Interfaces & C# Collections

By Ira Zavushchak

softserve

Agenda

- Interface declaration
- Interface implementation
- Built-in .Net interfaces
- Task 5.1
- Collections in C#
  - ✔ ArrayList & List<>
  - ✔ Dictionary
  - ✔ Queue
  - ✔ Stack
- Task 5.2
- Homework 5

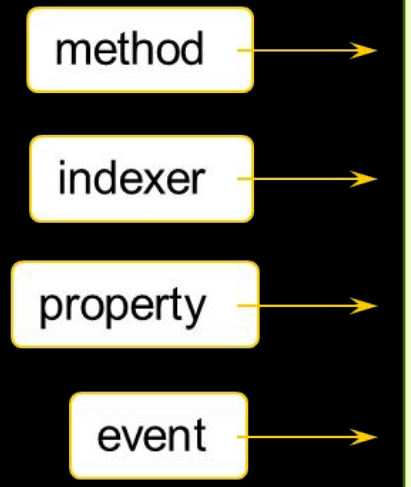
Interface declaration

- **An interface contains definitions for a group of related functionalities that a class or a struct can implement**

```
modificator opt  interface INameOfInterface: listOfInterfaces opt
{
        //declaration of interface members
}
```

- **Interface** Includes only declaration of **method, properties, events, indexers**
- **Interface** can't contain constants, fields, operators, instance constructors, destructors, or types.
- **Interface members** are automatically **public**, and they can't include any access modifiers.
- **Interface members** can't be static.

Interface declaration




```
interface IMyInterface
{
  void Process(int arg1, double arg2);

  float this [int index] { get; set; }

  string Name { get; set; }

  event MouseEventHandler Mouse;
}
```

Interface Implementation

```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

```
class Dragon : IFighter
{
  public void Punch(int side)
  {
    Move(tail, s
  }
  public void Ki
  {
    Move(legs[si
  }
  public void Bl
  {
    Move(legs[Le
    Move(tail, U
  }
}
```

```
IFighter f = new Soldier();
f.Punch(Left);
f.Kick(Right);
f.Block();

IFighter s= new Dragon();
s.Punch(Left);
s.Kick(Right);
s.Block();
...
```

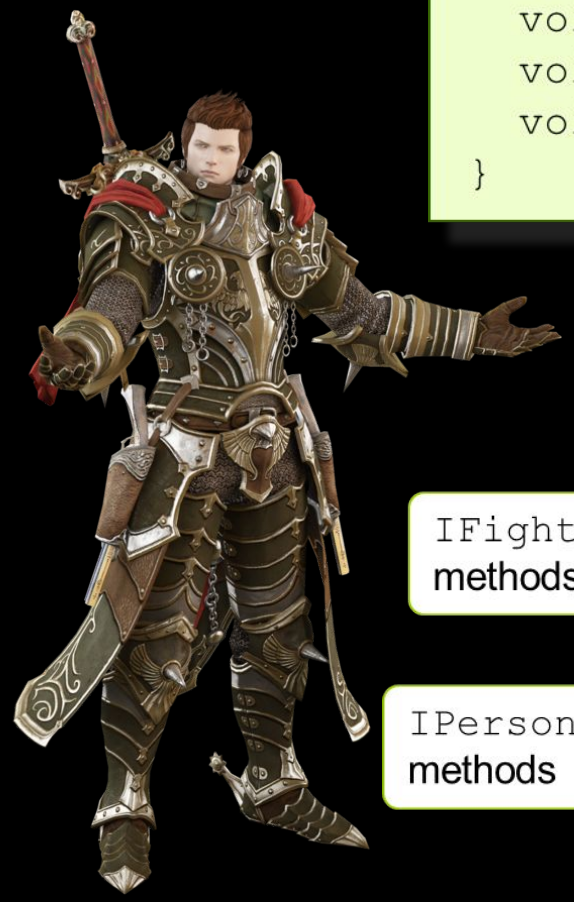
```
class Soldier : IFighter
{
  public void Punch(int side)
  {
    Move(arms[side], Forward);
  }
  public void Kick (int side)
  {
    Move(legs[side], Forward);
  }
  public void Block()
  {
                                  Left ], Up);
                                  Right], Up);

                                  Move(Arm a,Direction d)


                                  Move(Leg l,Direction d)
}
```

erve

Interface Implementation

- Any class or struct that implements the interface *must implement all its members.*
- By using interfaces, we may include *behavior from multiple sources in a class.*
- It is important in C# because the language *doesn't support multiple inheritance of classes.*
- We must use an interface for simulating *inheritance for structs*, because they can't actually inherit from another struct or class.

Interface Implementation



```
interface IFighter
{
  void Punch(int side);
  void Kick (int side);
  void Block();
}
```

```
interface IPerson
{
 string Name { get; set; }
 string Introduce();
}
```

IFighter methods

IPerson methods

```
class Soldier : IFighter, IPerson
{ private string name;
  public void Punch(int side) { ... }
  public void Kick (int side) { ... }
  public void Block() { ... }

  public string Name {get{return name;} set{name = value;}}
  public  string Introduce(){return "My name is"+this.name;}

  ...
}
```

softserve

FCL .Net Interfaces

- **IEnumerable:**

The IEnumerable interface allows foreach-loops on collections. It is often used in LINQ.

- **IDisposable:**

Provides a mechanism for releasing unmanaged resources.

- **ICollection:**

Defines methods to manipulate generic collections.

```
public interface IEnumerable {
    IEnumerator GetEnumerator();
}
```

```
public interface IDisposable {
    void Dispose();
}
```

```
public interface ICollection<T> : IEnumerable<T>, IEnumerable {
    void    Add(T item);
    void    Clear();
    Boolean Contains(T item);
    void    CopyTo(T[] array, Int32 arrayIndex);
    Boolean Remove(T item);
    Int32   Count      { get; } // Read-only property
    Boolean IsReadOnly { get; } // Read-only property
}
```

.Net Library Interfaces

```
public interface IComparable<T>
{
        Int32 CompareTo(T other);
}
```

Value	Meaning
Less than zero	This object is less than the other parameter.
Zero	This object is equal to other.
Greater than zero	This object is greater than other.

```
class Doctor:IComparable<Doctor>
{
    int CompareTo(Doctor other)
{
      return salary-other.salary;
}
...
}
```

```
public static void Main()
{
    Doctor [] doctors= new Doctor [5];
       //… input doctors

    Array.Sort(doctors);
```

Task 5.1

- Develop interface **IFlyable** with method **Fly**().
- Create two classes **Bird** (with fields: name and canFly) and **Plane** (with fields: mark and highFly) , which implement interface IFlyable.
- Create **List** of **IFlyable** objects and add some Birds and Planes to it. Call Fly() method for every item from the list of it.

C# Collections.

Generic collections

softserve

C# Collections

- .NET framework provides specialized classes for data storage and retrieval.
- There are two distinct collection types in C#:
  - The ***standard collections*** from the System.Collections namespace
  - The ***generic collections*** from System.Collections.Generic
  - Generic collections are more flexible and safe, and are the preferred way to work with data.

C# Collections

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

ArrayList

- **ArrayList** is a special array that provides us with some functionality over and above that of the standard Array.
- Unlike arrays, an ArrayList *can hold data of multiple data types*.
- We can dynamically resize it by simply adding and removing elements.

create `ArrayList` ⟶

```
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
  ...
}
```

softserve

List<T>

- ***List<T>*** is a strongly typed list of objects that can be accessed by index.
- It can be found under System.Collections.Generic namespace

```
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); }
}
```

List&ArrayList example


```
using System;
using System.Collections;
using System.Collections.Generic;

namespace Collections
{
    class Program
    {
        static void Main(string[] args)
        {
            // неузагальнена колекція ArrayList
            ArrayList objectList = new ArrayList() { 1, 2, "string", 'c', 2.0f };

            object obj = 45.8;

            objectList.Add(obj);  //додаємо оголошений об'єкт 45,8
            objectList.Add("string2");  //додаємо string2
            objectList.RemoveAt(0); // видалення першого елементу
            foreach (object o in objectList)
            {
                Console.WriteLine(o);
            }
            Console.WriteLine("Загальне число елементів колекції: {0}", objectList.Count);
            Console.WriteLine();
            Console.WriteLine();
            // узагальнена колекція List
            List<string> languages = new List<string>() { "C#", "C", "C++", "Java" };
            languages.Add("Python");
            languages.RemoveAt(1); // видалення другого елементу
            foreach (string s in languages)
            {
                Console.WriteLine(s);
            }

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

softserve

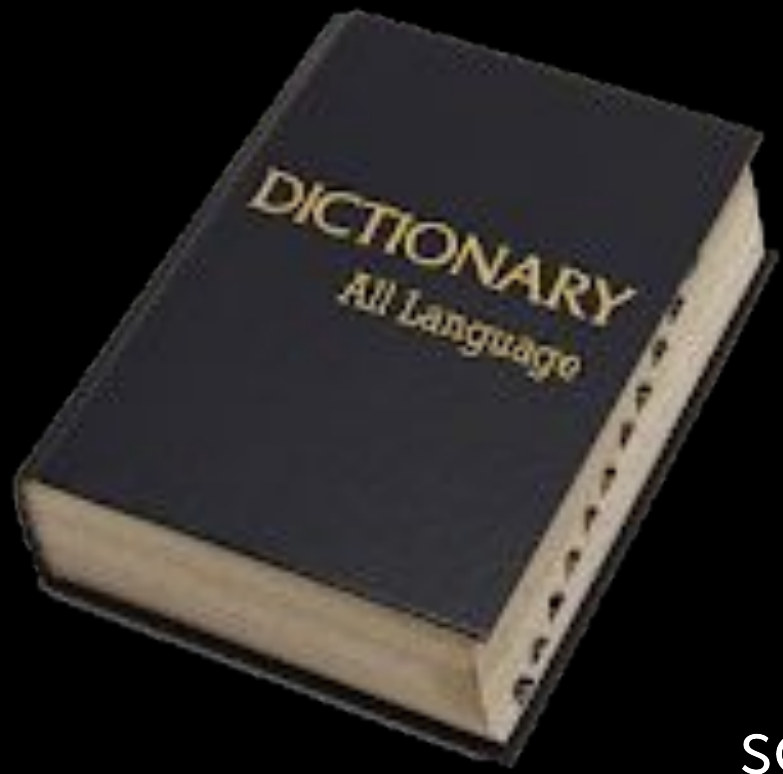
Using IEnumerable interface

```
static void Display(IEnumerable<int> values)
{
     foreach (int value in values)
     {
         Console.WriteLine(value);
     }
}
```

```
static void Main()
{
     int[] values = { 1, 2, 3 };
     List<int> values2 = new List<int>() { 1, 2, 3 };

     // Pass to a method that receives IEnumerable.
      Display(values);
      Display(values2);
}
```

Dictionary

- A **Dictionary**, also called an associative array, is a *collection of unique keys* and a *collection of values*
- Each key is associated with one value.
- Retrieving and adding values is very fast.



Dictionary

- Dictionary where we map domain names to their country names:

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

- Retrieve values by their keys and print the number of items:

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

- Print both keys and values of the dictionary:

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

Dictionary example

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

namespace Dictionary
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<char, Person> people = new Dictionary<char, Person>();
            people.Add('b', new Person() { Name = "Bill" });
            people.Add('t', new Person() { Name = "Tom" });
            people.Add('j', new Person() { Name = "John" });

            foreach (KeyValuePair<char, Person> keyValue in people)
                {
                 // keyValue.Value представляє класс Person
                 Console.WriteLine(keyValue.Key + " - " + keyValue.Value.Name);
                }

            // перебір ключів
            Console.WriteLine();
            foreach (char c in people.Keys)
                {
                 Console.WriteLine(c);
                }

            // перебір значень
            Console.WriteLine();
            foreach (Person p in people.Values)
            Console.WriteLine(p.Name);
            Console.ReadLine();
        }
    }
    class Person
        {
         public string Name { get; set; }
        }
}
```

Queue

- A ***Queue*** is a *First-In-First-Out* (FIFO) data structure.
- The first element added to the queue will be the first one to be removed.
- Queues may be used to process messages as they appear or serve customers as they come.
- Methods:
  - ✔ **Clear**(); removes all elements from the Queue.
  - ✔ **Contains**(object obj); determines whether an element is in the Queue.
  - ✔ **Dequeue**(); removes and returns the object at the beginning of the Queue.
  - ✔ **Enqueue**(object obj); adds an object to the end of the Queue.
  - ✔ **ToArray**(); Copies the Queue to a new array.

Stack

- A ***stack*** is a *Last-In-First-Out* (LIFO) data structure.
- The last element added to the queue will be the first one to be removed.
- The C language uses a stack to store local data in a function. The stack is also used when implementing calculators.

```
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);
        }
```

Queue & Stack example


```
using System;
using System.Collections.Generic;
namespace Collections
{
    class Program
    {
        static void Main(string[] args)
        {
            Queue<int> numbers = new Queue<int>();
            numbers.Enqueue(3); // черга 3  Enqueue - додає елемент в кінець черги
            numbers.Enqueue(5); // черга 3, 5
            numbers.Enqueue(8); // черга 3, 5, 8
            // отримуємо перший елемент черги
            int queueElement = numbers.Dequeue(); //Dequeue: витягує і повертає перший елемент черги
            Console.WriteLine(queueElement);
            Console.WriteLine();
            Queue<Person> persons = new Queue<Person>();
            persons.Enqueue(new Person() { Name = "Tom" });
            persons.Enqueue(new Person() { Name = "Bill" });
            persons.Enqueue(new Person() { Name = "John" });
            // Peek - отримуємо перший елемент черги без його вилучення
            Person pp = persons.Peek();
            Console.WriteLine(pp.Name);
            Console.WriteLine();
            Console.WriteLine("Зараз в черзі {0} людей", persons.Count);
            // тепер в черзі Tom, Bill, John
            foreach (Person p in persons)
            {
                Console.WriteLine(p.Name);
            }
            // Вилучаємо перший елемент з черги - Tom
            Person person = persons.Dequeue(); // тепер в черзі Bill, John
            Console.WriteLine();
            Console.WriteLine(person.Name);
            Console.ReadLine();
        }
    }
    class Person
    {
        public string Name { get; set; }
    }
}
```

```
using System;
using System.Collections.Generic;
namespace Collections
{
    class Program
    {
        static void Main(string[] args)
        {
            Stack<int> numbers = new Stack<int>();
            numbers.Push(3); // в стеку 3
            numbers.Push(5); // в стеку 5, 3
            numbers.Push(8); // в стеку 8, 5, 3

            // так як у вершині стеку буде число 8, то воно вилучається
            int stackElement = numbers.Pop(); // Pop - витягує і повертає перший елемент
            Console.WriteLine(stackElement);
            Console.WriteLine();

            Stack<Person> persons = new Stack<Person>();
            persons.Push(new Person() { Name = "Tom" }); //Push - додає елемент в стек на перше місце
            persons.Push(new Person() { Name = "Bill" });
            persons.Push(new Person() { Name = "John" });

            foreach (Person p in persons)
            {
                Console.WriteLine(p.Name);
            }

            // Перший елемент в стеку
            Person person = persons.Pop(); // тепер в стеку Bill, Tom
            Console.WriteLine();
            Console.WriteLine(person.Name);

            Console.ReadLine();
        }
    }

    class Person
    {
        public string Name { get; set; }
    }
}
```

Task 5.2

- Develop interface **IFlyable** with method **Fly**().
- Create two classes **Bird** (with fields: name and canFly) and **Plane** (with fields: mark and highFly) , which implement interface IFlyable.
- Create **List** of **IFlyable** objects and add some Birds and Planes to it. Call Fly() method for every item from the list of it.

- Declare **myColl** of 10 integers and fill it from Console.

  1) Find and print all positions of element -10 in the collection

  2) Remove from collection elements, which are greater then 20. Print collection

  3) Insert elements 1,-3,-4 in positions 2, 8, 5. Print collection

  4) Sort and print collection

Use next Collections for this tasks: **List** or **ArrayList**

Homework 5

- Create interface **IDeveloper** with property **Tool**, methods **Create**() and **Destroy**()
- Create two classes **Programmer** (with field **language)** and **Builder** (with field **tool**), which implement this interface.
- Create array of **IDeveloper** and add some **Programmers** and **Builders** to it. Call **Create**() and **Destroy**() methods, property Tool for all of it
- Implement interface IComparable for clases and sort array of IDeveloper

- Create Console Application project in VS. In the **Main**() method declare **Dictionary<uint,string>**.
- Add to Dictionary from Console seven pairs (ID, Name) of some persons.
- Ask user to enter ID, then find and write corresponding Name from your Dictionary. If you can't find this ID - say about it to user.