Лабораторна робота №6.

Тема : Успадкування. Інтерфейси. Програмування інтерфейсних методів.

Мета роботи:

Ознайомлення із програмуванням інтерфейсів. Застосування стандартних інтерфейсів .NET для порівняння, перебору, впорядкування та клонування об'єктів.

Теоретичні відомості

Інтерфейси

У мові C# для повного відділення структури класу від його реалізації використовується механізм інтерфейсів. Інтерфейс є розширенням ідеї абстрактних класів і методів. Синтаксис інтерфейсів подібний синтаксису абстрактних класів. Оголошення інтерфейсів виконується за допомогою ключового слова interface. При цьому методи в інтерфейсі не підтримують реалізацію.

Членами інтерфейсу можуть бути методи, властивості, індексатори й події. Інтерфейс може реалізовуватися довільною кількістю класів. Один клас, у свою чергу, може реалізовувати будь-яке число інтерфейсів. Кожний клас, що включає інтерфейс, повинен реалізовувати його методи. В інтерфейсі для методів неявним чином задається тип public. У цьому випадку також не допускається явний специфікатор доступу.

Іншими словами, інтерфейс визначає поведінку, яка підтримується, що реалізують цей інтерфейс класами. Основна ідея використання інтерфейсу полягає в тому, щоб до об'єктів таких класів можна було звертатися однаковим образом. Кожний клас може визначати елементи інтерфейсу по-своєму. Так досягається поліморфізм: об'єкти різних класів по-різному реагують на виклики того самого методу.

Синтаксис інтерфейсу аналогічний синтаксису класу:

```
[атрибути] [специфікатори] interface Ім'я_інтерфейсу
            [:список_батьківських_ інтерфейсів] {
      Оголошення _ властивостей _ і _ методів
};
```

Для інтерфейсу можуть бути зазначені специфікатори **new**, **public**, **protected**, **internal** і **private**. Специфікатор **new** застосовується для вкладених інтерфейсів і має такий же зміст, що як і відповідає модифікатор методу класу. Інші специфікатори управляють видимістю інтерфейсу. За замовчуванням інтерфейс доступний тільки зі збірка, у якім він описаний **(internal**).

Інтерфейс може успадковувати властивості декількох інтерфейсів, у цьому випадку *предки* перелічуються через кому. *Тіло інтерфейсу* становлять абстрактні методи, шаблони властивостей і індексаторів, а також події.

Від абстрактного класу *інтерфейс* відрізняється деякими деталями в синтаксисі та поведінці. Синтаксична відмінність полягає в тому, що методи *інтерфейсу* оголошуються без вказівки модифікатора доступу. Відмінність у поведінці полягає в більш жорстких вимогах до нащадків:

- елементи інтерфейсу за замовчуванням мають специфікатор доступу public і не можуть мати специфікаторів, заданих явно;
- інтерфейс не може містити полів і звичайних методів — усі елементи інтерфейсу повинні бути абстрактними;
- клас, у списку предків якого задається інтерфейс, повинен визначати *всі* його елементи, у той час як нащадок абстрактного класу може не перевизначати частина абстрактних методів предка (у цьому випадку похідний клас також буде абстрактним);
- клас може мати в списку предків кілька інтерфейсів, при цьому він повинен визначати всі їх методи.

Інтерфейси дозволяють частково впоратися з таким істотним недоліком мови C#, як відсутність *множинного спадкування* класів. Хоча реалізація *множинного спадкування* зустрічається з рядом проблем, його відсутність істотна знижує виразну потужність мови. У мові C# повного *множинного спадкування* класів немає. Щоб частково згладити цей пробіл, допускається *множинне спадкування інтерфейсів*. Забезпечити можливість класу мати декілька батьків - один повноцінний клас, а інші у вигляді *інтерфейсів*, - у цьому й полягає основне *призначення інтерфейсів*. Відзначимо одне важливе *призначення інтерфейсів*. *Інтерфейс* дозволяє описувати деякі бажані властивості, якими можуть мати об'єкти різних класів.

Як приклад розглянемо інтерфейс Iaction, що визначає базову поведінку персонажів комп'ютерної гри, що зустрічалися в попередніх главах. Допустимо, що будь-який персонаж повинен уміти виводити себе на екран, атакувати й красиво вмирати:

```
interface Iaction
{
    void Draw();
    int Attack(int a);
    void Die();
    int Power { get; }
}
```

Операції is і as

При роботі з об'єктом через об'єкт типу інтерфейсу буває необхідно переконатися, що об'єкт підтримує даний інтерфейс. Перевірка виконується за допомогою бінарної операції **is**.

Допустимо, ми оформили якісь дії з об'єктами у вигляді методу з параметром типу **object**. Перш ніж використовувати цей параметр усередині методу для звертання до методів, описаних у похідних класах, потрібно виконати перетворення до похідного класу. Для безпечного перетворення слід перевірити, чи можливо воно, наприклад так:

```
static void Act( object A )
{
    if ( A is Iaction )
    {
        Iaction Actor = (Iaction) A;
        Actor.Draw();
    }
}
```

У метод `Act` можна передавати будь-які об'єкти, але на екран будуть виведені тільки ті, які підтримують інтерфейс `Iaction`.

Недоліком використання операції **is** є те, що перетворення фактично виконується двічі: при перевірці й при властиво перетворенні. Більш ефективної є інша операція — **as**. Вона виконує перетворення до заданого типу, а якщо це неможливо, формує результат **null**, наприклад:

```
static void Act( object A )
{
    Iaction Actor = A as Iaction;
    if ( Actor != null ) Actor.Draw();
}
```

Обидві операції застосовуються як до інтерфейсів, так і до класів.

Стандартні інтерфейси .NET

У бібліотеці класів .NET визначена набір стандартних інтерфейсів, які задають бажану поведінку об'єктів. Наприклад, інтерфейс **Icomparable** задає метод порівняння об'єктів на більше-менше, що дозволяє виконувати їхнє сортування. Реалізація інтерфейсів **Ienumerable** і **Ienumerator** дає можливість переглядати вміст об'єкта за допомогою конструкції **foreach**, а реалізація інтерфейсу **Icloneable** — клонувати об'єкти.

Стандартні інтерфейси підтримуються багатьма стандартними класами бібліотеки. Наприклад, робота з масивами за допомогою циклу **foreach** можлива саме тому, що тип **Array** реалізує інтерфейси **Ienumerable** і **Ienumerator**. Можна створювати й власні класи, які підтримують стандартні інтерфейси, що дозволить використовувати об'єкти цих класів стандартними способами.

Порівняння об'єктів

Інтерфейс **Icomparable** визначений у просторі імен **System**. Він містить усього один метод **Compareto**, що повертає результат порівняння двох об'єктів — поточного й переданого йому в якості параметра:

```
interface Icomparable
{
    int Compareto( object obj );
}
```

Метод повинен повертати:

- 0, якщо поточний об'єкт і параметр рівні;
- від'ємне число, якщо поточний об'єкт менший параметра;
- додатне число, якщо поточний об'єкт більший параметра.

Інтерфейс **Icomparer** визначений у просторі імен **System.Collections**. Він містить один метод **Compare**, який повертає результат порівняння двох об'єктів, переданих йому в якості параметрів:

```
interface Icomparer
{
    int Compare( object ob1, object ob2 )
}
```

Метод повинен повертати:

- 0, якщо об'єкти рівні;
- від'ємне число, якщо перший об'єкт менший - другого;
- додатне число, якщо перший об'єкт більший - другого.

Принцип застосування цього інтерфейсу полягає в там, що для кожного критерію впорядкування об'єктів створюється допоміжний клас, який реалізує цей інтерфейс. Об'єкт цього класу передається в стандартний метод впорядкування масиву як другий аргумент.

Клонування об'єктів

Клонування — це створення копії об'єкта. Як відомо, при присвоєнні одного об'єкта посилального типу іншому копіюється посилання, а не сам об'єкт. Якщо необхідно скопіювати в іншу область пам'яті поля об'єкта, можна скористатися методом **Memberwiseclone**, який будь-який об'єкт успадковує від класу **object**. При цьому об'єкти, на які вказують поля об'єкта, у свою чергу, що є посиланнями, не копіюються. Це називається *поверхневим клонуванням*.

Для створення повністю незалежних об'єктів необхідно *глибоке клонування*, коли в пам'яті створюється дублікат усього дерева об'єктів, тобто об'єктів, на які посилаються поля об'єкта, поля полів, і так далі. Алгоритм глибокого клонування досить складний, оскільки вимагає рекурсивного обходу всіх посилань об'єкта й відстеження циклічних залежностей.

Об'єкт, що має власні алгоритми клонування, повинен оголошуватися як спадкоємець інтерфейсу **Icloneable** і перевизначати його єдиний метод **Clone**.

Перебір об'єктів (інтерфейс Ienumerable) та ітератори

Оператор **foreach** є зручним засобом перебору елементів об'єкта. Масиви та всі стандартні колекції бібліотеки .NET дозволяють виконувати такий перебір завдяки тому, що в них реалізовані інтерфейси **Ienumerable** і **Ienumerator**. Для застосування оператора **foreach** до користувацького типу даних потрібно реалізувати в ньому ці інтерфейси.

```
public interface IEnumerable
{
```

```
    IEnumerator GetEnumerator();
    }
```

Інтерфейс **Ienumerable** (*перерахований*) визначає всього один метод — **Getenumerator** об'єкт, який повертає тип Ienumerator (*перерахунок*), який можна використовувати для перегляду елементів об'єкта.

```
public interface IEnumerator
{
    bool MoveNext();
    object Current {get;}
    void Reset();
    }
```

Інтерфейс **Ienumerator** задає три елементи:

- властивість **Current**, що повертає поточний елемент об'єкта;
- метод **Movenext**, що просуває *перерахунок* на наступний елемент об'єкта;
- метод **Reset**, що встановлює *перерахунок* у початок перегляду.

Цикл **foreach** використовує ці методи для перебору елементів, з яких складається об'єкт.

Таким чином, якщо потрібно, щоб для перебору елементів класу міг застосовуватися цикл **foreach**, необхідно реалізувати чотири методи: **Getenumerator**, **Current**, **Movenext** і **Reset**.

```
using System;
using System.Collections;

namespace ConsoleApplication1
{
    class MyInt : IEnumerable, IEnumerator
    {
        int[] ints = { 12, 13, 1, 4 };
        int index = -1;

        // Реалізуємо інтерфейс IEnumerable
        public IEnumerator GetEnumerator()
        {
            return this;
        }

        // Реалізуємо інтерфейс IEnumerator
        public bool MoveNext()
        {
            if (index == ints.Length - 1)
            {
                Reset();
                return false;
            }

            index++;
            return true;
        }

        public void Reset()
        {
            index = -1;
        }

        public object Current
        {
            get
            {
```

```
                return ints[index];
            }
        }
    }

    class Program
    {
        static void Main()
        {
            MyInt mi = new MyInt();

            foreach (int i in mi)
                Console.Write(i+"\t");

            Console.ReadLine();
        }
    }
}
```

У **C#** версії **2.0** були введені засоби, що полегшують виконання перебору в об'єкті — ітератори. *Ітератор* представляє собою блок коду, що задає послідовність перебору елементів об'єкта. На кожному проході циклу **foreach** виконується один крок ітератора, що закінчується видачею чергового значення. Видача значення виконується за допомогою ключового слова **yield**.

Блок ітератора синтаксично це звичайний блок і може зустрічатися в тілі методу, операції або частині **get** властивості, якщо відповідне значення, яке повертається, має **тип** Ienumerable **або** Ienumerator.

У тілі блоку ітератора можуть зустрічатися дві конструкції:

- **yield return** формує поточне значення ітерації;
- **yield break** сигналізує про завершення ітерації.

Ключове слово **yield** має спеціальне значення для компілятора тільки в цих конструкціях. Код блоку ітератора виконується не так, як звичайні блоки. Компілятор формує службовий об'єкт-*перерахунок*, при виклику методу **Movenext** якого виконується код блоку ітератора, що видає чергове значення за допомогою ключового слова **yield**. Наступний виклик методу **Movenext** об'єкта-перерахунка відновляє виконання блоку ітератора з моменту, на якім він був припинений у попередній раз.

```
using System;
using System.Collections;

namespace ConsoleApplication1
{
    class Letter
    {
        char ch = 'A';
        int end;
        public Letter(int end)
        {
            this.end = end;
        }
        // ітератор, що повертає end-букв
        public IEnumerator GetEnumerator()
        {
            for (int i = 0; i < this.end; i++)
            {
                if (i == 33) yield break; // Выход із итератора
                yield return (char)(ch + i);
            }
```

```
        }
        // Створення іменованого ітератора
        public IEnumerable MyItr(int begin, int end)
        {
            for (int i = begin; i <= end; i++)
            {
                yield return (char)(ch + i);
            }
        }
    }
    class Program
    {
        static void Main()
        {
            Console.Write("Скільки букв вивести? ");
            int i = int.Parse(Console.ReadLine());

            Letter lt = new Letter(i);
            Console.WriteLine("\nРезультат: \n");

            foreach (char c in lt)
                Console.Write(c + " ");

            Console.Write("\n Введіть границі \n\nMin: ");
            int j = int.Parse(Console.ReadLine());
            Console.Write("Max: ");
            int y = int.Parse(Console.ReadLine());
            Console.WriteLine("\nРезультат: \n");

            foreach (char c in lt.MyItr(j,y))
                Console.Write(c + " ");

            Console.ReadLine();
        }
    }
}
```

Завдання до лабораторної роботи:

Порядок виконання роботи:

1) Створити проект C#.
2) Визначити ієрархію класів та класи. Повну структуру класів і їх взаємозв'язок продумати самостійно. У класах реалізувати абстрактні, віртуальні та не віртуальні методи, властивості, деструктори, індексатори та операції згідно з варіантом завдання. Для абстрактного класу визначити які методи повинні бути абстрактними.
3) Розробити програму мовою C# тестування всіх можливосте створених класів із виведення відповідної інформації. Для ідентифікація або типів під час виконання програми: використати оператори **`is`**, **`as`** або **`typeof`**.
4) Підготувати звіт у твердій копії та в електронному виді.

Завдання 1. Варіанти задач. Перебудувати ієрархії в лабораторної робота №3 (Побудувати ієрархію класів відповідно до варіанта завдання. Згідно завдання вибрати базовий клас та похідні. В класах задати поля, які характерні для кожного класу. Для всіх класів розробити метод Show(), який виводить дані про

об’єкт класу.), з визначення нових сутностей таким чином щоб базовими були декілька інтерфейсів користувача та інтерфейси .NET.

1.1. Студент, викладач, персона, завідувач кафедри.
1.2. Службовець, персона, робітник, інженер.
1.3. Робітник, кадри, інженер, адміністрація.
1.4. Деталь, механізм, виріб, вузол.
1.5. Організація, страхова компанія, нафтогазова компанія, завод.
1.6. Журнал, книга, друковане видання, підручник.
1.7. Тест, іспит, випускний іспит, випробування.
1.8. Місце, область, місто, мегаполіс.
1.9. Іграшка, продукт, товар, молочний продукт.
1.10. Квитанція, накладна, документ, рахунок.
1.11. Автомобіль, поїзд, транспортний засіб, експрес.
1.12. Двигун, двигун внутрішнього згоряння, дизель, реактивний двигун.
1.13. Республіка, монархія, королівство, держава.
1.14. Савець, парнокопитне, птах, тварина.
1.15. Корабель, пароплав, вітрильник, корвет.

Завдання 2. Варіанти задач. Побудувати ієрархію із інтерфейсом, який успадковує інтерфейси .NET. Побудувати похідні класи, які реалізують інтерфейси відповідно до варіанта завдання. У кожному класі задати поля та методи, які характерні для цих нього, та визначити методи інтерфейсів.

2.1. Створити інтерфейс **Figure** з методами обчислення площі й периметра, а також методом, що виводять інформацію про фігуру на екран. Створити похідні класи: **Rectangle** (прямокутник), **Circle** (коло), **Triangle** (трикутник) зі своїми методами обчислення площі й периметра. Створити масив **n** фігур і вивести повну інформацію про фігури на екран.
2.2. Створити інтерфейс **Function** з методом обчислення значення функції **y=f(x)** у заданій точці. Створити похідні класи: **Line (y=ax+b)**, **Kub (y=ax2+bx+c)**, **Hyperbola** зі своїми методами обчислення значення в заданій точці. Створити масив n функцій і вивести повну інформацію про значення даних функцій у точці x.
2.3. Створити інтерфейс **Видання** з методами, що дозволяють вивести на екран інформацію про видання, а також визначити чи є дане видання шуканим. Створити похідні класи: **Книга** (назва, прізвище автора, рік видання, видавництво), **Стаття** (назва, прізвище автора, назва журналу, його номер і рік видання), **Електронний ресурс** (назва, прізвище автора, посилання, анотація) зі своїми методами висновку інформації на екран. Створити каталог (масив) з **n** видань, вивести повну інформацію з каталогу, а також організувати пошук видань на прізвище автора.
2.4. Створити інтерфейс **Trans** з методами, що дозволяють вивести на екран інформацію про транспортний засіб, а також визначити

вантажопідйомність транспортного засобу. Створити похідні класи: **Легкова_машина** (марка, номер, швидкість, вантажопідйомність), **Мотоцикл** (марка, номер, швидкість, вантажопідйомність, наявність коляски, при цьому якщо коляска відсутня, то вантажопідйомність рівна 0), Вантажівка (марка, номер, швидкість, вантажопідйомність, наявність причепа, при цьому якщо є причіп, то вантажопідйомність збільшується у два рази) зі своїми методами висновку інформації на екран, і визначення вантажопідйомності. Створити базу (масив) з **n** машин, вивести повну інформацію з бази на екран, а також організувати пошук машин, що задовольняють вимогам вантажопідйомності.

2.5. Створити інтерфейс **Persona** з методами, що дозволяють вивести на екран інформацію про персону, а також визначити її вік (на момент поточної дати). Створити похідні класи: **Абітурієнт** (прізвище, дата народження, факультет), **Студент** (прізвище, дата народження, факультет, курс), **Викладати** (прізвище, дата народження, факультет, посада, стаж), зі своїми методами висновку інформації на екран, і визначення віку. Створити базу (масив) з **n** персон, вивести повну інформацію з бази на екран, а також організувати пошук персон, чий вік попадає в заданий діапазон.

2.6. Створити інтерфейс **Товар** з методами, що дозволяють вивести на екран інформацію про товар, а також визначити, чи відповідає вона строку придатності на поточну дату. Створити похідні класи: **Продукт** (назва, ціна, дата виробництва, строк придатності), **Партія** (назва, ціна, кількість шт, дата виробництва, строк придатності), **Комплект** (назви, ціна, перелік продуктів) зі своїми методами висновку інформації на екран, і визначення відповідності строку придатності. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук простроченого товару (на момент поточної дати).

2.7. Створити інтерфейс **Товар** з методами, що дозволяють вивести на екран інформацію про товар, а також визначити, чи відповідає вона шуканому типу. Створити похідні класи: **Іграшка** (назва, ціна, виробник, матеріал, вік, на який розрахована), **Книга** (назва, автор, ціна, видавництво, вік, на який розрахована), **Спорт-інвертар** (назва, ціна, виробник, вік, на який розрахована), зі своїми методами висновку інформації на екран, і визначення відповідності шуканому типу. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук товарів певного типу.

2.8. Створити інтерфейс **Телефонний_довідник** з методами, що дозволяють вивести на екран інформацію про записи в телефонному довіднику, а також визначити відповідність запису критерію пошуку. Створити похідні класи: **Персона** (прізвище, адреса, номер телефону),

Організація (назва, адреса, телефон, факс, контактна особа), **Друг** (прізвище, адреса, номер телефону, дата народження) зі своїми методами висновку інформації на екран, і визначення відповідності шуканому типу. Створити базу (масив) з **n** товарів, вивести повну інформацію з бази на екран, а також організувати пошук у базі на прізвище.

2.9. Створити абстрактний клас **Клієнт** із методами, що дозволяють вивести на екран інформацію про клієнтів банку, а також визначити відповідність клієнта критерію пошуку. Створити похідні класи: **Вкладник** (прізвище, дата відкриття внеску, розмір внеску, відсоток по внескові), **Кредитор** (прізвище, дата видачі кредиту, розмір кредиту, відсоток по кредиту, остача боргу), **Організація** (назва, дата відкриття рахунку, номер рахунку, сума на рахунку) зі своїми методами висновку інформації на екран, і визначення відповідності даті (відкриття внеску, видачі кредиту, відкриття рахунку). Створити базу (масив) з **n** клієнтів, вивести повну інформацію з бази на екран, а також організувати пошук клієнтів, що почали співробітничати з банком у задану дату.

2.10. Створити інтерфейс **Програмне_забезпечення** з методами, що дозволяють вивести на екран інформацію про програмне забезпечення, а також визначити відповідність можливості використання (на момент поточної дати). Створити похідні класи: **Вільне** (назва, виробник), **Умовно-безкоштовне** (назва, виробник, дата установки, строк безкоштовного використання), **Комерційне** (назва, виробник, ціна, дата установки, строк використання) зі своїми методами висновку інформації на екран, і визначення можливості використання на поточну дату. Створити базу (масив) з **n** видів програмного забезпечення, вивести повну інформацію з бази на екран, а також організувати пошук програмного забезпечення, яке припустимо використовувати на поточну дату.

Завдання 3. Варіанти задач. До одного з розроблених класів в лабораторної робота №4 або №5 добавити стандартні інтерфейси .NET перерахування щоб можна застосовувати оператор **foreach**.

Контрольні запитання