Лабораторна робота №8

Тема: Класи-колекції в C#.

Мета: Вивчити особливості роботи з класами-колекціями у C#.

Завдання: Здобути навички створення, відлагодження та тестування проектів обробки даних з використанням класів-колекцій мовою ***C#***.

(max: 5 балів)

Теоретичні відомості:

1. Абстрактні структури даних

Будь-яка програма призначена для обробки даних, від способу організації яких залежить її алгоритм. Для різних завдань необхідні різні способи зберігання і обробки даних, тому вибір структур даних повинен передувати створенню алгоритмів і ґрунтуватися на вимогах до функціональності і швидкодії програми. Найчастіше в програмах використовуються *масив, список, стек, черга, бінарне дерево, хеш-таблиця, граф і множина.*

Масив — це скінченна сукупність однотипних величин. Масив займає безперервну область пам'яті і надає прямий доступ до своїх елементів за індексом. Пам'ять під масив виділяється до початку роботи з ним і потім не змінюється. Нагадаємо, що в C# є *одновимірні, прямокутні і зубчасті масиви*.

У **списку** кожен елемент зв'язаний з наступним і, можливо, з попереднім. У першому випадку список називається *однозв'язним*, в другому — *двозв'язним*. Якщо останній елемент зв'язати покажчиком з першим, вийде **кільцевий** список. Кількість елементів в списку може змінюватися в процесі роботи програми.

Черга — окремий випадок однонаправленого списку, додавання елементів в який виконується в один кінець, а вибірка — з іншого кінця. Інші операції з чергою не визначені. При вибірці елемент виключається з черги. Говорять, що черга реалізує принцип обслуговування FIFO (First In — First Out, першим прийшов, — першим пішов).

Бінарне дерево — динамічна структура даних, що складається з вузлів, кожен з яких містить, окрім даних, не більше двох посилань на різні бінарні піддерева. На кожен вузол є рівно одне посилання. Початковий вузол називається коренем дерева.

Хеш-таблиця, **асоціативний масив**, або **словник** — це масив, доступ до елементів якого здійснюється не за номером, а за деяким ключем. Можна сказати, що це таблиця, що складається з пар "ключ-значення" Хеш-таблиця ефективно реалізує операцію пошуку значення по ключу. При цьому ключ перетворюється в число (хеш-код), яке використовується для швидкого знаходження потрібного значення в хеш-таблиці.

Граф — це сукупність вузлів і ребер, що сполучають різні вузли. Багато реальних практичних задач можна описати в термінах графів, що робить їх структурою даних, яка часто використовується при написанні програм.

Множина — це невпорядкована сукупність елементів. Для множини визначені операції перевірки належності елементу множині, включення і виключення елементу, а також об'єднання, пересічення і віднімання множин.

Ці структури даних називаються *абстрактними*, оскільки в них не задається *реалізація допустимих операцій*.

У бібліотеках більшості сучасних об'єктно-орієнтованих мов програмування представлені стандартні класи, що реалізують основні абстрактні структури даних. Такі класи називаються *колекціями*, або *контейнерами*. Для кожного типу колекції визначені методи роботи з її елементами, не залежні від конкретного типу даних, що зберігаються, тому один і той самий вигляд колекції можна використовувати для зберігання даних різних типів. Використання колекцій дозволяє скоротити терміни розробки програм і підвищити їх надійність. Вивчення можливостей стандартних колекцій і їх грамотне використання є необхідною умовою створення ефективних і професійних програм.

Кожен вид колекції підтримує свій набір операцій над даними, і швидкодія цих операцій може бути різною. Вибір вигляду колекції залежить від того, що потрібно робити з даними в програмі і які вимоги пред'являються до її швидкодії.

2. Колекції. Простір імен System.Collections

У C# під колекцією розуміється деяка група об'єктів. Колекції спрощують реалізацію багатьох завдань програмування, пропонуючи вже готові рішення для побудови структур даних. Всі колекції розроблені на основі певних інтерфейсів, тому стандартизують спосіб обробки групи об'єктів. Середовище .NET Framework підтримує три основні типи колекцій: *загального призначення, спеціалізовані і орієнтовані на побітову організацію даних.*

У просторі імен System.Collections визначені набори стандартних колекцій і інтерфейсів, які реалізовані в цих колекціях. У таблиці 1 наведені найбільш важливі інтерфейси. Інтерфейси IComparer, IEnumerable, IEnumerator ми розглядали в лекції 13.

Простір імен System.Collections.Specialized включає спеціалізовані колекції, наприклад, колекцію рядків StringCollection і хеш-таблицю із строковими ключами StringDictionary.

Таблиця 1. Інтерфейси простору імен System.Collections

Інтерфейс	Призначення
ICollection	Визначає загальні характеристики (наприклад, розмір) для набору елементів
IComparer	Дозволяє порівнювати два об'єкти
IDictionary	Дозволяє представляти вміст об'єкту у вигляді пар "ім'я-значення" (хеш-таблиці)
IDictionaryEnumerator	Використовується для нумерації вмісту об'єкту, що підтримує інтерфейс IDictionary
IEnumerable	Повертає інтерфейс IEnumerator для вказаного об'єкту
IEnumerator	Зазвичай використовується для підтримки оператора foreach відносно об'єктів
IHashCodeProvider	Повертає хэш-код для реалізації типу із застосуванням вибраного користувачем алгоритму хешування
Ilist	Підтримує методи додавання, видалення і індексування елементів в списку об'єктів

У таблиці 2 перелічені основні колекції, визначені в просторі System.Collections.

Таблиця 2. Класи колекцій загального призначення (з простору імен System.Collections)

Клас	Призначення	Найважливіші з реалізованих інтерфейсів
ArrayList	Динамічний масив	IList, ICollection, IEnumerable, ICloneable
BitArray	Компактний масив для зберігання бітових значень	ICollection, IEnumerable, ICloneable
Hashtable	Хеш-таблиця	IDictionary, ICollection, IEnumerable, ICloneable
Queue	Черга	ICollection, ICloneable, IEnumerable
SortedList	Колекція, відсортована по ключах. Доступ до елементів — по ключу або по індексу	IDictionary, ICollection, IEnumerable, ICloneable
Stack	Стек	ICollection, IEnumerable

Розглянемо деякі з цих класів.

3. Клас Stack

Стек — окремий випадок однонаправленого списку, додавання елементів в який і вибірка з якого виконуються з одного кінця, який називається вершиною стека (головою – head). Інші операції із стеком не визначені. При вибірці елемент виключається із стека. Говорять, що стек реалізує принцип обслуговування LIFO (Last In — First Out, останнім прийшов, — першим пішов).

У C# реалізацію стеку представляє клас **Stack**, який реалізує інтерфейси `ICollection, IEnumerable i ICloneable. Stack` - це динамічна колекція, розмір якої змінюється.

У класі Stack визначені наступні конструктори:

`public Stack();` //створює порожній стек, початкова кількість елементів якого дорівнює 10

`public Stack(int capacity);` //створює порожній стек, початкова кількість елементів якого дорівнює capacity

`public Stack(ICollection c);` //створює стек, який містить елементи колекції, заданої параметром c, початкова кількість елементів якого дорівнює 10.

Окрім методів, визначених в інтерфейсах, що реалізуються класом `Stack`, в цьому класі визначені власні методи

Таблиця 3. Методи класу Stack

Метод	Опис
`public virtual bool Contains(object v)`	Повертає значення true, якщо об'єкт **v** міститься у визиваючому стеку, інакше повертає значення false.
`public virtual void Clear()`	Встановлює властивість Count рівній нулю, тим самим очищаючи стек.
`public virtual object Peek()`	Повертає елемент, розташований у вершині стека, але не витягуючи його із стека
`public virtual object Pop()`	Повертає елемент, розташований у вершині стека, і витягує його із стека
`public virtual void Push(object v)`	Поміщає об'єкт v у стек
`public virtual object[] ToArray()`	Повертає масив, який містить копії елементів викликаючого стека

Розглянемо декілька прикладів використання стека.

Приклад 1. Для заданого значення n запишемо в стек всі числа від 1 до n, а потім будемо вибирати їх із стека:

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
namespace Lab8_1
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.Write("n= ");
            int n = int.Parse(Console.ReadLine());
            Stack intStack = new Stack();
            for (int i = 1; i <= n; i++)
                intStack.Push(i);
            Console.WriteLine("Розмірність стека " + intStack.Count);
            Console.WriteLine("Верхній елемент стека = " + intStack.Peek());
            Console.WriteLine("Розмірність стека " + intStack.Count);
            Console.Write("Вміст стека = ");
            while (intStack.Count != 0)
                Console.Write("{0} ", intStack.Pop());
            Console.WriteLine("\nНова розмірність стека " + intStack.Count);
            Console.ReadKey();
        }
    }
     }
```

Приклад 2. У текстовому файлі міститься математичний вираз. Необхідно перевірити баланс круглих дужок у виразі.

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

namespace Lab8_2
{
   class Program
   {
      static void Main(string[] args)
      {
        StreamReader fileIn=new StreamReader("t.txt");
        string line=fileIn.ReadToEnd();
        fileIn.Close();
        Stack duzshki=new Stack();
        bool flag=true;
       //перевіряємо баланс дужок
        for ( int i=0; i<line.Length;i++)
        {
         //якщо поточний символ дужка, що відкривається, то поміщаємо її в стек
          if (line[i]== '(') duzshki.Push(i);
          else if (line[i]== ')') //если поточний символ дужка, що закривається, то
          {
          //якщо стек порожній, то для закриваючої дужки не вистачає парної відкриваючої
           if (duzshki.Count == 0)
           { flag = false; Console.WriteLine("Можливо в позиції " + i + "зайва ) дужка"); }
           else duzshki.Pop(); //иначе витягуємо парну дужку
           }
         }
        //якщо після перегляду рядка стек виявився порожнім, то дужки збалансовані
         if (duzshki.Count == 0) { if (flag) Console.WriteLine("дужки збалансовані"); }
         else //инакше баланс дужок порушений
         {
          Console.Write("Можливо зайва (дужка в позиції:");
          while (duzshki.Count != 0)
          {
            Console.Write("{0} ", (int) duzshki.Pop());
          }
          Console.WriteLine();
         }
      }
   }
}
```

в файлі t.txt знаходиться вираз:
(1+2)-4*(a-3)/(2-7+6)

4. Клас Queue (Черга)

АТД черга - це окремий випадок однонаправленого списку, додавання елементів в який виконується в один кінець (хвіст), а вибірка робиться з іншого кінця (голови). Інші операції з чергою не визначені. При вибірці елемент виключається з черги. Говорять, що черга реалізує принцип обслуговування FIFO (fist in - fist out, першим прийшов, - першим вийшов).

У C# реалізацію АТД черга представляє клас **`Queue`**, який так як і стек реалізує інтерфейси `ICollection, IEnumerable i ICloneable`. Queue - це динамічна колекція, розмір якої змінюється. При необхідності збільшення місткості черги відбувається з коефіцієнтом зростання за умовчанням рівним 2.0.

У класі `Queue` визначені наступні конструктори:

`public Queue();` //створює порожню чергу, початкова місткість якої дорівнює 32

`public Queue (int capacity);` // створює порожню чергу, початкова місткість якої рівна capacity

`public Queue (int capacity, float n);` //створює порожню чергу, початкова місткість якої рівна capacity, і коефіцієнт зростання встановлюється параметром n

`public Queue (ICollection c);` //створює чергу, яка містить елементи колекції, заданої параметром c, і аналогічною місткістю

Окрім методів, визначених в інтерфейсах, що реалізуються класом Queue, в цьому класі визначені власні методи:

Таблиця 4. Методи класу Queue

Метод	Опис
`public virtual bool Contains (object v)`	Повертає значення true, якщо об'єкт v міститься в черзі, інакше повертає значення `false`
`public virtual void clear ()`	Встановлює властивість Count рівною нулю, тим самим очищаючи чергу
`public virtual object Dequeue ()`	Повертає об'єкт з початку черги, видаляючи його з черги
`public virtual object Peek ()`	Повертає об'єкт з початку черги, не видаляючи його з черги
`public virtual void Enqueue(object v)`	Додає об'єкт v в кінець черги
`public virtual object [ ] ToArray ()`	Повертає масив, який містить копії елементів черги
`public virtual void TrimToSizeO`	Встановлює властивість Capacity рівною значенню властивості Count

Розглянемо декілька прикладів використання черги.

Приклад 3. Для заданого значення n запишемо в чергу всі числа від 1 до n, а потім будемо вибирати їх з черги:

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using System.IO;
namespace Lab8_3
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.Write("n= ");
         int n = int.Parse(Console.ReadLine());
         Queue intQ = new Queue();
         for (int i = 1; i <= n; i++)
            intQ.Enqueue(i);
         Console.WriteLine("Розмірність черги " + intQ.Count);
         Console.WriteLine("Верхній елемент черги = " + intQ.Peek());
         Console.WriteLine("Розмірність черги " + intQ.Count);
         Console.Write("Вміст черги = ");
         while (intQ.Count != 0)
            Console.Write("{0} ", intQ.Dequeue());
         Console.WriteLine("\nНова розмірність черги " + intQ.Count);
      }
   }
}
```

Приклад 4. У текстовому файлі записана інформація про людей (прізвище, ім'я, по батькові, вік, вага через пробіл). Вивести на екран спочатку інформацію про людей молодше 40 років, а потім інформацію про всіх інших.

```
using System;
using System.Collections;
using System.Linq;
using System.Text;
using System.IO;
namespace Lab8_4
{
   public struct one //структура для зберігання даних про одну людину
   {
      public string f;
      public string i;
      public string про;
      public int age;
      public float massa;
   }
   class Program
   {
      static void Main(string[] args)
      {
         StreamReader fileIn = new StreamReader("t.txt", Encoding.GetEncoding(1251));
         string line;
         Queue people = new Queue();
         one a;
         Console.WriteLine("ВІК МЕНШЕ 40 РОКІВ");
         while ((line = fileIn.ReadLine()) != null) //читаємо до кінця файлу
         {
            string[] temp = line.Split(' '); //розбиваємо рядок на складові елементи
            //заповнюємо структуру
            a.f = temp[0];
            a.i = temp[1];
            a.o = temp[2];
            a.age = int.Parse(temp[3]);
            a.massa = float.Parse(temp[4]);
            // якщо вік менше 40 років, то виводимо дані на екран, інакше поміщаємо їх в
            //черга для тимчасового зберігання
            if (a.age < 40)
               Console.WriteLine(a.f + "\t" + a.i + "\t" + a.o + "\t" + a.age + "\t" + a.massa);
            else people.Enqueue(a);
         }
         fileIn.Close();
         Console.WriteLine("ВІК 40 РОКІВ І СТАРШЕ");
         while (people.Count != 0) // вибираємо з черги дані
         {
            a = (one)people.Dequeue();
            Console.WriteLine(a.f + "\t" + a.i + "\t" + a.o + "\t" + a.age + "\t" + a.massa);
         }
         Console.ReadKey();
      }
   }
}
```

5. Клас ArrayList

Основним недоліком звичайних масивів є те, що об'єм пам'яті, необхідний для зберігання їх елементів, має бути виділений до початку роботи з масивом. Об'єкт класу `ArrayList` є масивом змінної

довжини, елементами якого є посилання на об'єкти. Будь-який об'єкт класу `ArrayList` створюється з деяким початковим розміром.

Клас `ArrayList` реалізує інтерфейси `ICollection`, `IList`, `IEnumerable` і `ICloneable`. У класі `ArrayList` визначені наступні конструктори:

```
public ArrayList()          //створює порожній масив з максимальною ємкістю рівною 16 елементам, при поточній розмірності 0
public ArrayList(int capacity) // створює масив із заданою ємкістю capacity, при поточній розмірності 0
public ArrayList(ICollection c) // створює масив, який ініціалізується елементами колекції c
```

Крім методів, визначених в інтерфейсах, які реалізує клас `ArrayList`, в ньому визначені і власні методи:

Таблиця 5. Методи класу ArrayList

Метод	Опис
`public virtual void AddRange (ICollection c)`	Додає елементи з колекції **c** в кінець визиваючої колекції
`public virtual int BinarySearch (object v)`	У визиваючій відсортованій колекції виконує пошук значення, заданого параметром **v**. Повертає індекс знайденого елементу. Якщо потрібне значення не знайдене, повертає від'ємне значення.
`public virtual int BinarySearch (object v, IComparer comp)`	У визиваючій відсортованій колекції виконує пошук значення, яке задане параметром **v**, на основі методу порівняння об'єктів, заданого параметром **comp**. Повертає індекс знайденого елементу. Якщо потрібне значення не знайденен, повертає від'ємне значення.
`public virtual int BinarySearch (int startIdx, int count, object v, IComparer comp)`	В визиваючій відсортованій колекції виконує пошук значення, заданого параметром **v**, на основі методу порівняння об'єктів, заданого параметром comp. Пошук починається з елементу, індекс якого дорівнює значенню **startIdx**, і включає **count** елементів. Метод повертає індекс знайденого елементу. Якщо потрібне значення не знайдене, повертає від'ємне значення.
`public virtual void CopyTo(Array ar, int startIdx)`	Копіює вміст визиваючої колекції, починаючи з елементу, індекс якого дорівнює значенню **startIdx**, в масив, заданий параметром **ar**. Масив має бути одновимірним і сумісним за типом з елементами колекції.
`public virtual void CopyTo(int srcIdx, Array ar, int destIdx, int count)`	Копіює count елементів визиваючої колекції, починаючи з елементу, індекс якого дорівнює значенню **srcIdx**, в масив, заданий параметром **ar**, починаючи з елементу, індекс якого дорівнює значенню **destIdx**. Масив має бути одновимірним і сумісним за типом з елементами колекції
`public virtual ArrayList GetRange(int idx, int count)`	Повертає частину визиваючої колекції типа ArrayList. Діапазон колекції, яка повертається, починається з індексу idx і включає count елементів. Об'єкт, що повертається, посилається на ті ж елементи, що і визиваючий об'єкт
`public static ArrayList FixedSize(ArrayList ar)`	Перетворює колекцію **ar** на ArrayList-масив з фіксованим розміром і повертає результат
`public virtual void InsertRange(int startIdx, ICollection c)`	Вставляє елементи колекції, заданої параметром **c**, в колекцію, починаючи з індексу, заданого параметром **startIdx**

`public virtual int LastIndexOf(object v)`	Повертає індекс останнього входження об'єкту **v** в колекції. Якщо шуканий об'єкт не знайдений, повертає від'ємне значення
`public static ArrayList Readonly(ArrayList ar)`	Перетворює колекцію **ar** на ArrayList-масив, призначений лише для читання
`public virtual void RemoveRange(int idx, int count)`	Видаляє **count** елементів із колекції, починаючи з елементу, індекс якого дорівнює значенню **idx**
`public virtual void Reverse()`	Розташовує елементи колекції в зворотному порядку
`public virtual void Reverse(int startIdx, int count)`	Розташовує в зворотному порядку **count** елементів колекції, починаючи з індексу **startIdx**
`public virtual void SetRange(int startIdx, ICollection c)`	Замінює елементи колекції, починаючи з індексу **startIdx**, елементами колекції, заданої параметром **c**
`public virtual void Sort()`	Сортує колекцію в порядку збільшення елементів
`public virtual void Sort(IComparer comp)`	Сортує колекцію на основі методу порівняння об'єктів, заданого параметром **comp**. Якщо параметр **comp** має нульове значення, для кожного об'єкту використовується стандартний метод порівняння
`public virtual void Sort ( int startidx, int endidx, icomparer comp)`	Сортує частину колекції на основі методу порівняння об'єктів, заданого параметром **comp**. Сортування починається з індексу **startidx** і закінчується індексом **endidx**. Якщо параметр **comp** має нульове значення, для кожного об'єкту використовується стандартний метод порівняння
`public virtual object [ ] ToArray ()`	Повертає масив, який містить копії елементів визиваючого об'єкту
`public virtual Array ToArray (Type type)`	Повертає масив, який містить копії елементів об'єкту. Тип елементів в цьому масиві задається параметром **type**
`public virtual void TrimToSize()`	Встановлює властивість **Capacity** рівним значенню властивості **Count**

За замовчанням при створенні об'єкту типу `ArrayList` будується масив з **16** елементів типу `object`. Можна задати бажану кількість елементів в масиві, передавши його в конструктор або встановивши як значення властивості `Capacity`, наприклад:

```
ArrayList arr1 = new ArrayList();      // створюється масив з 16  елементів
ArrayList arr2 = new ArrayList(1000); // створюється масив з 1000 елементів
ArrayList arr3 = new ArrayList();
arr3.Capacity = 1000;                  // кількість елементів задається
```

Клас `ArrayList` реалізований через клас `Array`, тобто містить закрите поле цього класу. Оскільки всі типи в C# є нащадками класу `object`, масив може містити елементи довільного типа. Навіть якщо в масиві зберігаються звичайні цілі числа, тобто елементи значимого типа, внутрішній клас є масивом посилань на екземпляри типу `object`, які є упакованими типами-значеннями. Відповідно, при занесенні в масив виконується упаковка, а при виборі — розпаковування елементу.

Якщо при додаванні елементу в масив виявляється, що фактична кількість елементів масиву перевищує його ємкість, вона автоматично подвоюється, тобто відбувається повторне виділення пам'яті і переписування туди всіх існуючих елементів. Приклад занесення елементів в екземпляр класу `ArrayList`:

```
arr1.Add( 123 );    arr1.Add( -2 );    arr1.Add( "Вася" );
```

Доступ до елементу виконується за індексом, проте при цьому необхідно явним чином привести отримане посилання до цільового типу, наприклад:

```
int    a = (int) arr1[0];
```

```
int    b = (int) arr1[1];
string s = (string) arr1[2];
```

Спроба приведення до типу, який не відповідає типу що зберігається в елементі, викликає генерацію виключення `InvalidCastException`.

Розглянемо приклад використання класу `ArrayList`.

```
using System;
using System.Collections;

namespace Lab8_5
{
    class Program
    {
        static void ArrayPrint(string s, ArrayList a)
        {
            Console.WriteLine(s);
            foreach (int i in a)
                Console.Write(i + " ");
            Console.WriteLine();
        }

        static void Main(string[] args)
        {
            ArrayList myArray = new ArrayList();
            Console.WriteLine("Початкова ємкість масиву: " +
myArray.Capacity);
            Console.WriteLine("Початкова кількість елементів: " +
myArray.Count);

            Console.WriteLine("\nДобавили 5 цифр");
            for (int i = 0; i < 5; i++) myArray.Add(i);
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " +
myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nОптимізуємо ємність масиву");
            myArray.Capacity = myArray.Count;
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " +
myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nДодаємо елементи в масив");
            myArray.Add(10);
            myArray.Insert(1, 0);
            myArray.AddRange(myArray);
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " +
myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nВидаляємо елементи з масиву");
            myArray.Remove(0);
            myArray.RemoveAt(10);
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " +
myArray.Count);
            ArrayPrint("Вміст масиву", myArray);

            Console.WriteLine("\nВидаляємо весь масив");
```

```
            myArray.Clear();
            Console.WriteLine("Поточна ємність масиву: " + myArray.Capacity);
            Console.WriteLine("Поточна кількість елементів: " +
myArray.Count);
            ArrayPrint("Вміст масиву", myArray);
            Console.ReadKey();
        }
    }
}
```

6. Класи-прототипи або класи-узагальнення (generics)

У попередньому пункті було показано, що при використанні колекції `ArrayList` дозволено зберігати різні типи даних. Це відрізняється від масиву, де типи даних в масиві повинні бути одного і того ж типу. Але у `ArrayList` також існує проблема. Ми обговорювали раніше, що усе, що ви зберігаєте в `ArrayList` автоматично переводиться в тип `Object`, оскільки він є кореневим в .NET. Пригадаємо, що поліморфізм дозволяє базовому класу представляти підклас. .NET використовує `Object` як базовий клас для усіх інших типів, що створюються в C#.

Той факт, що `ArrayList` зберігає усі елементи як Object також означає, що під час проходження вам потрібно приводити усі об'єкти назад в той тип, у якому вони були спочатку. Це може бути проблемою або навіть може викликати помилку. Для вирішення цієї помилки ви можете використовувати **Класи-прототипи** або за інакшою термінологією **Класи-Узагальнення – (Generics)**.

Класи-прототипи (generics)— це класи, що мають у якості параметрів типи даних. Найчастіше їх застосовують для зберігання даних, тобто як контейнерні класи, або колекції. Починаючи з другої версії бібліотеки .NET додані параметризовані колекції для представлення основних структур даних — *стека, черги, списку, словника* тощо. Ці колекції, розташовані в просторі імен `System.Collections.Generic`, дублюють аналогічні колекції простору імен `System.Collections`.

Класи-прототипи називають також *родовими* або *шаблонними*, оскільки вони є зразками, за якими під час виконання програми будуються конкретні класи.

Класи-узагальнення працюють з параметром типу `T`, в класі чи при заданні інтерфейсу. Вам не потрібно задавати тип `T` до тих пір поки ви не створите екземпляр класу. Для створення классу-узагальнення вам потрібно:

1. Додати параметр типу `T` в трикутних дужках після ім'я класу.
2. Використовуйте параметр типу `T` замість назви типу членів вашого класу.

Наступний приклад демонструє як створювати клас узагальнення:

```
// Створення класу Узагальнення
public class CustomList<T>
{
   public T this[int index] { get; set; }
   public void Add(T item)
   {
      // Логіка методу.
   }
   public void Remove(T item)
   {
      // Логіка методу.
   }
}
```

Коли створюється екземпляр классу-узагальнення, ви задаєте тип, що ви хочете використовувати як параметр типу. Наприклад, якщо ви хочете використовувати вами заданий список для зберігання об'єктів типу Coffee, ви будете задавати Coffee як параметр типу.

Наступний приклад показує як створити екземпляр класу узагальнення:

```
...
//створення екземпляру класу узагальнення
CustomList<Coffee> clc = new CustomList<Coffee>;
Coffee coffee1 = new Coffee();
Coffee coffee2 = new Coffee();
clc.Add(coffee1);
clc.Add(coffee2);
Coffee firstCoffee = clc[0];
...
```

Коли створюється екземпляр класу, то кожен об'єкт `T` буде замінений на об'єкт типу, що ви задали. Наприклад, якщо ви задаєте `CustomList` клас з параметрами типу `Coffee`:

- Метод `Add` буде приймати лише аргументи типу `Coffee`.
- Метод `Remove` буде приймати лише аргументи типу `Coffee`.
- Індексатор повертатиме значення типу `Coffee`.

У таблиці 6 наведено відповідність між звичайними і параметризованими колекціями бібліотеки .NET (параметри, що визначають типи даних, які зберігаються в колекції, вказані в кутових дужках).

Таблиця 6. Параметризовані колекції

Клас-прототип	Звичайний клас
`Comparer<T>`	`Comparer`
`Dictionary<K,T>`	`HashTable`
`LinkedList<T>`	–
`List<T>`	`ArrayList`
`Queue<T>`	`Queue`
`SortedDictionary<K,T>`	`SortedList`
`Stack<T>`	`Stack<T>`

Як приклад розглянемо використання універсального "двійника" класу `ArrayList` — класу `List<T>` — для зберігання колекції об'єктів класу Person, а також для зберігання цілих чисел.

Приклад 6

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

namespace Lab8_6
{
    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;
        }
        public void Passport()
        {
            Console.WriteLine("Name = {0} Age = {1}", Name, Age);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<Person> pers = new List<Person>();
            pers.Add(new Person("Іваночко",18));
            pers.Add(new Person("Петришин",20));
            pers.Add(new Person("Мороз", 3));
            foreach (Person x in pers) x.Passport();
```

```
            List<int> lint = new List<int>();
            lint.Add(5); lint.Add(1); lint.Add(3);
            lint.Sort();
            int a = lint[2];
            Console.WriteLine(a);
            foreach (int x in lint) Console.Write(x + " ");
            Console.ReadLine();
        }
    }
}
```

У цьому прикладі створюється дві колекції. Перша (`pers`) містить елементи класу `Person`.

Колекція `lint` складається з цілих чисел, причому для роботи з ними не потрібні явні перетворення типу при отриманні елементу з колекції.

Індивідуальні завдання:

(Для виконання індивідуальних завдань № варінта є порядковим номером прізвища студента в списку групи. Усі проекти завантажити у власні репозиторії на Git Hub та виконати їх збірку/build у середовищі Travis CI. Посилання на репозиторії проектів вказати у звіті та обов'язково долучати скріни успішної збірки в Travis CI.)

1. Розробити консольний застосунок, у якому реалізувати клас-прототип List<T> — для зберігання колекції об'єктів класу згідно свого варіанту для роботи з базою даних, що зберігається у текстовому файлі. Передбачити не менше 5 полів різного типу (придумайте самостійно). Початковий масив записів у файлі не менше 3. Для ідентифікації спроби введення з клавіатури некоректних даних описати виключення. Реалізувати методи для:

– додавання записів;

– редагування записів;

– знищення записів;

– виведення впорядкованої за різними параметрами інформації з файла на екран.

Реалізувати заповнення колекції об'єктами за допомогою методу Add, а видалення об'єктів – Remove.

Меню програми реалізувати по натисненню на певні клавіші: наприклад, Enter – вихід, п - пошук, р – редагування тощо.

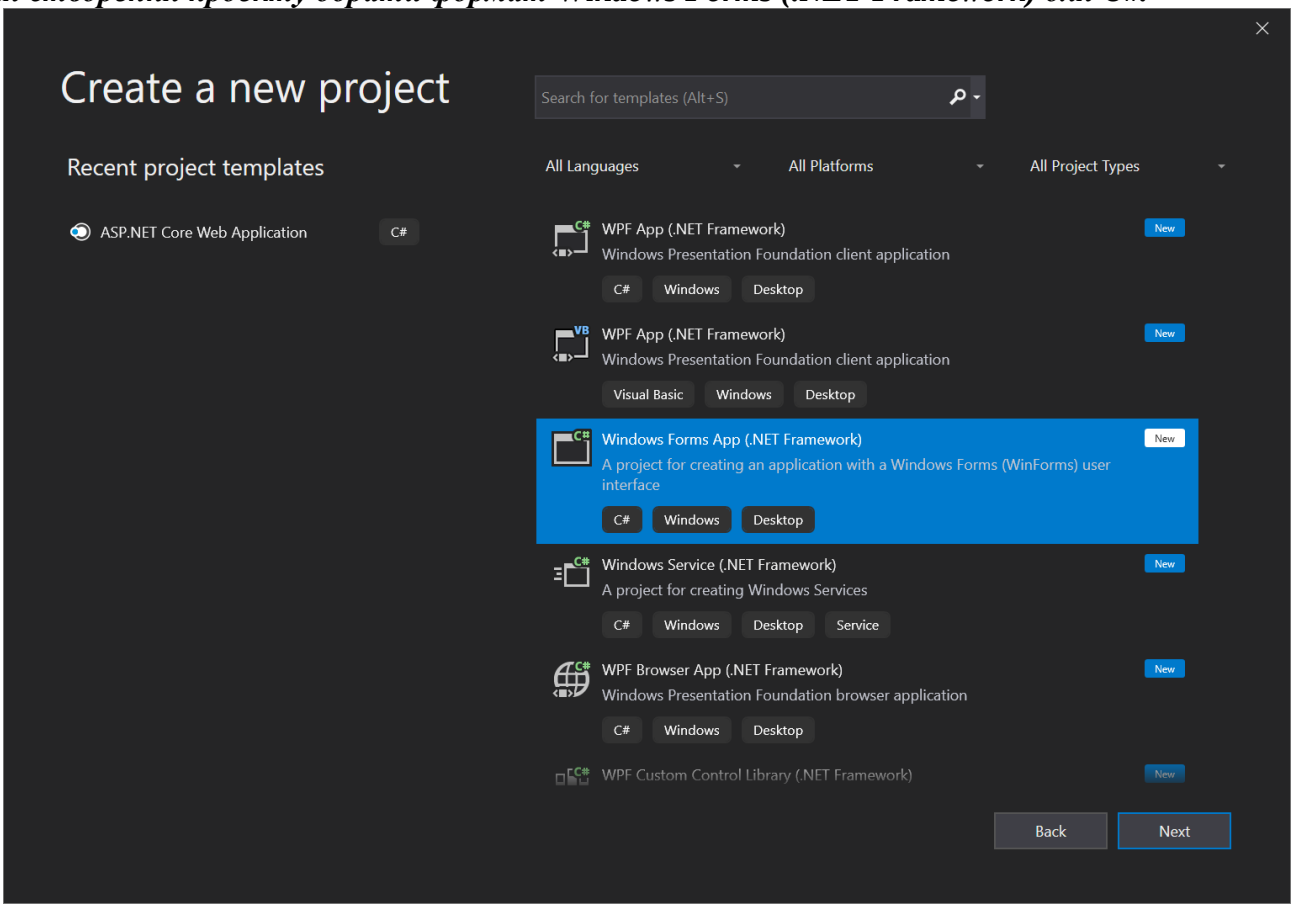
(3 бала)

№ варіанта	Клас	№ варіанта	Клас
1.	Пацієнти сімейного лікаря	13	Профілі клієнтів інтернет-магазину
2.	Викладачі університету	14	Камера схову
3.	Власники квартир ОСББ	15	Склад товарів
4.	Плейлист	16.	Каса продажу квитків
5.	Служба кур'єрської доставки	17.	Успішність студентів
6	Асортимент виробів	18.	Реєстр виборців закріплених за виборчою дільницею
7	Метеорологічні дані	19.	Відділ кадрів
8	Склад автозапчастин	20.	Розклад руху літаків
9	Бібліотека школи	21.	Меню ресторану
10	Розклад пар	22.	Записна книжка
11	Список файлів	23.	ЖРЕПи міста
12	Архів програмного забезпечення	24.	Абоненти кабельної мережі
25.		Рахунки банку	

2. Перетворити консольний застосунок з попереднього завдання у застосунок типу Windows Forms таким чином, щоб меню програми було реалізовано кнопками, а для виведення, редагування, додавання інформації були окремі вікна. Для сортування передбачити можливість вибору параметра сортування на головній формі і відображення результатів сортування у таблиці.

(2 бала)

Для створення проекту обрати формат Windows Forms (.NET Framework) для C#:



Додаткове завдання. <u>Не є частиною лабораторної роботи.</u> Не є обов'язковим для виконання. Оцінюється окремо. Оформлюється окремим звітом. Надає можливість отримати додаткові бали.

Оскільки виконувати будуть не всі студенти, кількість варіантів менша за кількість студентів групи. Вам дозволено обирати собі варіант самостійно. Для цього пишете Ользі Іванівні свої побажання, щоб не було повторень і «випадкових» збігів. Завдання, виконані без попереднього погодження варіанту зі мною, перевірятись і оцінюватись не будуть!!!

Ви можете виконати <u>лише одне</u> додаткове завдання.

3. Реалізувати застосунок типу Windows Forms для зображення предметної області згідно варіанту. Для опису даних використати класи-колекції. При описі класів самостійно визначити необхідні поля, властивості та методи вводу/виводу. Деякі методи класу-предка можуть бути віртуальними і абстрактними.

(8 балів)

1. Створити клас TPrism, який представляє правильну призму і містить методи для знаходження площі поверхні та об'єму. На основі цього класу створити класи- нащадки TPrism3 та TPrism4,

які представляють правильну трикутну та чотирикутну призми. З клавіатури вводиться дані для створення правильної трикутної та чотирикутної призми. На їх основі поступово створити *m* правильних призм (трикутних та чотирикутних), об'єм кожної з яких на 5 більше попередньої. Для трикутних призм знайти сумарний об'єм, а для чотирикутних – суму площ поверхні.

2. Створити клас TNumber з віртуальними методами для знаходження суми цифр та знаходження першої/останньої цифри. На основі цього класу створити класи- нащадки TIntNumber та TRealNumber, у яких реалізовано перевизначені віртуальні методи. Створити *m* об'єктів цілих чисел та *n* об'єктів дійсних чисел (дані згенерувати випадковим чином). Знайти суму перших цифр цілих чисел та суму останніх цифр дійсних чисел.
3. Створити клас TBody, який представляє просторову геометричну фігуру з методами обчислення площі поверхні та об'єму. На основі цього класу створити класи нащадки TParallelepiped та TBall. Випадковим чином створити певну кількість паралелепіпедів та куль, щоб їх сумарна кількість дорівнювала *n* . Знайти сумарну площу поверхонь усіх геометричних тіл.
4. Створити клас TTriangle з віртуальними методами для обчислення площі та периметру. На основі цього класу створити класи, які представляють рівносторонні, прямокутні та рівнобедрені трикутники. Випадковим чином створити певну кількість трикутників кожного виду, щоб їх сумарна кількість дорівнювала *n*. Для рівносторонніх та прямокутних обчислити суму площ, а для рівнобедрених – суму всіх периметрів.
5. Створити клас TQuadrangle, який представляє чотирикутник і містить віртуальні методи для обчислення площі та периметру. На основі цього класу створити класи, які представляють прямокутник, квадрат, паралелограм (квадрат створити на основі прямокутника). Випадковим чином створити певну кількість чотирикутників кожного виду, щоб їх сумарна кількість дорівнювала *n*. Обчислити суму площ прямокутників та квадратів і суму периметрів паралелограмів.
6. Створити клас TMatrix, який представляє матрицю і містить методи для обчислення детермінанта та суми елементів матриці. На основі цього класу створити класи, які представляють квадратні матриці порядку 2, та порядку 3. За допомогою цих класів обчислити вираз

$$S = \left(\sum_{i=1}^{3} \sum_{j=1}^{3} a_{ij} \right) + |A| + |B|,$$

де $A = \|a_{ij}\|_1^3$ – матриця порядку 3, а $B = \|b_{ij}\|_1^2$ – матриця порядку 2.

7. Створити клас TVector, який представляє вектор і місить методи для обчислення довжини вектора та скалярного добутку векторів. На основі цього класу створити класи, які представляють вектори з просторів R^2 та R^3. За допомогою цих класів обчислити значення виразу:

$$S = \langle a,b \rangle + \langle c,d \rangle + |a|,$$

де $a,b \in R^3$, а $c,d \in R^2$.

8. Створити клас TVector, який представляє вектор і місить методи для визначення того, чи є інший вектор паралельним/перпендикулярним до нього та метод знаходження довжини вектора. На основі цього класу створити класи, які представляють вектори з просторів R^3 . У масиві зберегти 3 двовимірні та 4 тривимірні вектори. Знайти суму довжин паралельних до першого по порядку двовимірних векторів та суму перпендикулярних до першого по порядку тривимірних векторів.
9. Створити клас TEquation, який представляє рівняння і містить віртуальні методи для знаходження коренів рівняння та перевірки чи є деяке значення коренем рівняння. На основі цього класу створити класи нащадки, які представляють лінійні рівняння та квадратні рівняння. Випадковим чином згенерувати дані для *n* лінійних рівнянь та *m* квадратних рівнянь. Знайти суму коренів для кожного із видів рівнянь (за умови, що вони існують).
10. Створити клас TSystemLinearEquation, який представляє систему лінійних алгебраїчних рівнянь і містить методи для введення/виведення коефіцієнтів, знаходження коренів та перевірки того, чи є деякий набір чисел розв'язком системи. На основі цього класу створити класи-нащадки, які представляють системи двох та трьох лінійних рівнянь відповідно з двома та трьома невідомими. Випадковим чином згенерувавши дані, знайти розв'язок систем лінійних алгебраїчних рівнянь обох видів.
11. Створити клас TVSeries, який представляє прогресію і містить методи для знаходження *n* -го члена прогресії і знаходження суми перших *n* членів цієї прогресі. На основі цього класу створити класи нащадки, які представляють арифметичні та геометричні прогресії. Випадковим

чином згенерувати дані для *n* прогресій (геометрична, арифметична, геометрична, арифметична, і т.д.). Знайти суму перших *m* членів прогресії, *n* -товий член якої є найбільшим.

12. Створити клас TTriad, який представляє трійку цілих чисел і містить методи для їх збільшення/зменшення на 1. Реалізувати класи нащадки TDate (“число.місяць.рік”) та TTime (“години.хвилини.секунди”). Випадковим чином згенерувати *n* дат та *m* об’єктів-часу. Визначити, які із дат мають значення, що є допустимими, якщо їх трактувати як час. Всі інші дати зменшити на 1.
13. Створити клас TPair, який представляє пару чисел і містить методи для їх збільшення/зменшення на 1. Реалізувати класи нащадки TTime (“години.хвилини”) та TMoney (“гривні.копійки”). Згенерувати поступово випадковим чином *n* пар (час, гроші), де час – тривалість виконання роботи, а гроші – вартість однієї хвилини роботи працівників. Обчислити витрати на виконання кожної із робіт.
14. Створити клас TArray, який представляє одновимірний масив і містить методи введення/виведення, збільшення/зменшення всіх елементів на 1 та знаходження середнього арифметичного. Реалізувати класи-нащадки, що представляють одновимірні масиви з елементами цілого та дійсного типів. Випадковим чином створивши *m* масивів кожного виду, знайти масив, середнє арифметичне елементів якого є найбільшим.
15. Створити клас TLine, що представляє пряму і містить методи для визначення того, чи є інша пряма паралельною/перпендикулярною до неї, чи належить вказана точка прямій. Реалізувати класи-нащадки, що представляє пряму на площині і в просторі. Випадковим чином згенерувати дані для створення *n* прямих у просторі, та *m* прямих на площині. Визначити, чи є серед заданих прямих у просторі така, що є перпендикулярною до всіх інших прямих у просторі.