C# Колекції

Клакович Л.М.

Колекції FCL

- Визначені в просторі імен **System.Collections** та **System.Collections.Generic**
- Більшість колекцій реалізують інтерфейс **ICollection, IComparer, IEnumerable, IList, IDictionary і IDictionaryEnumerator** або їх узагальнені варіанти.
- Рекомендується використовувати **колекції-узагальнення** замість колекцій, оскільки вони підтримують безпеку типів.

Узагальнення в бібліотеці FCL

System.Collections.Generic	System.Collections
List<T>	ArrayList
Dictionary<K,T>	HashTable
SortedList<K,T>, SortedDictionary<K,T>	SortedList
Stack<T>	Stack
Queue<T>	Queue
LinkedList<T> O(1)	-
IList<T>	IList
IDictionary<K,T>	IDictionary
ICollection<T>	ICollection
IEnumerator<T>	IEnumerator
IEnumerable<T>	IEnumerable
IComparer<T>	IComparer
IComparable<T>	IComparable

System.Collections

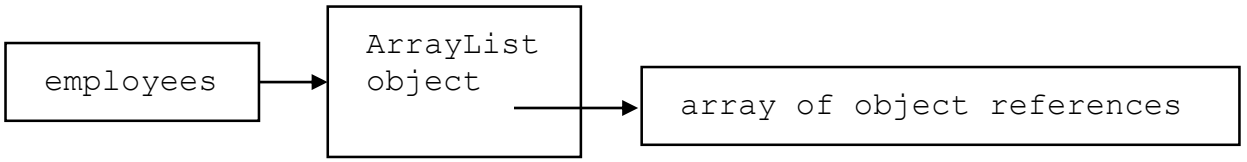
- **System.Collections** namespace
- **ArrayList**, **HashTable**, **SortedList, Queue, Stack**:
  - Колекції можуть містити наперед **невизначену** кількість елементів.
  - Елементи однієї колекції можуть бути **різного типу**.
  - Позиції об’єктів в колекції можуть змінюватися з часом, при додаванні/видаленні нових об’єктів

ArrayList

- **ArrayList**.
  - Містить подібну функціональність як і стандартний масив.
  - Можемо динамічно змінювати розмір при додаванні чи видаленні елементів

create `ArrayList` to store `Employees` →

```
using System.Collections;

class Department
{
  ArrayList employees = new ArrayList();
  ...
}
```



ArrayList

add new elements →

remove →

containment testing →

read/write existing element →

control of memory
in underlying array →

```
public class ArrayList : IList, ICloneable
{
  int  Add   (object value) // at the end
  void Insert(int index, object value) ...

  void Remove  (object value) ...
  void RemoveAt(int    index) ...
  void Clear   () ...

  bool Contains(object value) ...
  int  IndexOf (object value) ...

  object this[int index] { get... set.. }

  int  Capacity { get... set... }
  void TrimToSize() //minimize memory
  ...
}
```

Sort

- Потрібно забезпечити метод порівняння
  - реалізуючи **`IComparable`**
  - Окремо визначивши об”єкт **`IComparer`**
  - Прості типи мають вбудований порівнювач

built-in compare →

separate compare →

```
public class ArrayList : IList, ICloneable
{
  void Sort() ...

  void Sort(IComparer comparer) ...
  void Sort(int index, int count, IComparer comparer) ...
  ...
}
```

ArrayList

- **Переваги `ArrayList`**
  - Підтримує автоматичний **resizing.**
  - Вставка елементів через **insert** (початково масив без елементів)
  - Видалення елементів **(remove)**.
  - Легко використовувати.
- **Недоліки `ArrayLists`**
  - Найбільшим недоліком є швидкість. Дорого коштує розміщення суцільного масиву в динамічній пам’яті

Hashtable

- Представляє колекцію пар ключ/значення (**key/value),** розміщення яких базується на hash-коді ключа.
- Ключі повинні перевизначати методи **GetHashCode()** і **Equals()**.

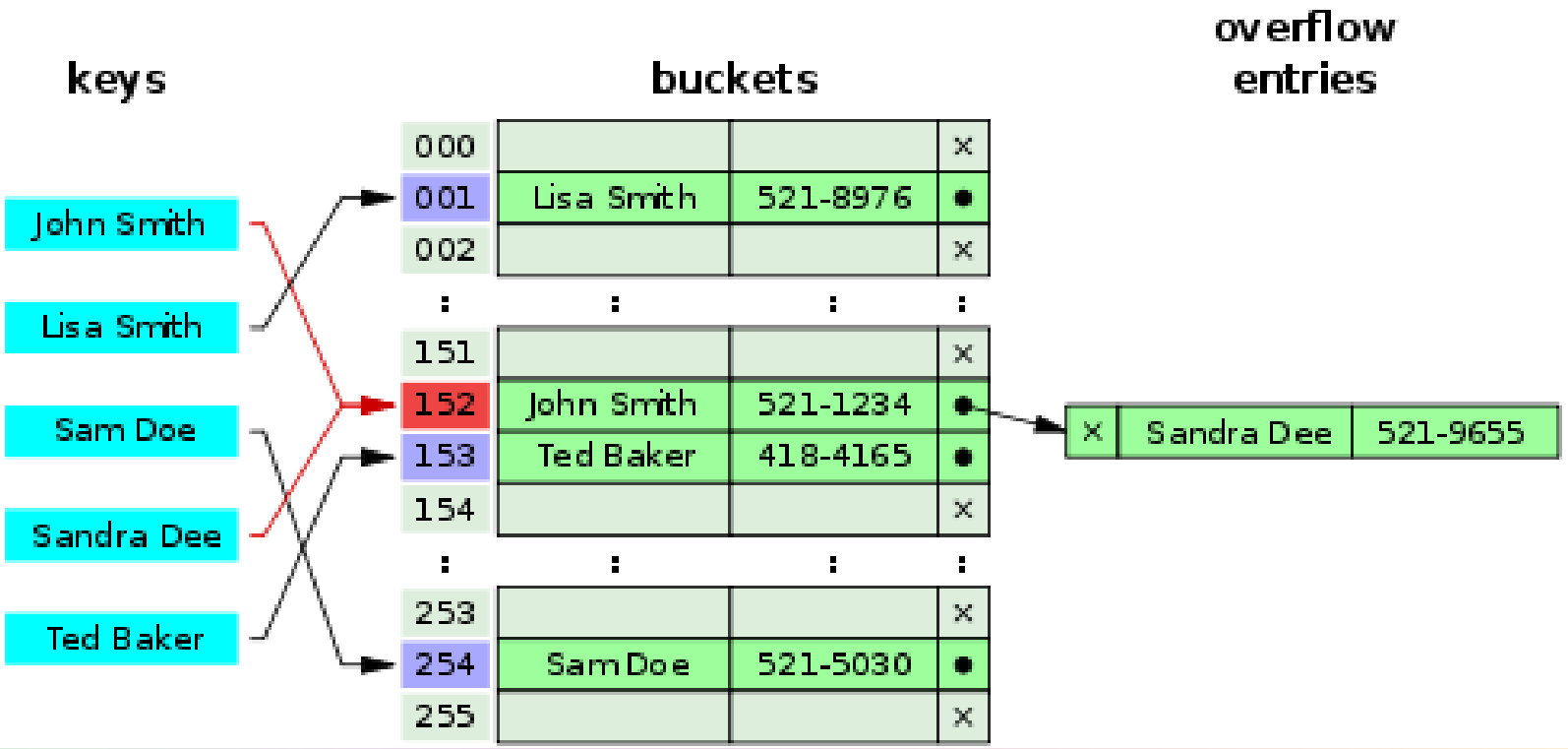
- **Переваги Hashtable:**
  - **Ключі можуть бути числовим, текстовим типами і навіть типом-date**. Не можуть бути null.
  - Легко **додавати та отримувати** елементи.
  - Швидкий перегляд всіх елементів.

create →
add →
update →
retrieve →

```
Hashtable ages = new Hashtable();

ages["Ann"] = 27;
ages["Bob"] = 32;
ages.Add("Tom", 15);

ages["Ann"] = 28;

int a = (int)ages["Ann"];
```

Hash Table

- **Hash function** – функція, яка мапить об’єкт до цілого числа (`Object.GetHashCode()`)
- Використовуючи хеш значення **ключа**, Hash Table визначає **корзину**, де зберігати чи шукати значення.



Hashtable

- **Недоліки:**
  - `Hashtable` повільніші до змін елементів, але швидші до використання в циклі ніж `ArrayList`
  - **Ключі** повинні бути **унікальні**
  - Внутрішнє сортування, яке не використовується.

enumerate entries →

get key and value →

```
Hashtable ages = new Hashtable();

ages["Ann"] = 27;
ages["Bob"] = 32;
ages["Tom"] = 15;

foreach (DictionaryEntry entry in ages)
{
  string name = (string)entry.Key;
  int    age  = (int)   entry.Value;
  ...
}
```

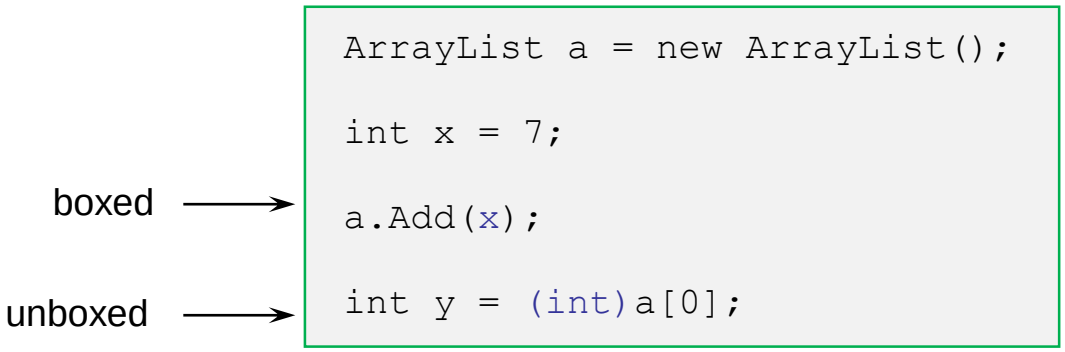
SortedList

- **SortedList –** представляє колекцію пар ключ/значення **(key/value)**, сортованих за ключами
- Доступ до елементів за **ключем** і за **індексом**.
- Внутрішня будова **SortedList** – два масиви
- Додавання елементів відбувається згідно методу Compare() (Icomparer).

```
SortedList stlShippers = new SortedList();

stlShippers["cp"]="Canada Post";

stlShippers["fe"]="Federal Express";

stlShippers["us"]="United State Postal Service";

lblOut.Text = "The full name of shipper = " +

                stlShippers[txtCodeIn.Text];
```

Collections. Недоліки

- Нема перевірки типів при компілюванні
  - Не перешкоджає додаванню елементів небажаних типів
  - Runtime errors!
- Всі елементи зберігаються як object
  - Вимагається приведення типів
  - Продуктивність знижується при цьому




```
ArrayList a = new ArrayList();

int x = 7;

a.Add(x);

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

The Generic Solution

- Відкриті конструйовані типи
  - Класи визначаються без конкретного типу
  - Тип встановлюється під-час утворення об’єкта
  - Гарантується безпека типів підчас компіляції

System.Collections.Generic	System.Collections
List<T>	ArrayList
Dictionary<K,T>	HashTable
SortedList<K,T>, SortedDictionary<K,T>	SortedList
Stack<T>	Stack
Queue<T>	Queue
LinkedList<T> O(1)	-
IList<T>	IList
IDictionary<K,T>	IDictionary
ICollection<T>	ICollection
IEnumerator<T>	IEnumerator
IEnumerable<T>	IEnumerable
IComparer<T>	IComparer
IComparable<T>	IComparable

List<T>

- `[SerializableAttribute]`

```
public class List<T> : IList<T>, ICollection<T>,
        IEnumerable<T>, IList, ICollection, IEnumerable
```

  - **List class** є узагальненням **ArrayList**. Реалізує інтерфейс `IList` з динамічно змінним розміром.
  - List використовує порівняння на **рівність** в методах **Contains**, **IndexOf**, **LastIndexOf**, and **Remove**
  - та **порівняння** при сортуванні у методах **BinarySearch**() та **Sort()**
    - Якщо тип **T** реалізує `IComparable` інтерфейс - `CompareTo()`.

List<T>

```
using System.Collections.Generic;
static void Main()
{
        List<string> langs = new List<string>();
        langs.Add("Java");
        langs.Add("C#");
        langs.Add("C++");
        langs.Add("Javascript");

       Console.WriteLine(langs.Contains("C#"));
       Console.WriteLine(langs[1]);

       langs.Remove("C#");
       Console.WriteLine(langs.Contains("C#"));

       langs.Insert(2, "Haskell");
       langs.Sort();
       foreach(string lang in langs)
          { Console.WriteLine(lang); }
}
```

Dictionary<TKey, TValue>

- **Dictionary**, називають ще асоціативним масивом – це *колекція унікальних ключів і колекція значень*
- Кожен ключ пов'язаний з одним значенням, використовуючи **хеш код** ключа.
- Отримання та додавання даних є дуже швидким.

Dictionary <TKey, TValue>

- Словник, де ми мапимо назви доменів і назви країн:

```
Dictionary<string,string> domains = new Dictionary<string,string>();
domains.Add("de", "Germany");
domains.Add("sk", "Slovakia");
domains.Add("us", "United States");
```

- Отримання значень через ключі:

```
Console.WriteLine(domains["sk"]);
Console.WriteLine(domains["de"]);
Console.WriteLine("Dictionary has {0} items", domains.Count);
```

- Отримання ключів та значень:

```
foreach(KeyValuePair<string, string> kvp in domains)
{
   Console.WriteLine("Key = {0}, Value = {1}", kvp.Key, kvp.Value);
}
```

Queue

- ***Queue*** це *First-In-First-Out* (FIFO) структура даних.
- Черги можуть бути використані для обробки повідомлень
- Методи
  - **Clear**();
  - **Contains**(object obj);
  - **Dequeue**(); Видаляє і повертає елемент
  - **Enqueue**(object obj); Додає елемент
  - **ToArray**();

Queue<T>

```
Queue<string> msgs = new Queue<string>();

        msgs.Enqueue("Message 1");
        msgs.Enqueue("Message 2");
        msgs.Enqueue("Message 3");


        Console.WriteLine(msgs.Dequeue());
        Console.WriteLine(msgs.Peek());
        Console.WriteLine(msgs.Peek());

        Console.WriteLine();

        foreach(string msg in msgs)
        {
            Console.WriteLine(msg);
        }
```

Stack <T>

- ***stack*** це *Last-In-First-Out* (LIFO) структурою даних.
- CLR використовує stack для зберігання локальних змінних в функціях.
- stack також використовується для реалізації калькулятора.

Stack <T>

```
Stack<int> stc = new Stack<int>();

        stc.Push(1);
        stc.Push(4);
        stc.Push(3);
        stc.Push(6);

        Console.WriteLine(stc.Pop());
        Console.WriteLine(stc.Peek());
        Console.WriteLine(stc.Peek());

        Console.WriteLine();

        foreach(int item in stc)
        {
            Console.WriteLine(item);
        }
```

Огляд колекцій

- **Доступ до елементів:**
  - Queue і Queue<T> надають доступ **FIFO** .
  - Stack і Stack<T> надають доступ **LIFO**
  - LinkedList<T> однаковий доступ як з голови так і з хвоста.
  - Решта колекцій надають довільний доступ
- **Доступ до елементів через індекс та ключ:**
  - ArrayList, StringCollection, List<T> доступ за індексом (починаючи з 0)
  - Hashtable, SortedList, ListDictionary, StringDictionary, Dictionary<TKey, Tvalue>, SortedDictionary<TKey, TValue> доступ до елементів через ключ
  - NameObjectCollectionBase, NameValueCollection, KeyedCollection<TKey, Titem> і SortedList<TKey, Tvalue> надають доступ як за індексом так і за ключем.
- **Елементи колекцій містять:**
  - **Одне значення**: колекції, що реалізують інтерфейс IList або IList<T>.
  - **Один ключ і одне значення**: колекції, що реалізують інтерфейс IDictionary або IDictionary<TKey, Tvalue> .
  - **Один ключ з багатьма значеннями**: колекція NameValueCollection.

Огляд колекцій

- **Сортовані колекції:**
  - Колекція Hashtable сортує елементи за hash-кодами.
  - Колекції SortedList, SortedDictionary<TKey, Tvalue>, SortedList<TKey, Tvalue> сортує елементи за ключем, використовуючи реалізацію інтерфейсу IComparer <T>
  - ArrayList, List<T> надають метод **Sort**, що містить реалізацію IComparer (чи його узагальнення) як параметр
- **Швидкий пошук і видобування інформації**
  - ListDictionary є швидшою ніж Hashtable для невеликих колекцій (до 10 елементів).
  - Dictionary<TKey, Tvalue> забезпечує швидший перегляд ніж SortedDictionary<TKey, Tvalue>.
- **Колекції, що приймають тільки рядки**
  - StringCollection (базується на IList) і StringDictionary (базуєтьсяна IDictionary) в просторі імен **System.Collections.Specialized**.

Допоміжні типи

- **KeyValuePair**<K,V> - пара ключ/значення – елементи колекцій, що реалізують інтерфейс IDictionary<TKey, Tvalue> (**Dictionary** і **SortedDictionary**)
  DictionaryEntry (**Hashtable**)
  - властивості Key, Value.

```
foreach (KeyValuePair<int, string> kvp in myDictionary)
{
    Console.WriteLine("Key = {0}, Value =
                                {1}", kvp.Key, kvp.Value);
}
```