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Inheritance
{
    class BaseClass
    {
        public int field1;
        public int field2;
```

```
        public int field3;
    }
}
```

DerivedClass.cs

```
using System;

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        public int field4;
        public int field5;
    }
}
```

Приклад 6 Inheritance

Program.cs

```
using System;

// Приведення до базового типу.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();

            // UpCast
            BaseClass instanceUp = instance;
            instanceUp.Method();

            // DownCast
            DerivedClass instanceDown = (DerivedClass)instanceUp;
            instanceDown.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Inheritance
{
    class BaseClass
    {
        public void Method()
        {
            Console.WriteLine("Method from BaseClass");
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        // Заміщення метода базового класу.

        public void Method()
```

```
        {
            Console.WriteLine("Method from DerivedClass");
        }
    }
}
```

Приклад 7 Virtual

Program.cs

```
using System;

// Приведення до базового типу.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();

            // UpCast
            BaseClass instanceUp = instance;
            instanceUp.Method();

            // DownCast
            DerivedClass instanceDown = (DerivedClass)instanceUp;
            instanceDown.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Inheritance
{
    class BaseClass
    {
        public virtual void Method()
        {
            Console.WriteLine("Method from BaseClass");
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        // Перевизначення метода базового класу.

        public override void Method()
        {
            Console.WriteLine("Method from DerivedClass");
        }
    }
}
```

Приклад 8 base.Method

Program.cs

```
using System;
```

```
namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Inheritance
{
    class BaseClass
    {
        public virtual void Method()
        {
            Console.WriteLine("Method from BaseClass");
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        // Перевизначення метода базового класу.

        public override void Method()
        {
            // Виклик метода базового класу.
            base.Method();

            Console.WriteLine("Method from DerivedClass");
        }
    }
}
```

Приклад 9 IsAs

```
using System;

// Оператори is та as.

namespace Inheritance
{
    class ClassA { /* ... */ }

    class ClassB : ClassA { /* ... */ }

    class Program
    {
        static void Main()
        {
            ClassB b = new ClassB();
            ClassA a = null;

            //------------------------------------------- is ----------------------
----------------------
```

```
            // Оператор is - перевіряє сумісність об'єкта з заданим типом.
            // Якщо наданий об'єкт може бути приведеним до наданого типу не викликаючи
виключення,
            // вираз is приймає значення true.

            // Наприклад, в наступному коді визначається, чи є об'єкт екземпляром типу
A або типу, похідного від A:

            if (b is ClassA)
            {
                a = (ClassA)b;
            }
            else
            {
                a = null;
            }

            //------------------------------------------- as-----------------------
---------------------
            // Оператор as використовується для виконання перетворень між сумісними
типами-посиланнями.
            // Оператор as подіюний до оператора приведення. Однак, якщо перетворення
неможливе,
            // as повертає значення null, а не викликає виключення.

            // В загальному вигляді логіка роботи оператора as є механізмом
використання оператора is
            // (приклад на 25 рядку), лише в скороченому вигляді.

            a = b as ClassA;

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 10 As

Program.cs

```
using System;

// Cast з використанням оператора as.

namespace Inheritance
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();

            // UpCast
            BaseClass instanceUp = instance as BaseClass;
            instanceUp.Method();

            // DownCast
            DerivedClass instanceDown = instanceUp as DerivedClass;
            instanceDown.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;
```

```
namespace Inheritance
{
    class BaseClass
    {
        public virtual void Method()
        {
            Console.WriteLine("Method from BaseClass");
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Inheritance
{
    class DerivedClass : BaseClass
    {
        // Перевизначення метода базового класу.

        public override void Method()
        {
            Console.WriteLine("Method from DerivedClass");
        }
    }
}
```

Приклад 11 Sealed

```
using System;

// Герметизовані класи.

namespace Inheritance
{
    sealed class SealedClass
    {
        public int x;
        public int y;
    }

    // Спроба наслідування від SealedClass призводить до помилки компілятора.

    class DerivedClass //: SealedClass   // Error
    {
    }

    class Program
    {
        static void Main()
        {
            SealedClass instance = new SealedClass();
            instance.x = 100;
            instance.y = 200;

            Console.WriteLine("x = {0}, y = {1}", instance.x, instance.y);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 SealedMethods

```
using System;

// Sealed методи.

namespace Inheritance
```

```
{
    partial class ClassA
    {
        public virtual void Method1() { Console.WriteLine("ClassA.Method1"); }
        public virtual void Method2() { Console.WriteLine("ClassA.Method2"); }
    }
    partial class ClassA
    {

    }

    class ClassB : ClassA
    {
        sealed public override void Method1() { Console.WriteLine("ClassB.Method1"); }
        public override void Method2() { Console.WriteLine("ClassB.Method2"); }
    }

    class ClassC : ClassB
    {
        // Спроба перевизначити Method1 призводить до помилки компілятора: CS0239.
        // public override void Method1() { Console.WriteLine("ClassC.Method1"); }

        // Перевизначення Method2 дозволено.
        public override void Method2() { Console.WriteLine("ClassC.Method2"); }
    }

    class Program
    {
        static void Main()
        {
            ClassA instanceA = new ClassA();
            instanceA.Method1();
            instanceA.Method2();

            Console.WriteLine(new string('-', 15));

            ClassB instanceB = new ClassB();
            instanceB.Method1();
            instanceB.Method2();

            Console.WriteLine(new string('-', 15));

            ClassC instanceC = new ClassC();
            instanceC.Method1();
            instanceC.Method2();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Необхідно:

Створити клас з ім'ям Printer.

В телі класу створити метод void Print(string value), який виводить на екран значення аргумента.

Реалізуйте можливість того, щоб в разі наслідування від даного класу інших класів та виклику відповідного метода їх екземпляра, рядки, передані в якості аргументів методів, виводились різними кольорами.
Обов'язково використайте приведення типів.

Завдання 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Необхідно:
Створити клас, що представляє учбовий клас ClassRoom.
Створити клас учень Pupil. В тілі класу створити методи void Study(), void Read(), void Write(), void Relax().
Створити три похідні класи ExcelentPupil, GoodPupil, BadPupil від базового класу Pupil та перевизначити кожен з методів, в залежності від успішності учня.
Конструктор класу ClassRoom приймає аргументи типу Pupil, клас повинен складатися з 4 учнів. Передбачити можливість того, що користувач може передати 2 або 3 аргументи.
Вивести інформацію про те, як всі учні екземпляра класу ClassRoom вміють навчатися, читати, писати, відпочівати.

Завдання 3

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас Vehicle.
В тілі класу створити поля: координати та параметри засобів пересування (ціна, швидкість, рік випуску).
Створити три похідні класи Plane, Car, Ship.
Для класу Plane має бути визначена висота та кількість пасажирів.
Для класу Ship – кількість пасажирів та порт припису.
Написати програму, яка виводить на екран інформацію про кожен засіб пересування.

Завдання 4

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям DocumentWorker.
В телі класу створити три методи OpenDocument(), EditDocument(), SaveDocument().
В тіло кожного з методів додати виведення на екран наступних рядків: «Документ відкритий», «Редагування документу доступне у версії Про», «Збереження документу доступне у версії Про».
Створити похідний клас ProDocumentWorker.

Перевизначити відповідні методи, при перевизначенні методів вивести наступні рядки: «Документ відредагований», «Документ збережений у старому форматі, збереження в інших форматах доступне у версії Експерт».
Створити похідний клас ExpertDocumentWorker від базового класу ProDocumentWorker.
Перевизначити відповідний метод. При виклику даного методу необхідно виводити на екран «Документ збережений у новому форматі».
В тілі метода Main() реалізувати можливість прийому від користувача ключа доступу **pro** та **exp**. Якщо користувач не вводить ключ, він може користуватися лише безкоштовною версією (створюється екзкмпляр базового класу), якщо користувач ввів номер ключа доступу **pro** та **exp**, то повинен створитися екземпляр відповідної версії класу, приведений до базового – DocumentWorker.

Індивідуальні завдання

1. Описати базовий клас Елемент.
Закриті поля:
- ім'я елемента (рядок символів);
- кількість входів елемента;
- кількість виходів елемента.

Методи:
- конструктор класу без параметрів;
- конструктор, що задає ім'я і встановлює рівною 1 кількість входів і виходів;
- конструктор, що задає значення всіх полів елемента.

Властивості:
- ім'я елемента (тільки читання);
- кількість входів елемента;
- кількість виходів елемента.

2. На основі класу Елемент описати похідний клас Комбінаційний, що представляє собою комбінаційний елемент (двійковий вентиль), який може мати кілька входів і один вихід.
Поле – масив значень входів.
Методи:
- конструктори;
- метод, що задає значення на входах екземпляра класу;
- метод, що дозволяє опитувати стан окремого входу екземпляр класу;
- метод, що обчислює значення виходу (за варіантом завдання).

3. На основі класу Елемент описати похідний клас Пам'ять, що представляє собою тригер. Тригер має входи, відповідні типу тригера (див. варіант завдання), і входи установки і скидання. Всі тригери вважаються синхронними, сам синхровхід до складу тригера не включається.
Поля:

- масив значень входів об'єкта класу, в масиві враховуються всі входи (керуючі та інформаційні);
- стан на прямому виході тригера;
- стан на інверсному виході тригера.

Методи:

- конструктор (за замовчуванням скидає екземпляр класу);
- конструктор копіювання;
- метод, що задає значення на входах екземпляра класу;
- методи, що дозволяють опитувати стану окремого входу екземпляр класу;
- метод, що обчислює стан екземпляра класу (за варіантом завдання) в залежності від поточного стану і значень на входах;
- метод, що перевизначає операцію == для екземплярів класу.

4. Створити клас Регістр, використовуючи клас Пам'ять як вкладений клас.

Поля:

- стан входу «Скидання» – один для екземпляра класу;
- стан входу «Установка» – один для екземпляра класу;
- масив типу Пам'ять розмірності, заданої у варіанті;
- масив (масиви), що містить значення на відповідних входах елементів масиву типу Пам'ять.

Методи:

- метод, що задає значення на входах екземпляра класу;
- метод, що дозволяє опитувати стан окремого виходу екземпляра класу;
- метод, що обчислює значення нового стану екземпляра класу.

Всі поля класів Елемент, Комбінаційний і Пам'ять повинні бути описані з ключовим словом private.

У завданні перераховані тільки обов'язкові члени і методи класу. Можна ставити додаткові члени і методи, якщо вони не скасовують обов'язкові і забезпечують додаткові зручності при роботі з даними класами, наприклад, описати функції обчислення виходу/стану як віртуальні.

5. Для перевірки функціонування створених класів написати програму, яка використовує ці класи. У програмі повинні бути продемонстровані всі властивості створених класів.

Конкретний тип комбінаційного елемента, тип тригера і розрядність регістра вибираються відповідно до варіанта завдання:

Варіант завдання

Варіант	***Комбінаційний елемент***	***Число входів***	***Тригер***	***Розрядність регістра***
1	AND-NOT	4	RS	8
2	OR	5	RST	10
3	MOD2-NOT	6	D	12
4	AND	8	T	8
5	OR-NOT	8	V	9
6	AND	4	RS	10
7	OR-NOT	5	JK	11

8	MOD2	5	D	8
9	AND	4	T	10
10	OR	3	JK	8
11	AND-NOT	3	RS	12
12	OR-NOT	4	RST	4
13	MOD2	5	D	10
14	MOD2-NOT	6	T	10
15	OR-NOT	8	V	10
16	AND	8	JK	6
17	AND-NOT	8	RS	10
18	OR	8	T	10
19	MOD2	6	JK	8
20	MOD2-NOT	5	V	10

Додаткові завдання

Варіант 1

Створити клас Point (точка). На його основі створити класи ColoredPoint і Line (лінія). На основі класу Line створити класи ColoredLine і PolyLine (багатокутник).

У класах описати наступні елементи:

- конструктори з параметрами і конструктори за замовчуванням;
- властивості для установки і отримання значень всіх координат, а також для зміни кольору і отримання поточного кольору;
- для ліній – методи зміни кута повороту ліній щодо першого точки;
- для багатокутника – метод масштабування.

Варіант 2

Описати базовий клас Рядок.

Обов'язкові поля класу:

- поле для зберігання символів рядка;
- значення типу word для зберігання довжини рядка в байтах.

Реалізувати обов'язкові методи наступного призначення:

- конструктор без параметрів;
- конструктор, що приймає як параметр рядковий літерал;
- конструктор, що приймає як параметр символ;
- метод отримання довжини рядка;
- метод очищення рядки (зробити рядок порожнім).

Описати похідний від Рядок клас Комплексне_число.

Рядки даного класу складаються з двох полів, розділених символом *i*. Перше поле задає значення дійсної частини числа, друге – значення уявної. Кожне з полів може містити лише символи десяткових цифр і символи – і +, що задають знак числа. Символи - або + можуть знаходитись тільки в першій позиції числа, причому символ + може бути відсутнім, в цьому випадку число вважається додатним. Якщо в складі ініціалізуючого рядка будуть зустрінуті будь-які символи, відмінні від допустимих, клас Комплексне_число приймає нульове значення. Приклади рядків:

```
33 i 12
-7 i 00
+5 i -21
```

Для класу Комплексне_число визначити наступні методи:

- перевірка на рівність;
- додавання чисел;
- множення чисел.

Варіант 3

Описати базовий клас Рядок відповідно до Варіанта 2.
Описати похідний від Рядок клас Десятковий_рядок.
Рядки даного класу можуть містити тільки символи десяткових цифр і символи - і +, що задають знак числа. Символи - або + можуть знаходитись тільки в першій позиції числа, причому символ + може бути відсутнім, в цьому випадку число вважається додатним. Якщо в складі ініціалізуючого рядка будуть зустрінуті будь-які символи, відмінні від допустимих, клас Десятковий_рядок приймає нульове значення. Вміст цих рядків розглядається як десяткове число.
Для класу визначити наступні методи:

- конструктор, що приймає як параметр число;
- арифметична різниця рядків;
- перевірка на більше (за значенням);
- перевірка на менше (за значенням).

Варіант 4

Описати базовий клас Рядок відповідно до Варіанта 2.
Описати похідний від Рядок клас Бітовий_рядок.
Рядки даного класу можуть містити тільки символи «0» або «1». Якщо в складі ініціалізуючого рядка будуть зустрінуті будь-які символи, відмінні від допустимих, клас Бітовий_рядок приймає нульове значення. Вміст цих рядків розглядається як двійкове число. Від'ємні числа зберігаються в додатковому коді.
Для класу Бітовий_рядок визначити наступні методи:

- конструктор, що приймає як параметр рядковий літерал;
- деструктор;
- зміна знака на протилежний (переведення числа в додатковий код);
- присвоювання;
- обчислення арифметичної суми рядків;
- перевірка на рівність.

У разі необхідності більш короткий бітовий рядок розширюється вліво знаковим розрядом.

Контрольні питання

1. Що таке наслідування? Які недоліки наслідування вам відомі?
2. Що таке модифікатори доступу та де їх використовують?
3. Що таке поліморфізм?
4. Навіщо використовується ключове слово virtual?
5. Що таке Cast, Upcast, Downcast?
6. Поясніть призначення ключового слова sealed?

Комп'ютерний практикум №3

АБСТРАКТНІ КЛАСИ ТА ІНТЕРФЕЙСИ

Мета роботи: розгляд абстрактних класів та інтерфейсів [1-6, 8, 9, 19-21].

Теоретичні відомості

- **Абстракція** – в об'єктно-орієнтованому програмуванні це надання об'єкту характеристик, які відрізняють його від інших об'єктів, чітко визначаючи його концептуальні границі.
- Основна ідея абстракції полягає в тому, щоб відокремити спосіб використання складових об'єктів даних від деталей їх реалізації, у вигляді більш простих об'єктів, подібно до того, як функціональна абстракція поділяє спосіб використання функції та деталей її реалізації, у термінах більш примітивних функцій, таким чином, дані обробляються функцією високого рівня за допомогою функцій низького рівня.
- **Абстрактний клас** – в об'єктно-орієнтованому програмуванні це базовий клас, який не передбачає створення екземплярів. Абстрактні класи реалізують на практиці один з принципів ООП – поліморфізм. Абстрактний клас може містити (і не містити) абстрактні методи та властивості. Абстрактний метод не реалізується для класу, в якому описаний, проте має бути реалізований для його неабстрактних нащадків. Абстрактні класи є найбільш загальними абстракціями, тобто мають найбільший обсяг і найменший зміст.
- Ключове слово abstract може використовуватися з класами, методами, властивостями, індексаторами та подіями.
- Ключове слово abstract при створенні класу вказує, що клас призначений лише для використання як базовий клас для інших класів. Члени, позначені як абстрактні або включені до абстрактного класу, повинні бути реалізовані за допомогою класів, похідних від абстрактних класів.
- Можливості абстрактних класів:
  - екземпляр абстрактного класу створити не можна;
  - абстрактні класи можуть містити як абстрактні, так і прості (неабстрактні) члени;
  - Неабстрактний клас, що є похідним від абстрактного, повинен містити фактичні реалізації всіх наслідуваних абстрактних методів та методів доступу.
- Можливості абстрактних методів:
  - абстрактний метод є неявним віртуальним методом;
  - створення абстрактних методів допускається лише в абстрактних класах;

  - тіло абстрактного методу відсутнє; створення методу просто закінчується двокрапкою, а після сигнатури ставити фігурні дужки ({}) не потрібно;
  - реалізація надається методом перевизначення override, що є членом неабстрактного класу.
- Абстрактний клас повинен надавати реалізацію для всіх членів інтерфейсу.
- Абстрактний клас, що реалізує інтерфейс, може відображати методи інтерфейсу в абстрактних методах.
- Переваги використання абстрактних класів:
  - загальний код в одній реалізації у вигляді конкретних та абстрактних членів;
  - зміна значення полів або неабстрактних членів абстрактного класу призводить до відповідної зміни у всіх його похідних класах;
  - наявність реалізації за замовчуванням.
- **Інтерфейс** (від латів. inter – «між», і face – «поверхня») – семантична та синтаксична конструкція в коді програми, що використовується для специфікації послуг, які надаються класом або компонентом. Інтерфейс визначає границю взаємодії між класами або компонентами, специфікуючи певну абстракцію, яку здійснює реалізуюча сторона. На відміну від багатьох інших видів інтерфейсів, інтерфейс ООП є строго формалізованим елементом об'єктно-орієнтованої мови і, як семантична конструкція, широко використовується кодом програми.
- Неможливо створити екземпляр інтерфейсу.
- Інтерфейси та члени інтерфейсів є абстрактними. Інтерфейси не мають реалізації за замовчуванням.
- Інтерфейс містить лише сигнатури методів, властивостей, подій чи індексаторів.
- Члени інтерфейсів автоматично є відкритими, абстрактними, і вони не можуть мати модифікатори доступу.
- Інтерфейси не можуть містити константи, поля, оператори, конструктори екземплярів, деструктори або типи.
- Клас або структура, які реалізують інтерфейс, повинні реалізувати члени цього інтерфейсу, вказані під час його створення.
- Однак якщо базовий клас реалізує інтерфейс, похідний клас успадковує цю реалізацію.
- Інтерфейс може успадковуватись від одного або декількох базових інтерфейсів.
- Базовий клас також може реалізувати члени інтерфейсу за допомогою віртуальних членів. І тут похідний клас може змінити поведінку інтерфейсу шляхом перевизначення віртуальних членів.

- Якщо клас реалізує два інтерфейси, що містять член з однаковою сигнатурою, то при реалізації цього члена в класі обидва інтерфейси будуть використовувати цей член для своєї реалізації.
- Якщо члени двох інтерфейсів з однаковою сигнатурою методів повинні виконувати різні дії при їх реалізації, необхідно скористатися явною реалізацією члена інтерфейсу – технікою явного вказування імені члена імені інтерфейсу, якому належить даний член. Це досягається шляхом включення до імені члена класу імені інтерфейсу з точкою. Цей член у похідному класі буде позначений за замовчуванням як прихований.
- Перевага використання інтерфейсів:
  - клас чи структура може реалізувати кілька інтерфейсів;
  - якщо клас або структура реалізує інтерфейс, вона отримує лише імена та сигнатури методу;
  - інтерфейси визначають поведінку екземплярів похідних класів;
  - базовий клас може мати непотрібний функціонал, отриманий від інших його базових класів, чого можна уникнути, застосовуючи інтерфейси.
- **Впровадження залежностей** (Dependency Injection, DI) – чудова методика для створення слабко зв'язаних додатків. Вона надає можливості для спрощення коду, вилучення та обробки залежностей між об'єктами та автоматичного створення екземплярів залежного об'єкта.
- Впровадження залежностей описує процес розробки додатків – замість зазначення конкретних залежностей у додатку під час розробки та створення необхідних об'єктів у коді під час виконання додаток вирішує, які об'єкти йому потрібні, а потім створює та впроваджує їх у додаток.
- Використання впровадження залежностей надає кілька переваг:
  - послаблення зв'язку між класами;
  - створення коду, який краще піддається перевірці;
  - спрощення тестування.

Практичні приклади

Приклад 1 Abstraction

```
using System;

// Абстрактний клас.

namespace Abstraction
{
    // Абстрактний клас.
    abstract class AbstractClass
    {
        public abstract void Method();
    }

    // Конкретний клас.
    class ConcreteClass : AbstractClass
    {
        public override void Method()
        {
```

```
            Console.WriteLine("Implementation");
        }
    }

    class Program
    {
        static void Main()
        {
            AbstractClass instance = new ConcreteClass();

            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Abstraction

```
using System;

// Абстрактний клас може бути успадкованим від конкретного класу.

namespace Abstraction
{
    // Конкретний клас A.
    class ConcreteClassA
    {
        public void Operation()
        {
            Console.WriteLine("ConcreteClassA.Operation");
        }
    }

    // Абстрактний клас.
    abstract class AbstractClass : ConcreteClassA
    {
        public abstract void Method();
    }

    // Конкретний клас B.
    class ConcreteClassB : AbstractClass
    {
        public override void Method()
        {
            Console.WriteLine("ConcreteClassB.Method");
        }
    }

    class Program
    {
        static void Main()
        {
            AbstractClass instance = new ConcreteClassB();

            instance.Method();
            instance.Operation();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Abstraction

```
using System;

// Абстрактний клас може бути успадкованим від абстрактного класу.
```

```
// Реалізація абстрактного метода з базового абстрактного класу, в похідному
абстрактному класі - не обов'язкова.

namespace Abstraction
{
    // Абстрактний клас A.
    abstract class AbstractClassA
    {
        public abstract void OperationA();
    }

    // Абстрактний клас B.
    abstract class AbstractClassB : AbstractClassA
    {
        public abstract void OperationB();
    }

    // Конкретний клас.
    class ConcreteClass : AbstractClassB
    {
        public override void OperationA()
        {
            Console.WriteLine("ConcreteClass.OperationA");
        }

        public override void OperationB()
        {
            Console.WriteLine("ConcreteClass.OperationB");
        }
    }

    class Program
    {
        static void Main()
        {
            AbstractClassA instance = new ConcreteClass();

            instance.OperationA();

            //instance.OperationB();  // Питання: чому недоступний даний метод?

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Abstraction

Program.cs

```
using System;

namespace Abstraction
{
    class Program
    {
        static void Main()
        {
            ConcreteDerivedClass instance = new ConcreteDerivedClass();

            instance.SimpleMethod();
            instance.VirtualMethod();
            instance.AbstractMethod();

            //Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Abstraction
{
    abstract class AbstractBaseClass
    {
        // 1.
        // Звичайний метод передається похідному класу як при наслідуванні від
конкретного класу.
        public void SimpleMethod()
        {
            Console.WriteLine("AbstractBaseClass.SimpleMethod");
        }

        // 2.
        // Віртуальний метод передається похідному класу як при наслідуванні від
конкретного класу.
        virtual public void VirtualMethod()
        {
            Console.WriteLine("AbstractBaseClass.VirtualMethod");
        }

        // 3.
        // Абстрактний метод - реалізується в похідному класі.
        abstract public void AbstractMethod();
    }
}
```

DerivedClass.cs

```
using System;

namespace Abstraction
{
    class ConcreteDerivedClass : AbstractBaseClass
    {
        // Перевизначаємо віртуальний метод VirtualMethod() базового абстрактного
класу.
        // Якщо ми не перевизначимо віртуальний метод, то буде використаний метод з
базового класу.

        public override void VirtualMethod()
        {
            Console.WriteLine("DerivedClass.VirtualMethod();");
        }

        // Реалізуємо абстрактний метод AbstractMethod() базового абстрактного класу.

        public override void AbstractMethod()
        {
            Console.WriteLine("DerivedClass.AbstractMethod();");
        }
    }
}
```

Приклад 5 Abstraction

Program.cs

```
using System;

namespace Abstraction
{
    class Program
    {
        static void Main()
        {
            ConcreteDerivedClass instance = new ConcreteDerivedClass();

            instance.SimpleMethod();
            instance.VirtualMethod();
```

```
            instance.AbstractMethod();

            //Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Abstraction
{
    abstract class AbstractBaseClass
    {
        // 1.
        // Звичайний метод передається похідному класу як при наслідуванні від
конкретного класу.
        public void SimpleMethod()
        {
            Console.WriteLine("AbstractBaseClass.SimpleMethod");
        }

        // 2.
        // Віртуальний метод передається похідному класу як при наслідуванні від
конкретного класу.
        virtual public void VirtualMethod()
        {
            Console.WriteLine("AbstractBaseClass.VirtualMethod");
        }

        // 3.
        // Абстрактний метод - реалізується в похідному класі.
        abstract public void AbstractMethod();
    }
}
```

DerivedClass.cs

```
using System;

namespace Abstraction
{
    class ConcreteDerivedClass : AbstractBaseClass
    {
        // Перевизначаємо віртуальний метод VirtualMethod() базового абстрактного
класу.
        // Якщо ми не перевизначимо віртуальний метод, то буде використаний метод з
базового класу.

        //public override void VirtualMethod()
        //{
        //    Console.WriteLine("DerivedClass.VirtualMethod();");
        //}

        // Реалізуємо абстрактний метод AbstractMethod() базового абстрактного класу.

        public override void AbstractMethod()
        {
            Console.WriteLine("DerivedClass.AbstractMethod();");
        }
    }
}
```

Приклад 6 Abstraction

```
using System;

namespace Abstraction
{
    abstract class AbstractClass
```

```
    {
        // Конструктор (відпрацьовує першим).
        public AbstractClass()
        {
            Console.WriteLine("1 AbstractClass()");

            // Викликається реалізація метода з похідного класу - ConcreteClass.
            this.AbstractMethod();

            Console.WriteLine("2 AbstractClass()");
        }

        public abstract void AbstractMethod();
    }

    class ConcreteClass : AbstractClass
    {
        string s = "FIRST";

        // Конструктор (відпрацьовує другим).
        public ConcreteClass()
        {
            Console.WriteLine("3 ConcreteClass()");
            s = "SECOND";
        }

        public override void AbstractMethod()
        {
            Console.WriteLine("Реалізація метода AbstractMethod() в ConcreteClass
{0}", s);
        }
    }

    class Program
    {
        static void Main()
        {
            AbstractClass instance = new ConcreteClass();

            Console.WriteLine(new string('-', 55));

            instance.AbstractMethod();

            // Затримка.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 Interfaces

```
using System;

// Інтерфейси.

namespace Interfaces
{
    interface IInterface
    {
        void Method();
    }

    class MyClass : IInterface
    {
        public void Method()
        {
            Console.WriteLine("Метод - реалізація Інтерфейса.");
        }
    }
```

```
    class Program
    {
        static void Main()
        {
            MyClass my = new MyClass();

            my.Method();

            //Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 Interfaces

Program.cs

```
using System;

// Множинне наслідування.

namespace Interfaces
{
    class Program
    {
        static void Main()
        {
            Interface1 instance1 = new DerivedClass();
            Interface2 instance2 = new DerivedClass();

            instance1.Method1();
            // instance1.Method2();

            instance2.Method2();
            // instance2.Method1();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Interfaces
{
    class DerivedClass : Interface1, Interface2
    {
        public void Method1()
        {
            Console.WriteLine("Реалізація метода Method1() з Interface1");
        }

        public void Method2()
        {
            Console.WriteLine("Реалізація метода Method2() з Interface2");
        }
    }
}
```

Interface1.cs

```
namespace Interfaces
{
    interface Interface1
    {
        void Method1();
    }
}
```

Interface1.cs

```
namespace Interfaces
{
    interface Interface2
    {
        void Method2();
    }
}
```

Приклад 9 Interfaces

Program.cs

```
using System;

// Явне вказування імені інтерфейсу в імені метода.

namespace Interfaces
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();

            //instance. -- // На екземплярі не бачимо методів інтерфейсів.

            // Приведемо екземпляр класу DerivedClass - instance, до базового
інтерфейсного типу Interface1

            Interface1 instance1 = instance as Interface1;
            instance1.Method();

            Interface2 instance2 = instance as Interface2;
            instance2.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

DerivedClass.cs

```
using System;

namespace Interfaces
{
    class DerivedClass : Interface1, Interface2
    {
        // На 10-ому рядку реалізуємо метод з ім'ям Method з базового інтерфейсу
Interface1
        // При реалізації метода використовуємо техніку явного вказування імені
інтерфейсу в імені метода, якому належить даний метод.

        // За замовчуванням одноіменні методи є private,
        // але явно вказувати модифікатори доступу неприпустимо.
        void Interface1.Method()
        {
            Console.WriteLine("Реалізація метода Method() з Interface1");
        }

        void Interface2.Method()
        {
            Console.WriteLine("Реалізація метода Method() з Interface2");
        }
    }
}
```

Interface1.cs

```
namespace Interfaces
{
```

```
    interface Interface1
    {
        void Method();
    }
}
```

Interface1.cs

```
namespace Interfaces
{
    interface Interface2
    {
        void Method();
    }
}
```

Приклад 10 Interfaces

Program.cs

```
using System;

// Множинне наслідування від одного класу та декількох інтерфейсів.

namespace Interfaces
{
    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();
            instance.Method1();
            instance.Method2();

            Console.WriteLine(new string('-', 40));

            BaseClass instance0 = instance as BaseClass;
            instance0.Method();

            Interface1 instance1 = instance as Interface1;
            instance1.Method1();

            Interface2 instance2 = instance as Interface2;
            instance2.Method2();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

BaseClass.cs

```
using System;

namespace Interfaces
{
    class BaseClass
    {
        public void Method()
        {
            Console.WriteLine("Метод класу - BaseClass.");
        }
    }
}
```

DerivedClass.cs

```
using System;

// В C# припустиме множинне наслідування лише від інтерфейсів.
// Множинне наслідування реалізації (т.т. від двох та більше класів або структур)
неприпустиме.
```

```
// Припустиме множинне наслідування від одного класу та багатьох інтерфейсів.

namespace Interfaces
{
    class DerivedClass : BaseClass, Interface1, Interface2
    {
        public void Method1()
        {
            Console.WriteLine("Реалізація метода Method1() з Interface1");
        }

        public void Method2()
        {
            Console.WriteLine("Реалізація метода Method2() з Interface2");
        }
    }
}
```

Interface1.cs

```
namespace Interfaces
{
    interface Interface1
    {
        void Method1();
    }
}
```

Interface1.cs

```
namespace Interfaces
{
    interface Interface2
    {
        void Method2();
    }
}
```

Приклад 11 Interfaces

```
using System;

// Наслідування інтерфейса від інтерфейса.

namespace Interfaces
{
    interface IInterface1
    {
        void Method1();
    }

    interface IInterface2 : IInterface1
    {
        void Method2();
    }

    class ConcreteClass : IInterface2
    {
        public void Method1()
        {
            Console.WriteLine("Method1 - реалізація інтерфейса IInterface1");
        }

        public void Method2()
        {
            Console.WriteLine("Method2 - реалізація інтерфейса IInterface2");
        }
    }

    class Program
    {
        static void Main()
```

```
        {
            ConcreteClass instance = new ConcreteClass();
            instance.Method1();
            instance.Method2();

            IInterface1 instance1 = instance as IInterface1;
            instance1.Method1();

            IInterface2 instance2 = instance as IInterface2;
            instance2.Method1();
            instance2.Method2();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 Interfaces

```
using System;

// Наслідування інтерфейса від інтерфейса, у яких співпадають імена членів.

namespace Interfaces
{
    interface IInterface1
    {
        void Method();
    }

    interface IInterface2 : IInterface1
    {
        // Без new - помилки не буде, але буде попередження компілятора.
        new void Method();
    }

    class ConcreteClass : IInterface2
    {
        void IInterface1.Method()
        {
            Console.WriteLine("Method - реалізація інтерфейса IInterface1");
        }

        void IInterface2.Method()
        {
            Console.WriteLine("Method - реалізація інтерфейса IInterface2");
        }
    }

    class Program
    {
        static void Main()
        {
            ConcreteClass instance = new ConcreteClass();
            //instance.Method();

            IInterface1 instance1 = instance as IInterface1;
            instance1.Method();

            IInterface2 instance2 = instance as IInterface2;
            instance2.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 13 Interfaces

```
using System;

// Наслідування від інтерфейсів, у яких співпадають імена членів.

// Об'єднання реалізації одноіменних абстрактних членів.

namespace Interfaces
{
    interface IInterface1
    {
        void Method();
    }

    interface IInterface2
    {
        void Method();
    }

    class ConcreteClass : IInterface1, IInterface2
    {
        public void Method()
        {
            Console.WriteLine("Method - реалізація інтерфейса IInterface (1-2)");
        }
    }

    class Program
    {
        static void Main()
        {
            ConcreteClass instance = new ConcreteClass();
            instance.Method();

            IInterface1 instance1 = instance as IInterface1;
            instance1.Method();

            IInterface2 instance2 = instance as IInterface2;
            instance2.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 14 Interfaces

```
using System;

// Наслідування абстрактних класів від інтерфейсів.

namespace Interfaces
{
    interface Interface
    {
        void Method();
    }

    abstract class AbstractClass : Interface
    {
        // Реалізація абстрактного метода з інтерфейса в абстрактному класі
обв'язкова.
        public void Method()
        {
            Console.WriteLine("Метод - реалізація інтерфейса в абстрактному класі.");
        }
    }
```

```
    class ConcreteClass : AbstractClass
    {
    }

    class Program
    {
        static void Main()
        {
            ConcreteClass instance = new ConcreteClass();
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 15 Interfaces

```
using System;

// Наслідування абстрактних класів від інтерфейсів.

namespace Interfaces
{
    interface Interface
    {
        void Method();
    }

    abstract class AbstractClass : Interface
    {
        // Заміщення абстрактного метода з інтерфейса в абстрактному класі
обов'язкове.
        public abstract void Method();
    }

    class ConcreteClass : AbstractClass
    {
        // Реалізація абстрактного метода з абстрактного класу в конкретному класі
обов'язкова.
        public void Method()
        {
            Console.WriteLine("Метод - реалізація інтерфейса в абстрактному класі.");
        }
    }

    class Program
    {
        static void Main()
        {
            ConcreteClass instance = new ConcreteClass();
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 16 Interfaces

```
using System;

// Наслідування від класу та інтерфейса, у яких співпадають сигнатури членів.

namespace Interfaces
{
    interface IInterface
    {
```

```
        void Method();
    }

    class BaseClass
    {
        public void Method()
        {
            Console.WriteLine("BaseClass.Method()");
        }
    }

    class DerivedClass : BaseClass, IInterface
    {
        // Реалізація інтерфейса не обв'язкова, оскільки
        // сигнатури методів в класі та інтерфейсі співпадають.
    }

    class Program
    {
        static void Main()
        {
            DerivedClass instance = new DerivedClass();
            instance.Method();

            IInterface instance1 = instance as IInterface;
            instance1.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи VisualStudio, створіть проєкт за шаблоном ConsoleApplication.
Необхідно:
Створити клас AbstractHandler.
В тілі класу створіть методи void Open(), void Create(), void Chenge(), void Save().
Створити похідні класи XMLHandler, TXTHandler, DOCHandler від базового класу AbstractHandler.
Написати програму, яка буде виконувати визначення документа і для кожного формату повинні бути методи відкриття, створення, редагування, збереження певного формату документу.

Завдання 2

Використовуючи VisualStudio, створіть проєкт за шаблоном ConsoleApplication.
Необхідно:
Створити 2 інтерфейси IPlayable та IRecodable. В кожному інтерфейсі створіть по 3 методи void Play() / void Stop() / void Record() / void Pause() / void Stop() відповідно. Створіть похідний клас Player від базових інтерфейсів IPlayable та IRecodable.
Написати програму, яка буде виконувати відтворення треку та запис.

Завдання 3

Використовуючи VisualStudio, створити проєкт за шаблоном ConsoleApplication.
Необхідно:
Змінити Приклад 12 з КП 1 (робота з докуементом) і створити загальний абстрактний клас для всіх частин документа.

Індивідуальні завдання

Варіант 1

Створити абстрактний клас Figure з віртуальними методами обчислення площі та периметра. Створити похідні класи Rectangle (прямокутник), Circle (коло), Trapezium (трапеція) з власними методами обчислення площі та периметра. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Площа трапеції обчислюється за формулою: $S = (a+b) \times h/2$.

Варіант 2

Створити абстрактний клас Number з віртуальними методами – арифметичними операціями. Створити похідні класи Integer (ціле) та Double (дійсне). Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Варіант 3

Створити абстрактний клас Body (тіло) з віртуальними методами обчислення площі поверхні та об'єму. Створити похідні класи Parallelepiped (перелелепіпед) та Ball (куля) з власними методами обчислення площі поверхні та об'єма. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Варіант 4

Створити абстрактний клас Currency (валюта) для роботи з грошовими сумами. Визначити віртуальні методи переводу в гривні і виводу на екран. Реалізувати похідні класи Dolar (прямокутник), Euro (коло), Trapezium (трапеція) з власними методами переводу і виведення на екран. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних.

Варіант 5

Створити абстрактний клас Triangle для надання трикутника з віртуальними методами обчислення площі та периметра. Поля даних повинні включати дві сторони та кут між ними. Визначити класи-нащадки: прямокутний трикутник, рівнобедрений трикутник, рівнобічний трикутник з власними методами обчислення площі та периметра.

Варіант 6

Створити абстрактний клас Root (корінь) з віртуальними методами обчислення коренів і виведення результату на екран. Створити похідні класи

Linear (лінійне рівняння) та Square (квадратне рівняння) з власними методами обчислення коренів і виведення на екран.

Варіант 7

Створити абстрактний клас Function з віртуальними методами обчислення значення функції $y = f(x)$ в точці x і виводу результату на екран. Створити похідні класи Rectangle (еліпс) та Circle (гіпербола) з власними методами обчислення y в залежності від вхідного параметра x. Рівняння еліпса: $x^2/a^2 + y^2/b^2 = 1$, гіперболи: $x^2/a^2 - y^2/b^2 = 1$.

Варіант 8

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи FuzzyNumber та Complex. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас FuzzyNumber призначений для роботи з нечіткими числами, які надаються трійками чисел $(x-e_1, x, x+e_2)$. Для чисел $A = (A-a_l, A, A+a_r)$ та $B = (B-b_l, B, B+b_r)$ арифметичні операції виконуються за наступними формулами: додавання $A+B = (A+B-a_l-b_l, A+B, A+B+a_r+b_r)$, віднімання $A-B = (A-B-a_l-b_l, A-B, A-B+a_r+b_r)$, множення $A \cdot B = (A\times B - B\times a_l - A\times b_l + a_l\times b_l, A\times B, A\times B + B\times a_r + A\times b_r + a_r\times b_r)$, ділення $A/B = ((A-a_l)/(B+b_r), A/B, (A+a_r)/(B-b_l)), B>0$. Комплексне число надається парою дійсних чисел (a,b), де a – дійсна частина, b – уявна частина. В класі Complex мають бути присутні методи: додавання add() $(a,b)+(c,d) = (a+b, c+d)$; віднімання sub() $(a,b)-(c,d) = (a-b, c-d)$; множення mul() $(a,b)\times(c,d) = (ac-bd, ad-bc)$; ділення div() $(a,b)/(c,d) = (ac+bd, bc-ad)/(c^2-d^2)$; порівняння equ() $(a,b) = (c,d)$, *якщо* $(a=c)$ *та* $(b=d)$.

Варіант 9

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи Complex та Rational. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Комплексне число надається парою дійсних чисел (a,b), де a – дійсна частина, b – уявна частина. В класі Complex мають бути присутні методи: додавання add() $(a,b)+(c,d) = (a+b, c+d)$; віднімання sub() $(a,b)-(c,d) = (a-b, c-d)$; множення mul() $(a,b)\times(c,d) = (ac-bd, ad-bc)$; ділення div() $(a,b)/(c,d) = (ac+bd, bc-ad)/(c^2-d^2)$; порівняння equ() $(a,b) = (c,d)$, *якщо* $(a=c)$ *та* $(b=d)$. Раціональний дріб надається парою

цілих чисел (a,b), де a – чисельник, b – знаменник. В класі Rational мають бути присутні методи: додавання add() $(a,b)+(c,d)=(ad+bc,bd)$; віднімання sub() $(a,b)-(c,d)=(ad-bc,bd)$; множення mul() $(a,b)\times(c,d)=(ac,bd)$; ділення div() $(a,b)/(c,d)=(ad,bc)$; порівняння equal, greate, less. Також в класі Rational повинний бути реалізований метод скорочення дробу reduce, який викликається при виконанні арифметичних операцій.

Варіант 10

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи Money та Fraction. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас Money призначений для роботи з грошовими сумами. Число має бути представленим двома полями: для гривень та копійок. В класі реалізуються операції додавання, віднімання, множення та ділення на дробове число. Клас Fraction призначений для роботи з дробовими числами. Число має бути представленим двома полями: ціла частина та дробова частина. В класі реалізуються операції додавання, віднімання, множення.

Варіант 11

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи Money1 та Money2. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас Money1 призначений для роботи з грошовими сумами. Число має бути представленим двома полями: для гривень та копійок. В класі реалізуються операції додавання, віднімання, множення та ділення на дробове число. Номінали української гривні можуть приймати значення 1, 2, 5, 10, 20, 50, 100, 200, 500, 1000. Клас Money2 призначений для роботи з грошовими сумами. В ньому сума має бути представлена полями-номіналами, значеннями яких повинна бути кількість купюр даного номіналу. В класі реалізується додавання сум, віднімання сум, множення та ділення суми юна дробове число.

Варіант 12

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи Money та Complex. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас Money призначений для роботи з грошовими сумами. Число має бути представленим двома полями: для гривень та копійок. В класі реалізуються операції додавання, віднімання, множення та ділення на дробове число. Комплексне число надається парою дійсних чисел (a,b), де a – дійсна частина, b – уявна частина. В класі Complex мають бути присутні методи: додавання add() $(a,b)+(c,d)=(a+b,c+d)$; віднімання sub() $(a,b)-(c,d)=(a-b,c-d)$; множення mul() $(a,b)\times(c,d)=(ac-bd,ad-bc)$;

ділення div() $(a,b)/(c,d)=(ac+bd,bc-ad)/(c^2-d^2)$; порівняння equ() $(a,b)=(c,d)$, *якщо* $(a=c)ma(b=d)$.

Варіант 13

Створити абстрактний клас Pair з віртуальними методами обчислення арифметичних операцій. Створити похідні класи FuzzyNumber та Fraction. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас FuzzyNumber призначений для роботи з нечіткими числами, які надаються трійками чисел $(x-e_1,x,x+e_2)$. Для чисел $A=(A-a_l,A,A+a_r)$ та $B=(B-b_l,B,B+b_r)$ арифметичні операції виконуються за наступними формулами: додавання $A+B=(A+B-a_l-b_l,A+B,A+B+a_r+b_r)$, віднімання $A-B=(A-B-a_l-b_l,A-B,A-B+a_r+b_r)$, множення $A\cdot B=(A\times B-B\times a_l-A\times b_l+a_l\times b_l,A\times B,A\times B+B\times a_r+A\times b_r+a_r\times b_r)$, ділення $A/B=((A-a_l)/(B+b_r),A/B,(A+a_r)/(B-b_l)),B>0$. Клас Fraction призначений для роботи з дробовими числами. Число має бути представленим двома полями: ціла частина та дробова частина. В класі реалізуються операції додавання, віднімання, множення.

Варіант 14

Створити абстрактний клас Triad з віртуальними методами збільшення на 1. Створити похідні класи Date та Time. Визначити, які поля необхідні, які з них можна задати в базовому класі, а які – в похідних. Клас Date призначений для роботи з датами у форматі «рік.місяць.день». Дата представлена структурою з трьома полями: для року, для місяця та для дня. В класі реалізуються операції обчислення дати через задану кількість днів, віднімання заданої кількості днів від дати, обчислення кількості днів між датами. Клас Time призначений для роботи з часом у форматі «година.хвилина.секунда». В класі реалізуються операції обчислення різниці між двома моментами часу в секундах, переведення в секунди, переведення в хвилини.

Варіант 15

Створити абстрактний клас Integer (ціле) з віртуальними арифметичними операціями та методом виведення на екран. Створити похідні класи Decimal (десяткове) та Binary (двійкове), які реалізують власні арифметичні операції і функцію виведення на екран. Число представлене масивом, кожен елемент якого – цифра.

Варіант 16

Створити абстрактний клас Series (прогресія) з віртуальними методами обчислення *j*-го елемента прогресії і суми прогресії. Створити похідні класи Linear (арифметична) та Exponential (геометрична). Арифметична прогресія: $a_j=a_0+jd$, $j=0,1,2,...$; сума арифметичної прогресії $s_n=(n+1)(a_0+a_n)/2$;

геометрична прогресія: $a_j = a_0 r^j$, $j = 0,1,2,...$; сума геометричної прогресії: $s_n = (a_0 - a_n r)/(1-r)$.

Варіант 17

Створити абстрактний клас Norm з віртуальним методом обчислення норми та модуля. Створити похідні класи Complex, та Vector3D з власними методами обчислення норми та модуля. Модуль для комплексного числа обчислюється як корінь з суми квадратів дійсної та уявної частин; норма для комплексних чисел обчислюється як модуль в квадраті. Модуль вектора обчислюється як корінь квадратний з суми квадратів координат; норма вектора обчислюється як максимальне з абсолютних значень координат.

Варіант 18

Створити абстрактний клас Container з віртуальними методами sort() та поелементної обробки контейнера foreach(). Створити похідні класи Bubble (бульбашка) та Choice (вибір). В першому класі сортування реалізується методом бульбашки, а поелементна обробка полягає у взятті квадратного кореня. В другому класі сортування реалізується методом вибору, а поелементна обробка – обчислення логарифму.

Варіант 19

Створити абстрактний клас Array з віртуальними методами додавання та поелементної обробки масиву foreach(). Створити похідні класи SortArray та XorArray. В першому класі операція додавання реалізується як об'єднання множин, а поелементна обробка полягає у сортуванні. В другому класі операція додавання реалізується як виключаюче АБО, а поелементна обробка – обчислення кореня.

Варіант 20

Створити абстрактний клас Array додавання та поелементної обробки масиву foreach(). Створити похідні класи AndArray та OrArray. В першому класі операція додавання реалізується як перетин множин, а поелементна обробка полягає у взятті кореня квадратного. В другому класі операція додавання реалізується як об'єднання, а поелементна обробка – обчислення логарифму.

Контрольні питання

1. Що таке абстрактний клас?
2. Що таке інтерфейс?
3. Чим абстрактний клас відрізняється від інтерфейсу?
4. Що таке множинне наслідування?
5. Чим абстрактний клас відрізняється від конкретного?
6. Які члени можуть бути абстрактними?

Комп'ютерний практикум №4

СТРУКТУРИ

Мета роботи: розгляд структур, їх членів, вкладених структур; ознайомлення з наслідуванням структур; розгляд принципів пакування та розпаковування, коваріантності та контрваріантності структур; ознайомлення з різновидами структур [1-6, 8, 9, 22-26].

Теоретичні відомості

- **Структура (класична)** – це конструкція мови, що дозволяє містити набір полів різних типів.
- **Структура C#** – це конструкція мови, що складається з ключового слова struct, ідентифікатора і тіла. Структура може містити у своєму тілі поля, властивості та методи. Також структури можуть включати у своє тіло інші структури та класи, але такий підхід не є поширеною технікою.
- Структури повинні використовуватися тільки для зберігання маленьких, єдиних, бажано незмінних значень, які не будуть часто упаковуватися.
- В C# структурам забороняється мати конструктор за замовчуванням.
- Структури доцільно використовувати для спрощення списків параметрів методів.
- Відмінності класів від структур:
  - розміщення: в області стеку (класи – керована купа (heap));
  - копіювання: створюється окрема копія об'єкта, яка після копіювання живе «своїм життям» (класи – створюється посилання на той самий клас (так званий instance));
  - наслідування: не дозволяється доповнення своїми властивостями; від нього не можна успадковувати (клас – дозволяє, крім випадків, коли клас створювався з ключовим словом sealed, що не дозволяє наслідування);
  - передача параметрів: як локальні копії змінних (у класах – як посилання);
  - конструктор: так, крім конструктора за замовчуванням, який не потребує параметрів (у класах – так, без обмежень);
  - звільнення змінної: при виході за границі видимості (у класах – під час прибирання сміття (garbage collector)).
- **Пакування-перетворення** – це процес перетворення структурного типу на тип object або будь-який інший тип інтерфейсу, реалізований цим типом.
- Коли структурний тип упаковується середовищем CLR, воно створює програму-оболонку значення всередині System.Object і зберігає її в керованій купі.

- Операція **розпакування-перетворення** витягує структурний тип з об'єкта.
- Пакування-перетворення є неявним; розпакування-перетворення є явним.
- Концепція пакування та розпакування лежить в основі єдиної системи типів C#, в якій значення будь-якого типу можна розглядати як об'єкт.
- **DateTime** – структура, яка представляє поточний час, зазвичай виражений як дата і час доби.
- DateTime представляє дату та час в діапазоні від 00:00:00 1 січня 0001 (н. е.) до 23:59:59 31 грудня 9999 (н. е.).
- Значення часу вимірюються в 100-наносекундних одиницях, які називаються тактами, і точна дата представляється числом тактів з 00:00 1 січня 0001 н. е. (н. е.) у календарі GregorianCalendar (за винятком тактів, доданих секундами коригування). Наприклад, значення тактів, що дорівнює 31241376000000000L, є п'ятницею 1 січня 0100 00:00:00. Значення DateTime завжди виражається у контексті явно визначеного чи заданого за умовчанням календаря.
- Способи створення нового значення DateTime:
  - шляхом виклику будь-якої з перевантажених версій конструктора DateTime, які дозволяють вказати певні елементи значення дати та часу (наприклад, рік, місяць, день чи кількість інтервалів);
  - використовуючи будь-який синтаксис компілятора для оголошення значень дати та часу. (DateTime.Now);
  - шляхом аналізу рядкового подання значення дати та часу (метод (Parse)).
- **Перелічуваний тип** (перелічення, enumeration, enumerated type) – в програмуванні тип даних, чия множина значень є обмеженим списком ідентифікаторів.
- Перелічуваний тип визначається як набір ідентифікаторів, які з погляду мови відіграють таку саму роль, як і звичайні іменовані константи, але пов'язані з цим типом.
- Джеффрі Ріхтер: Перелічення є структурою з низкою статичних констант.
- Рекомендується використовувати int як основний тип перелічення.
- Рекомендується називати перелічення іменниками в однині.
- Використання перелічень дозволяє зробити вихідні коди програм більш зручними для читання, оскільки дозволяють замінити «магічні числа», що кодують певні значення, на імена, що читаються.
- Застереження Ніклауса Вірта під час використання перелічень: «Непродумане використання перелічень призводить до демографічного вибуху серед типів, що, своєю чергою, веде не до ясності програм, а до багатослівності».
- Не рекомендується створювати перелічення з одним значенням.

Практичні приклади

Приклад 1 Structure

```
using System;

// Структури.

// В структурах неможна ініціалізувати поля безпосередньо в місці створення.

namespace Structure
{
    struct MyStruct
    {
        public int field;
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу, без виклика конструктора.
            MyStruct instance;

            instance.field = 1; // Закоментувати.

            // Спроба виведення значення неініціалізованого поля призведе до помилки.
            Console.WriteLine(instance.field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Structure

```
using System;

// Структури.

// Якщо в структурі є члени, які звертаються до поля і немає конструктора користувача,
//  то необхідно при створенні екземпляра викликати конструктор за замовчуванням.
(Інакше будет помилка.)

namespace Structure
{
    struct MyStruct
    {
        private int field;

        public int Field
        {
            get { return field; }
            set { field = value; }
        }

        public void Show()
        {
            Console.WriteLine(field);
        }
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу з викликом конструктора за
замовчуванням.
            MyStruct instance = new MyStruct();
```

```
            instance.Field = 1;

            Console.WriteLine(instance.Field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Structure

```
using System;

// Структури.

namespace Structure
{
    struct MyStruct
    {
        public int field;

        // Конструктори за замовчуванням неможна задавати явно.
        //public MyStruct()
        //{
        //}

        // Якщо в структурі є конструктор користувача, то необхідно в ньому
ініціалізувати всі поля.
        public MyStruct(int value)
        {
            Console.WriteLine("Constructor");
            this.field = value;
        }
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу з викликом конструктора за
замовчуванням.
            MyStruct instance = new MyStruct();

            Console.WriteLine(instance.field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Structure

```
using System;

// Структури.

namespace Structure
{
    struct MyStruct
    {
        public int field;

        // Конструктори за замовчуванням неможна задавати явно.
        //public MyStruct()
        //{
        //}
```

```
        // Якщо в структурі є конструктор користувача, то необхідно в ньому
ініціалізувати всі поля.
        public MyStruct(int value)
        {
            Console.WriteLine("Constructor");
            this.field = value;
        }
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу з викликом конструктора
користувача.
            MyStruct instance = new MyStruct(1);

            Console.WriteLine(instance.field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Structure

```
using System;

// Структури.

// Структури можуть містити статичні члени.

// Статичні структури неприпустимі.

namespace Structure
{
    struct MyStruct
    {
        public static int Field
        {
            get;
            set;
        }

        public static void Show()
        {
            Console.WriteLine(Field);
        }
    }

    class Program
    {
        static void Main()
        {
            // Ініціалізація статичних полів необов'язкова.
            //MyStruct.Field = 1;

            MyStruct.Show();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 Structure

```
using System;

// Структури. Статичний конструктор.

namespace Structure
{
    struct MyStruct
    {
        public int field;

        // Статичний конструктор завжди відпрацьовує першим.
        static MyStruct()
        {
            Console.WriteLine("Static Constructor");
        }

        // Якщо в структурі є конструктор користувача, то необхідно в ньому
ініціаліувати всі поля.
        public MyStruct(int value)
        {
            Console.WriteLine("Constructor");
            this.field = value;
        }
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу з викликом конструктора
користувача.
            MyStruct instance = new MyStruct { field = 0 };

            Console.WriteLine(instance.field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 Structure

```
using System;

// Структури. Конструктори.

namespace Structure
{
    struct MyStruct
    {
        public int field;

        // Конструктор користувача з параметрами.
        public MyStruct(int value)
        {
            this.field = value;

            Console.WriteLine(value);
        }
    }

    class Program
    {
        static void Main()
        {
            // Створення екземпляра структурного типу без виклика конструктора.
            MyStruct instance;
```

```
            // Неможна використовувати не ініціалізовану змінну.
            // Оскільки конструктор не викликався змінна field залишилась не
ініціалізованою.

            //Console.WriteLine(instance.field);   // Прибрати коментар.

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 Structure

```
using System;

// Структури.

// В структурах можна створювати автоматично реалізуємі властивості,
// при цьому потрібно використовувати конструктор при побудові екземпляра.

namespace Structure
{
    struct MyStruct
    {
        public int MyProperty { get; set; }
    }

    class Program
    {
        static void Main()
        {
            MyStruct instance = new MyStruct();

            instance.MyProperty = 1;
            Console.WriteLine(instance.MyProperty);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 9 Structure s

```
using System;

// Структури.

// Приклад паганого дизайну.

namespace Structure
{
    struct ZipCode
    {
        // Поля
        int fiveDigitCode;
        int plusFourExtension;

        // Властивості.
        public int FiveDigitCode
        {
            get { return fiveDigitCode; }
            set { fiveDigitCode = value; }
        }

        public int PlusFourExtension
        {
            get { return plusFourExtension; }
            set { plusFourExtension = value; }
```

```
        }
    }

    class Program
    {
        static void Main()
        {
            ZipCode zipCode = new ZipCode();

            zipCode.FiveDigitCode = 12345;
            zipCode.PlusFourExtension = 1234;

            Console.WriteLine(zipCode.FiveDigitCode);
            Console.WriteLine(zipCode.PlusFourExtension);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 10 Structure

```
using System;

// Структури.

// Приклад гарного дизайну.

namespace Structure
{
    struct ZipCode
    {
        // Поля
        int fiveDigitCode;
        int plusFourExtension;

        // Конструктори.
        public ZipCode(int fiveDigitCode, int plusFourExtension)
        {
            this.fiveDigitCode = fiveDigitCode;
            this.plusFourExtension = plusFourExtension;
        }

        public ZipCode(int fiveDigitCode)
            : this(fiveDigitCode, 0)
        {
        }

        // Властивості.
        public int FiveDigitCode
        {
            get { return fiveDigitCode; }
        }

        public int PlusFourExtension
        {
            get { return plusFourExtension; }
        }
    }

    class Program
    {
        static void Main()
        {
            ZipCode zipCode = new ZipCode(12345, 1234);

            Console.WriteLine(zipCode.FiveDigitCode);
            Console.WriteLine(zipCode.PlusFourExtension);
```

```
            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 11 Inheritance

```
using System;

// Структуры. Наслідування.

// Наслідування структур дозволене лише від інтерфейсів.
// Наслідування структур від класів та структур заборонене.

namespace Inheritance
{
    interface IInterface
    {
        void Method();
    }

    struct MyStruct : IInterface
    {
        public void Method()
        {
            Console.WriteLine("Method");
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct instance;

            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 Inheritance

```
using System;

// Структури. Наслідування.

// Всі структури неявно наслідуються від абстрактного класу - ValueType

namespace Inheritance
{
    struct MyStruct // : ValueType
    {
        public void Method()
        {
            Console.WriteLine("Method");
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct instance = new MyStruct();

            ValueType valueType = instance as ValueType;
```

```
            Console.WriteLine("instance  = {0}", instance.GetHashCode());
            Console.WriteLine("valueType = {0}", valueType.GetHashCode());
            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 13 Inheritance

```
using System;

// Структури. Наслідування.

// Від структур неможна наслідуватись.
namespace Inheritance
{
    struct MyStruct
    {
        // Структури не можуть мати protected членів.
        //protected int field;
    }

    class MyClass //: MyStruct
    {
    }

    class Program
    {
        static void Main()
        {
            Console.WriteLine("Від структур неможна наслідуватись.");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 14 Nested

```
using System;

// Nested structures.

namespace Nested
{
    struct MyStruct
    {
        public struct Nested
        {
            public void Method()
            {
                Console.WriteLine("Nested");
            }
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct.Nested instance = new MyStruct.Nested();
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 15 Nested

```
using System;

// Nested structures.

// Структури можуть містити вкладені класи.

namespace Nested
{
    struct MyStruct
    {
        public class Nested
        {
            public void Method()
            {
                Console.WriteLine("Nested");
            }
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct.Nested instance = new MyStruct.Nested();
            instance.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 16 Boxing

```
using System;

// Пакування та розпакування.

// Пакування (Boxing) - перетворення структурного типу (типу значення) в тип-посилання
// (Object або будь-який інший тип інтерфейса, що реалізується цим типом значення).
// Коли тип значення пакується середовищем CLR, воно створює програму-оболонку
значення всередині
// System.Object і зберігає її в керуємій купі.

// Розпакування (Unboxing) - перетворення типу-посилання в структурний тип.
// Операція розпакування-перетворення вилучає тип значення з об'єкта.
// Перед розпакуванням середовище виконання перевіряє сумісність між об'єктом та
структурою,
// в яку буде відбуватися розпакування.

namespace Boxing
{
    class Program
    {
        static void Main()
        {

            short a = 25;

            // Пакування  змінної - a  (Boxing).
            object o = a;

            // Розпакування об'єкта (UnBoxing).
            short b = (short)o;

            // Розпакування має проводитись тільки в тей тип, з якого проводилось
пакування.
            //byte s = (byte)o;
```

```
            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 17 Boxing

```
using System;

// Пакування та розпакування

namespace Boxing
{
    interface IInterface
    {
        void Method();
    }

    struct MyStruct : IInterface
    {
        public void Method()
        {
            Console.WriteLine("Method");
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct my;
            my.Method();
            // Пакування (Boxing).
            IInterface boxed = my;

            boxed.Method();

            // Розпакування об'єкта (UnBoxing).
            MyStruct unBoxed = (MyStruct)boxed;

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 18 Boxing

```
using System;

// Пакування та розпакування

namespace Boxing
{
    struct MyStruct //: ValueType
    {
        public void Method()
        {
            Console.WriteLine("Method");
        }
    }

    class Program
    {
        static void Main()
        {
            MyStruct my = new MyStruct();

            // Пакування (Boxing).
```

```
            ValueType boxed = my;

            // Розпакування об'єкта (UnBoxing).
            MyStruct unBoxed = (MyStruct)boxed;

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 19 ArrayCovariant

```
using System;

// Коваріантність не застосовується до масивів елементів структурних типів.

namespace ArrayCovariant
{
    public interface IAnimal
    {
        void Voice();
    }

    public struct Dog : IAnimal
    {
        public void Voice()
        {
            Console.WriteLine("Gav-Gav");
        }
    }

    class Program
    {
        static void Main()
        {
            // 1
            Dog[] dogs = { new Dog(), new Dog(), new Dog() };

            //IAnimal[] animal = dogs; // Коваріантність.
            //dogs = array;  // Контрваріантність.
            ValueType i = new Int32() as ValueType;

            // 2
            int[] vector = new int[3] { 1, 2, 3 };
            //object[] array = vector; // Коваріантність


            // Затримкадержка.
            Console.ReadKey();
        }
    }
}
```

Приклад 20 ArrayCovariant

```
using System;

// Коваріантність структурних типів.

namespace Covariant
{
    class Program
    {
        static void Main()
        {
            uint[] array = new uint[3];

            Console.WriteLine("array  {0} {1}", array is uint[], array is int[]);
```

```
            object @object = array;

            Console.WriteLine("object {0} {1}", @object is uint[], @object is int[]);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 21 DateTime

```
using System;

// Структура DateTime надає поточний час, зазвичай виражений як дата та час доби.
// Тип значення DateTime надає дату і час в діапазоні від 00:00:00 1 січня 0001 року
(н. е.)
// до 23:59:59 31 грудня 9999 року (н. е.)

namespace WorkDateTime
{
    class Program
    {
        static void Main()
        {
            // DateTime.Now повертає об'єкт System.DateTime, якому присвоєні поточні
дата і час
            // доби даного ПК
            DateTime now = DateTime.Now;

            Console.WriteLine("Поточні дата і час : {0}", now);
            Console.WriteLine("Рік: {0}", now.Year);
            Console.WriteLine("Місяць: {0}", now.Month);
            Console.WriteLine("День місяця: {0}", now.Day);

            Console.WriteLine("Поточний час - {0}:{1}:{2}", now.Hour, now.Minute,
now.Second);

            Console.WriteLine("День тижня : {0}", now.DayOfWeek);
            Console.WriteLine("Це {0}-й день року", now.DayOfYear);

            //Отримуємо дату поточного ПК і значення часу, що дорівнює півночі
(00:00:00).
            Console.WriteLine(DateTime.Now.Date);

            // Затримка.
            Console.ReadKey();
        }
    }
}
```

Приклад 22 DateTime

```
using System;

// DateTime надає момент(значеннння) часу, тоді як TimeSpan надає інтервал(проміжок)
часу.
// Це означає, що можна відняти один екземпляр DateTime від іншого для отримання об
єкта TimeSpan,
// який представляє собою часовий інтервал між ними.
// Або можна додати додатне значення TimeSpan до поточного значення DateTime, щоб
отримати значення
// DateTime, яке представляє собою майбутню дату.

namespace DateTime2
{
    class Program
    {
        static void Main()
        {
```

```
            // Створення Нової дати. DateTime(рр, мм, дд)
            DateTime newYearDate = new DateTime(2013, 1, 1);
            DateTime today = DateTime.Now;

            // Представляє інтервал часу.
            TimeSpan left = newYearDate - today;
            Console.WriteLine("До нового року залишилось " + left.Days + " днів");
            Console.WriteLine("До нового року залишилось " + left.TotalHours + "
годин");

            // Створення Нової дати і часу. DateTime(рр, мм, дд, гг, хв, сек)
            DateTime newDate = new DateTime(2012, 12, 05, 23, 11, 11);

            Console.WriteLine(newDate);                          // Виведення значення
дати і часу на екран
            Console.WriteLine(newDate.TimeOfDay);                // Виведення значення
часу, встановленного
                                                                 // користувачем на
екран

            // Перетворює задане рядкове надання дати і часу в його еквівалент
            Console.WriteLine(DateTime.Parse("3/12/2012"));

            Console.WriteLine(DateTime.Parse("05 march 2012"));  // Місяць написати
локальною мовою OS.

            // Затримка.
            Console.ReadKey();
        }
    }
}
```

Приклад 23 DateTime

```
using System;

// Форматоване виведення дати і часу

namespace DateTime3
{
    class Program
    {
        static void Main()
        {
            DateTime now = DateTime.Now;

            Console.WriteLine("Дата місяць(прописом) рік : {0:D}", now);
            Console.WriteLine("Дата.місяць.рік : {0:d}", now);
            Console.WriteLine("Дата місяць(прописом) рік час(гг:хх:сс) : {0:F}", now);
            Console.WriteLine("Дата місяць(прописом) рік час(гг:хх) : {0:f}", now);
            Console.WriteLine("Дата.місяць.рік час(гг:хх:сс) : {0:G}", now);
            Console.WriteLine("Дата.місяць.рік час(гг:хх) : {0:g}", now);
            Console.WriteLine("Поточний місяць и дата : {0:M}", now);
            Console.WriteLine("Поточний місяць и год : {0:Y}", now);
            Console.WriteLine("Час(гг:хх:сс) : {0:T}", now);
            Console.WriteLine("Час(гг:хх) : {0:t}", now);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 24 Enums

```
using System;

// Перелічення.

// Перелічення - це набір іменованих констант, які зберігають числові значення.
```

```
// Перелічення визначає іменовані константи, кожній з яких відповідає числове
значення.
// Всі перелічення в C# походять від єдиного Базового класу System.Enum

namespace Enums
{
    // При компіляції - компілятор надає замість імен,
    // встановлені їм у відповідність числові значення. [ім'я] = [число]
    // За замовчуванням типом даних констант перелічення буде int.
    // Можна використовувати будь-який цілий тип даних C# (byte, sbyte, short, ushort,
int, uint, long, ulong)

    enum EnumType : byte // Явно вказуємо використовувати тип byte.
    {
        Zero = 0,
        One = 1,
        Two = 2,
        Three = 3
    }

    class Program
    {
        static void Main()
        {
            Console.WriteLine(EnumType.One);
            Console.WriteLine((byte)EnumType.One);

            EnumType digit = EnumType.Zero;
            Console.WriteLine(digit);
            Console.WriteLine((byte)digit);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 25 Enums

```
using System;

// Перелічення.

// Всі перелічення в C# походять від єдиного Базового класу System.Enum

namespace Enums
{
    enum EnumType : byte
    {
        Zero = 0,
        One = 1,
        Two = 2,
        Three = 3
    }

    class Program
    {
        static void Main()
        {
            Enum one = EnumType.One;
            Console.WriteLine(one);

            EnumType digit = EnumType.Zero;
            Enum zero = digit;
            Console.WriteLine(zero);

            Console.WriteLine(new string('-', 10));

            for (EnumType number = EnumType.Zero; number <= EnumType.Three; number++)
            {
```

```
                Console.WriteLine(number);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 26 Enums

```
using System;

// Перелічення.

namespace Enums
{
    // Можна використовувати псевдонім для будь-якого цілого типу даних C# (byte,
sbyte, short, ushort, int,
    // uint, long, ulong)

    // Неможна використовувати будь-який системний цілий тип даних C# (Byte, SByte,
Int16, UInt16, Int32,
    // UInt32, Int64, UInt64)

    enum EnumType //: Int32 // Помилка.
    {
        Zero = 0,
        One = 1,
        Two = 2,
        Three = 3
    }

    class Program
    {
        static void Main()
        {
            Console.WriteLine(EnumType.One);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 27 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One,   // = 1
        one = One,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
        Six = Nine,
        //Seven,
        Eight = 8, // Якщо не вказати значення явно, то на 13 рядку буде помилка!
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;

// Перелічення.

namespace Enums
{
    class Program
    {
        // Enum, як аргумент метода
        public static void MethodEnum(EnumType e)
        {
            switch (e)
            {
                case EnumType.Zero:
                    Console.WriteLine("Число 0");
                    break;
                case EnumType.Two:
                    Console.WriteLine("Число 2");
                    break;
                case EnumType.Five:
                    Console.WriteLine("Число 5");
                    break;
                case EnumType.Ten:
                    Console.WriteLine("Число 10");
                    break;

                default: break;
            }
        }

        static void Main()
        {
            MethodEnum(EnumType.Five);

            EnumType digit = EnumType.Ten;
            MethodEnum(digit);

            int i = (int)(++digit);
            Console.WriteLine(i);

            Console.WriteLine(digit); // Змінна змінилась.
            Console.WriteLine((int)EnumType.Ten); // Константа не змінилась.

            digit++;
            digit = digit + 5;

            // Неприпустимо.
            //digit = ++EnumType.One;
            //digit = EnumType.One + EnumType.Two;

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 28 Enums

```
using System;

// Перелічення. Значення за замовчуванням для перелічувань.

namespace Enums
{
    enum EnumType : byte
    {
        Zero = 0,
        One = 1,
        Two = 2,
```

```
        Three = 3
    }

    class Program
    {
        static void Main()
        {
            // Змінній five типу EnumType може бути призначене будь-яке значення, що
входить в діапазон
            // базового типу, значення не обмежені іменованими константами.

            EnumType five = (EnumType)5;

            Console.WriteLine(five);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 29 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One = 1,
        one = One,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
        //Six,
        Seven,
        Eight = 8,
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;

// Перелічення. Отримання інформації про тип елементів перелічення.

namespace Enums
{
    class Program
    {
        static void Main()
        {
            EnumType digit = EnumType.Ten;

            // Enum.GetUnderlyingType() - показує тип, який використовується для
надання
            // числових значень елементіов перелічення.
            Type @enum = digit.GetType();
            Console.Write("1. Перелічення {0} складається з елементів типу: ",
@enum.Name);
            Type @const = Enum.GetUnderlyingType(@enum);
            Console.WriteLine(@const);

            // оператор typeof - повертає екземпляр класу System.Type, який містить
інформацію
            // про той тип, який був переданий в якості аргумента даного оператора.
```

```
            Console.Write("2. Переліченння EnumType складається з елементів типу: ");

            Console.WriteLine(Enum.GetUnderlyingType(typeof(EnumType)));

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 30 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One = 1,
        one = One,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
        //Six,
        Seven,
        Eight = 8,
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;

// Перелічення. Форматоване виведення елементів перелічення

namespace Enums
{
    class Program
    {
        static void Main()
        {
            EnumType digit = EnumType.Ten;

            Console.WriteLine("Число {0}", digit.ToString());

            // Enum.Format() - Дозволяє робити більш точне форматування за рахунок
вказування прапорця,
            // а також отримувати імена елементів перелічення за їх числовим значенням

            // Виведення в 16-овому форматі. Прапорець "x" - hex (16-овий формат)
            Console.WriteLine("Hex значення {0}", Enum.Format(typeof(EnumType),
EnumType.Ten, "x"));

            // Виведення в 10-овому форматі. Прапорець "D" - dec (10-овий формат)
            Console.WriteLine("Dec значення {0}", Enum.Format(typeof(EnumType), digit,
"D"));

            // Виведення в рядковому форматі. Прапорець "G" - str (Рядковий формат)
            Console.WriteLine("Str значення {0}", Enum.Format(typeof(EnumType), 10,
"G"));

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 31 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One = 1,
        one = One,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
        Six = Nine,
        //Seven,
        Eight = 8, // Якщо не вказати значення явно, то на 13 рядку буде помилка!
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;

// Пошук елемента переліченння за ім'ям константи.

namespace Enums
{
    class Program
    {
        static void Main()
        {
            // Знаходимо елемент переліченння за ім'ям константи.
            object element = Enum.Parse(typeof(EnumType), "Infinite");
            EnumType number = (EnumType)element;

            Console.WriteLine("Значення константи {0}: {1}", number, (byte)number);

            // Enum.IsDefined() - Дозволяє визначити, чи є символьний рядок елементом
переліченння.
            bool flag = Enum.IsDefined(typeof(EnumType), "one");

            if (flag == true)
                Console.WriteLine("Да, переліченння містить елемент з таким ім'ям.");
            else
                Console.WriteLine("Нет, переліченння не містить елемент с таким
ім'ям.");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 32 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One = 1,
        one = One,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
```

```
        //Six,
        Seven,
        Eight = 8,
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;

// Отримання інформації про кількість елементів переліченння. Отримання всіх елементів
переліченння і їх значень

namespace Enums
{
    class Program
    {
        static void Main()
        {
            // Enum.GetValues() - повертає екземпляр System.Array, при цьому кожному
елементу масива
            // буде відповідати член вказаного переліченння.

            // Розміщуємо в масиві элементи переліченння.
            Array array = Enum.GetValues(typeof(EnumType));

            // Отримуємо інформацію про кількість елементів в масиві.
            Console.WriteLine("Це переліченння містить {0} членів \n", array.Length);

            // Виведення на екран всіх елементів переліченння
            for (int i = 0; i < array.Length; i++)
            {
                Console.WriteLine("Ім'я константи: {0}, значення {0:D}",
array.GetValue(i));
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 33 Enums

EnumType.cs

```
namespace Enums
{
    enum EnumType
    {
        Zero,  // = 0
        One = 1,
        one = 1,
        Two = 2,
        Three, // = 3
        Four,  // = 4
        Five = 5,
        //Six,
        Seven,
        Eight = 8,
        Nine,
        Ten = 10,
        Infinite = 255
    }
}
```

Program.cs

```
using System;
```

```
// Порівняння елементів переліченння

namespace Enums
{
    class Program
    {
        static void Main()
        {
            EnumType x = EnumType.Five;
            EnumType y = EnumType.One;

            //if (x < y)                        // Перший варіант порівняння.
            if (EnumType.Five < EnumType.One) // Другий варіант порівняння.
                Console.WriteLine("X = {0} (меньше чем) Y = {1}.", x, y);
            else
                Console.WriteLine("Y = {0} (меньше чем) X = {1}.", y, x);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи VisualStudio, створіть проєкт за шаблоном ConsoleApplication.
Створіть структуру з ім'ям – **Notebook**.
Поля структури: модель, виробник, ціна.
В структурі має бути реалізований конструктор для ініціалізації полів та методів для виведення вмісту полів на екран.

Завдання 2

Створіть клас MyClass і структуру **MyStruct**, які містять поля public string change.
У класі Program створіть два методи:
• static void ClassTaker(MyClass myClass), який надає полю change екземпляра myClass значення «змінено».
• static void StruktTaker(**MyStruct** myStruct), який надає полю change екземпляра **myStruct** значення «змінено».
Продемонструйте різницю у використанні класів та структур, створивши у методі Main() екземпляри структури та класу. Змініть значення полів екземплярів на «не змінено», а потім викличте методи ClassTaker і StruktTaker. Виведіть на екран значення полів екземплярів. Проаналізуйте отримані результати.

Індивідуальні завдання

Варіант 1

Описати структуру з ім'ям STUDENT, що містить наступні поля:
• прізвище та ініціали;
• номер групи;
• успішність (масив з п'яти елементів).
Написати програму, що виконує наступні дії:

• введення з клавіатури даних в масив, що складається з десяти структур типу STUDENT (записи повинні бути впорядковані за зростанням номера групи);
• виведення на екран прізвищ і номерів груп для всіх студентів, включених, в масив, якщо середній бал студента більше 4,0 (якщо таких студентів немає, вивести відповідне повідомлення).

Варіант 2

Описати структуру з ім'ям STUDENT, що містить наступні поля:
• прізвище та ініціали;
• номер групи;
• успішність (масив з п'яти елементів).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з десяти структур типу STUDENT (записи повинні бути впорядковані за зростанням середнього бала);
• виведення на екран прізвищ і номерів груп для всіх студентів, які мають оцінки 4 і 5 (якщо таких студентів немає, вивести відповідне повідомлення).

Варіант 3

Описати структуру з ім'ям STUDENT, що містить наступні поля:
• прізвище та ініціали;
• номер групи;
• успішність (масив з п'яти елементів).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з десяти структур типу STUDENT (записи повинні бути впорядковані за алфавітом);
• виведення на екран прізвищ і номерів груп для всіх студентів, які мають хоча б одну оцінку 2 (якщо таких студентів немає, вивести відповідне повідомлення).

Варіант 4

Описати структуру з ім'ям AEROFLOT, що містить наступні поля:
• назва пункту призначення рейсу;
• номер рейсу;
• тип літака.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з семи елементів типу AEROFLOT (записи повинні бути впорядковані за зростанням номера рейсу);
• виведення на екран номерів рейсів і типів літаків, що вилітають в пункт призначення, назва якого співпала з назвою, введеною з клавіатури (якщо таких рейсів немає, вивести відповідне повідомлення).

Варіант 5

Описати структуру з ім'ям AEROFLOT, що містить наступні поля:
• назва пункту призначення рейсу;
• номер рейсу;
• тип літака.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з семи елементів типу AEROFLOT (записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення);

• виведення на екран пунктів призначення і номерів рейсів, що обслуговуються літаком, тип якого введено з клавіатури (якщо таких рейсів немає, вивести відповідне повідомлення).

Варіант 6

Описати структуру з ім'ям WORKER, що містить наступні поля:
• прізвище та ініціали працівника;
• назва займаної посади;
• рік зарахування на роботу.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з десяти структур типу WORKER (записи повинні бути впорядковані за алфавітом);
• виведення на екран прізвищ працівників, стаж роботи яких перевищує значення, введене з клавіатури (якщо таких працівників немає, вивести відповідне повідомлення).

Варіант 7

Описати структуру з ім'ям TRAIN, що містить наступні поля:
• назва пункту призначення;
• номер потяга;
• час відправлення.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу TRAIN (записи повинні бути розміщені в алфавітному порядку за назвами пунктів призначення);
• виведення на екран інформації про потяги, що відправляються після введеного з клавіатури часу (якщо таких потягів немає, вивести відповідне повідомлення).

Варіант 8

Описати структуру з ім'ям TRAIN, що містить наступні поля:
• назва пункту призначення;
• номер потяга;
• час відправлення.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з шести елементів типу TRAIN (записи повинні бути впорядковані за часом відправлення потяга);
• виведення на екран інформації про потяги, що прямують в пункт, назва якого введена з клавіатури (якщо таких потягів немає, вивести відповідне повідомлення).

Варіант 9

Описати структуру з ім'ям TRAIN, що містить наступні поля:
• назва пункту призначення;
• номер потяга;
• час відправлення.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу TRAIN (записи повинні бути впорядковані за номерами потягів);

• виведення на екран інформації про потяг, номер якого введено з клавіатури (якщо таких потягів немає, вивести відповідне повідомлення).

Варіант 10

Описати структуру з ім'ям MARSH, що містить наступні поля:
• назва початкового пункту маршруту;
• назва кінцевого пункту маршруту;
• номер маршруту.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу MARSH (записи повинні бути впорядковані за номерами маршрутів);
• виведення на екран інформації про маршрут, номер якого введений з клавіатури (якщо таких маршрутів немає, вивести відповідне повідомлення).

Варіант 11

Описати структуру з ім'ям MARSH, що містить наступні поля:
• назва початкового пункту маршруту;
• назва кінцевого пункту маршруту;
• номер маршруту.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу MARSH (записи повинні бути впорядковані за номерами маршрутів);
• виведення на екран інформації про маршрути, які починаються або закінчуються в пункті, назва якого введена з клавіатури (якщо таких маршрутів немає, вивести відповідне повідомлення).

Варіант 12

Описати структуру з ім'ям NOTE, що містить наступні поля:
• прізвище, ім'я;
• номер телефону;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу NOTE (записи повинні бути впорядковані за датою народження);
• виведення на екран інформації про людину, номер телефону якої введено з клавіатури (якщо такої немає, вивести відповідне повідомлення).

Варіант 13

Описати структуру з ім'ям NOTE, що містить наступні поля:
• прізвище, ім'я;
• номер телефону;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу NOTE (записи повинні бути розміщені за алфавітом);
• виведення на екран інформації про людей, чиї дні народження припадають на місяць, значення якого введено з клавіатури (якщо таких немає, вивести відповідне повідомлення).

Варіант 14

Описати структуру з ім'ям NOTE, що містить наступні поля:
• прізвище, ім'я;

• номер телефону;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу NOTE (записи повинні бути впорядковані за трьома першими цифрами номера телефону);
• виведення на екран інформації про людину, чиє прізвище введене з клавіатури (якщо такої немає, вивести відповідне повідомлення).

Варіант 15

Описати структуру з ім'ям ZNAK, що містить наступні поля:
• прізвище ім'я;
• знак зодіаку;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу ZNAK (записи повинні бути впорядковані за датою народження);
• виведення на екран інформації про людину, чиє прізвище введене з клавіатури (якщо такої немає, вивести відповідне повідомлення).

Варіант 16

Описати структуру з ім'ям ZNAK, що містить наступні поля:
• прізвище ім'я;
• знак зодіаку;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу ZNAK (записи повинні бути впорядковані за датою народження);
• виведення на екран інформації про людей, які народилися під знаком, назва котрого введена з клавіатури (якщо таких немає, вивести відповідне повідомлення).

Варіант 17

Описати структуру з ім'ям ZNAK, що містить наступні поля:
• прізвище ім'я;
• знак зодіаку;
• дата народження (масив з трьох чисел).
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу ZNAK (записи повинні бути впорядковані за знаками Зодіаку);
• виведення на екран інформації про людей, які народилися в місяць, значення котрого введене з клавіатури (якщо таких немає, вивести відповідне повідомлення).

Варіант 18

Описати структуру з ім'ям PRICE, що містить наступні поля:
• назва товару;
• назва магазину, в якому продається товар;
• вартість товару в гривнях.
Написати програму, що виконує наступні дії:

• введення з клавіатури даних в масив, що складається з восьми елементів типу PRICE (записи повинні бути впорядковані в алфавітному порядку за назвами товарів);
• виведення на екран інформації про товар, назва якого введена з клавіатури (якщо таких товарів немає, вивести відповідне повідомлення).

Варіант 19

Описати структуру з ім'ям PRICE, що містить наступні поля:
• назва товару;
• назва магазину, в якому продається товар;
• вартість товару в гривнях.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу PRICE (записи повинні бути впорядковані в алфавітному порядку за назвами магазинів);
• виведення на екран інформації про товари, що продаються в магазині, назва якого введена з клавіатури (якщо такого магазину немає, вивести відповідне повідомлення).

Варіант 20

Описати структуру з ім'ям ORDER, що містить наступні поля:
• розрахунковий рахунок платника;
• розрахунковий рахунок одержувача;
• перераховується сума в гривнях.
Написати програму, що виконує наступні дії:
• введення з клавіатури даних в масив, що складається з восьми елементів типу ORDER (записи повинні бути розміщені в алфавітному порядку за розрахунковими рахунками платників);
• виведення на екран інформації про суму, зняту з розрахункового рахунку платника, введеного з клавіатури (якщо такого розрахункового рахунку немає, вивести відповідне повідомлення).

Контрольні питання

1. Що таке структура?
2. Чи можна створювати структури без оператора new?
3. Які обмеження при використанні структур вам відомі?
4. Назвіть основні відмінності класів від структур.
5. Чому структури можуть наслідуватись лише від інтерфейсів?
6. Чи можна в структурі створювати конструктори?
7. Що таке пакування і розпаковування?
8. Які види пакування вам відомі?
9. Чи коваріантні масиви елементів структурного типу?
10. Яку структуру необхідно використовувати для отримання формату часу?
11. В яких випадках необхідно використовувати структуру TimeSpan?
12. Які види перелічень вам відомі?
13. Від якого класу наслідуються всі перелічення?

Комп'ютерний практикум № 5

ДЕЛЕГАТИ

Мета роботи: знайомство з поняттями делегата та комбінованого делегата, розгляд та використання анонімних методів, лямбда-операторів та лямбда-виразів [1-6, 27-29].

Теоретичні відомості

- **Делегат (delegate)** – безпечний вказівник на метод. В класі-делегаті суворо вказані тип значення, що повертається, і аргументи методу (сигнатура).
- Всі делегати є похідними від абстрактного класу System.MulticastDelegate, який у свою чергу успадковується від абстрактного класу Delegate.
- Екземпляр делегату може посилатися на будь-який статичний метод або метод екземпляра – за умови, що сигнатура методу збігається з сигнатурою делегата.
- **Анонімна функція** – це оператор або вираз "inline", який можна використовувати щоразу, коли очікується тип делегата. Його можна використовувати для ініціалізації іменованого делегата або підставити замість типу іменованого делегата як параметр методу.
- Існує два типи анонімних функцій – це лямбда-вирази та анонімні методи.
- **Лямбда-вираз** – це анонімна функція, що містить вирази і оператори та може використовуватися для створення делегатів.
- Правила, які застосовуються до області дії змінної в лямбда-виразах:
  - захоплена змінна не буде знищена збирачем сміття доти, доки делегат, який на неї посилається, не вийде за межі області;
  - змінна, введена в лямбда-вираз, невидима у зовнішньому методі;
  - лямбда-вираз не може безпосередньо захоплювати параметри ref або out з методу, що включає їх;
  - лямбда-вираз не може містити оператор goto, оператор break або оператор continue, для яких, мітка переходу знаходиться поза тілом або в тілі анонімної функції.
- У всіх лямбда-виразах використовується лямбда-оператор =>, який читається як «переходить в». Ліва частина лямбда-оператора визначає параметри введення (якщо є), а права частина містить вираз або блок оператора. Лямбда-вираз x => x * x читається як «x перетворюється на x, x разів».
- Оператор => має той самий пріоритет, що і оператор присвоєння (=) і є право асоціативним.

Практичні приклади

Приклад 1 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    // Клас, метод якого буде сполучений з делегатом.
    static class MyClass
    {
        // Створюємо статичний метод, який плануємо сполучити з делегатом.
        public static void Method()
        {
            Console.WriteLine("Рядок виведено методом, сполученим з делегатом.");
        }
    }

    // На 21-ому рядку створюємо клас-делегата з ім'ям MyDelegate,
    // метод, який буде сполучений з екземпляром даного класа-делегата,
    // не буде нічого приймати и не буде нічого повертати.

    public delegate void MyDelegate();  // Створюємо клас делегата. (1)

    class Program
    {
        static void Main()
        {
            MyDelegate myDelegate = new MyDelegate(MyClass.Method); // Створюємо
екземпляр делегата. (2)

            myDelegate.Invoke(); // Викликаємо метод, сполучений з делегатом. (3)

            myDelegate();        // Інший спосіб виклика метода, сполученого з
делегатом. (3')

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    // Клас, метод якого буде сполучений з делегатом.
    class MyClass
    {
        // Створюємо метод, який плануємо сполучити з делегатом.
        public void Method()
        {
            Console.WriteLine("Рядок виведено методом, сполученим з делегатом.");
        }
    }

    // На 21-ому рядку створюємо клас-делегата з ім'ям MyDelegate,
    // метод, який буде сполучений з екземпляром даного класа-делегата,
    // не буде нічого приймати и не буде нічого повертати.

    public delegate void MyDelegate();  // Створюємо клас делегата. (1)

    class Program
    {
```

```
        static void Main()
        {
            MyClass instance = new MyClass();

            MyDelegate myDelegate = new MyDelegate(instance.Method); // Створюємо
екземпляр делегата. (2)

            myDelegate.Invoke();    // Викликаємо метод, сполучений з делегатом. (3)

            myDelegate();           // Інший спосіб виклика метода, сполученого з
делегатом. (3')

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    // Клас, метод якого буде сполучений з делегатом.
    class MyClass
    {
        // Створюємо метод, який плануємо сполучити з делегатом.
        public string Method(string name)
        {
            return "Hello " + name;
        }
    }

    // На 21-ому рядку створюємо клас-делегата з ім'ям MyDelegate,
    // метод, який буде сполучений з екземпляром даного класа-делегата,
    // буде приймати один рядковий аргумент і повертати рядкове значення.

    public delegate string MyDelegate(string name);  // Створюємо клас делегата. (1)

    class Program
    {
        static void Main()
        {
            MyClass instance = new MyClass();

            MyDelegate myDelegate = new MyDelegate(instance.Method); // Створюємо
екземпляр делегата і сполучаємо з ним метод. (2)

            string greeting = myDelegate.Invoke("Jeffrey Richter"); // Викликаємо
метод, сполучений з делегатом. (3)

            Console.WriteLine(greeting);

            greeting = myDelegate("Grady Booch"); // Інший спосіб виклика метода,
сполученого з делегатом. (3')

            Console.WriteLine(greeting);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Delegates

```
using System;
```

```
// Комбіновані (групові) делегати.

namespace Delegates
{
    public delegate void MyDelegate();

    class Program
    {
        // Методи.
        public static void Method1()
        {
            Console.WriteLine("Method1");
        }

        public static void Method2()
        {
            Console.WriteLine("Method2");
        }

        public static void Method3()
        {
            Console.WriteLine("Method3");
        }

        static void Main()
        {
            MyDelegate myDelegate = null;
            MyDelegate myDelegate1 = new MyDelegate(Method1);
            MyDelegate myDelegate2 = new MyDelegate(Method2);
            MyDelegate myDelegate3 = new MyDelegate(Method3);

            // Комбінуємо делегати.
            myDelegate = myDelegate1 + myDelegate2 + myDelegate3;

            Console.WriteLine("Введіть число від 1 до 7");
            string choice = Console.ReadLine();

            switch (choice)
            {
                case "1":
                    {
                        myDelegate1.Invoke();
                        break;
                    }
                case "2":
                    {
                        myDelegate2.Invoke();
                        break;
                    }
                case "3":
                    {
                        myDelegate3.Invoke();
                        break;
                    }
                case "4":
                    {
                        MyDelegate myDelegate4 = myDelegate - myDelegate1;
                        myDelegate4.Invoke();
                        break;
                    }
                case "5":
                    {
                        MyDelegate myDelegate5 = myDelegate - myDelegate2;
                        myDelegate5.Invoke();
                        break;
                    }
                case "6":
                    {
                        MyDelegate myDelegate6 = myDelegate - myDelegate3;
```

```
                        myDelegate6.Invoke();
                        break;
                    }
                case "7":
                    {
                        myDelegate.Invoke();
                        break;
                    }
                default:
                    {
                        Console.WriteLine("Ви ввели неприпустиме значення.");
                        break;
                    }
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Delegates

```
using System;

// Анонімні (лямбда) методи.

namespace Delegates
{
    // Створюємо клас делегата.
    public delegate void MyDelegate();

    class Program
    {
        static void Main()
        {
            // Створюємо екземпляр класа-делегата MyDelegate і сполучаємо з ним
анонімний метод.
            MyDelegate myDelegate = delegate { Console.WriteLine("Hello world!"); };

            // Виклик анонімного метода, сполученого з делегатом.
            myDelegate();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 Delegates

```
using System;

// Анонімні (лямбда) методи.

namespace Delegates
{
    // Створюємо клас делегата.
    public delegate int MyDelegate(int a, int b);

    class Program
    {
        static void Main()
        {
            int summand1 = 1, summand2 = 2, sum = 0;

            MyDelegate myDelegate = delegate(int a, int b) { return a + b; };

            sum = myDelegate(summand1, summand2);
```

```
            Console.WriteLine("{0} + {1} = {2}", summand1, summand2, sum);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 Delegates

```
using System;

// Анонімні (лямбда) методи.

namespace Delegates
{
    // Створюємо клас делегата.
    public delegate void MyDelegate(ref int a, ref int b, out int c);

    class Program
    {
        static void Main()
        {
            int summand1 = 1, summand2 = 2, sum;

            MyDelegate myDelegate = delegate(ref int a, ref int b, out int c) { a++;
b++; c = a + b; };

            myDelegate(ref summand1, ref summand2, out sum);

            Console.WriteLine("{0} + {1} = {2}", summand1, summand2, sum);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 Delegates

```
using System;

// Лямбда-вираз та лямбда-оператори.

namespace Delegates
{
    public delegate int MyDelegate(int a);

    class Program
    {
        static void Main()
        {
            MyDelegate myDelegate;

            myDelegate = delegate(int x) { return x * 2; }; // Лямбда-Метод

            myDelegate = (x) => { return x * 2; };          // Лямбда-Оператор.

            myDelegate = x => x * 2;                        // Лямбда-Вираз.

            int result = myDelegate(4);
            Console.WriteLine(result);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 9 Delegates

```
using System;

// Лямбда-вираз та лямбда-оператори.

namespace Delegates
{
    public delegate void MyDelegate();

    class Program
    {
        static void Main()
        {
            MyDelegate myDelegate;

            myDelegate = delegate { Console.WriteLine("Hello 1"); }; // Лямбда-Метод

            myDelegate += () => { Console.WriteLine("Hello 2"); };    // Лямбда-
Оператор.

            myDelegate += () => Console.WriteLine("Hello 3");         // Лямбда-Вираз.

            myDelegate();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 10 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    delegate Delegate2 Delegate1();  // 1.
    delegate void Delegate2();       // 2.


    class Program
    {
        public static Delegate2 Method1()      // 1.
        {
            return new Delegate2(Method2);
        }

        public static void Method2()           // 2.
        {
            Console.WriteLine("Hello world!");
        }

        static void Main()
        {
            Delegate1 delegate1 = new Delegate1(Method1);

            Delegate2 delegate2 = delegate1();

            delegate2();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 11 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    delegate Delegate3 Functional(Delegate1 delegate1, Delegate2 delegate2);
    delegate string Delegate1();
    delegate string Delegate2();
    delegate string Delegate3();

    class Program
    {
        public static Delegate3 MethodF(Delegate1 delegate1, Delegate2 delegate2)
        {
            return delegate { return delegate1.Invoke() + delegate2.Invoke(); };
        }

        public static string Method1() { return "Hello "; }
        public static string Method2() { return "world!"; }

        static void Main()
        {
            Functional functional = new Functional(MethodF);

            Delegate3 delegate3 = functional.Invoke(new Delegate1(Method1), new
Delegate2(Method2));

            Console.WriteLine(delegate3.Invoke());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 Delegates

```
using System;

// Делегати.

namespace Delegates
{
    delegate MyDelegate Functional(MyDelegate delegate1, MyDelegate delegate2);
    delegate string MyDelegate();

    class Program
    {
        static void Main()
        {
            MyDelegate delegate1 = () => "Hello ", delegate2 = () => "world!";

            Functional functional = delegate(MyDelegate d1, MyDelegate d2) { return
delegate() { return d1.Invoke() + d2.Invoke(); }; };
            //Functional functional = delegate(MyDelegate d1, MyDelegate d2) { return
() => d1.Invoke() + d2.Invoke(); };
            //Functional functional = (MyDelegate d1, MyDelegate d2) => () => d1() +
d2();

            //Console.WriteLine((functional.Invoke(delegate1,delegate2)).Invoke());
            Console.WriteLine(functional(delegate1, delegate2)());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 13 Delegates

```
using System;

// Делегати. (Припущення делегата)

namespace Delegates
{
    static class MyClass
    {
        public static void Method()
        {
            Console.WriteLine("Рядок виведено методом, сполученим з делегатом.");
        }
    }

    public delegate void MyDelegate();

    class Program
    {
        static void Main()
        {
           // MyDelegate myDelegate = new MyDelegate(MyClass.Method);

            MyDelegate myDelegate = MyClass.Method; // Припущення делегата.
            myDelegate();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 14 Delegates

```
using System;

// Рекурсія в лямбда-операторах.

namespace Delegates
{
    delegate void MyDelegate(int argument);

    class Program
    {
        static void Main()
        {
            MyDelegate my = null; // Необхідно обов'язково присвоїти null.

            // Необхідно окреме присвоєння посилання на делегат з сполученим лямбда-
оператором,
            // в месці створення змінної, неприпустимо одразу створювати лямбда-
оператор.

            my = (int i) =>
            {
                i--;
                Console.WriteLine("Begin {0}", i);

                if (i > 0)
                {
                    my(i);
                }

                Console.WriteLine("End {0}", i);
            };

            my(3);
```

```
            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Створіть чотири лямбда оператора для виконання арифметичних дій: (Add – додавання, Sub – віднімання, Mul – множення, Div – ділення). Кожен лямбда оператор повинен приймати два аргументи та повертати результат обчислення. Лямбда оператор ділення повинен робити перевірку ділення на нуль.
Написати програму, яка виконуватиме арифметичні дії, вказані користувачем.

Завдання 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Створіть анонімний метод, який приймає як аргумент масив делегатів і повертає середнє арифметичне повертаємих значень методів, повідомлених з делегатами в масиві. Методи, повідомлені з делегатами із масиву, повертають випадкове значення типу int.

Індивідуальні завдання

Варіант 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує радіус R та обчислює довжину кола за формулою: $D = 2\pi R$. Метод повинен бути оголошений як статичний.

Варіант 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує радіус R та обчислює площу кола за формулою: $S = \pi R^2$. Метод повинен бути оголошений як статичний.

Варіант 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує радіус R та обчислює об'єм кулі за формулою: $V = \frac{4}{3}\pi R^3$. Метод повинен бути оголошений як статичний.

Варіант 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує радіус R і висоту h та обчислює площу поверхні циліндра за формулою: $S = 2\pi R(R + h)$. Метод повинен бути оголошений як статичний.

Варіант 5

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує радіус R і висоту h та обчислює об'єм циліндра за формулою: $V = \pi R^2 h$. Метод повинен бути оголошений як статичний.

Варіант 6

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжину сторони a і обчислює площу квадрата за формулою: $S = a^2$. Метод повинен бути оголошений як статичний.

Варіант 7

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжину діагоналі d і обчислює периметр квадрата за формулою: $P = 2\sqrt{2}d$. Метод повинен бути оголошений як статичний.

Варіант 8

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжини сторін a і b та обчислює периметр прямокутника за формулою: $P = 2(a+b)$.
Метод повинен бути оголошений як статичний.

Варіант 9

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжини катетів a і b та обчислює гіпотинузу прямокутного трикутника за формулою: $c = \sqrt{a^2 + b^2}$. Метод повинен бути оголошений як статичний.

Варіант 10

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжини сторін a, b і c та обчислює периметр прямокутного трикутника за формулою: $P = a + b + c$. Метод повинен бути оголошений як статичний.

Варіант 11

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжину гіпотинузи c і гострий кут α та обчислює периметр прямокутного трикутника за формулою: $P = (1 + \sin\alpha + \cos\alpha)c$. Метод повинен бути оголошений як статичний.

Варіант 12

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.
Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.
Реалізувати виклик метода за допомогою делегата, який отримує довжину катета a і гострий кут α, що лежить навпроти катета, та обчислює периметр прямокутного трикутника за формулою: $P = a\left(\frac{1}{\operatorname{tg}\alpha} + \frac{1}{\sin\alpha} + 1\right)$. Метод повинен бути оголошений як статичний.

Варіант 13

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину катета a і прилеглий до нього кут β та обчислює периметр прямокутного трикутника за формулою: $P = a\left(\frac{1}{\text{ctg}\beta} + \frac{1}{\cos\beta} + 1\right)$. Метод повинен бути оголошений як статичний.

Варіант 14

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідним параметром змінну типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину ребра H і обчислює площу поверхні куба за формулою: $S = 6H^2$. Метод повинен бути оголошений як статичний.

Варіант 15

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину сторони a і висоту h та обчислює площу ромба за формулою: $S = ah$. Метод повинен бути оголошений як статичний.

Варіант 16

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину діоганалей d_1 і d_2 та обчислює площу ромба за формулою: $S = \frac{1}{2}d_1 d_2$. Метод повинен бути оголошений як статичний.

Варіант 17

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину сторони a і кут α та обчислює площу ромба за формулою: $S = a^2 \sin\alpha$. Метод повинен бути оголошений як статичний.

Варіант 18

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину основи трикутника b і довжину висоти трикутника h та обчислює площу рівнобедреного трикутника за формулою: $S = \frac{1}{2}bh$. Метод повинен бути оголошений як статичний.

Варіант 19

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує довжину основи трикутника b і довжину однієї з рівних сторін a та обчислює площу рівнобедреного трикутника за формулою: $S = \frac{1}{2}b\sqrt{\left(a+\frac{1}{2}b\right)\left(a-\frac{1}{2}b\right)}$. Метод повинен бути оголошений як статичний.

Варіант 20

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Оголосити тип делегата, який посилається на метод. Вимоги до сигнатури метода: метод отримує вхідними параметрами змінні типу double, метод повертає значення типу double, яке є результатом обчислення.

Реалізувати виклик метода за допомогою делегата, який отримує площу основи $S_{осн}$ і довжину висоти піраміди H та обчислює об'єм піраміди за формулою: $V = \frac{1}{3}S_{осн}H$. Метод повинен бути оголошений як статичний.

Контрольні питання

1. Що таке делегат?
2. Що таке комбінований делегат?
3. Що таке анонімний метод?
4. Які існують правила при виклику методів, повідомлених з делегатами?
5. Що таке лямбда-вираз?
6. Чим лямбда-вираз відрізняється від лямбда-оператора?

Комп'ютерний практикум №6

УНІВЕРСАЛЬНІ ШАБЛОНИ

Мета роботи: розгляд універсальних типів мові C# [1-6, 30-34].

Теоретичні відомості

- **Узагальнення** – елемент коду, здатний адаптуватися до виконання загальних (подібних) дій над різними типами даних.
- Універсальні шаблони були додані в мову C# версії 2.0 та середовище CLR. Цю можливість CTS (Common Type System – загальна система типів) назвали узагальненнями (generics).
- Узагальнення забезпечують більшу продуктивність, тому що не відбувається операції «пакування-розпакування" (boxing-unboxing).
- Узагальнення забезпечують безпеку типів, оскільки можуть містити лише типи, які задаються під час оголошення.
- Узагальнення дозволяють створювати відкриті (open-ended) типи, які перетворюються на закриті під час виконання.
- **Ідентифікатор <T>** – це вказівник місця заповнення, замість якого підставляється будь-який тип.
- Перезавантаження узагальнених типів розрызняються кількістю параметрів типу, а не їх іменами.
- Правильне перезавантаження: MyClass<T>{}, MyClass<T, R>{}
- Приклад відкритого типу: MyClass<T>
- Приклад закритого типу: MyClass<int>
- Кожен закритий тип отримує свою копію набору статичних полів.
- Загальні відомості про універсальні шаблони:
  - використовуйте універсальні типи для досягнення максимального рівня повторного використання коду, безпеки типу та продуктивності;
  - найчастішим випадком використання універсальних шаблонів є створення класів колекції;
  - можна створювати власні універсальні інтерфейси, класи, методи, події та делегати;
  - доступ універсальних класів до методів можна обмежити певними типами даних
- **Коваріантність узагальнень** – upcast параметрів типів.
- **Контраваріантність узагальнень** – downcast параметрів типів.
- Коваріантність та контраваріантність узагальнень в мові C# обмежена делегатами та інтерфейсами.
- Ключове слово default застосовується для встановлення стандартного значення для параметра типу.

- Тип Nullable<T> представляє типи значень із порожніми (нульовими) значеннями.
- При визначенні універсального типу можна обмежити види типів, які можна використовувати клієнтським кодом в якості аргументів типу при ініціалізації відповідного класу. При спробі клієнтського коду створити екземпляр класу за допомогою типу, який не допускається обмеженням, виникає помилка компіляції. Це називається **обмеженнями**.
- Обмеження визначаються за допомогою контекстно-залежного ключового слова where.
- Обмеження new вказує, що аргумент будь-якого типу у оголошенні загального класу повинен мати відкритий конструктор без параметрів. Використовувати обмеження new можна лише в тому випадку, якщо тип не є абстрактним.
- where T : **struct** – аргумент типу має бути структурного типу, крім Nullable.
- where T : class – аргумент типу повинен мати тип-посиланя; це також поширюється на тип будь-якого класу, інтерфейсу, делегату чи масиву.
- where T : <base class name> – аргумент типу повинен сам або бути похідним від зазначеного базового класу
- where T : U – аргумент типу, що постачається для T, повинен бути сам або бути похідним від аргументу, що поставляється дня U. Це називається **неприкритим обмеженням типу**.
- Колекція ArrayList – колекція з динамічним збільшенням розміру до потрібного значення.
- Місткість колекції ArrayList – це кількість елементів, яку може містити ArrayList. При додаванні елементів до колекції ArrayList її ємність автоматично збільшується за рахунок перерозподілу внутрішнього масиву.
- Якщо розмір колекції можна оцінити, доцільно вказати для неї початкову ємність, щоб позбавитися необхідності виконувати операції зміни розміру при додаванні елементів до колекції ArrayList.
- List<T> – клас, що представляє строго типізований список об'єктів, доступних за індексом. Підтримує методи пошуку за списком, виконання сортування та інших операцій зі списками.
- Dictionary <TKey, TValue> – клас, що представляє колекцію ключів та значень.

Практичні приклади

Приклад 1 Generics

```
using System;

// Універсальні шаблони.

namespace Generics
{
```

```
    // На 9-ому рядку створюємо клас з ім'ям MyClass, параметризований Вказівником
Місця Заповнення Типом - T

    class MyClass<T>
    {
        public T field;

        public void Method()
        {
            Console.WriteLine(field.GetType());
        }
    }

    class Program
    {
        static void Main()
        {
            // Створюємо екземпляр класу MyClass і в якості параметра типа (тип
MyClass) передаємо тип int.
            MyClass<int> instance1 = new MyClass<int>();
            instance1.Method();

            // Створюємо екземпляр класу MyClass і в якості параметра типа (тип
MyClass) передаємо тип long.
            MyClass<long> instance2 = new MyClass<long>();
            instance2.Method();

            // Створюємо екземпляр класу MyClass і в якості параметра типа (тип
MyClass) передаємо тип string.
            MyClass<string> instance3 = new MyClass<string>();
            instance3.field = "ABC";
            instance3.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Generics

```
using System;

// Універсальні шаблони.

namespace Generics
{
    class MyGenerics<TYPE1, TYPE2>
    {
        // Поля
        private TYPE1 variable1;
        private TYPE2 variable2;

        // Конструктор.
        public MyGenerics(TYPE1 argument1, TYPE2 argument2)
        {
            this.variable1 = argument1;
            this.variable2 = argument2;
        }

        // Властивості.
        public TYPE1 Variable1
        {
            get { return this.variable1; }
            set { this.variable1 = value; }
        }

        public TYPE2 Variable2
        {
            get { return this.variable2; }
```

```
            set { this.variable2 = value; }
        }
    }

    class Program
    {
        static void Main()
        {
            MyGenerics<int, int> instance1 = new MyGenerics<int, int>(1, 2);
            Console.WriteLine(instance1.Variable1 + instance1.Variable2);

            MyGenerics<string, int> instance2 = new MyGenerics<string, int>("Number ",
1);
            Console.WriteLine(instance2.Variable1 + instance2.Variable2);

            MyGenerics<string, string> instance3 = new MyGenerics<string,
string>("Hello ", "World");
            Console.WriteLine(instance3.Variable1 + instance3.Variable2);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Generics

```
using System;

// Універсальні шаблони. (Універсальний метод)

namespace Generics
{
    class MyClass
    {
        public void Method<T>(T argument)
        {
            T variable = argument;

            Console.WriteLine(variable);
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass instance = new MyClass();

            instance.Method<string>("Hello world!");

            instance.Method("Привіт світ!");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Generics

```
using System;

// Універсальні шаблони. (Універсальний делегат)

namespace Generics
{
    // На 11-ому рядку створюємо клас-делегата з ім'ям  MyDelegate, параметризований
двома Вказівниками Місця Заповнення Типом - T та R,
```

```
    // метод який буде сповіщений з екземпляром даного класа-делегата, буде приймати
один аргумент, типу Вказівника Місця Заповнення Типом - T,
    // і повертати значення типа Вказівника Місця Заповнення Типом - R.

    delegate R MyDelegate<T, R>(T t);

    class Program
    {
        public static int Add(int i)
        {
            return ++i;
        }

        public static string Concatenation(string s)
        {
            return "Hello " + s + "!";
        }


        static void Main()
        {
            MyDelegate<int, int> myDelegate1 = new MyDelegate<int, int>(Add);
            int i = myDelegate1.Invoke(1);
            Console.WriteLine(i);

            MyDelegate<string, string> myDelegate2 = new MyDelegate<string,
string>(Concatenation);
            string s = myDelegate2("Alex");
            Console.WriteLine(s);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Generics

```
using System;

// Коваріантність узагальнень.

namespace Generics
{
    public abstract class Shape { }
    public class Circle : Shape { }

    public interface IContainer<T>
    {
        T Figure { get; set; }
    }

    public class Container<T> : IContainer<T>
    {
        public T Figure { get; set; }

        public Container(T figure)
        {
            this.Figure = figure;
        }
    }

    class Program
    {
        static void Main()
        {
            Circle circle = new Circle();

            IContainer<Circle> container = new Container<Circle>(circle);
```

```
            Console.WriteLine(container.Figure.ToString());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 Generics

```
using System;

// Коваріантність узагальнень.

namespace Generics
{
    public abstract class Shape { }
    public class Circle : Shape { }

    public interface IContainer<T>
    {
        T Figure { get; set; }
    }

    public class Container<T> : IContainer<T>
    {
        public T Figure { get; set; }

        public Container(T figure)
        {
            this.Figure = figure;
        }
    }

    class Program
    {
        static void Main()
        {
            Circle circle = new Circle();

            IContainer<Shape> container = new Container<Shape>(circle);

            Console.WriteLine(container.Figure.ToString());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 Generics

```
using System;

// Коваріантність узагальнень.
// Коваріантність узагальнень в C# обмежена інтерфейсами і делегатами.

namespace Generics
{
    public abstract class Shape { }
    public class Circle : Shape { }

    public interface IContainer<out T>
    {
        T Figure { get; }
    }

    public class Container<T> : IContainer<T>
    {
        private T figure;
```

```
        public Container(T figure)
        {
            this.figure = figure;
        }

        public T Figure
        {
            get { return figure; }
        }
    }

    class Program
    {
        static void Main()
        {
            Circle circle = new Circle();

            IContainer<Shape> container = new Container<Circle>(circle);

            Console.WriteLine(container.Figure.ToString());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 Generics

```
using System;

// Контрваріантність узагальнень.
// Контрваріантність узагальнень в C# обмежена інтерфейсами і делегатами.

namespace Generics
{
    public abstract class Shape { }
    public class Circle : Shape { }

    public interface IContainer<in T>
    {
        T Figure { set; }
    }

    public class Container<T> : IContainer<T>
    {
        private T figure;

        public Container(T figure)
        {
            this.figure = figure;
        }

        public T Figure
        {
            set { figure = value; }
        }

        public override string ToString()
        {
            return figure.GetType().ToString();
        }
    }

    class Program
    {
        static void Main()
        {
            Shape shape = new Circle();
```

```
            IContainer<Circle> container = new Container<Shape>(shape);

            Console.WriteLine(container.ToString());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 9 Generics

```
using System;

// Коваріантність узагальнень.
// Коваріантність узагальнень в C# 4.0 обмежена інтерфейсами і делегатами.

namespace Generics
{
    class Animal { }
    class Cat : Animal { }

    class Program
    {
        delegate T MyDelegate<out T>();   // out - Для повертаємого значення.

        public static Cat CatCreator()
        {
            return new Cat();
        }

        static void Main()
        {
            MyDelegate<Cat> delegateCat = new MyDelegate<Cat>(CatCreator);
            MyDelegate<Animal> delegateAnimal = delegateCat;    // Від похідного до
базового.

            Animal animal = delegateAnimal.Invoke();

            Console.WriteLine(animal.GetType().Name);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 10 Generics

```
using System;

// Контрваріантність узагальнень.
// Контрваріантність узагальнень в C# обмежена інтерфейсами і делегатами.

namespace Generics
{
    class Animal { }
    class Cat : Animal { }

    class Program
    {
        delegate void MyDelegate<in T>(T a);  // in - Для аргумента.

        public static void CatUser(Animal animal)
        {
            Console.WriteLine(animal.GetType().Name);
        }

        static void Main()
```

```
        {

            MyDelegate<Animal> delegateAnimal = new MyDelegate<Animal>(CatUser);
            MyDelegate<Cat> delegateCat = delegateAnimal;    // Від базового до
похідного.

            delegateAnimal(new Animal());
            delegateCat(new Cat());

            //delegateCat(new Animal()); // Неможливо.

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 11 Generics

```
using System;

// Часткові класи та методи.

// Часткові методи не можуть мати out параметри.

namespace Generics
{
    public partial class MyClass<T>
    {
        partial void PartialMethod<T>(T a, ref T b);
    }

    public partial class MyClass<T>
    {
        partial void PartialMethod<T>(T a, ref T b)
        {
            b = default(T);
            Console.WriteLine("{0}, {1}", a, b);
        }

        public void Proxy(T a, ref T b)
        {
            PartialMethod(a, ref b);
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass<int> instance = new MyClass<int>();

            int operand1 = 1, operand2 = 2;

            instance.Proxy(operand1, ref operand2);

            Console.WriteLine(operand2);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 Generics

```
using System;

// Паганий приклад. Так робити не рекомендується.
```

```
namespace Generics
{
    class MyClass<T>
    {
        public T Add(T a, T b)
        {
            if (typeof(T) == typeof(int))
                return (T)(Object)((int)(object)a + (int)(object)b);

            if (typeof(T) == typeof(double))
                return (T)(Object)((double)(object)a + (double)(object)b);

            return (T)(object)0;
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass<int> my = new MyClass<int>();
            int sum = my.Add(2, 3);

            Console.WriteLine(sum);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 13 NullableTypes

```
using System;

namespace NullableTypes
{
    class Program
    {
        static void Main()
        {
            // a - містить невідоме значення.
            int? a = null;
            int? b = a + 4; // b = null
            int? c = a * 5; // c = null

            Console.WriteLine("->{0}<-", a);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 14 NullableTypes

```
using System;

namespace NullableTypes
{
    class Program
    {
        static void Main()
        {
            // a - містить невідоме значення.
            int? a = null;
            int? b = -5; // b = -5

            // При порівнянні операндів один з яких = null - результатом порівняння
завжди буде - false.
```

```
            // Виходячи з цього, неможна розраховувати на істинність (правильність)
результату.
            if (a >= b)
            {
                Console.WriteLine("a >= b");
            }
            else
            {
                Console.WriteLine("a < b");
            }

            // Порівнювати операнди (Nullable) є сенс лише для перевірки - чи обидва
містять null?
            // І якщо обидва операнди містять null, то результатом порівняння буде -
true.

            b = null;

            if (a == b)
            {
                Console.WriteLine("a == b");
            }
            else
            {
                Console.WriteLine("a != b");
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 15 NullableTypes

```
using System;

// Операція поглинання - ??

namespace NullableTypes
{
    class Program
    {
        static void Main()
        {
            int? a = null;
            int? b;

            b = a ?? 10; // b = 10
            Console.WriteLine(b);

            a = 3;
            b = a ?? 10; // b = 3
            Console.WriteLine(b);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 16 NullableTypes

```
using System;

namespace NullableTypes
{
    // Значення за замовчуванням для NullableTypes == null.

    class Program
```

```
    {
        static int? i;

        static void Main()
        {
            if (i == null)
            {
                Console.WriteLine("null");
            }
            else if (i == 0)
            {
                Console.WriteLine("0");
            }
            else
            {
                Console.WriteLine("Другое...");
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 17 NullableTypes

```
using System;

namespace NullableTypes
{
    class Program
    {
        static void Main()
        {
            Nullable<int> a = 1;

            if (a.HasValue == true)
            {
                Console.WriteLine("a is {0}.", a.Value);
            }

            // Коротка нотація Nullable типу - "?", доступна лише для C#.
            int? b = 1;

            if (b.HasValue == true)
            {
                Console.WriteLine("b is {0}", b.Value);
            }

            // Неявно типизована локальна змінна не може бути - Nullable.
            // var? c = null;

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 18 Constraint

```
using System;

// Обмеження параметрів типу.

namespace GenericsConstraints
{
    // where T : class  -   Аргумент типу повинен мати тип-посилання, це також
розповсюджується на тип будь-якого класу, інтерфейсу, делегата або масива.
    class MyClass1<T> where T : class
    {
```

```
        public T variable;
    }

    // where T : struct  -  Аргумент типу повинен мати тип значення. Допускається
вказування будь-якого типу значення, крім Nullable.
    class MyClass2<T> where T : struct
    {
        public T variable;
    }

    class Program
    {
        static void Main()
        {
            MyClass1<string> instance1 = new MyClass1<string>();
            //MyClass1<int> instance1 = new MyClass1<int>();                //
Помилка.    int - структурний тип.

            MyClass2<int> instance2 = new MyClass2<int>();
            //MyClass2<string> instance2 = new MyClass2<string>();          //
Помилка.    string - тип-посилання.

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 19 NewConstraint

```
using System;

// Обмеження параметрів типу

namespace GenericsConstraints
{
    // where T : new()  -  Аргумент типу повинен мати відкритий конструктор без
параметрів.

    // При використанні з іншими обмеженнями обмеження new() повинно встановлюватись
останнім:
    // class MyClass<T> where T : class, new()    { /* ... */ }

    class MyClass<T> where T : new()
    {
        public T instance = new T();

        public void GetValues()
        {
            Console.WriteLine(instance.ToString());
        }
    }

    class TestClass
    {
        public int MyIntProperty { get; set; }
        public string MyStringProperty { get; set; }

        public override string ToString()
        {
            return string.Format("{0} - {1}", MyIntProperty, MyStringProperty);
        }
    }

    class Program
    {
        static void Main()
        {
            MyClass<TestClass> foo = new MyClass<TestClass>();
            foo.instance.MyIntProperty = 1;
```

```
            foo.instance.MyStringProperty = "Hello World!";
            foo.GetValues();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 20 BaseConstraint

```
using System;

// Обмеження параметрів типу

namespace GenericsConstraints
{
    class Base { }
    class Derived : Base { }

    // where T : Base -  Аргумент типу повинен бути або бути похідним від вказаного
базового класу.
    class MyClass<T> where T : Base
    {
    }

    class Program
    {
        static void Main()
        {
            MyClass<Base> mc1 = new MyClass<Base>();

            MyClass<Derived> mc2 = new MyClass<Derived>();

            //MyClass<string> mc3 = new MyClass<string>();     // Помилка.

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 21 ConstraintTU

```
using System;

// Обмеження параметрів типу

namespace GenericsConstraints
{
    interface IInterface { /* ... */ }
    interface IInterface<U> { /* ... */ }

    class Derived : IInterface, IInterface<object> { /* ... */ }

    // where T : IInterface, IInterface<object>  -  Аргумент типу повинен бути або
реалізовувати вказаний інтерфейс.
    // Можна встановити декілька обмежень інтерфейса.
    // Обмежуючий інтерфейс також може бути універсальним.
    class MyClass<T> where T : IInterface, IInterface<object> { /* ... */ }

    class MyClass2<T> where T : IInterface { /* ... */ }

    class MyClass3<T> where T : IInterface<object> { /* ... */ }


    class Program
    {
        static void Main()
        {
```

```
            // В якості аргумента типу підходить Derived, оскільки він наслідується
від обох інтерфейсів.
            MyClass<Derived> my1 = new MyClass<Derived>();
            // MyClass<IInterface> my1 = new MyClass<IInterface>();    // Помилка.
            // MyClass<IInterface<object>> my1 = new MyClass<IInterface<object>>();
// Помилка.

            MyClass2<IInterface> my2 = new MyClass2<IInterface>();
            MyClass2<Derived> my3 = new MyClass2<Derived>();

            MyClass3<IInterface<object>> my4 = new MyClass3<IInterface<object>>();
            MyClass3<Derived> my5 = new MyClass3<Derived>();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 22 ConstraintTU2

```
using System;

// Обмеження параметрів типу

namespace GenericsConstraints
{
    interface IInterface { /* ... */ }
    interface IInterface<U> : IInterface { /* ... */ }

    class Derived : IInterface, IInterface<object> { /* ... */ }

    class Derived2 : IInterface<object> { /* ... */ }

    // where T : IInterface, IInterface<object>  -  Аргумент типу повинен бути або
реалізовувати вказаний інтерфейс.
    // Можна встановити декілька обмежень інтерфейса. Обмежуючий інтерфейс також може
бути універсальним.

    class MyClass<T> where T : IInterface, IInterface<object> { /* ... */ }

    class MyClass2<T> where T : IInterface<object> { /* ... */ }


    class Program
    {
        static void Main()
        {
            //  В якості аргумента типу підходить Derived, оскільки він наслідується
від обох інтерфейсів.
            MyClass<Derived> my1 = new MyClass<Derived>();
            //MyClass<IInterface> my1 = new MyClass<IInterface>();    // Помилка.

            //  Аргумент типу підходить, оскільки IInterface<object> наслідується від
IInterface
            MyClass<IInterface<object>> my2 = new MyClass<IInterface<object>>();

            MyClass2<Derived> my3 = new MyClass2<Derived>();
            MyClass2<Derived2> my4 = new MyClass2<Derived2>();
            MyClass2<IInterface<object>> my5 = new MyClass2<IInterface<object>>();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 23 ConstraintNaked

```
using System;
```

```
// Обмеження параметрів типу - "naked"

namespace GenericsConstraints
{
    // Аргумент типу, який надається в якості T, повинен співпадати з аргументом, який
надається в якості U, або бути похідним від нього.
    // Это называется ограничением типа "Naked".

    class MyClass<T, R, U> where T : U
    {
    }

    class Program
    {
        static void Main()
        {
            MyClass<int, Object, int> my1 = new MyClass<int, Object, int>();

            MyClass<string, Object, string> my2 = new MyClass<string, Object,
string>();

            // Не співпадає перший і третій аргументи типів - T та U (string та int).
            //MyClass<string, Object, int> my2 = new MyClass<string, Object, int>();
// ПОМИЛКА!

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 24 ArrayList_ListGen

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace Collection
{
    class Program
    {
        static void Main()
        {
            ArrayList arrayList = new ArrayList();

            // Boxing.
            arrayList.Add(1);
            arrayList.Add((object)2);


            // Unboxing
            int i1 = (int)arrayList[0];

            for (int i = 0; i < arrayList.Count; i++)
                Console.WriteLine((int)arrayList[i]);


            Console.WriteLine(new string('-', 3));


            List<int> list = new List<int>();

            // Пакування немає.
            list.Add(3);
            list.Add(4);

            // Розпакування немає.
            int i3 = list[0];
```

```
            for (int i = 0; i < list.Count; i++)
                Console.WriteLine(list[i]);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 25 Dictionary

```
using System;
using System.Collections.Generic;

// Словники (ключ/значення)

namespace DictionaryWork
{
    class Program
    {
        static void Main()
        {
            Dictionary<int, string> dictionary = new Dictionary<int, string>();

            dictionary.Add(0, "Ноль");
            dictionary.Add(1, "Один");
            dictionary.Add(2, "Два");
            dictionary.Add(3, "Три");

            Console.WriteLine(dictionary.ContainsValue("Ноль"));

            for (int i = 0; i < dictionary.Count; i++)
                Console.WriteLine(dictionary[i]);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створіть клас MyList<T>. Реалізуйте у найпростішому наближенні можливість використання його екземпляра аналогічно екземпляру класу List<T>. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Завдання 2

Використовуючи VisualStudio, створити проєкт за шаблоном Console Application.

Створіть клас MyClass<T>, який містить статичний фабричний метод – T FacrotyMethod(), який породжуватиме екземпляри типу, зазначеного як параметр типу (вказівника місця заповнення типом – T).

Завдання 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створіть клас Dictionary <TKey, TValue>. Реалізуйте у найпростішому наближенні можливість використання його екземпляра аналогічно екземпляру класу Dictionary із простору імен System.Collections.Generic. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання пар елементів, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості пар елементів.

Завдання 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створіть клас ArrayList. Реалізуйте у найпростішому наближенні можливість використання його екземпляра аналогічно екземпляру класу ArrayList із простору імен System.Collections.

Індивідуальні завдання

Варіант 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас CarCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення парку машин. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання машин з назвою машини і року її випуску, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас LibraryCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення бібліотеки. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання книг з назвою книги і року її видавництва, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас DepartmentCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення бази даних кафедри. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання викладача з прізвищем викладача і роком долучення до колективу кафедри, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас SensorCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку датчиків. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання датчика з назвою його маркування і року його виготовлення, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 5

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас SubjectCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку вибіркових дисциплін. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання дисциплін з назвою дисципліни і курсом, на якому вона викладається, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 6

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас ProjectCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку наукових проєктів. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання проєктів з назвою проєкту і роком його початку, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 7

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас DoctorCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку лікарів. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання лікарів з прізвищем лікаря і його спеціалізацією, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 8

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас MobileCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку операторів мобільного зв'язку. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання оператора з його назвою і кількістю користувачів, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 9

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас SportCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку спортсменів останніх Олімпійських ігор. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання спортсмена з його прізвищем і спеціалізацією, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 10

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас ShopCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку продукції магазину. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання товару з його назвою і кількістю, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 11

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас UrbanTransportCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку міського транспорту. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання міського транспорту з його назвою і кількістю, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 12

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас PharmacyCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення переліку ліків в аптеці. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання ліків з їх назвою і кількістю, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 13

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас PopulationCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення бази даних країн світу. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання країни з її назвою і кількістю населення, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 14

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас AirplaneCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення бази даних літаків. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання літаків з назвою компанії виробника і їх кількістю, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 15

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас LaptopCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для створення бази даних ноутбуків. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання ноутбуків з назвою компанії виробника і їх кількістю, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 16

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас UniversityCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для переліку факультетів університету. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання факультетів з назвою факультета і роком його створення, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 17

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас ZooCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для бази даних тварин в зоопарку. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання тварини з її назвою і роком народження, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 18

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас PhoneCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для бази даних мобільних телефонів. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання телефону з назвою компанії-виробника і роком випуску, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 19

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас ScientistCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для бази даних вчених-лауреатів Нобелівської премії з фізики. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання вчених з їх прізвищем і роком нагородження, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Варіант 20

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть клас ProgrammingLanguageCollection<T>. Реалізуйте у найпростішому наближенні можливість використання екземпляра для бази даних мов програмування. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання мов з їх назвою і роком створення, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів.

Контрольні питання

1. Що таке узагальнення?
2. Що таке закритий тип?
3. Що таке відкритий тип?
4. Поясніть поняття коваріантності та контраваріантності узагальнень.
5. Які переваги використання узагальнень вам відомі?
6. Що таке Nullable тип?
7. Що таке операція поглинання?
8. Які типи обмежень для узагальнень ви знаєте?
9. Що таке обмеження узагальнень?
10. Які види обмежень узагальнень ви знаєте?
11. Поясніть поняття коваріантності та контраваріантності узагальнень.
12. Які узагальнені колекції ви знаєте?

Комп'ютерний практикум № 7

ПОТОКИ

Мета роботи: знайомство з багатопоточністю; розгляд критичних секцій як засобів синхронізації доступу декількох потоків до ресурсів, що розділяються [1-6, 35-37].

Теоретичні відомості

- **Масив** – це іменований набір однотипних змінних, розміщених в пам'яті безпосередньо один за одним, доступ до яких здійснюється за індексом.
- C# підтримує паралельне виконання коду через багатопоточність.
- Потоки та процеси – це пов'язані поняття в обчислювальній техніці. Обидва є послідовністю інструкцій, які мають виконуватися процесором в певному порядку.
- **Потік** – це незалежний шлях реалізації, здатний виконуватися одночасно з іншими потоками.
- **Потік (thread)** – шлях виконання дій всередині додатка, що реалізується.
- **Потік** – елементарна одиниця виконання, яку можна планувати засобами операційної системи.
- **Потоки вводу-виводу (stream)** – надають можливість писати і читати байти з допоміжного пристрою, яким може бути один з декількох пристроїв зберігання інформації (місце на диску, оперативної пам'яті тощо).
- **Завдання (task)** – шлях виконання дій всередині програми, що реалізується. Виконанням завдань керує планувальник завдань, а не планувальник потоків (у разі потоків), який працює з пулом потоків. Завдання можна сприймати як оболонку для пулу потоків та кращого способу планування потоків (хоча за рахунок додаткових накладних витрат). Існуючі методи пулу потоків продовжать працювати, але завдання набагато легше використовувати, і вони пропонують додаткову функціональність.
- Процеси існують в операційній системі та відповідають тому, що користувачі бачать як програми або додатки. Потік існує всередині процесу. Кожен процес складається з одного або кількох потоків.
- Програма на C# запускається як єдиний потік, що автоматично створюється CLR та операційною системою («головний» або первинний потік), і стає багатопотоковою за допомогою створення додаткових потоків.
- Процес може створювати один або більше потоків для виконання частин програмного коду, пов'язаного з процесом. Слід використовувати делегат ThreadStart або ParameterizedThreadStart для завдання

програмного коду, керованого потоком. За допомогою делегата ParameterizedThreadStart можна передавати дані в потокову процедуру.

- Існують два різновиди потоків: пріоритетний та фоновий. Відмінність між ними полягає в тому, що процес не завершиться до тих пір, поки не закінчиться пріоритетний потік, тоді як фонові потоки завершуються автоматично після закінчення всіх пріоритетних потоків.
- За замовчуванням створюваний потік стає пріоритетним.
- Для того, щоб зробити потік фоновим, достатньо присвоїти логічне значення true властивості IsBackground. А логічне значення false вказує на те, що потік є пріоритетним.
- У разі використання декількох потоків доводиться координувати їх дії – такий процес, що називається **синхронізацією**.
- Основна причина застосування синхронізації – необхідність розділяти серед двох або більше потоків загальний ресурс (ресурс, що розділяється), який може бути одночасно доступний тільки одному потоку.
- В основу синхронізації покладено поняття блокування, за допомогою якого організується керування доступом до кодового блоку (критичної секції). Коли доступний для кожного потоку об'єкт (об'єкт синхронізації доступу) заблокований одним потоком, інші потоки не можуть отримати доступ до заблокованого кодового блоку (критичної секції). Коли блокування знімається одним потоком, об'єкт (об'єкт синхронізації доступу) стає доступним для використання в іншому потоці.
- Об'єктом синхронізації доступу до ресурсу, що розділяється, вважається такий об'єкт, який представляє синхронізований ресурс. У деяких випадках їм виявляється екземпляр самого ресурсу або довільний екземпляр класу, що використовується для синхронізації.
- Ключове слово lock не дозволить одному потоку увійти до важливого розділу коду в той момент, коли в ньому знаходиться інший потік. При спробі входу іншого потоку до заблокованого коду потрібно буде дочекатися зняття блокування об'єкта.
- Ключове слово lock викликає Monitor.Enter() на початку блоку та Monitor.Exit() наприкінці блоку.

Практичні приклади

Приклад 1 Threads

```
using System;
using System.Threading;

// Потоки.

namespace Threads
{
    class Program
    {
        // Метод, який планується виконувати в окремому потоці.
        static void WriteSecond()
        {
            while (true)
            {
```

```
                Console.WriteLine(new string(' ', 10) + "Secondary");
            }
        }

        static void Main()
        {
            ThreadStart writeSecond = new ThreadStart(WriteSecond);
            Thread thread = new Thread(writeSecond);
            thread.Start();

            while (true)
            {
                Console.WriteLine("Primary");
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Threads

```
using System;
using System.Threading;

// Потоки.

namespace Threads
{
    class Program
    {
        // Статичний метод, який планується виконувати одночасно в головному
(первинному) і у вторинному потоках.
        static void WriteSecond()
        {
            // CLR призначає кожному потоку свій стек і методи мають свої власні
локальні змінні.
            // Окремий екземпляр змінної counter створюється в стеку каждого потока,
            // тому для кожного потока виводяться, свої значення counter - 0,1,2.
            int counter = 0;

            while (counter < 10)
            {
                Console.WriteLine("Thread Id {0}, counter = {1}",
Thread.CurrentThread.GetHashCode(), counter);
                counter++;
            }
        }

        static void Main()
        {
            // Робота вторинного потока.
            Thread thread = new Thread(WriteSecond);
            thread.Start();

            // Робота первинного потока.
            WriteSecond();


            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Threads

```
using System;
using System.Threading;

// Потоки.
```

```
namespace Threads
{
    class Program
    {
        static void WriteSecond()
        {
            // Thread.CurrentThread - повертає посилання на екземпляр поточного
потока.
            Thread thread = Thread.CurrentThread;

            // Присвоює потоку ім'я.
            thread.Name = "Secondary";

            // Виводимо на екран інформацію про поточний потік.
            Console.WriteLine("ID потока {0}: {1}", thread.Name,
thread.GetHashCode());

            for (int counter = 0; counter < 10; counter++)
            {
                Console.WriteLine(new string(' ', 15) + thread.Name + " " + counter);
                // Призупиняємо виконання поточного потока.
                Thread.Sleep(1000);
            }
        }

        static void Main()
        {
            // Отримуємо посилання на екземпляр поточного потока.
            Thread primaryThread = Thread.CurrentThread;

            // Присвоюємо потоку ім'я.
            primaryThread.Name = "Primary";

            // Виводимо на екран інформацію про поточний потік.
            Console.WriteLine("ID потока {0}: {1}", primaryThread.Name,
primaryThread.GetHashCode());


            // Робота вторинного потока.
            Thread secondaryThread = new Thread(WriteSecond);
            secondaryThread.Start();

            // Робота первинного потока.
            for (int counter = 0; counter < 10; counter++)
            {
                Console.WriteLine(primaryThread.Name + " " + counter);
                // Призупиняємо виконання поточного потока.
                Thread.Sleep(1500);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Threads

```
using System;
using System.Threading;

// Потоки. Передача даних в потік.

namespace Threads
{
    class Program
    {
        // Метод, який планується виконувати в окремому потоці.
        static void WriteSecond(object argument)
        {
            for (int i = 0; i < 10; i++)
            {
```

```
                Console.WriteLine(argument);
                Thread.Sleep(1000);
            }
        }

        static void Main()
        {
            ParameterizedThreadStart writeSecond = new
ParameterizedThreadStart(WriteSecond);
            Thread thread = new Thread(writeSecond);
            thread.Start("Hello");

            Thread.Sleep(500);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Threads

```
using System;
using System.Threading;

// Потоки. Анонімні методи.

namespace Threads
{
    class Program
    {
        static void Main()
        {
            int counter = 0;

            // ThreadStart
            Thread thread = new Thread(delegate() { Console.WriteLine("1. counter =
{0}", ++counter); });
            thread.Start();

            Thread.Sleep(100);
            Console.WriteLine("2. counter = {0}", counter);

            // ParameterizedThreadStart
            thread = new Thread((object argument) => { Console.WriteLine("3. counter =
{0}", (int)argument); });
            thread.Start(counter);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 Threads

```
using System;
using System.Threading;

// Основні та фонові потоки. За замовчуванням властивість IsBackground дорівнює false.

namespace Threads
{
    class Program
    {
        // Метод, який планується виконувати в окремому потоці.
        static void WriteSecond()
        {
            while (true)
            {
                Console.WriteLine(new string(' ', 15) + "Secondary");
                Thread.Sleep(500);
            }
```

```
        }

        static void Main()
        {
            // Робота вторинного потока.
            ThreadStart writeSecond = new ThreadStart(WriteSecond);
            Thread thread = new Thread(writeSecond);
            thread.Start();

            // Робота первинного потока.
            for (int i = 0; i < 10; i++)
            {
                Console.WriteLine("Primary");
                Thread.Sleep(500);
            }

            // Завершити роботу вторинного потока
            //thread.IsBackground = true;
        }
    }
}
```

Приклад 7 CriticalSection

```
using System;
using System.Threading;

// Критичная секція (critical section).

// lock - блокує блок кода так, що в кожний окремий момент часу, цей блок кода
// зможе використати лише один потік. Всі інші потоки чекають поки поточний потік,
закінчить роботу.

namespace CriticalSection
{
    class MyClass
    {
        object block = new object();

        public void Method()
        {
            int hash = Thread.CurrentThread.GetHashCode();

            lock (block) // Закоментувати lock.
            {
                for (int counter = 0; counter < 10; counter++)
                {
                    Console.WriteLine("Потік # {0}: крок {1}", hash, counter);
                    Thread.Sleep(100);
                }
                Console.WriteLine(new string('-', 20));
            }
        }

        class Program
        {
            static void Main()
            {
                Console.SetWindowSize(80, 40);

                MyClass instance = new MyClass();

                for (int i = 0; i < 3; i++)
                {
                    new Thread(instance.Method).Start();
                }

                Thread.Sleep(500);

                // Delay.
                Console.ReadKey();
            }
```

```
        }
    }
}
```

Приклад 8 CriticalSection

```
using System;
using System.Threading;

// Критична секція (critical section).

// lock - це скорочене використання System.Threading.Monitor.
// Monitor.Enter(this) - блокує блок кода так, що його може використовувати тільки поточний потік.
// Всі інші потоки чекають поки поточний потік, закінчить роботу і викличе Monitor.Exit(this).

namespace CriticalSection
{
    class MyClass
    {
        object block = new object();

        public void Method()
        {
            int hash = Thread.CurrentThread.GetHashCode();

            Monitor.Enter(block); // Закоментувати.

            for (int counter = 0; counter < 10; counter++)
            {
                Console.WriteLine("Поток # {0}: шаг {1}", hash, counter);
                Thread.Sleep(100);
            }
            Console.WriteLine(new string('-', 20));

            Monitor.Exit(block);  // Закоментувати.
        }
    }

    class Program
    {
        static void Main()
        {
            Console.SetWindowSize(80, 40);
            MyClass instance = new MyClass();
            for (int i = 0; i < 3; i++)
            {
                new Thread(instance.Method).Start();
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 9 CriticalSection

```
using System;
using System.Threading;

// Критична секція (critical section).

namespace CriticalSection
{
    class Program
    {
        static object locker = new object();

        static void WriteSecond()
        {
            for (int i = 0; i < 20; i++)
            {
```

```
                lock (locker)
                {
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    Console.WriteLine(new string(' ', 10) + "Secondary");
                    Console.ForegroundColor = ConsoleColor.Gray;
                    Thread.Sleep(100);
                }
            }
        }

        static void Main()
        {
            Console.SetWindowSize(80, 45);

            ThreadStart writeSecond = new ThreadStart(WriteSecond);
            Thread thread = new Thread(writeSecond);
            thread.Start();

            for (int i = 0; i < 20; i++)
            {
                lock (locker)
                {
                    Console.ForegroundColor = ConsoleColor.Green;
                    Console.WriteLine("Primary");
                    Console.ForegroundColor = ConsoleColor.Gray;
                    Thread.Sleep(100);
                }
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

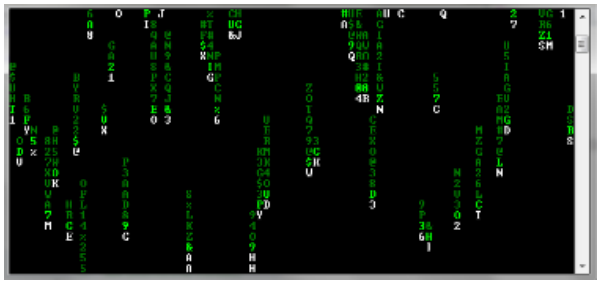
Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

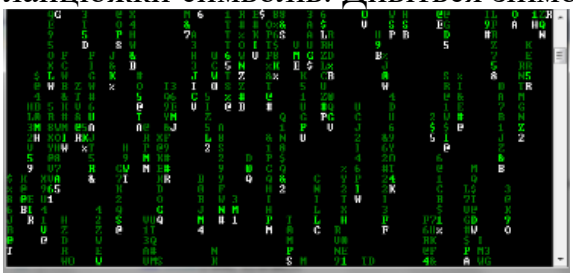
Створіть програму, яка буде виводити на екран ланцюжки символів, що падають. Довжина кожного ланцюжка визначається випадково. Перший символ ланцюжка – білий, другий символ – світло-зелений, решта символів темно-зелені. Під час падіння ланцюжка на кожному кроці всі символи змінюють своє значення. Дійшовши до кінця, ланцюжок зникає і зверху формується новий ланцюжок. Дивіться знімок екрана, представлений як зразок.



Завдання 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Розширте завдання 1 так, щоб в одному стовпці одночасно могло бути два ланцюжки символів. Дивіться знімок екрана, представлений як зразок.



Індивідуальні завдання

Варіант 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 20 випадкових дійсних чисел в діапазоні від -100 до 100, а Thread2 – 20 випадкових літер німецької абетки.

Варіант 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 20 випадкових цілих чисел в діапазоні від -50 до 50, а Thread2 – 20 випадкових літер української абетки.

Варіант 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 10 парних чисел, а Thread2 – 10 символів «*».

Варіант 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 10 непарних чисел, а Thread2 – 10 символів «+».

Варіант 5

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 15 випадкових літер англійскої абетки, а Thread2 – 20 випадкових літер української абетки.

Варіант 6

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 30 випадкових цілих чисел в діапазоні від 0 до 200, а Thread2 – 10 символів «-».

Варіант 7

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 10 символів «=», а Thread2 – 50 випадкових дробових чисел в діапазоні від -5 до 5.

Варіант 8

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 20 символів «@», а Thread2 – 50 випадкових парних чисел.

Варіант 9

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 15 символів «#», а Thread2 – 40 випадкових непарних чисел.

Варіант 10

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 20 символів «!», а Thread2 – 10 випадкових цілих чисел в діапазоні від -70 до 70.

Варіант 11

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 10 випадкових дробових чисел у діапазоні, що задається користувачем, а Thread2 – 20 символів «//».

Варіант 12

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 30 символів «\/», а Thread2 – 20 випадкових цілих чисел у діапазоні, що задається користувачем.

Варіант 13

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 20 нулів, а Thread2 – 10 символів «/\».

Варіант 14

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран 15 символів «|», а Thread2 – 20 одиниць.

Варіант 15

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 15 дійсних чисел в діапазоні від 0 до 15.

Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран суму всіх елементів масиву, а Thread2 – добуток всіх елементів.

Варіант 16

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 20 цілих чисел в діапазоні від 0 до 15. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран середнє арифметичне значення, а Thread2 – середнє геометричне значення елементів масиву.

Варіант 17

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 30 цілих чисел в діапазоні від -10 до 15. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран значення елементів масиву, піднесені до другого ступеня, а Thread2 – значення елементів масиву, піднесені до третього ступеня.

Варіант 18

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 10 цілих чисел в діапазоні від -20 до 20. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран суму елементів масиву, а Thread2 – кількість непарних елементів масиву.

Варіант 19

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 15 цілих чисел в діапазоні від 0 до 30. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран добуток елементів масиву, а Thread2 – кількість парних елементів масиву.

Варіант 20

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створіть масив з 10 цілих чисел в діапазоні від 0 до 40. Створіть програму, яка буде виводити на екран результати роботи двох потоків Thread1 і Thread2. Thread1 виводить на екран кількість непарних, а Thread2 – кількість парних елементів масиву.

Контрольні питання

1. Що таке потік, критична секція, ресурс, який розділяється?
2. Який клас потрібно використовувати для створення екземпляра потоку?
3. Який делегат використовується для передачі метода в потік?
4. Яким способом можна передати в потік параметри?
5. Що таке об'єкт синхронізації доступу до ресурсу, що розділяється?
6. Чим відрізняється використання оператора lock від класу Monitor?

Комп'ютерний практикум №8

КОЛЕКЦІЇ

Мета роботи: розгляд роботи колекцій та оператора yield [1-6, 38-42].

Теоретичні відомості

- **Колекція** – це набір об'єктів подібних типів, згрупованих разом.
- **Місткість колекції** – це кількість елементів, яку вона може містити.
- **Кількість елементів колекції** – це кількість елементів, які вона реально містить.
- **Нижня границя колекції** – це індекс першого елемента.
- **Перелючувач** – це об'єкт, який виконує ітерацію у пов'язаній з ним колекції. Можна вважати, що він є вказівником на будь-який елемент колекції. Перелічувач може бути пов'язаний лише з однією колекцією, але колекція може мати кілька перелічувачів. Оператор foreach використовує перелічувач і полегшує поводження з ним.
- Оператор foreach передбачає, що всі елементи колекції мають один і той самий тип.
- Всі перелічувачі засновані на інтерфейсі IEnumerator або на універсальному інтерфейсі IEnumerator<T>, для чого їм необхідно мати наведені нижче члени:
  - властивість Current вказує на поточний елемент колекції;
  - метод MoveNext() переміщує перелічувач до наступного елемента колекції;
  - метод Reset() переміщує перелічувач на початок колекції. Властивість Current вказує на положення перед першим елементом.
- Ключове слово yield повідомляє компілятор, що метод, в якому воно міститься, є блоком ітератора. Для реалізації поведінки, визначеної у блоці ітератора, компілятор створює клас. У блоці ітератора ключове слово yield використовується спільно з ключовим словом return для надання значення об'єкту перелічувача, наприклад, значення, що повертається в кожному циклі оператора foreach. Ключове слово yield завжди використовується разом із ключовим словом break для позначення кінця ітерації.
- Оператор yield не може використовуватись в анонімних методах.

Практичні приклади

Приклад 1 UserCollection

Element.cs

```
using System;

namespace InterIEnumerable
{
```

```
    // Екземпляри цього класу буде містити колекція - UserCollection.
    public class Element
    {
        // Поля.

        private string name;
        private int field1;
        private int field2;

        // Конструктор.
        public Element(string s, int a, int b)
        {
            name = s;
            field1 = a;
            field2 = b;
        }

        // Властивості.

        public int Field1
        {
            get { return field1; }
            set { field1 = value; }
        }

        public int Field2
        {
            get { return field2; }
            set { field2 = value; }
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
}
```

UserCollection.cs

```
using System;
using System.Collections;

namespace InterIEnumerable
{
    // Клас UserCollection колекція (набір) об'єктів класу Element.
    // Для застосування foreach, необхідно, щоб клас реалізовував інтерфейс -
IEnumerable.
    public class UserCollection : IEnumerable, IEnumerator
    {
        public Element[] elementsArray = null;

        public UserCollection()
        {
            elementsArray = new Element[4];
            elementsArray[0] = new Element("A", 1, 10);
            elementsArray[1] = new Element("B", 2, 20);
            elementsArray[2] = new Element("C", 3, 30);
            elementsArray[3] = new Element("D", 4, 40);
        }

        // Вказівник поточної позиції елемента в масиві.
        int position = -1;

        // -------------------------------------------------------------------------
        // Реалізація інтерфейса IEnumerator.

        // Пересунути внутрішній вказівник (position) на одну позицію.
        public bool MoveNext()
        {
            if (position < elementsArray.Length - 1)
```

```
            {
                position++;
                return true;
            }
            else
            {
                return false;
            }
        }

        // Встановити вказівник (position) перед початком набору.
        public void Reset()
        {
            position = -1;
        }

        // Отримати поточний елемент набору.
        public object Current
        {
            get { return elementsArray[position]; }
        }

        // -------------------------------------------------------------------------
        // Реалізація інтерфейса - IEnumerable.

        IEnumerator IEnumerable.GetEnumerator()
        {
            return this as IEnumerator;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;

namespace InterIEnumerable
{
    class Program
    {
        static void Main()
        {
            UserCollection myCollection = new UserCollection();

            // Використовуємо foreach для звернення до кожного об'єкта Element
всередині масиву myCollection.
            foreach (Element element in myCollection)
            {
                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }

            //myCollection.Reset(); // Прибрати коментар для перевірки.

            Console.Write(new string('-', 29) + "\n");

            // Використовуємо foreach, для повторного звернення до кожного об'єкта
Element всередині масиву myCollection.
            foreach (Element element in myCollection)
            {
                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }

            Console.Write(new string('-', 29) + "\n");


            // -------------------------------------------------------------------------
            // Так працює foreach.

            UserCollection myElementsCollection = new UserCollection();
```

```
            // foreach - приводить колекцію до інтерфейсного типу IEnumerable.
            IEnumerable enumerable = myElementsCollection as IEnumerable;

            // foreach - приводит колекцію до інтерфейсного типу, викликаючи метод -
GetEnumerator().
            IEnumerator enumerator = enumerable.GetEnumerator();


            while (enumerator.MoveNext()) // Переміщуєм курсор на 1 крок вперед (з -1
на 0) тощо.
            {
                Element element = enumerator.Current as Element;

                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }


            enumerator.Reset();

            // Затримка.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Yield

UserCollection.cs

```
using System.Collections;
using System.Collections.Generic;
using System;
using System.Threading;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            yield return "Hello world!";
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.WriteLine(element);

            Console.WriteLine(new string('-', 12));

            //------------------------------------------------------------------------
            // Так працює foreach.

            IEnumerable enumerable = UserCollection.Power();

            IEnumerator enumerator = enumerable.GetEnumerator();

            while (enumerator.MoveNext()) // Переміщуєм курсор на 1 крок вперед.
            {
                String element = enumerator.Current as String;
```

```
                Console.WriteLine(element);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 3 Yield

UserCollection.cs

```
using System;
using System.Threading;
using System.Collections;
using System.Collections.Generic;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            return new ClassPower(-2);
        }

        private sealed class ClassPower : IEnumerable<object>,  IEnumerator<object>,
IEnumerator, IDisposable
        {
            // Поля.
            private int state;
            private object current;
            private int initialThreadId;

            // Конструктор.
            public ClassPower(int state)
            {
                this.state = state;
                this.initialThreadId = Thread.CurrentThread.ManagedThreadId;
            }

            //private bool IEnumerator.MoveNext() // Так в Рефлекторі
            bool IEnumerator.MoveNext()
            {
                switch (this.state)
                {
                    case 0:
                        this.state = -1;
                        this.current = "Hello world!";
                        this.state = 1;
                        return true;

                    case 1:
                        this.state = -1;
                        break;
                }
                return false;
            }

            IEnumerator<object> IEnumerable<object>.GetEnumerator()
            {
                if ((Thread.CurrentThread.ManagedThreadId == this.initialThreadId) &&
(this.state == -2))
                {
                   this.state = 0;
                   return this;
                }
                return new UserCollection.ClassPower(0);
            }
```

```
            IEnumerator IEnumerable.GetEnumerator()
            {
                // Так в Рефлекторі
                //return
this.System.Collections.Generic.IEnumerable<System.Object>.GetEnumerator();

                return (this as IEnumerable<object>).GetEnumerator();
            }

            void IEnumerator.Reset()
            {
                throw new NotSupportedException();
            }

            void IDisposable.Dispose()
            {
            }

            // Властивості.
            object IEnumerator<object>.Current
            {
                get
                {
                    return this.current;
                }
            }

            object IEnumerator.Current
            {
                get
                {
                    return this.current;
                }
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.WriteLine(element);

            Console.WriteLine(new string('-', 12));

            //-------------------------------------------------------------------------
            // Так працює foreach.

            IEnumerable enumerable = UserCollection.Power();

            IEnumerator enumerator = enumerable.GetEnumerator();
            enumerator.Reset();

            while (enumerator.MoveNext()) // Переміщуєм курсор на 1 крок вперед.
            {
                String element = enumerator.Current as String;

                Console.WriteLine(element);
            }

            // Delay.
```

```
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Yield

UserCollection.cs

```
using System.Collections;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            yield break;
        }
    }
}
```

Program.cs

```
using System;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.Write(element);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Yield

UserCollection.cs

```
using System.Collections.Generic;
using System.Collections;
using System.Threading;
using System;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            return new ClassPower(-2);
        }

        private sealed class ClassPower : IEnumerable<object>, IEnumerable,
IEnumerator<object>, IEnumerator, IDisposable
        {
            // Поля.
            private int state1;
            private object current2;
            private int initialThreadId_1;

            // Конструктор.
            public ClassPower(int state1)
            {
                this.state1 = state1;
                this.initialThreadId_1 = Thread.CurrentThread.ManagedThreadId;
            }
```

```
            //private bool IEnumerator.MoveNext() // Так в Рефлекторі.
            bool IEnumerator.MoveNext()
            {
                if (this.state1 == 0)
                {
                    this.state1 = -1;
                }
                return false;
            }

            IEnumerator<object> IEnumerable<object>.GetEnumerator()
            {
                if ((Thread.CurrentThread.ManagedThreadId == this.initialThreadId_1)
&& (this.state1 == -2))
                {
                    this.state1 = 0;
                    return this;
                }
                return new UserCollection.ClassPower(0);
            }

            IEnumerator IEnumerable.GetEnumerator()
            {
                return (this as IEnumerable<object>).GetEnumerator();
            }

            void IEnumerator.Reset()
            {
                throw new NotSupportedException();
            }

            void IDisposable.Dispose()
            {
            }

            // Властивості.
            object IEnumerator<object>.Current
            {
                get
                {
                    return this.current2;
                }
            }

            object IEnumerator.Current
            {
                get
                {
                    return this.current2;
                }
            }
        }
    }
}
```

Program.cs

```
using System;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.Write(element);

            // Delay.
            Console.ReadKey();
        }
    }
```

```
}
```

Приклад 6 Yield

UserCollection.cs

```
using System;
using System.Collections;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            while (true)
                yield return "Hello world!";
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.WriteLine(element);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 Yield

UserCollection.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Threading;

namespace Yield
{
    class UserCollection
    {
        public static IEnumerable Power()
        {
            return new ClassPower(-2);
        }

        private sealed class ClassPower : IEnumerable<object>, IEnumerable,
IEnumerator<object>, IEnumerator, IDisposable
        {
            // Поля.
            private int state;
            private object current;
            private int initialThreadId;

            // Конструктор.
            public ClassPower(int state)
            {
                this.state = state;
                this.initialThreadId = Thread.CurrentThread.ManagedThreadId;
            }
```

```
            //private bool IEnumerator.MoveNext() // Так в Рефлекторі
            bool IEnumerator.MoveNext()
            {
                //----------------------------------------
                //switch (this.state)
                //{
                //    case 0:
                //        this.state = -1;
                //        break;
                //                                              // НЕПОТРІБНИЙ БЛОК.
                //    case 1:
                //        this.state = -1;
                //        break;

                //    default:
                //        return false;
                //}
                //----------------------------------------
                this.current = "Hello world!";
                this.state = 1;
                return true;
            }

            IEnumerator<object> IEnumerable<object>.GetEnumerator()
            {
                if ((Thread.CurrentThread.ManagedThreadId == this.initialThreadId) &&
(this.state == -2))
                {
                    this.state = 0;
                    return this;
                }
                return new UserCollection.ClassPower(0);
            }

            IEnumerator IEnumerable.GetEnumerator()
            {
                // Так в Рефлекторі
                //return
this.System.Collections.Generic.IEnumerable<System.Object>.GetEnumerator();

                return (this as IEnumerable<object>).GetEnumerator();
            }

            void IEnumerator.Reset()
            {
                throw new NotSupportedException();
            }

            void IDisposable.Dispose()
            {
            }

            // Властивості.
            object IEnumerator<object>.Current
            {
                get
                {
                    return this.current;
                }
            }

            object IEnumerator.Current
            {
                get
                {
                    return this.current;
                }
            }
        }
    }
```

```
}
```

Program.cs

```
using System;
using System.Collections;

namespace Yield
{
    class Program
    {
        static void Main()
        {
            foreach (string element in UserCollection.Power())
                Console.WriteLine(element);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 InterIEnumerable

Element.cs

```
using System;

namespace InterIEnumerable
{
    // Екземпляри цього класу буде містити колекція - UserCollection.
    public class Element
    {
        // Поля.

        private string name;
        private int field1;
        private int field2;

        // Конструктор.
        public Element(string s, int a, int b)
        {
            name = s;
            field1 = a;
            field2 = b;
        }

        // Властивості.

        public int Field1
        {
            get { return field1; }
            set { field1 = value; }
        }

        public int Field2
        {
            get { return field2; }
            set { field2 = value; }
        }

        public string Name
        {
            get { return name; }
            set { name = value; }
        }
    }
}
```

UserCollection.cs

```
using System.Collections;

namespace InterIEnumerable
{
```

```
    public class UserCollection : IEnumerable
    {
        public Element[] elementsArray = null;

        public UserCollection()
        {
            elementsArray = new Element[4];
            elementsArray[0] = new Element("A", 1, 10);
            elementsArray[1] = new Element("B", 2, 20);
            elementsArray[2] = new Element("C", 3, 30);
            elementsArray[3] = new Element("D", 4, 40);
        }

        // Вказівник поточної позиції елементу в масиві.
        int position = -1;

        // Встановити вказівник (position) перед початком набору.
        public void Reset()
        {
            position = -1;
        }

        // ------------------------------------------------------------------------
        // Реалізація інтерфейсу - IEnumerable.

        public IEnumerator GetEnumerator()
        {
            while (true)
            {
                if (position < elementsArray.Length - 1)
                {
                    position++;
                    yield return elementsArray[position];
                }
                else
                {
                    Reset();
                    yield break;  // Вихід з циклу.
                }
            }
        }
    }
}
```

Program.cs

```
using System;
using System.Collections;

namespace InterIEnumerable
{
    class Program
    {
        static void Main()
        {
            UserCollection myCollection = new UserCollection();

            // Використовуємо foreach для звернення до кожного об'єкта Element
всередині масиву myCollection.
            foreach (Element element in myCollection)
            {
                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }

            Console.Write(new string('-', 29) + "\n");

            // Використовуємо foreach для повторного звернення до кожного об'єкта
Element всередині масиву myCollection.
            foreach (Element element in myCollection)
            {
```

```
                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }

            Console.Write(new string('-', 29) + "\n");


            // ------------------------------------------------------------------------
            // Так працює foreach.

            UserCollection myElementsCollection = new UserCollection();

            // foreach - приводить колекцію до інтерфейсного типу IEnumerable.
            IEnumerable enumerable = myElementsCollection as IEnumerable;

            // foreach - приводить колекцію до інтерфейсного типу, викликаючи метод -
GetEnumerator().
            IEnumerator enumerator = enumerable.GetEnumerator();


            while (enumerator.MoveNext()) // Переміщаємо курсор на 1 крок вперед (з -1
на 0) тощо.
            {
                Element element = enumerator.Current as Element;

                Console.WriteLine("Name: {0}  Field1: {1} Field2: {2}", element.Name,
element.Field1, element.Field2);
            }

            // Затримка.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створіть колекцію MyList<T>. Реалізуйте у найпростішому наближенні можливість використання її екземпляра аналогічно екземпляру класу List<T>. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізуйте можливість перебору елементів колекції у циклі foreach.

Завдання 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створіть колекцію MyDictionary<TKey, TValue>. Реалізуйте у найпростішому наближенні можливість використання її екземпляра аналогічно екземпляру класу Dictionary<TKey, TValue>. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної

кількості елементів. Реалізуйте можливість перебору елементів колекції у циклі foreach.

Індивідуальні завдання

Варіант 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію цілих чисел MyList<int>. Видалити з неї всі числа, які без залишку діляться на перший елемент. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію цілих чисел MyList<int>. Вставити 0 між будь-якими двома парними елементами колекції. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію цілих чисел MyList<int>. Вставити 1 між будь-якими двома непарними елементами колекції. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію дійсних чисел MyList<double>. Видалити з неї всі числа, які без залишку діляться на останній елемент. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 5

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію дійсних чисел MyList<double>. Вставити 7,7 між будь-якими двома непарними елементами колекції. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 6

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію дійсних чисел MyList<double>. Вставити 5,5 між будь-якими двома парними елементами колекції. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 7

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі прізвище, ім'я і посаду працівників компанії. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 8

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву країни, її столиці і кількість мешканців. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 9

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву університету, місто, в якому він знаходиться і кількість студентів, які в ньому навчаються. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 10

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву посібника з метрології, прізвище його автора і рік видання. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 11

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву мови програмування, прізвище її розробника і рік розробки. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 12

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву періодичного видання зі 152 спеціальності, категорію та країну видання. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 13

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву конкурсу студентських наукових робіт, країну і періодичність проведення. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 14

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву конференції з інформаційно-вимірювальної техніки, країну та періодичність проведення. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 15

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву автівки, тип двигуна і рік виготовлення. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 16

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву зброї, призначення і назву компанії-виробника. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 17

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву картини, прізвище художника, рік створення картини. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 18

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі назву теореми, прізвище її автора і рік створення. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 19

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі тип мобільного телефону, назву компанії-виробника і рік виготовлення. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Варіант 20

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію користувача, яка містить в собі тип жорсткого диску, назву компанії-виробника і обсяг пам'яті. Мінімально необхідний інтерфейс взаємодії з екземпляром повинен включати метод додавання елемента, індексатор для отримання значення елемента за вказаним індексом і властивість тільки для читання для отримання загальної кількості елементів. Реалізувати можливість перебору елементів нової колекції у циклі foreach.

Контрольні питання

1. Що таке колекція?
2. Перерахуйте члени інтерфейсу IEnumerable.
3. Перерахуйте члени інтерфейсу IEnumerator.
4. Що таке оператор yield?
5. Перерахуйте члени інтерфейсу IEnumerable<T>.
6. Перерахуйте члени інтерфейсу IEnumerator<T>.
7. Опишіть алгоритм роботи циклу foreach.
8. Чи можна використовувати оператор yield в анонімних методах?

Комп'ютерний практикум №9

LINQ

Мета роботи: розгляд поняття анонімних і динамічних типів, знайомство з особливостями роботи в LINQ [1-6, 43-46].

Теоретичні відомості

- Анонімні типи пропонують зручний спосіб інкапсуляції набору властивостей в один об'єкт без необхідності попереднього явного визначення типу.
- Ім'я типу створюється компілятором та недоступно на рівні вихідного коду.
- Анонімні типи є типами-посиланнями, які походять безпосередньо від класу object.
- В анонімних типах можуть бути вкладені анонімні типи.
- Анонімні типи зазвичай використовуються в реченні select виразу запиту для повернення підмножини властивостей кожного об'єкта у вихідній послідовності.
- **Language Integrated Query** (LINQ) – проєкт Microsoft по додаванню синтаксиса мови запитів, що нагадує SQL, в мови програмування платформи .NET Framework.
- LINQ – надає стандартні, легко вивчаємі шаблони для створення запитів та оновлення даних; технологія може бути розширена для підтримки потенційно будь-якого типу сховища даних.
- LINQ є революційною іновацією в .NET Framework, яка є мостом між світом об'єктів та світом даних.
- Всі операції запиту LINQ складаються з трьох різних дій:
  - отримання джерела даних;
  - створення запиту;
  - виконання запиту.
- Вираз запиту має починатися реченням from і закінчуватися реченням select або group.
- Вираз запиту може включати вкладені запити, що починаються з from.
- Ключове слово from визначає наступне:
  - джерело даних, стосовно якого запит або вкладений запит буде виконуватись;
  - локальну змінну діапазону, що представляє кожен елемент вихідної послідовності.
- where – використовується у виразі запиту для вказування елементів, що повертаються з джерела даних, у виразі запиту. Воно застосовує логічну умову (предикат) до кожного вихідного елементу (з посиланням на

змінну діапазону) і повертає елементи, для яких задана умова є істинною. В одному виразі запиту може бути кілька where, а в одному where – кілька частин виразу предикату.

- У виразі запиту select задає тип значень, які отримуються під час виконання запиту. Результат ґрунтується на аналізі всіх попередніх запитів та на будь-яких виразах всередині речення select. Вираз запиту має завершуватися select або group.
- group – повертає послідовність об'єктів IGrouping<TKey, TElement>, що містять нуль або більше елементів, які відповідають значенню ключа групи.
- Щоб виконати додаткові запити для кожної з груп, можна вказати тимчасовий ідентифікатор, скориставшись для цього контекстним ключовим словом into. При використанні ключового слова into необхідно продовжити запит і завершити його інструкцією select або group.
- orderby –·здійснює сортування поверненої послідовності за зростанням або за спаданням у виразі запиту. Для виконання однієї або кількох операцій наступного сортування можна вказати кілька ключів. Сортування виконується функцією порівняння за замовчуванням для цього типу елементів. За замовчуванням використовується порядок сортування за зростанням. Можна також вказати функцію користувача за замовчуванням.
- join – використовується для зв'язування елементів з різних послідовностей джерел, які не мають прямих зв'язків в об'єктній моделі. Єдина вимога – елементи у кожному джерелі повинні спільно використовувати деяке значення, яке можна порівняти щодо рівності.
- join в якості вводу приймає дві послідовності джерел. Елементи в кожній послідовності мають бути властивістю або містити властивість, яку можна порівняти з відповідною властивістю в іншій послідовності. join порівнює зазначені ключі щодо рівності за допомогою спеціального ключового слова equals.
- Будь-який LINQ-запит трансформується у послідовність викликів розширяючих методів.
- let – представляє новий локальний ідентифікатор, на який можна посилатися в іншій частині запиту. Його можна надати, як локальну змінну, видиму лише всередині виразу запиту.
- dynamic – це тип даних, який є статичним, але об'єкт типу dynamic обходить перевірку статичного типу. Найчастіше він функціонує, як тип object. Під час компіляції передбачається, що елементи з типом dynamic підтримують будь-які операції. Тому розробнику не потрібно стежити, звідки об'єкт отримує своє значення.
- На відміну від ключового слова var, об'єкт, оголошений як dynamic, може змінювати тип під час виконання. У разі використання ключового слова var визначення типу об'єкта відкладається. Але як тільки його

визначено компілятором, змінювати його вже не можна. Що ж до об'єкта dynamic, можна не просто змінити його тип, але робити це багаторазово. Це відрізняється від приведення об'єкта від одного типу до іншого. При приведенні об'єкта створюється новий об'єкт з іншим, але сумісним типом.

- «Події» не можуть бути типом dynamic.

Практичні приклади

Приклад 1 Anonymous

```
using System;

// Анонімні типи.

// Анонімні типи надають зручний спосіб інкапсуляції набору властивостей в один об’єкт без необхідності попереднього явного визначення типу.
// Ім’я типу створюється компілятором і недоступно на рівні вихідного кода.
// Анонімні типи є типами-посиланнями, які походять безпосередньо від класу object.
// Компілятор присвоює їм імена, недивлячись на те, що ці імена недоступні для додатку.

namespace Anonymous
{

    class Program
    {
        static void Main()
        {
            var instance = new  { Name = "Alex", Age = 27 };
            // instance.Name = “xxx”;

            Console.WriteLine("Name = {0}, Age = {1}", instance.Name, instance.Age);

            Type type = instance.GetType();

            Console.WriteLine(type.ToString());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 2 Anonymous

```
using System;

// Анонімні типи.

namespace Anonymous
{
    class Program
    {
        static void Main()
        {
            // Анонімні типи в анонімних типах.

            var instance = new { Name = "Alex", Age = 27, Id = new { Number = 123 } };

            Console.WriteLine("Name = {0}, Age = {1}, Id = {2}", instance.Name,
instance.Age, instance.Id.Number);


            // Delay.
            Console.ReadKey();
```

```
        }
    }
}
```

Приклад 3 Anonymous

```
using System;

// Анонімні типи. (Сильне посилання)

namespace Anonymous
{
    class MyClass
    {
        public int field;

        public void Method()
        {
            Console.WriteLine(field);
        }
    }

    class Program
    {
        static void Main()
        {
            var instance = new { My = new MyClass() };

            instance.My.field = 1;
            instance.My.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 4 Anonymous

```
using System;
// Анонімні типи. (Слабке посилання)

namespace Anonymous
{
    class MyClass
    {
        public int field;

        public void Method()
        {
            Console.WriteLine(field);
        }
    }

    class Program
    {
        static void Main()
        {
            new
            {

                My = new MyClass { field = 1 }

            }.My.Method();

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 5 Anonymous

```
using System;

// Анонімні типи.

namespace Anonymous
{
    delegate void MyDelegate(string @string);

    class Program
    {
        static void Main()
        {
            var instance = new
            {
                MyDel = new MyDelegate((string @string) => Console.WriteLine(@string))
            };

            instance.MyDel.Invoke("Hello world!");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 6 LINQ

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

namespace LINQ
{
    public class Employee
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public decimal Salary { get; set; }
        public DateTime StartDate { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // База даних співробітників.
            var employees = new List<Employee>
            {
                new Employee
                {
                    FirstName = "Ivan",
                    LastName = "Ivanov",
                    Salary = 94000,
                    StartDate = DateTime.Parse("1/4/1992")
                },
                new Employee
                {
                    FirstName = "Petro",
                    LastName = "Petrov",
                    Salary = 123000,
                    StartDate = DateTime.Parse("12/3/1985")
                },
                new Employee
                {
                    FirstName = "Andrii",
                    LastName = "Andriiv",
                    Salary = 1000000,
                    StartDate = DateTime.Parse("1/12/2005")
```

```
                }
            };

            // Вираз запиту.
            var query = // query - змінна запиту.
                        from employee in employees // from - оголошує змінну діапазона
employee.
                        where employee.Salary > 100000 // where - фільтр
                        orderby employee.LastName, employee.FirstName
                        select new // select - операція проекції.
                        {
                            LastName = employee.LastName,
                            FirstName = employee.FirstName
                        };

            Console.WriteLine("Співробітники з високою зарплатнею: ");

            foreach (var item in query)
                Console.WriteLine("{0} {1}", item.LastName, item.FirstName);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 7 LINQ

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

namespace LINQ
{
    public class Employee
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public decimal Salary { get; set; }
        public DateTime StartDate { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // База даних співробітників.
            var employees = new List<Employee>
            {
                new Employee
                {
                    FirstName = "Ivan",
                    LastName = "Ivanov",
                    Salary = 94000,
                    StartDate = DateTime.Parse("1/4/1992")
                },
                new Employee
                {
                    FirstName = "Petro",
                    LastName = "Petrov",
                    Salary = 123000,
                    StartDate = DateTime.Parse("12/3/1985")
                },
                new Employee
                {
                    FirstName = "Andrii",
                    LastName = "Andriiv",
                    Salary = 1000000,
                    StartDate = DateTime.Parse("1/12/2005")
                }
```

```
            };

            // Вираз запиту. (використання викликів розширюючих методів.)
            var query = // query - змінна запиту.
                        employees
                        .Where(emp => emp.Salary > 100000)
                        .OrderBy(emp => emp.LastName)
                        .OrderBy(emp => emp.FirstName)
                        .Select(emp => new
                        {
                            LastName = emp.LastName,
                            FirstName = emp.FirstName
                        });

            Console.WriteLine("Співробітники з високою зарплатнею: ");

            foreach (var item in query)
                Console.WriteLine("{0} {1}", item.LastName, item.FirstName);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 8 LINQ

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

namespace LINQ
{
    public class Employee
    {
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public decimal Salary { get; set; }
        public DateTime StartDate { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // База даних співробітників.
            var employees = new List<Employee>
            {
                new Employee
                {
                    FirstName = "Ivan",
                    LastName = "Ivanov",
                    Salary = 94000,
                    StartDate = DateTime.Parse("1/4/1992")
                },
                new Employee
                {
                    FirstName = "Petro",
                    LastName = "Petrov",
                    Salary = 123000,
                    StartDate = DateTime.Parse("12/3/1985")
                },
                new Employee
                {
                    FirstName = "Andrii",
                    LastName = "Andriiv",
                    Salary = 1000000,
                    StartDate = DateTime.Parse("1/12/2005")
                }
            };
```

```
            // Вираз запиту.(Використання викликів статичних методів.)
            var query = // query - змінна запиту.
                Enumerable.Select(
                Enumerable.OrderBy(
                Enumerable.OrderBy(
                Enumerable.Where(
                employees, emp => emp.Salary > 100000),
                emp => emp.LastName),
                emp => emp.FirstName),
                emp => new { LastName = emp.LastName, FirstName = emp.FirstName });

            Console.WriteLine("Співробітники з високою зарплатнею: ");

            foreach (var item in query)
                Console.WriteLine("{0} {1}", item.LastName, item.FirstName);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 9 LINQ

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

// Власна реалізація where та select.

namespace LINQ
{
    public static class MySet
    {
        public static IEnumerable<T> Where<T>(this IEnumerable<T> source, Func<T,
bool> predicate)
        {
            Console.WriteLine("Викликана власна реалізація Where.");
            return System.Linq.Enumerable.Where(source, predicate);
        }

        public static IEnumerable<R> Select<T, R>(this IEnumerable<T> source, Func<T,
R> selector)
        {
            Console.WriteLine("Викликана власна реалізація Select.");
            return System.Linq.Enumerable.Select(source, selector);
        }
    }

    class Program
    {
        static void Main()
        {
            int[] numbers = { 1, 2, 3, 4 };

            var query = from x in numbers
                        where x % 2 == 0
                        select x * 2;

            foreach (var item in query)
                Console.WriteLine(item);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 10 LINQ

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

//Таблиця множення.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            // Конструкція from схожа на оператор foreach.
            // Використання декількох конструкцій from, аналогічно вкладеним
операторам foreach.

            var query = from x in Enumerable.Range(1, 9) // Таблиця множення від 1 до
9.
                        from y in Enumerable.Range(1, 10)
                        select new
                        {
                            X = x,
                            Y = y,
                            Product = x * y
                        };

            foreach (var item in query)
                Console.WriteLine("{0} * {1} = {2}", item.X, item.Y, item.Product);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 11 LINQ

```
using System;
using System.Collections;
using System.Linq;

// Застосування запиту до колекції, яка підтримує IEnumerable не параметризований.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            ArrayList numbers = new ArrayList();
            numbers.Add(1);
            numbers.Add(2);

            // ЯВНЕ вказування типу Int32 змінної діапазону - n. (var - НЕМОЖЛИВО
використовувати, тому що IEnumerable не параметризований!)
            var query = from int n in numbers
                        select n * 2;

            foreach (var item in query)
                Console.WriteLine(item);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 12 LINQ

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

// join (операція об'єднання) - встановлює співвідношення даних із двох різних джерел.

namespace LINQ
{
    public class EmployeeID
    {
        public string Id { get; set; }
        public string Name { get; set; }
    }

    public class EmployeeNationality
    {
        public string Id { get; set; }
        public string Nationality { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // Побудувати колекцію співробітників.
            var employees = new List<EmployeeID>
            {
                new EmployeeID {Id = "111", Name = "Ivan Ivanov"},
                new EmployeeID {Id = "222", Name = "Andrii Andriiv"},
                new EmployeeID {Id = "333", Name = "Petro Petrov"},
                new EmployeeID {Id = "444", Name = "Alex Olexiiv"}
            };

            // Побудувати колекцію національностей.
            var empNationalities = new List<EmployeeNationality>
            {
                new EmployeeNationality {Id = "111", Nationality = "Pole"},
                new EmployeeNationality {Id = "222", Nationality = "Ukrainian"},
                new EmployeeNationality {Id = "333", Nationality = "American"},
            };

            // Побудувати запит.
            // Отримання списку імен усіх співробітників разом з їх національностями,
при цьому відсортувавши список за спаданням.
            var query = from emp in employees
                        join n in empNationalities
                        on emp.Id equals n.Id
                        orderby n.Nationality descending // ascending - за зростанням
| descending - за спаданням.
                        select new
                        {
                            Id = emp.Id,
                            Name = emp.Name,
                            Nationality = n.Nationality
                        };

            foreach (var person in query)
                Console.WriteLine("{0}, {1}, \t{2}", person.Id, person.Name,
person.Nationality);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 13 LINQ

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

// orderby - використовується для сортування (послідовності) результату запиту.

namespace LINQ
{
    public class Employee
    {
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string Nationality { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // Побудувати колекцію співробітників з національностями.
            var employees = new List<Employee>
            {
                new Employee {LastName = "Ivanov", FirstName = "Ivan", Nationality =
"Pole"},
                new Employee {LastName = "Andriiv", FirstName = "Andrii", Nationality
= "Ukrainian"},
                new Employee {LastName = "Petrov", FirstName = "Petro", Nationality =
"American"}
            };

            // Побудувати запит.
            // Отримання списку імен усіх співробітників разом із їхніми
національностями.
            var query = from emp in employees
                        orderby emp.Nationality ascending,
                        emp.LastName descending, // ascending - за зростанням |
descending - за спаданням.
                        emp.FirstName descending // за замовчуванням - ascending.
                        select emp;

            foreach (var person in query)
                Console.WriteLine("{0}, \t{1}, \t{2}", person.LastName,
person.FirstName, person.Nationality);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 14 LINQ

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

// select - (операція проєкції) використовується для побудови кінцевого результату
запиту.

namespace LINQ
{
    public class Result
    {
        public Result(int input, int output)
        {
            Input = input;
            Output = output;
        }
```

```
        public int Input { get; set; }
        public int Output { get; set; }
    }

    class Program
    {
        static void Main()
        {
            int[] numbers = { 1, 2, 3, 4 };

            // Побудувати запит.
            var query = from x in numbers
                        select new Result(x, x * 2);

            numbers[0] = 777; // Вираз запиту виконується в момент звернення до
змінної запиту в foreach.

            foreach (var item in query)
                Console.WriteLine("Input = {0}, \t Output = {1}", item.Input,
item.Output);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 15 LINQ

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

// select - (операція проєкції) використовується для побудови кінцевого результату
запиту.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            int[] numbers = { 1, 2, 3, 4 };

            // Побудувати запит.
            var query = from x in numbers
                        select new { Input = x, Output = x * 2 };

            foreach (var item in query)
                Console.WriteLine("Input = {0}, \t Output = {1}", item.Input,
item.Output);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 16 LINQ

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

// let - представляє новий локальний ідентифікатор, на який можна посилатися в іншій
частині запиту.
// Його можна надати, як локальну змінну видиму лише всередині виразу запиту.

namespace LINQ
{
```

```
    public class Employee
    {
        public string LastName { get; set; }
        public string FirstName { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // Побудувати колекцію співробітників.
            var employees = new List<Employee>
            {
                new Employee {LastName = "Ivanov", FirstName = "Ivan"},
                new Employee {LastName = "Andreev", FirstName = "Andrew"},
                new Employee {LastName = "Petrov", FirstName = "Petr"}
            };

            // Побудувати запит.
            var query = from emp in employees
                        let fullName = emp.FirstName + " " + emp.LastName // let -
новий локальний ідентифікатор.
                        orderby fullName descending
                        select fullName;

            foreach (var person in query)
                Console.WriteLine(person);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 17 LINQ

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

// let - представляє новий локальний ідентифікатор, на який можна посилатися в іншій
частині запиту.
// Його можна надати, як локальну змінну видиму лише всередині виразу запиту.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            // Побудувати запит.
            var query = from x in Enumerable.Range(0, 10)
                        let innerRange = Enumerable.Range(0, 10)
                        from y in innerRange
                        select new { X = x, Y = y, Product = x * y };

            foreach (var item in query)
                Console.WriteLine("{0} * {1} = {2}", item.X, item.Y, item.Product);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 18 LINQ

```
using System;
using System.Linq;
```

```
// group - є засобом для поділу введення запиту.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

            // Побудувати запит.
            // Розділення чисел на парні та непарні.
            var query = from x in numbers
                        group x by x % 3;

            foreach (var group in query)
            {
                Console.WriteLine("mod{0} == {0}", group.Key);

                foreach (var number in group)
                    Console.WriteLine("{0}, ", number);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 19 LINQ

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

namespace LINQ
{
    public class Employee
    {
        public string LastName { get; set; }
        public string FirstName { get; set; }
        public string Nationality { get; set; }
    }

    class Program
    {
        static void Main()
        {
            // Побудувати колекцію співробітників з національностями.
            var employees = new List<Employee>
            {
                new Employee {LastName = "Andriiv", FirstName = "Andrii",  Nationality
= "Ukrainian"},
                new Employee {LastName = "Ivanov",  FirstName = "Ivan",    Nationality
= "Pole"},
                new Employee {LastName = "Andriiv", FirstName = "Ivan",  Nationality =
"Ukrainian"},
                new Employee {LastName = "Petrov",  FirstName = "Petro",
Nationality = "American"},
                new Employee {LastName = "Andriiv", FirstName = "Sergii",  Nationality
= "Ukrainian"},
                new Employee {LastName = "Petrov",  FirstName = "Slava",
Nationality = "American"}
            };

            // Побудувати запит.
            var query = from emp in employees
                        group emp by new { Nationality = emp.Nationality, LastName =
emp.LastName };
```

```
            foreach (var group in query)
            {
                Console.WriteLine(group.Key);

                foreach (var employee in group)
                    Console.WriteLine(employee.FirstName);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 20 LINQ

```
using System;
using System.Linq;

// into - подібно до let, дозволяє визначити локальний по відношенню до запиту
ідентифікатор.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
            int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

            // Побудувати запит.
            // Розділення чисел на парні та непарні.
            var query = from x in numbers
                        group x by x % 2 into partition
                        where partition.Key == 0
                        select new { Key = partition.Key, Count = partition.Count(),
Group = partition };


            foreach (var item in query)
            {
                Console.WriteLine("mod2 == {0}", item.Key);
                Console.WriteLine("Count == {0}", item.Count);

                foreach (var number in item.Group)
                    Console.Write("{0}, ", number);

                Console.WriteLine();
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 21 LINQ

```
using System;
using System.Linq;

// into - подібно до let, дозволяє визначити локальний по відношенню до запиту
ідентифікатор.

namespace LINQ
{
    class Program
    {
        static void Main()
        {
```

```
            int[] numbers = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

            // Побудувати запит.
            // Розділення чисел на парні та непарні.
            var query = from x in numbers
                        group x by x % 2 into partition
                        select new { Key = partition.Key, Count = partition.Count(),
Group = partition };

            foreach (var item in query)
            {
                Console.WriteLine("mod2 == {0}", item.Key);
                Console.WriteLine("Count == {0}", item.Count);

                foreach (var number in item.Group)
                    Console.Write("{0}, ", number);

                Console.WriteLine("\n");
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 22 Dynamic

```
using System;

// Динамічні типи даних. (Локальні змінні)

namespace Dynamic
{
    class Program
    {
        static void Main()
        {
            dynamic variable = 1;

            Console.WriteLine(variable);

            variable = "Hello world!";

            Console.WriteLine(variable);

            variable = DateTime.Now;

            Console.WriteLine(variable);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 23 Dynamic

```
using System;

// Динамічні типи даних. (Статичні поля)

namespace Dynamic
{
    class Program
    {
        static dynamic field = 1;

        static void Main()
        {
```

```
            Console.WriteLine(field);

            field = "Hello world!";

            Console.WriteLine(field);

            field = DateTime.Now;

            Console.WriteLine(field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 24 Dynamic

```
using System;

// Динамічні типи даних.  (Нестатичні поля)

namespace Dynamic
{
    class Program
    {
        dynamic field = 1, field2 = "Hello", field3 = true;

        static void Main()
        {
            dynamic instance = new Program();

            Console.WriteLine(instance.field);

            instance.field = "Hello world!";

            Console.WriteLine(instance.field);

            instance.field = DateTime.Now;

            Console.WriteLine(instance.field);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 25 Dynamic

```
using System;

// Динамічні типи даних.

namespace Dynamic
{
    class Program
    {
        static void Main()
        {
            for (dynamic i = 0; i < 10; i++)
            {
                Console.WriteLine("Hello world!");
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 26 Dynamic

```
using System;

// Динамічні типи даних. (Динамічні типи аргументів та повертаємих значень методів.)

namespace Dynamic
{
    class Program
    {
        static dynamic Method(dynamic argument)
        {
            return "Hello " + argument + "!";
        }

        static void Main()
        {
            string @string = Method("friend");

            Console.WriteLine(@string);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 27 Dynamic

```
using System;

// Динамічні типи даних.

namespace Dynamic
{
    class MyClass
    {
        dynamic field;

        public MyClass(dynamic argument)
        {
            field = argument;
        }

        // Динамічні властивості.

        public dynamic MyAutoProperty { get; set; }

        public dynamic Field
        {
            get { return field; }
            set { field = value; }
        }

        // Динамічні методи.

        public dynamic Method(dynamic argument)
        {
            return argument;
        }

        // Динамічні масиви і індексатори.

        dynamic[] array = new dynamic[3];

        public dynamic this[dynamic index]
        {
            get { return array[index]; }
            set { array[index] = value; }
        }
```

```
    }

    class Program
    {
        static void Main()
        {
            dynamic my = new MyClass("Hello");

            Console.WriteLine(my.Field);

            string variable = my.Method("World");

            Console.WriteLine(variable);

            my[0] = "Zero";
            my[1] = "One";
            my[2] = "Two";

            for (dynamic i = 0; i < 3; i++)
            {
                Console.WriteLine(my[i]);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 28 Dynamic

```
using System;

// Динамічні типи даних. (Динамічні типи в делегатах)

namespace Dynamic
{
    delegate dynamic MyDelegate(dynamic argument);

    class Program
    {
        static dynamic Method(dynamic argument)
        {
            return argument;
        }

        static void Main()
        {
            dynamic myDelegate = new MyDelegate(Method);

            dynamic @string = myDelegate("Hello world!");

            Console.WriteLine(@string);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 29 Dynamic

```
using System;

// Динамічні типи даних. (Динамічні типи в параметризованих делегатах)

namespace Dynamic
{
    delegate R MyDelegate<T, R>(T argument);
```

```
    class Program
    {
        static dynamic Method(dynamic argument)
        {
            return argument;
        }

        static void Main()
        {
            dynamic myDelegate = new MyDelegate<dynamic, dynamic>(Method);

            dynamic @string = myDelegate("Hello world!");

            Console.WriteLine(@string);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 30 Dynamic

```
using System;

// Динамічні типи даних. (Події)

namespace Dynamic
{
    delegate dynamic MyDelegate(dynamic sender, dynamic e);

    class MyClass
    {
        dynamic myEvent;

        public event MyDelegate MyEvent
        {
            add { myEvent += value; }
            remove { myEvent -= value; }
        }

        public dynamic Method(dynamic sender, dynamic e)
        {
            myEvent.Invoke(sender, e);
            return default(dynamic);
        }
    }

    class Program
    {
        static dynamic Handler(dynamic sender, dynamic e)
        {
            Console.WriteLine("sender = {0}, e = {1}", sender, e);
            return default(dynamic);
        }

        static void Main()
        {
            dynamic my = new MyClass();
            my.MyEvent += new MyDelegate(Handler);
            //Console.WriteLine();
            my.Method("user", "mouse");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 31 Dynamic

```
using System;

// Динамічні типи даних. (Події)

namespace Dynamic
{
    delegate dynamic MyDelegate(dynamic sender, dynamic e);

    class MyClass
    {
        public dynamic MyEvent = default(dynamic); // Не є подією.

        public dynamic Method(dynamic sender, dynamic e)
        {
            MyEvent.Invoke(sender, e);
            return default(dynamic);
        }
    }

    class Program
    {
        static dynamic Handler(dynamic sender, dynamic e)
        {
            Console.WriteLine("sender = {0}, e = {1}", sender, e);
            return default(dynamic);
        }

        static void Main()
        {
            dynamic my = new MyClass();
            my.MyEvent += new MyDelegate(Handler);

            my.Method("user", "mouse");

            my.MyEvent.Invoke("user", "mouse");

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 32 Dynamic

```
using System;

// Динамічні типи даних. (Наслідування)

namespace Dynamic
{
    class Base
    {
        public dynamic field;

        public dynamic Method()
        {
            return this.field;
        }
    }

    class Derived : Base
    {

    }

    class Program
    {
        static void Main()
```

```
        {
            dynamic instance = new Derived();

            // Динамічні поля повинні бути проініціалізовані перед використанням.
            instance.field = "Hello";

            Console.WriteLine(instance.Method());

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 33 Dynamic

```
using System;

// Динамічні типи даних.

namespace Dynamic
{
    class MyClass
    {

    }

    class Program
    {
        static dynamic FactoryMethod()
        {
            return new MyClass();
        }

        static void Main()
        {
            dynamic instance = FactoryMethod() as dynamic;

            if (false)
            {
                instance.Method(7);                   // Виклик методу.
                instance.field = instance.Property;   // Отримання та встановлення
значень полів та властивостей.
                instance["one"] = instance[2];        // Отримання та встановлення
значень через індексатори.
                dynamic variable = instance + 3;      // Виклик операторів.
                variable = instance(5, 7);            // Виклик делегатів.
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 34 Dynamic

```
using System;

// Динамічні типи даних. (Анонімні типи)

namespace Dynamic
{
    class Program
    {
        static void Main()
        {
            dynamic instance = new { Name = "Alex", Age = 18 };

            Console.WriteLine(instance.Name);
```

```
            Console.WriteLine(instance.Age);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 35 Dynamic

```
using System;

// Динамічні типи даних. (Динамічні типи аргументів і повертаємих значень методів.)

namespace Dynamic
{
    class Program
    {
        static dynamic Method(ref dynamic argument1, out dynamic argument2)
        {
            argument1 = ++argument1;
            argument2 = 2;

            return default(dynamic);
        }

        static void Main()
        {
            dynamic variable1 = 0, variable2;

            Method(ref variable1, out variable2);

            Console.WriteLine(variable1);
            Console.WriteLine(variable2);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 36 Dynamic

```
using System;

// Динамічні типи даних.

namespace Dynamic
{
    struct Point
    {
        dynamic x, y;

        public Point(dynamic x, dynamic y)
        {
            this.x = x;
            this.y = y;
        }

        // Один з параметрів бінарного оператора повинен мати існуючий тип.

        //public static dynamic operator +(dynamic pointA, dynamic pointB) - так
неприпустимо.

        public static dynamic operator +(Point pointA, dynamic pointB)
        {
            return new Point(pointA.x + pointB.x, pointA.y + pointB.y);
        }

        // В унарних операторах неприпустимо використовувати динамічні типи (загалом).
```

```
        // public static dynamic operator ++(dynamic pointA) - так неприпустимо.

        public static Point operator ++(Point pointA)
        {
            return new Point(pointA.x + 1, pointA.y +1);
        }

        public override string ToString()
        {
            return string.Format("x = {0}, y = {1}", x, y);
        }
    }

    class Program
    {
        static void Main()
        {
            dynamic a = new Point(1, 1), b = new Point(2, 2), c = a + b;

            Console.WriteLine(c);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 37 Dynamic

```
using System;

// Динамічні типи даних.

namespace Dynamic
{
    class Calculator
    {
        public dynamic Add(dynamic a, dynamic b)
        {
            return a + b;
        }
    }

    class Program
    {
        static void Main()
        {
            dynamic calculator = new Calculator();

            Console.WriteLine(calculator.Add(2, 3));
            Console.WriteLine(calculator.Add("1", 2));

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Приклад 38 Dynamic

```
using System;
using System.Collections;

// Динамічні типи даних. (Анонімні типи)

namespace Dynamic
{
    class UserCollection
    {
```

```
        public static IEnumerable Generator()
        {
            yield return new { Key = 0, Value = "Zero" };
            yield return new { Key = 1, Value = "One" };
            yield return new { Key = 2, Value = "Two" };
        }
    }

    class Program
    {
        static void Main()
        {
            foreach (dynamic item in UserCollection.Generator())
            {
                Console.WriteLine("Key = {0}, Value = {1}", item.Key, item.Value);
            }

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Завдання для виконання

Загальні завдання

Завдання 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Уявіть, що ви пишете додаток для Автостанції та вам необхідно створити просту колекцію автомобілів з наступними даними: марка автомобіля, модель, рік випуску, колір. А також другу колекцію з моделлю автомобіля, ім'ям покупця і його номером телефону. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про покупця одного з автомобілів і повну характеристику придбаної ними моделі.

Завдання 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Використовуючи динамічні і анонімні типи даних, створіть Англо-Український словник на 10 слів і виведіть на екран його вміст.

Завдання 3

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.

Створіть клас Calculator, методи якого приймають аргументи і повертають значення типу dynamic.

Індивідуальні завдання

Варіант 1

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Читачі. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про читачів, номер телефону яких починається з 067.

Варіант 2

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Викладачі. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію по викладачів, які працюють на посаді асистента.

Варіант 3

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Датчики. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про датчики, ціна яких не перевищує 4000 грн.

Варіант 4

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Автівки. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про легкові автівки з пробігом до 10000 км.

Варіант 5

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Дисципліни. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про дисципліни, які викладаються тричі на тиждень.

Варіант 6

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Проєкти. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про проєкти, які розпочалися у 2019 році.

Варіант 7

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Лікарі. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію, про лікарів-офтальмологів.

Варіант 8

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application.

Створити колекцію Оператори зв'язку. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про кількість абонентів, підключених до «Київстар» у 2021 році.

Варіант 9

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Спортсмени. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про спортсменів, які одержали нагороди на зимових олімпійських іграх.

Варіант 10

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Магазини. Використовуючи найпростіший

LINQ запит, необхідно вивести на екран інформацію продукцію, ціна закупівлі якої не перевищує 500 грн.

Варіант 11

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Вид транспорту. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про міський транспорт, що здійснює перевезення пасажирів за маршрутом Нивки-Оболонь.

Варіант 12

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Ліки. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про ліки, ціна яких перевищує 1000 грн.

Варіант 13

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Населення. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про країни, чисельність населення яких не перевищує 50 млн.

Варіант 14

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Літаки. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про маршрути літаків компанії Boeing.

Варіант 15

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Ноутбуки. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про ноутбуки, ціна яких перевищує 60000 грн.

Варіант 16

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Факультети КПІ ім. Ігоря Сікорського. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про факультети з найбільшою кількістю вступників на перший курс.

Варіант 17

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Тварини. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про тварин з найдовшою тривалістю життя.

Варіант 18

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Шахові турніри. Використовуючи

найпростіший LINQ запит, необхідно вивести на екран інформацію про шахові турніри, що проводились в Канаді у 2021 році.

Варіант 19

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Кораблі. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про кораблі, що курсують Індійським океаном.

Варіант 20

Використовуючи Visual Studio, створіть проєкт за шаблоном Console Application. Створити колекцію Вчені. Використовуючи найпростіший LINQ запит, необхідно вивести на екран інформацію про вчених-лауреатів Нобелівської премії з фізики.

Контрольні питання

1. Що таке анонімні типи?
2. Що таке LINQ?
3. Що таке вираз запиту?
4. Що таке змінна запиту?
5. Що таке змінна діапазону?
6. Що таке динамічний тип?
7. Що робить оператор dynamic?

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Troelsen Andrew. With .NET and .NET Core. Berkeley, CA : Apress : Imprint : Apress, 2017. 1372 p. Online resource: https://opac.kpi.ua/F/?func=direct&doc_number=000627408&local_base=KPI01

2. Основи програмування мовою C# : навч. посіб. / Л.В. Соловей та ін. Харків: НТУ "ХПІ", 2019. 487 с. Онлайн ресурс: https://opac.kpi.ua/F/?func=direct&doc_number=000604001&local_base=KPI01

3. Данилець Є. В., Райко Г. О. Візуальне програмування мовою C# для економістів : навч. посіб. Херсон : Вишемирський В. С., 2019. 276 с. Онлайн ресурс: https://opac.kpi.ua/F/?func=direct&doc_number=000604910&local_base=KPI01

4. Deitel Paul J., Deitel Harvey. C# 6 for Programmers. Pearson, 2016. 768 p.

5. Troelsen Andrew, Japikse Philip. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11st ed. Edition – Apress, 2022. 1705 p.

6. Beginning Visual C# 2010 / Karli Watson, Christian Nagel, Jacob Hammer Pedersen, Jon Reid, Morgan Skinner. Wiley Publishing, 2010. 1037 p.

7. Добролюбова М. В., Маркін М. О., Філіппова М. В. Обчислювальна техніка та програмування. Процедурна складова мови C# : навч. посіб. Київ : КПІ ім. Ігоря Сікорського, 2021. 154 с. https://ela.kpi.ua/handle/123456789/42415

8. Добролюбова М.В. Обчислювальна техніка, основи алгоритмізації та програмування [електронний ресурс]: метод. вказівки до курсу комп. практикумів з кредитного модуля «ОТтаП - 1» для бакалаврів спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» спеціалізацій «Інформаційні вимірювальні технології та системи» і «Метрологія та вимірювальна техніка». Київ : НТУУ «КПІ», 2017. 108 с.:

Гриф «Рекомендовано ВР ФАКС НТУУ «КПІ»», протокол № 3 від 30 жовтня 2017 р.

9. Добролюбова М.В. Обчислювальна техніка, основи алгоритмізації та програмування [електронний ресурс]: метод. вказівки до курсу практичних робіт з кредитного модуля «ОТтаП - 1» для бакалаврів спеціальності 152 «Метрологія та інформаційно-вимірювальна техніка» спеціалізацій «Інформаційні вимірювальні технології та системи» і «Метрологія та вимірювальна техніка». Київ : НТУУ «КПІ», 2017. 203 с.: Гриф «Рекомендовано ВР ФАКС НТУУ «КПІ»», протокол № 11 від 26 червня 2017 р.

10. MSDN: Класи class (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/x9afc042.aspx (дата звернення: 02.01.2023).

11. MSDN: Об'єкти (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173110.aspx (дата звернення: 02.01.2023).

12. MSDN: Використання конструкторів (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173115(VS.90).aspx (дата звернення: 02.01.2023).

13. Автоматично реалізуємі властивості (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/bb384054.aspx (дата звернення: 02.01.2023).

14. MSDN: Часткові класи та методи (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/wa80x488.aspx (дата звернення: 02.01.2023).

15. MSDN: readonly (Довідник з C#). URL : http://msdn.microsoft.com/ru-ru/library/acdd6hb7.aspx (дата звернення: 02.01.2023).

16. MSDN: Поліморфізм. URL : http://msdn.microsoft.com/ru-ru/library/z165t2xk(v=VS.90).aspx (дата звернення: 03.01.2023).

17. MSDN: Модифікатори доступу (Довідник з C#). URL : http://msdn.microsoft.com/ru-ru/library/wxh6fsc7(v=VS.90).aspx (дата звернення: 03.01.2023).

18. MSDN: Наслідування (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173149.aspx (дата звернення: 03.01.2023).

19. MSDN: Абстрактні класи (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/sf985hc5.aspx (дата звернення: 04.01.2023).

20. MSDN: Інтерфейси (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173156.aspx (дата звернення: 04.01.2023).

21. MSDN: Інтерфейс interface (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/87d83y5b.aspx (дата звернення: 04.01.2023).

22. MSDN: Структури struct (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/saxz13w4.aspx (дата звернення: 05.01.2023).

23. MSDN: Класи та структури (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173109.aspx (дата звернення: 05.01.2023).

24. MSDN: Пакування-перетворення та розпакування-перетворення (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/yz2be5wk.aspx (дата звернення: 05.01.2023).

25. MSDN: Типи перелічень enum (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/cc138362.aspx (дата звернення: 05.01.2023).

26. MSDN: **DateTime**-структура. URL : http://msdn.microsoft.com/ru-ru/library/system.datetime.aspx (дата звернення: 05.01.2023).

27. MSDN: Делегати (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/ms173171.aspx (дата звернення: 06.01.2023).

28. MSDN: ключове слово delegate (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/900fyy8e(v=vs.90).aspx (дата звернення: 06.01.2023).

29. Анонімні методи (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/0yw3tz5k(v=vs.90).aspx (дата звернення: 06.01.2023).

30. MSDN: Універсальні шаблони (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/512aeb7t(v=VS.90).aspx (дата звернення: 07.01.2023).

31. MSDN: Ключове слово default (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/xwth0h0d(v=VS.90).aspx (дата звернення: 07.01.2023).

32. MSDN: Універсальні методи (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/twcad0zb(v=VS.90).aspx (дата звернення: 07.01.2023).

33. MSDN: Обмеження параметра типу. URL : http://msdn.microsoft.com/ru-ru/library/d5x73970(v=vs.90).aspx (дата звернення: 07.01.2023).

34. MSDN: Обмеження new. URL : http://msdn.microsoft.com/ru-ru/library/sd2w2ew5(v=vs.90).aspx (дата звернення: 07.01.2023).

35. MSDN: Потоки і робота з потоками. URL : http://msdn.microsoft.com/ru-ru/library/6kac2kdh.aspx (дата звернення: 08.01.2023).

36. MSDN: Клас Thread. URL : http://msdn.microsoft.com/ru-ru/library/system.threading.thread.aspx (дата звернення: 08.01.2023).

37. MSDN: Оператор params (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/c5kehkcz.aspx#Y491 (дата звернення: 08.01.2023).

38. MSDN: Визначення колекцій (Керівництво з програмування на C#). URL : https://msdn.microsoft.com/ru-ru/library/7y3x785f.aspx (дата звернення: 09.01.2023).

39. MSDN: Ключове слово yield (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/9k7k7cf0.aspx (дата звернення: 09.01.2023).

40. MSDN: Ітератори (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/dscyy5s0.aspx (дата звернення: 09.01.2023).

41. MSDN: Управління колекціями та їх створення. URL : http://msdn.microsoft.com/ru-ru/library/14ek9axh.aspx (дата звернення: 09.01.2023).

42. MSDN: Перелічення колекції. URL : http://msdn.microsoft.com/ru-ru/library/hwyysy67.aspx (дата звернення: 09.01.2023).

43. MSDN: Анонімні типи (Керівництво з програмування на C#). URL : http://msdn.microsoft.com/ru-ru/library/bb397696(v=VS.100).aspx (дата звернення: 10.01.2023).

44. MSDN: LINQ. URL : http://msdn.microsoft.com/ru-ru/library/bb397926(VS.90).aspx (дата звернення: 10.01.2023).

45. MSDN: dynamic (Довідник по C#). URL : http://msdn.microsoft.com/ru-ru/library/dd264741.aspx (дата звернення: 10.01.2023).

46. Діно Еспозито. Застосування ключового слова dynamic в C#. URL : http://msdn.microsoft.com/ru-ru/magazine/ee336309.aspx (дата звернення: 10.01.2023).

Додаток А

ЗРАЗОК ОФОРМЛЕННЯ РЕЗУЛЬТАТІВ ВИКОНАННЯ ІНДИВІДУАЛЬНОГО ЗАВДАННЯ

Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

кафедра інформаційно-вимірювальних технологій

Комп’ютерний практикум 1

на тему «Класи та об'єкти в мові C# »

з дисципліни «Технології програмування ІВС»

Виконав: студент групи ПВ-01

Соколюк Р.О.

Перевірила: Добролюбова М.В.

Київ -2023

Індивідуальне завдання:

Варіант 2

Використовуючи Visual Studio, створіть проект за шаблоном Console Application.
Необхідно:
Створити клас з ім'ям Rectangle.
В телі класу створити два поля, які описують довжини сторін double side1, side2.
Створити конструктор користувача Rectangle(double side1, double side2), в тілі якого поля side1 та side2 ініціалізуються значеннями аргументів.
Створити два методи, які обчислюють площу прямокутника – double AreaCalculator() та периметр прямокутника – double PerymeterCalculator().
Створити дві властивості double Area і double Perymeter з одним методом доступу get.
Написати програму, яка приймає від користувача довжини двох сторін прямокутника і виводить на екран периметр та площу.

Розв'язок завдання мовою C#:

Rectangle.cs

```
namespace Task_2
{
    class Rectangle
    {
        //Створення полів класу.
        double side1, side2;

        /// <summary>
        /// Побудова прямокутника.
        /// </summary>
        /// <param name="side1">Довжина першої сторони</param>
        /// <param name="side2">Довжина другої сторони</param>

        //Конструктор користувача
        public Rectangle(double side1, double side2)
        {
            this.side1 = side1;
            this.side2 = side2;
        }

        /// <summary>
        /// Розрахунок периметру.
        /// </summary>
        /// <returns>Периметр</returns>

        //Метод обчислення периметру прямокутника
        double PerimeterCalculator()
        {
            return (side1 + side2) * 2;
        }
        //Дана властивість лише для читання і завжди повертає результат виконання методу
PerimeterCalculator
        public double Perimeter
        {
            get
            {
                //Ключове слово this посилається на поточний екземпляр класу
                return this.PerimeterCalculator();
            }
        }

        /// <summary>
        /// Розрахунок площі
        /// </summary>
        /// <returns>Площа</returns>

        //Обчислення площі прямокутника
        double AreaCalculator()
        {
            return side1 * side2;
        }
```

```
        // Дана властивість лише для читання і завжди повертає результат виконання методу
AreaCalculator
        public double Area
        {
            get
            {
                //Ключове слово this посилається на поточний екземпляр класу
                return this.AreaCalculator();
            }
        }
    }
}
```

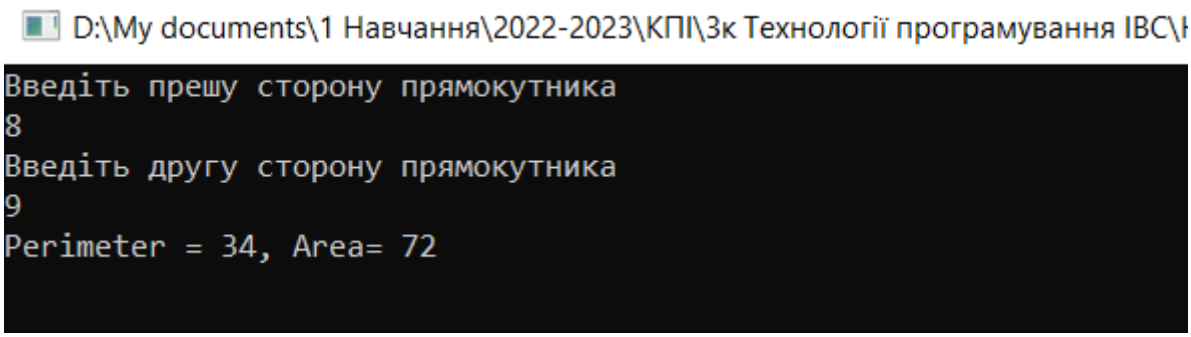
Program.cs

```
using System;

namespace Task_2
{
    class Program
    {
        static void Main()
        {
            Console.WriteLine("Введіть прешу сторону прямокутника");
            //Зчитування значень із стандартного вхідного потоку в змінну side
            double side1 = Convert.ToDouble(Console.ReadLine());

            Console.WriteLine("Введіть другу сторону прямокутника");
            //Зчитування значень із стандартного вхідного потоку в змінну side2
            double side2 = Convert.ToDouble(Console.ReadLine());

            //Створення екземпляру класу і передача в конструктор користувача двох аргументів
            Rectangle rectangle = new Rectangle(side1, side2);
            //Відображення периметру і площі прямокутника
            Console.WriteLine("Perimeter = {0}, Area= {1}", rectangle.Perimeter, rectangle.Area);

            // Delay.
            Console.ReadKey();
        }
    }
}
```

Результат виконання програми: