Практична робота 9.2. Вбудовані інтерфейси C#

План

1. Впорядкованість об'єктів і інтерфейс IComparable
2. Впорядкування об'єктів за кількома критеріями. Інтерфейс IComparer
3. Перелічуваність об'єктів і інтерфейси. Інтерфейс IEnumerable
4. Ітератори і інтерфейс IEnumerable

У попередній лекції ми говорили, що інтерфейс це повністю абстрактний клас, всі методи якого абстрактні (не містять реалізації).

Це означає, що клас, який наслідує інтерфейс, зобов'язаний повністю реалізувати *всі методи інтерфейсу*. У цьому відмінність від класу, що успадковує абстрактний клас, де похідний клас може реалізувати лише *деякі методи* абстрактного класу.

Головне призначення інтерфейсів – забезпечення множинного наслідування класів.

В бібліотеці FCL платформи .Net Framework є багато вбудованих інтерфейсів, які використовуються самими класами бібліотеки FCL, а також можуть використовуватися прикладними програмами. Розглянемо деякі з них.

1. Впорядкованість об'єктів і інтерфейс IComparable

Часто, коли створюється клас, бажано задати відношення порядку на його об'єктах для того, щоб можна було якимсь чином порівнювати об'єкти. Такий клас слід оголосити спадкоємцем інтерфейсу IComparable. Цей інтерфейс має всього один метод CompareTo (object obj), що повертає ціле значення, позитивне, негативне або рівне нулю, залежно від виконання відношення "більше", "менше" або "рівно".

Таким чином, метод CompareTo (object obj) повинен повертати:

- 0, якщо поточний об'єкт і параметр рівні;
- негативне число, якщо поточний об'єкт менше параметра;
- позитивне число, якщо поточний об'єкт більше параметра.

При використанні цього інтерфейсу в класі спочатку визначають метод CompareTo, а після цього вводять перевизначені операції для порівняння об'єктів звичним чином з використанням знаків операцій відношення (<,>,==).

Приклад 1 . Розглянемо наш клас Person

```
class Person
   {
      public string Name;       //ім'я
      public int Age;           // вік
      public string Role;       // роль
      public string GetName() { return Name; }
      public int GetAge() { return Age; }
   }
```

Введемо відношення порядку на класі Person, зробивши цей клас спадкоємцем інтерфейсу IComparable. Реалізуємо в цьому класі метод інтерфейсу CompareTo для порівняння об'єктів:

```
public class Person:IComparable
  {
    public int CompareTo( object pers)
    {
      const string s = "Порівнюваний об'єкт не належить класу Person";
      Person p = pers as Person;
      if (!p.Equals(null))
        return (Name.CompareTo(p.Name));
      throw new ArgumentException (s);
    }
    // інші компоненты класу
  }
```

Оскільки аргумент в методі інтерфейсу належить типу object, перед виконанням порівняння його потрібно привести до типу Person. Для приведення використовується операція **as**, що дозволяє перевірити коректність виконання приведення. Якщо приведення неможливе, то неможливо виконати і порівняння об'єктів. В цьому випадку генерується виключення, яке може обробити розумним чином лише клієнт класу, що намагався виконати порівняння. У самому класі Person можна лише пояснити ситуацію, передавши інформацію об'єкту виключення.

При перевірці на значення null використовується відношення Equals, а не звичайна рівність, яка буде перевизначена.

Приведення типів і операції as і is

Операція **is** використовується в логічних виразах. Логічний вираз

obj is T

вірний (true), якщо об'єкт obj належить типу T, і невірний (false) в іншому випадку.

Оператор призначення

obj = P as T;

призначає об'єкту **obj** об'єкт P, приведений до типу T, якщо таке приведення можливе, інакше об'єкту призначається значення null.

Порядок на класі Person

Визначивши метод CompareTo для класу Person, ми тим самим ввели відношення порядку для об'єктів цього класу. Звичайно, порівняння персон може виконуватися по різних критеріях: зросту, віку, зарплаті. В нашому випадку відношення порядку на об'єктах класу Person задається як відношення порядку на прізвищах персон. Оскільки рядки успадковують інтерфейс IComparable, для прізвищ персон викликається метод CompareTo, його результат і повертається як результат методу CompareTo для персон.

Введемо тепер в нашому класі Person перевантаження операцій відношення:

```
public static bool operator <(Person p1, Person p2)
```

```
        {
          return (p1.CompareTo(p2) < 0);
        }
        public static bool operator >(Person p1, Person p2)
        {
          return (p1.CompareTo(p2) > 0);
        }
        public static bool operator <=(Person p1, Person p2)
        {
          return (p1.CompareTo(p2) <= 0);
        }
        public static bool operator >=(Person p1, Person p2)
        {
          return (p1.CompareTo(p2) >=0);
        }
       }
```

Повний код програми

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

namespace Console1_Lab11
{
    public class Person :IComparable
    {
        public string Name;       //ім'я
        public int Age;           // вік
        public string Role;       // роль
        public string GetName() { return Name; }
        public int GetAge() { return Age; }
        public Person(string N)
        {
            this.Name = N;
        }
        public int CompareTo(object pers)
        {
            const string s = "Порівнюваний об'єкт не належить до класу Person";
            Person p = pers as Person;
            if (!p.Equals(null))
                return (Name.CompareTo(p.Name));
            throw new ArgumentException(s);
        }
        public static bool operator <(Person p1, Person p2)
        {
            return (p1.CompareTo(p2) < 0);
        }
        public static bool operator >(Person p1, Person p2)
        {
            return (p1.CompareTo(p2) > 0);
        }
        public static bool operator <=(Person p1, Person p2)
        {
            return (p1.CompareTo(p2) <= 0);
        }
```

```
        public static bool operator >=(Person p1, Person p2)
        {
            return (p1.CompareTo(p2) >= 0);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Person prep1 = new Person("Тітков");
            Person prep2 = new Person("Коротун");
            Person prep3 = new Person("Шилін");
            Person prep4 = new Person("Крячок");
            Person prep5 = new Person("Пожидаєва");
            Person prep6 = new Person("Проскудіна");
            Console.WriteLine("{0} > {1} = {2}", prep1.Name,
               prep2.Name, (prep1 > prep2));
            Console.WriteLine("{0} >= {1} = {2}", prep3.Name,
               prep4.Name, (prep3 >= prep4));
            Console.WriteLine("{0} != {1} = {2}", prep5.Name,
               prep6.Name, (prep5 != prep6));
            Console.ReadLine();
        }
    }

}
```

В цьому прикладі зроблене досить штучне порівняння об'єктів, яке просто демонструє можливості порівняння об'єктів класу. Зробимо порівняння об'єктів за віком. Змінемо конструктор класу з ініціалізацією віку:

```
        public Person(string N,int Age)
        {
            this.Name = N;
            this.Age = Age;

        }
```

Задамо в методі CompareTo операції порівняння за віком

```
        public int CompareTo(object pers)
        {
            const string s = "Порівнюваний об'єкт не належить до класу Person";
            Person p = pers as Person;
            if (!p.Equals(null))
                return (Age.CompareTo(p.Age));
           throw new ArgumentException(s);
        }
```

Можлива і наступна реалізація інтерфейсу:

```
        public int CompareTo(object pers)
        {
            Person p = (Person) pers;
            if (this.Age > p.Age) return 1;
            if (this.Age < p.Age) return -1;
            return 0;

        }
```

Для спрощення виводу на консоль створимо віртуальний метод

```
 virtual public void Passport()
    {
        Console.WriteLine("Name = {0} Age = {1}",Name, Age);

    }
```

Внесемо зміни в метод Main. Створимо масив персон, елементами якого є створені об'єкти класу prep1,...prep6. Так як це масив – ми можемо його відсортувати викликом методу Sort().

Приклад 2.

Повний код програми:

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

namespace Console2_Lab11
{
    public class Person : IComparable
    {
        public string Name;       //имя
        public int Age;           // возраст
        public string Role;       // роль
        public string GetName() { return Name; }
        public int GetAge() { return Age; }
        public Person(string N,int Age)
        {
            this.Name = N;
            this.Age = Age;
        }
        public int CompareTo(object pers)
        {
            Person p = (Person) pers;
            if (this.Age > p.Age) return 1;
            if (this.Age < p.Age) return -1;
            return 0;
        }

         public void Passport()
        {
            Console.WriteLine("Name = {0} Age = {1}",Name, Age);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Person prep1 = new Person("Іванов",48);
            Person prep2 = new Person("Петров",36);
            Person prep3 = new Person("Соколов",30);
            Person prep4 = new Person("Сидоров",50);
            Person prep5 = new Person("Орлов",50);
            Person prep6 = new Person("Федоров",60);
            Person [] group = new Person[6];
            group[0] = prep1;
```

```
            group[1] = prep2;
            group[2] = prep3;
            group[3] = prep4;
            group[4] = prep5;
            group[5] = prep6;
            Array.Sort(group);
            foreach (Person elem in group) elem.Passport();
            Console.ReadLine();
        }
    }
}
```

Інтерфейс IComparable дозволяє встановити порівняння об'єктів тільки за одним критерієм. Але у багатьох алгоритмах потрібно виконувати порівняння об'єктів за різними критеріями. У C# для цього використовується інтерфейс **IComparer.**

2. Впорядкування об'єктів за кількома критеріями. Інтерфейс IComparer

Інтерфейс IComparer визначений в просторі імен System.Collections. Він містить один метод Compare, що повертає результат порівняння двох об'єктів, переданих йому як параметри:

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

Принцип використання цього інтерфейсу полягає в тому, що для кожного критерію сортування об'єктів описується невеликий допоміжний клас, що реалізує цей інтерфейс. Об'єкт цього класу передається в стандартний метод сортування масиву другим аргументом.

Розглянемо приклад сортування об'єктів класу Person за віком і зарплатою. Модифікуємо клас Person, додамо в клас поле Zarplata, модифікуємо конструктор класу ініціалізацією цього поля.

Створимо два допоміжних класи: SortByAge, SortByZarplata, які є нащадками інтерфейсу **IComparer**. В них реалізовнані методи порівняння за потрібними критеріями (віком і зарплатою).

```
public class SortByAge : IComparer
  {
      //Сортування за віком
      int IComparer.Compare(object ob1, object ob2)
      {
          Person p1 = (Person)ob1;
          Person p2 = (Person)ob2;
          if (p1.Age > p2.Age) return 1;
          if (p1.Age < p2.Age) return -1;
          return 0;
      }
  }
```

```
        public class SortByZarplata : IComparer
//
        {
            // сортування за зарплатою
            int IComparer.Compare(object ob1, object ob2)
            {
                Person p1 = (Person)ob1;
                Person p2 = (Person)ob2;
                if (p1.Zarplata > p2.Zarplata) return 1;
                if (p1.Zarplata < p2.Zarplata) return -1;
                return 0;
             }
              }
```

Приклад 3.

Повний код програми.

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

namespace Console3_Lab11
{
    // Використання інтерфейсу IComparer
    public class Person
    {
        public string Name;       //имя
        public int Age;           // возраст
        public string Role;       // роль
        public int Zarplata;      // зарплата
        public string GetName() { return Name; }
        public int GetAge() { return Age; }

        public Person(string N,int Age, int Z)
        {
            this.Name = N;
            this.Age = Age;
            this.Zarplata = Z;
        }

         virtual public void Passport()
        {
            Console.WriteLine("Name = {0} Age = {1}  Zarplata = {2}", Name,
Age,Zarplata);
        }

         public class SortByAge : IComparer
         {
             //Сортування за віком
             int IComparer.Compare(object ob1, object ob2)
             {
                 Person p1 = (Person)ob1;
                 Person p2 = (Person)ob2;
                 if (p1.Age > p2.Age) return 1;
                 if (p1.Age < p2.Age) return -1;
                 return 0;
             }
         }

         public class SortByZarplata : IComparer
//
         {
```

```
            // сортування за зарплатою
            int IComparer.Compare(object ob1, object ob2)
            {
                Person p1 = (Person)ob1;
                Person p2 = (Person)ob2;
                if (p1.Zarplata > p2.Zarplata) return 1;
                if (p1.Zarplata < p2.Zarplata) return -1;
                return 0;
             }
        }

    }

    class Program
    {
        static void Main(string[] args)
        {
            Person prep1 = new Person("Іванов",48,1500);
            Person prep2 = new Person("Петров",36,3500);
            Person prep3 = new Person("Соколов",30,1000);
            Person prep4 = new Person("Сидоров",50,4500);
            Person prep5 = new Person("Орлов",50,6000);
            Person prep6 = new Person("Федоров",60,1300);
            Person [] group = new Person[6];
            group[0] = prep1;
            group[1] = prep2;
            group[2] = prep3;
            group[3] = prep4;
            group[4] = prep5;
            group[5] = prep6;
            Console.WriteLine("Сортування за віком:");
            Array.Sort(group,new Person.SortByAge());
            foreach (Person elem in group) elem.Passport();
            Console.WriteLine("Сортування за зарплатою:");
            Array.Sort(group,new Person.SortByZarplata());
            foreach (Person elem in group) elem.Passport();
            Console.ReadLine();

        }
    }

}
```

3. Перерахування об'єктів і інтерфейси. Інтерфейс IEnumerable

Якщо клас є деякою сукупність впорядкованих об'єктів, то щоб клас міг використати для їх перегляду цикл foreach, він має бути спадкоємцем інтерфейсу **Ienumerable,** мати в своєму складі ітератори - методи, що повертають результат типу IEnumerable.

Використання для роботи з масивами циклу foreach можливе саме тому, що тип Array реалізує інтерфейси IEnumerable і IEnumerator. Можна створювати і власні класи, що підтримують ці стандартні інтерфейси.

Інтерфейс впорядкованості IEnumerable

В цього інтерфейсу всього один метод - **GetEnumerator().** В методі немає жодних вхідних аргументів, так що, здається, організувати перерахування об'єктів класу досить просто - написати реалізацію одного методу. Проте є проблеми. Перша - метод GetEnumerator синтаксично визначається так:

IEnumerator GetEnumerator()

Це означає, що в результаті виклику методу повинен повертатися інтерфейсний об'єкт типу IEnumerator - ще одному інтерфейсу, пов'язаному з перерахуванням.

Метод GetEnumerator вимагає створення об'єкту нумератора, який реалізує методи інтерфейсу IEnumerator.

В інтерфейсі IEnumerator є декілька методів, і в сукупності саме вони і дозволяють організувати процес перехування.

Розглянемо приклад використання інтерфейсу IEnumerable.

Додамо в проект новий клас Persons, одним з полів якого буде масив об'єктів Person. Організуємо перерахування в цьому класі.

Приклад 4.

```
/// <summary>
/// Клас - спадкоємець інтерфейсу IEnumerable
/// що допускає перелічування об'єктів.
/// Перелічування  зводиться до перелічування  персон
/// заданих полем  container
/// </summary>
class Persons:IEnumerable
{
protected int size;
protected Person[] container;  // !!! Не Persons
Random rnd = new Random();
/// <summary>
/// Конструктор за замовчуванням.
/// Створює масив з 10 персон
/// </summary>
public Persons()
{
size = 10;
container = new Person[size];
FillContainer();
}
/// <summary>
/// Конструктор. Створює масив заданої розмірності
/// Створює його елементи, використовуючи рандомізацію.
/// </summary>
/// <param name="size">розмірність масиву</param>
public Persons(int size)
{
this.size = size;
container = new Person[size];
FillContainer();
}
```

```
/// <summary>
   /// Заповнення масиву person
   /// </summary>
void FillContainer()
   {
   for(int i = 0; i <size; i++)
   {
      int num = rnd.Next(3*size);
      int age = rnd.Next(27, 46);
      container[i]= new Person("агент_" + num, age);
   }
   }
   /// <summary>
   /// Конструктор, якому передається масив персон
   /// </summary>
   /// <param name="container"> масив Person</param>
 public Persons(Person[] container)
   {
   this.container = container;
   size = container.Length;
   }
}
```

В класі є поле container, яке є масивом з елементами класу Person. Набір конструкторів класу дозволяє передати класу масив персон або доручити самому класу його формування, використовуючи метод FillContainer. Оскільки клас оголошений спадкоємцем інтерфейсу IEnumerable, необхідно реалізувати метод GetEnumerator, що забезпечить можливість перераховувати об'єкти класу і, отже, використовувати цикл foreach.

Реалізація методу:

```
/// <summary>
   /// Реалізація методу інтерфейсу IEnumerable
   /// Зводиться до виклику відповідного методу
   /// для поля container, тобто масиву
   /// для якого цей метод реалізований в бібліотеці FCL
   /// </summary>
   /// <returns>нумератор - інтерфейсний об'єкт</returns>
   public IEnumerator GetEnumerator()
   {
         return container.GetEnumerator();  //подали container,
                                            // а отримали щось нове
   }
```

Повний код програми 4.

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

namespace Console4_Lab11
{
    public class Person
    {
        public string Name;       //ім'я
        public int Age;           // вік
        public string Role;       // роль
        public string GetName() { return Name; }
        public int GetAge() { return Age; }
        public Person(string N, int A)
        {
            this.Name = N;
            this.Age = A;
        }
    }

    class Persons:IEnumerable
    {
        protected int size;
        protected Person[] container;
        Random rnd = new Random();
        /// <summary>
        /// Конструктор за замовчанням.
        /// Створює масив з 10 персон
        /// </summary>
        public Persons()
        {
            size = 10;
            container = new Person[size];
            FillContainer();
        }

        /// <summary>
        /// Конструктор. Створює масив заданої розмірності
        /// Створює його елементи, використовуючи рандомізацію.
        /// </summary>
        /// <param name="size">розмірність масиву</param>
        public Persons(int size)
        {
            this.size = size;
            container = new Person[size];
            FillContainer();
        }
        /// <summary>
        /// Конструктор, якому передається масив персон
        /// </summary>
        /// <param name="container"> масив Person</param>
        public Persons(Person[] container)
        {
            this.container = container;
            size = container.Length;
        }

        /// <summary>
        /// Заповнення масиву person
        /// </summary>
        ///
```

```
        void FillContainer()
        {
            for (int i = 0; i < size; i++)
            {
                int num = rnd.Next(3 * size);
                int age = rnd.Next(27, 46);
                container[i] = new Person("агент_" + num, age);
            }
        }
        public IEnumerator GetEnumerator()
        {
            return container.GetEnumerator();
        }
    }
```

У тесті створюється об'єкт класу Persons і в циклі foreach об'єкти перебираються, повертаючи кожного разу черговий об'єкт класу Person. Метод ToString, визначений в класі Person, дозволяє виводити інформацію про об'єкти.

```
class Program
{

    static void Main(string[] args)
    {
        int size = 5;
        Persons agents = new Persons(size);
        foreach (Person agent in agents)
        {
            Console.WriteLine("Прізвище:  " + agent.Name.ToString() + "    Вік:  " +
agent.Age.ToString());
        }
        Console.ReadLine();
    }
}
```

Оператор foreach є зручним засобом перебору елементів об'єкту. Масиви і всі стандартні колекції бібліотеки .NET дозволяють виконувати такий перебір завдяки тому, що в них реалізовані інтерфейси IEnumerable і IEnumerator. Для застосування оператора foreach до типу даних користувача потрібно реалізувати в ньому ці інтерфейси.

4. Ітератори і інтерфейс IEnumerable

Інтерфейс IEnumerable визначає всього один метод — GetEnumerator, що повертає об'єкт типу IEnumerator (нумератор), який можна використовувати для перебору елементів об'єкту.

Інтерфейс IEnumerator задає три елементи:

- властивість Current, що повертає поточний елемент об'єкту;
- метод MoveNext, що пересуває нумератор на наступний елемент об'єкту;
- метод Reset, що встановлює нумератор в початок перегляду.

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

Таким чином, якщо потрібно для перебору елементів класу застосовувати цикл foreach, необхідно реалізувати чотири методи:

GetEnumerator, Current, MoveNext і Reset.

Це нецікава робота, а виконувати її доводиться часто, тому у версію 2.0 були введені засоби, що полегшують виконання перебору в об'єкті, — ітератори.

Ітератор – це блок коду, в якому задається послідовність перебору елементів об'єкту. Під час кожного проходу циклу foreach виконується один крок ітератора, що закінчується видачею чергового значення

foreach(var color in Color)

{……}.

Одна ітерція – одне “повернення чогось” з колекції Color в змінну color.

Повернення (видача) значення виконується за допомогою оператора **yield**, який дозволяє заповнювати контейнер елементами. Його синтаксис:

yield return <вираз>;

Кожне виконання оператора **yield** додає новий елемент в контейнер. Розглянемо простий приклад використання ітератора, який створює колекцію кольорів.

```
public System.Collections.IEnumerable Rainbow()
 {
     yield return "red";
     yield return "orange";
     yield return "yellow";
     yield return "green";
     yield return "blue";
     yield return "violet";
 }
```

Клієнти цього класу можуть працювати з цією колекцією, наприклад так:

```
string colors = "";
foreach(string s in tst.Rainbow())
  colors += s + "-";
```

В цьому прикладі tst – об'єкт класу Testing, а змінна s в циклі foreach набуде значень всіх кольорів, розміщених в контейнері за допомогою оператора yield. Слід зазначити, що реально жодні контейнери не створюються, а цикл foreach на кожному кроці викликає ітератор і створює новий елемент. Саме тому цикл foreach працює лише на читання елементів і не працює на запис.

Розглянемо ще один приклад створення ітератора для класу Person. Хай потрібно створити об'єкт, що містить масив екземплярів класу Person. Для підтримки перебору достатньо вказати, що клас Persons реалізує інтерфейс IEnumerable (оператор 1), і описати ітератор (оператор 2). Доступ до нього може бути здійснений через методи MoveNext і Current інтерфейсу IEnumerator.

Приклад 5.

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

namespace Console5_Lab11
{
    public class Person
    {
        public string Name;        //ім'я
        public int Age;            // вік
        public string Role;        // роль
        public string GetName() { return Name; }
        public int GetAge() { return Age; }
        public int Zarplata;       // зарплата
        public Person(string N, int A, int Z)
        {
            this.Name = N;
            this.Age = A;
            this.Zarplata = Z;
        }

        virtual public void Passport()
        {
            Console.WriteLine("Name = {0} Age = {1}  Zarplata = {2}", Name, Age,
Zarplata);
        }
    }

    class Persons:IEnumerable       //1
    {
        private Person[] mas;
        private int n;

        public Persons()
        {
           mas = new Person[10];
           n = 0;
        }
        public IEnumerator GetEnumerator()
        {
            for ( int i = 0; i < n; ++i ) yield return mas[i];      //2
        }
        public void Add( Person m )
        {
            if ( n >= 10 ) return;
            mas[n] = m;
            ++n;
        }
    }
    class Program
    {
        static void Main()
        {
```

```
            Persons pers = new Persons();
            pers.Add(new Person("Іванов", 48, 1500));
            pers.Add(new Person("Петров",36,3500));
            pers.Add(new Person("Соколов", 30,1000));
            foreach (Person x in pers) x.Passport();
            Console.ReadKey();
        }
    }
}
```

Блок ітератора синтаксично є звичайним блоком 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 значення, що видається на черговій ітерації;
- **yield break** сигналізує про завершення ітерації.

Ключове слово **yield** має спеціальне значення для компілятора лише в цих конструкціях.

Код блоку ітератора виконується не так, як звичайні блоки. Компілятор формує службовий об'єкт-нумератор, при виклику методу **MoveNext** якого виконується код блоку ітератора, що видає чергове значення за допомогою ключового слова yield. Наступний виклик методу MoveNext об'єкту-нумератора відновлює виконання блоку ітератора з моменту, на якому він був призупинений в попередній раз.

Висновки

В бібліотеці FCL є велика кількість стандартних (вбудованих) інтерфейсів, які можна використовувати в програмі. Ми цій лекції ми розглянули тільки деякі з них:

IComparable – впорядкування об'єктів за одним критерієм;

IComparer - впорядкування об'єктів за кількома критеріями;

IEnumerable – перелічування об'єктів.

Кожний з інтерфейсів має методи, реалізація яких в похідних класах дозволяє виконувати над об'єктами класу відповідні дії (порівняння, сортування, перелічування). Для того, щоб можна було реалізувати методи інтерфейсів в класі, потрібно зробити клас спадкоємцем інтерфейсу.

Використання вбудованих інтерфейсів в прикладній програмі може скоротити її текст і зробити її більш зрозумілою.

З інтерфейсом IEnumerable пов'язана така конструкція C# як **ітератори** - зручний спосіб перебору елементів колекції і доступу до них.

Крім цих інтерфейсів в C# є багато інших як от IClonable, ISerializable тощо вивчення яких виходить за рами нашого курсу.

Питання і завдання для самостійної роботи студента

1. Який стандартний інтерфейс визначає можливість порівняння об'єктів класу за одним критерієм? Які методи цього інтерфейсу потрібно реалізувати?

2. Яким чином потрібно реалізувати метод CompareTo (object obj)? Який результат повинен повертати цей метод?

3. Яке призначення операцій **as** та **is**? Яка між ними різниця?

4. Чи повинен клас реалізовувати всі методи своїх інтерфейсів-предків?

5. Як реалізувати перевизначення операцій відношення для порівняння об'єктів?

6. Перерахуєте стандартні інтерфейси .NET, які визначають можливості сортування і перегляду об'єктів за допомогою оператора foreach.

7. Яка різниця між інтерфейсами IComparable та IComparer?

8. Яке призначення інтерфейсу IEnumerable? Який його склад?

9. Опишіть, як використовується конструкція yield.