

Інтерфейси .Властивості інтерфейсів.

Інтерфейси схожі на класи, але у них повністю відсутня реалізація. Інакше, інтерфейси - це іменована колекція абстрактних членів. Інтерфейси відображають поведінку, яку даний клас чи структура можуть обирати для реалізації. Інтерфейс – це ще один спосіб реалізації поліморфізму, оскільки в різних класах члени одних і тих же інтерфейсів будуть реалізовані по різному, а значить будуть по різному реагувати на одні й тіж виклики.

Властивості інтерфейсів:

1. Інтерфейси можуть містити лише оголошення методів, властивостей, індексаторів та подій
2. Інтерфейси не можуть містити змінних.
3. При оголошенні членів інтерфейсів не дозволяються модифікатори. Члени інтерфейсів завжди неявно є `public` і не можуть бути `static`.
4. Інтерфейс верхнього рівня може бути оголошений з модифікатором `public` або `internal`, а внутрішній – з тими ж модифікаторами, що і інші члени класу.
5. Не можна напямую створити об'єкт інтерфейсу. Ім'я інтерфейсу може бути типом змінної, але містити змінна може тільки посилання на об'єкт класу, який реалізує цей інтерфейс.

Пиклад оголошення інтерфейсу:

```
interface IMy {  
    int X { get; set; }  
    int this[int i] { get; set;}  
    void fun(string str);  
}
```

Кажуть, що клас реалізує інтерфейс. Кожен клас, який реалізує інтерфейс, повинен повністю визначити кожен із членів інтерфейсу. Тобто реалізація інтерфейсу працює по принципу: все або нічого. Не можна обирати, що реалізувати, а що – ні.

Приклад використання інтерфейсу

Оголошення інтерфейсу

```
interface IFigure
{
    float Perim ();
    float Area();
    float Width {
        get; set;
    }
    float Height {
        get; set;
    }
}
```

Реалізація інтерфейсу

```
class Rectangle : IFigure {
    float width; float height;
    public Rectangle(float width, float height)
    {
        this.width = width;
        this.height = height;
    }
    public float Width{ get => width;
        set => width=value;}
    public float Height { get => height;
        set => height=value; }
    public float Area(){return height * width;}
    public float Perim(){return Width*2+Height*2;}
}
```

Використання в Main.

```
public static void Main(){  
    Rectangle obj1=new Rectangle(5,10);  
    obj1.Width=2;  
    Console.WriteLine("area="+obj1.Area());  
    IFigure obj2=new Rectangle(3,2);  
    obj2.Width=4;  
    Console.WriteLine("area="+obj2.Perim());  
    Console.ReadLine();  
}
```

Оскільки інтерфейси є типами .Net, можна створювати методи, які приймають інтерфейси в якості параметрів.

Приклад коду:

```
public void Metod(IFigure f){  
    float area=f.Area();  
}
```

Приклад виклику методу:

```
Program pr=new Program();  
Rectangle r= Rectangle(2,3);  
pr.Metod(r);
```

Клас наслідує клас та реалізує два інтерфейси.

```
interface IMy1 {
    void f1();
}

interface IMy2{
    void f2();
}

class My{
    public void Show(){
        Console.WriteLine("Show");
    }
}

class My1 : My, IMy1, IMy2 {
    public void f1(){
        Console.WriteLine("f1"); }
    public void f2(){
        Console.WriteLine("f2"); }
}

static void Main(string[] args){
    My1 m = new My1();
    m.Show();
    m.f1(); m.f2();
    Console.WriteLine("m=" + (m is IMy1));
    Console.ReadLine(); }
```


Усунення конфлікту імен за рахунок явної реалізації інтерфейсів.

Оскільки клас може реалізовувати скільки завгодно інтерфейсів, можлива ситуація, коли декілька інтерфейсів містять члени з однаковими іменами. Тому може виникнути необхідність усунення конфліктів на рівні імен.

В цьому випадку реалізація всіх членів буде мати модифікатор доступу `private` і змінити його на інший неможливо. Для виклику таких методів необхідно використати явне приведення типів. Щой продемонстровано в наступному слайді.

```
interface IMy1 {
    void f1(); }
interface IMy2{
    void f1(); }
class My : IMy1, IMy2 {
    void IMy1.f1() {
        Console.WriteLine("IMy1");
    }
    void IMy2.f1() {
        Console.WriteLine("IMy2");
    }
}
```

```
static void Main(string[] args)
{
    My m = new My();
    IMy1 im1 = (IMy1)m;
    im1.f1();
    IMy2 im2 = (IMy2)m;
    im2.f1();
    Console.ReadLine();
}
```

Результат:

```
IMy1
IMy2
```

Стандартні(бібліотечні) інтерфейси IComparable та IComparer.

```
static void Main(string[] args) {  
    int[] arr = { 1, 4, 2, 8, 5 };  
    Array.Sort(arr);  
    foreach (int i in arr)  
        Console.WriteLine("i=" + i);  
}
```

Результат:

i=1

i=2

i=4

i=5

i=8

Відсортуємо масив автомобілів:

```
class Car{
    string name;
    int power;
    public Car(string name, int power) {
this.name = name; this.power = power;
}
    public override string ToString()
    {
        return "name" + name +
"power" + power;
    }
}
```

```
static void Main(string[]
args) {
    Car[] car = {
new Car("Opel", 205),
new Car("Tiggo", 202),
new Car("Audi", 250) };
        Array.Sort(car);
    }
}
```

Результат:

При виконанні
отримаємо exception

Для усунення exception необхідно доповнити код класу:

```
class Car:Comparable
{
    string name;
    int power;
    public Car(string name, int power) {
        this.name = name; this.power = power;
    }
    public int CompareTo(object obj)
    {
        throw new NotImplementedException();
    }
}
```

```
class Car:IComparable{
    string name; int power;
    public Car(string name, int
power) { this.name = name;
this.power = power; }
    public int Power { get => power; }
public int CompareTo(object obj){
    Car car = (Car)obj;
return name.CompareTo(car.name);
}
public override string ToString() {
return "name=" + name + " power="
+ power; }
}
```

```
static void Main(string[] args) {
    Car[] car = { new
Car("Opel", 205), new
Car("Tiggo", 202), new
Car("Audi", 250) };
Array.Sort(car);
    foreach(Car name in car)
Console.WriteLine(name);
Console.ReadLine();
}
```

Результат:

```
name=Audi power=250
name=Opel power=205
name=Tiggo power=202
```

```
class Car:IComparable<Car>{
    string name; int power;
    public Car(string name, int
power) { this.name = name;
this.power = power; }
    public int Power { get => power; }
    public int CompareTo(Car other){
        return
name.CompareTo(other.name); }
    public override string ToString() {
return "name=" + name + "
power=" + power; }
}
```

```
static void Main(string[] args) {
    Car[] car = { new
Car("Opel", 205), new
Car("Tiggo", 202), new
Car("Audi", 250) };
    Array.Sort(car);
    foreach(Car name in car)
        Console.WriteLine(name);
        Console.ReadLine();
}
```

Результат:

name=Audi power=250

name=Opel power=205

name=Tiggo power=202

Додаємо додатковий клас для сортування

```
class SortPower:
    IComparer {
        public int
        Compare(object x,
        object y){
            Car c1 = (Car)x;
            Car c2 = (Car)y;
            return
            c1.Power - c2.Power;
        }
    }
```

```
static void Main(string[] args) {
    Car[] car = { new Car("Opel", 205
    new Car("Tiggo", 202), new
    Car("Audi", 250) };
    Array.Sort(car,new SortPower());
    foreach (Car name in car)
        Console.WriteLine(name);}
name=Tiggo power=202
name=Opel power=205
name=Audi power=250
```


Вносимо зміни в додатковий клас

```
class SortPower:
IComparer<Car> {
public int
    Compare(Car x, Car y) {
        return x.Power-y.Power;
    }
}
```

```
static void Main(string[] args) {
    Car[] car = { new Car("Opel", 205),
                  new Car("Tiggo", 202), new
                  Car("Audi", 250) };
    Array.Sort(car,new SortPower());
    foreach (Car name in car)
        Console.WriteLine(name);}
name=Tiggo power=202
name=Opel power=205
name=Audi power=250
```