Тема 9. Абстрактні класи. Інтерфейси

План практичного заняття

1. Абстрактні класи і методи
2. Приховані класи
3. Види відношень між класами
4. Клас Object – базовий клас ієрархії класів C#
5. Інтефейси та їх властивості

Зміст практичного заняття

Абстрактні класи і методи

При створенні ієрархії об'єктів для виключення коду, що повторюється, часто буває логічно виділити їх загальні властивості в один базовий клас. При цьому може виявитися, що створювати екземпляри такого класу не має сенсу, тому що жодні реальні об'єкти їм не відповідають. Такі класи називають абстрактними.

Визначає методи, які перевизначаються в похідних класах. Інтерфейс для всієї ієрархії.

Методи абстрактного класу можуть бути:

- порожніми (абстрактними) – містити лише оголошення параметрів, без тіла методу;
- непорожніми - виконувати якісь дії, містити тіло функції.

Якщо в класі є хоча б один абстрактний метод, весь клас - abstract. Похідний від абстрактного - abstract, якщо не перекриває всі абстрактні методи базового.

!!!Не можна використовувати конструктор абстрактного класу для створення його об'єкта.

В абстрактному класі визначаються лише загальні призначення методів, які повинні бути реалізовані в похідних класах, але сам по собі цей клас не містить реалізації методів, а тільки їх сигнатуру (тип значення, що повертається, ім'я методу і список параметрів).

При оголошенні абстрактного методу використовується модифікатор *abstract*. Абстрактний метод автоматично стає віртуальним, так що модифікатор virtual при оголошенні методу не використовується.

Абстрактний клас призначений тільки для створення ієрархії класів, не можна створити *об'єкт абстрактного класу.*

Якщо в класі є хоч би один абстрактний метод, весь клас також має бути описаний як абстрактний, наприклад:

Приклад 1. Застосування абстрактних класів (закінчити самостійно)

```
abstract class Person
{
  public string Name { get; set; }
  public Person(string name) { Name = name; }
  public abstract void Display();
}
```

```
class Client : Person
{
  public int Sum { get; set; }
  public Client(string name, int sum) : base(name) { Sum = sum; }
  public override void Display()
  {
    Console.WriteLine($"{Name} имеет счет на сумму {Sum}");
  }
}
class Employee : Person
{
  public string Position { get; set; }
  public Employee(string name, string position) : base(name)
  { Position = position; }
  public override void Display()
  {
    Console.WriteLine($"{Position} {Name}");
  }
}
```

Приклад 2. Переписати програму Student з практичної 4, додати клас Client (банку).

Використати як базовий для них абстрактний клас Person з абстрактим методом, який первизначити в похідних для визначення рейтингу студента та споживача кредиту.

Закриті класи та методи (sealed)

Класи, позначений як sealed, не можете мати похідних, а тикий метод не можна перевизначити.

```
sealed class FinalClass
{
  //тіло класу
}
class DerivedClass : FinalClass  //Неправильно! Помилка компіляції
{
  //тіло класу
}
```

Більшість вбудованих типів даних описана як sealed.

Визначення інтерфейсу

Термін інтерфейс в програмуванні має різні значення:

- інтерфейс користувача – засіб взаємодії користувача з програмою (консольний -, Windows, Web);
- інтерфейс між модулями програми – формальні і фактичні параметри методів;
- інтерфейс, як відкрита частина класу – поля і методи з модифікатором public;
- *інтерфейс, як окремий випадок абстрактного класу*, всі методи якого абстрактні.

Розглянемо останній випадок.

Отже, інтерфейс мітить лише оголошення абстрактних методів, шаблони властивостей і індексаторів та подій, які мають бути реалізовані в похідних класах. Інтерфейс не може містити константи, поля, операції, конструктори, деструктори, типи та будь-які статичні елементи

```
[атрибути][специфікатори] interface ім’я[ : базові]
тіло_інтерфейсу[;]
```

Інтерфейс може успадковувати методи декількох інтерфейсів, в цьому випадку предки перераховуються через кому.

Приклад. Методи і властивість – лише оголошені, але не реалізовані.

```
interface IAction
{
  void Draw();
  int Attack(int a);
  void Die();
  int Power
  {
    get;
  }
}
```

Інтерфейс визначає поведінку, яка реалізується класами, тобто які успаковують цей інтерфейс. Кожен клас може визначити елементи інтерфейсу по-своєму.

Призначення інтерфейсів – забезпечення множинного наслідування класів та однаковий спосіб використання об’єктів класів, які його реалізують.

Використання інтефейсів є ще однією реалізацією ідеї поліморфізму: об'єкти різних класів по-різному реагують на виклики одного і того ж методу.

Відмінність інтерфейсів від абстрактних класів

Синтаксична відмінність інтерфейсу від абстрактного класу – абстрактні методи інтерфейсу оголошуються без вказівки модифікатора доступу. Елементи інтерфейсу за замовчуванням мають специфікатор доступу public і не можуть мати явно заданих специфікаторів.

Семантична відмінність - в похідних класах повинні бути реалізовані всі методи інтерфейсу. Клас, похідний від інтерфейсу, має реалізувати всі методи інтерфейсу. Клас, в списку базових якого є інтерфейс, повинен визначати всі елементи інтерфейсу;

Нащадок абстрактного класу може й не визначати частину абстрактних методів базового (тоді похідний клас - абстрактний). Абстрактний клас є початковим етапом проектування класу, який в майбутньому отримає конкретну реалізацію. *Класи, які реалізують абстрактні методи абстрактного класу є нащадками одного базового* (не Object).

Інтерфейс вказує лише додаткові властивості класу, який той має реалізувати. Один і той самий інтерфейс дозволяє описувати властивості, які можуть мати різні *класи, які не обов’язково є нащадками одного класу* (не Object).

Інтерфейси та успадкування

Якщо набір різних дій має сенс лише для певної ієрархії класів, то їх краще реалізувати як віртуальні методи абстрактного базового класу. Однакові дії в межах ієрархії бажано повністю визначити в базовому класі.
Інтерфейси вартовикористовувати для задання спільних властивостей класів з різних ієрархій.
Інтерфейс може не мати або мати скільки завгодно інтерфейсів-предків, в останньому випадку він успадковує всі елементи всіх своїх базових інтерфейсів.
Клас може мати декілька базових інтерфейсів і має реалізувати всі їхні методи.
Сигнатури методів в інтерфейсі і реалізації повинні співпадати.
Для реалізованих елементів інтерфейсу в класі використовується специфікатор public.

Приклад 1. Розглянемо ще один приклад. Свтворимо класи Person, Student та Teacher, де реалізуємо властивості та методи інтерфейсу IRating.

Файл IRating.cs

```
namespace PersonExample
{
  public interface IRating
  {
    string Name { get; set; }
    string Role { get; set; }
    void Dispaly(double rating);
    string GetRole(int course);
  }
}
```

Створимо новий клас Person похідний від інтерфейсом IRating. В класі створимо конструктор і реалізацію шаблонів властивостей інтерфейсу IRating.

Файл Person.cs

```
namespace PersonExample {
  class Person
  {
    private string name;
    private string role;
    public Person() { }
    public Person(string name, string role)
    {
    this.name = name;
    this.role = role;
    }
    public string Name  {  get => name; set => name = value; }
    public string Role  {  get => role;  set => role=value;
    }
  }
}
```

Створимо новий клас Student, похідний від класу Person з інтерфейсом IRating. В класі Teacher створимо конструктор і реалізацію методів інтерфейсу IRating.

Файл Student.cs

```
using System;
namespace PersonExample
{
  class Student : Person, IRating
  {
```

```
    int age;
    private string department;
    private string group;
    private int course;

    public Student(string name, int age, string role, string department,
     string group, int course) : base(name, role)
    {
    this.age = age;
    this.department = department;
    this.group = group;
    this.course = course;
    }

    public int Age { get => age; set => age = value; }
    public int Course  {  get => course; set => course = value;  }
    public string Department {  get => department;  set => department = value;  }
    public string Group {  get => group;  set => group = value; }

    public string GetRole(int course) => (course <= 4) ? "бакалавр" : "магістр";
    public void Dispaly(double rating)
    {
    if (rating >= 82) Console.WriteLine("Привіт відмінникам");
    else
    {
      if (rating <= 45) Console.WriteLine("Перездача! Треба краще вчитися!");
      else      Console.WriteLine("Можна вчитися ще краще!");
    }
    }
  }
}
```

Створимо новий клас Teacher, похідний від класу Person з інтерфейсом IRating. В класі Teacher створимо конструктор і реалізацію методів інтерфейсу IRating.

Файл Teacher.cs

```
using System;
namespace PersonExample
{
  class Teacher : Person,IRating
  {
    string cathedra;

    public Teacher(string name, string role):base(name,role)
    {
    }
    public string Cathedra
    {
    get => cathedra;
    set => cathedra = value;
    }
    public string GetRole(int course)
    {
    switch (course)
    {
      case 1: return ("Не читаю");
      case 2: return ("Matlab");
      case 3: return ("C#");
      case 4: return ("Технологія програмування");
      case 5: return ("Semantic web and XML");
      case 6: return ("Парадигми програмування");
      default: return ("Неправильно заданий курс");
    }
```

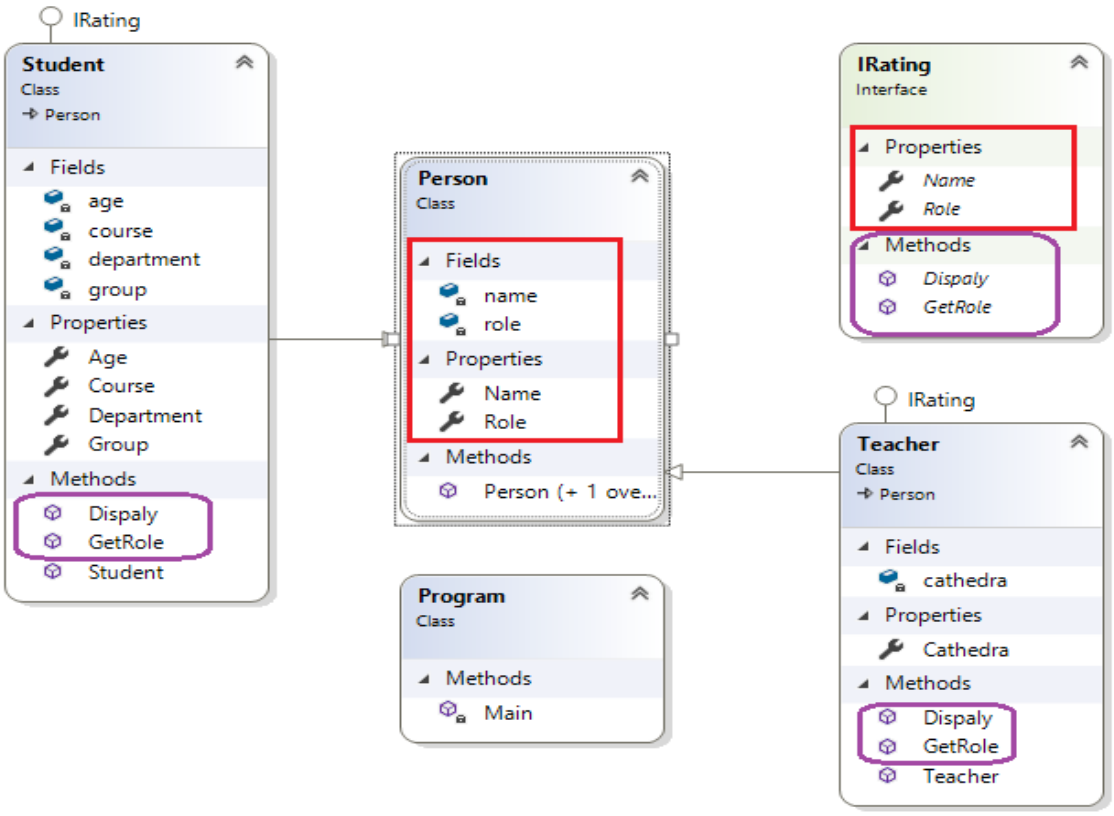
```
    }
    public void Dispaly(double rating)
    {
    Console.WriteLine("Рейтинг викладача" + rating);
    }
  }
}
```

В класі Program напишемо код для перевірки роботи класів.

Файл Program.cs

```
using System;
namespace PersonExamp     le
{
  class Program
  {
    static void Main(string[] args)
    {
    //Дані про студента
    Student student = new Student("Іваненко",20,"студент", "КЕПІТ", "ІПЗ-17", 3);
    Console.WriteLine("Дані про студента");

    Console.WriteLine($"Прізвище - {student.Name}");
    Console.WriteLine($"Вік - {student.Age}");
    Console.WriteLine($"Ви ще - {student.GetRole(student.Course)}");
    Console.WriteLine($"Факультет - {student.Department}");
    Console.WriteLine($"група - {student.Group}");
    Console.WriteLine($"курс - {student.Course}");
    Console.WriteLine($"Ваш рейтинг?");
    double rating = Convert.ToDouble(Console.ReadLine());
    student.Dispaly(rating);

    Console.WriteLine("\n\nДані про викладача");
    Teacher teacher  = new Teacher("Петренко А.В.", "доцент");
    teacher.Cathedra = "Програмної інженерії";

    Console.WriteLine($"Прізвище:  " + teacher.Name);
    Console.WriteLine($"Посада:  {teacher.Role}");
    Console.WriteLine($"Кафедра: {teacher.Cathedra}");

    teacher.Dispaly(100);   // Інтерфейсний метод
    Console.WriteLine("Які дисципліни ви викладаєте?");
    Console.WriteLine("На якому курсі?");
    int cource = int.Parse(Console.ReadLine());

    string disc = teacher.GetRole(cource);
    Console.WriteLine($"на {cource} курсі дисципліну {disc}");
    Console.ReadLine();
    }
  }
}
```

Побудувати діаграму класів



Приклад 2 (самостійно). Реалізувати ієрархію класів з використанням інтерфейсу так, як показно на малюнку

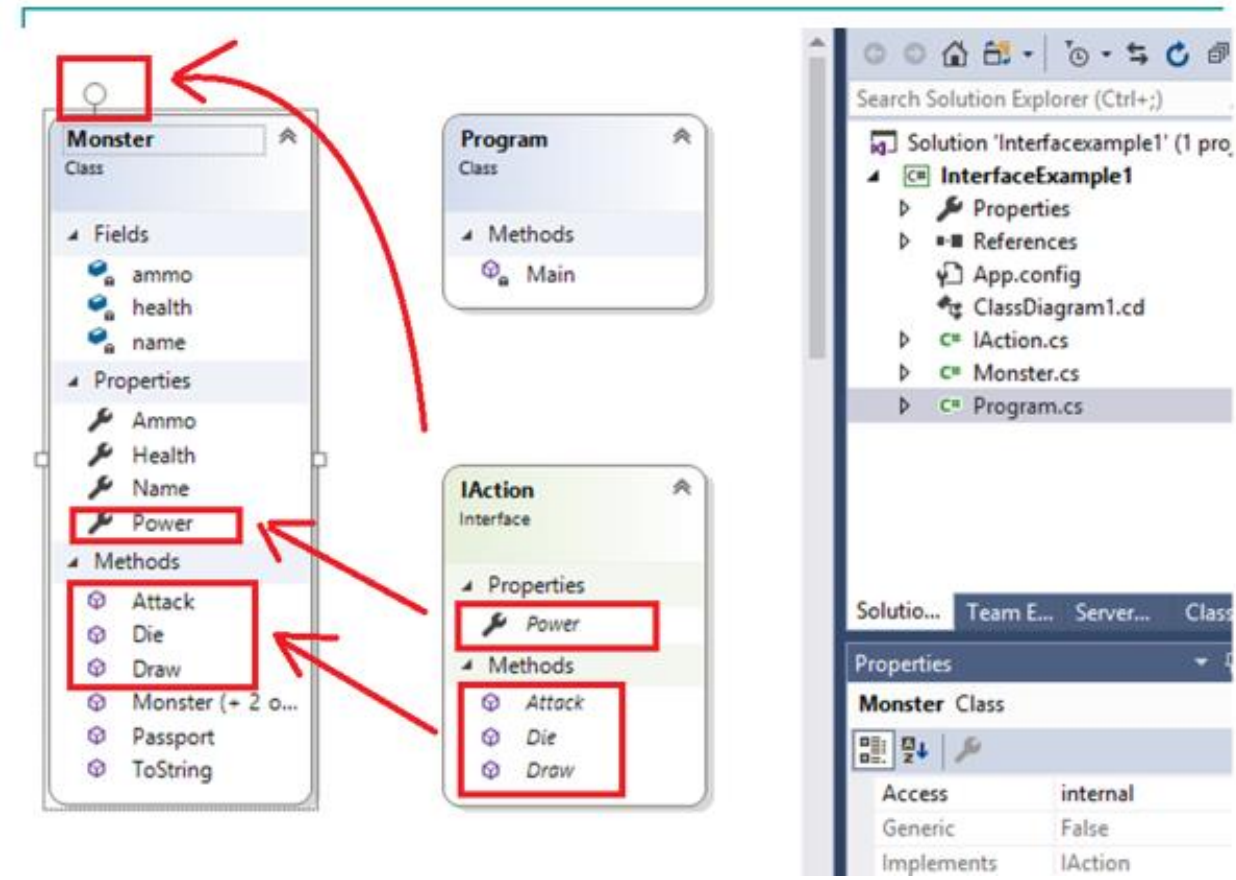
```
interface IAction
{
  void Draw();
  int Attack(int a);
  void Die();
  int Power { get; }
}
```

```
class Monster : IAction
{
    string name;      // private поля
    int health, ammo;
    public Monster()      // конструктор...
    public Monster(string name) ...
    public Monster(int health, int ammo, string name)...
    public int Health...
    public int Ammo...
    public string Name...
    public void Passport()              // метод...
    public override string ToString()...

    public void Draw() { Console.WriteLine("Тут був " + name); }
    public int Attack(int _ammo){
        ammo -= _ammo;
        if (ammo > 0) Console.WriteLine("Ба-бах!"); else ammo = 0;
        return ammo;
    }
    public void Die()
      {  Console.WriteLine("Monster " + name + " RIP"); health = 0; }
    public int Power { get { return ammo * health; }
    }
}
```

Головний блок має бути таким

```
class Program
{
  static void Main(string[] args)
  {
    // об'єкт класса Monster
    Monster Holum = new Monster(50, 50, "Голум");
    Vasyl.Draw();  //результат - Тут був Голум

    // об'єкт типу інтерфейс
    IAction Actor = new Monster(10, 10, "Орк");
    Actor.Draw();  // результат - Тут була Марія

    Console.ReadKey();
  }
}
```

Ієрархія має бути такою



Інтерфейси і властивості

В інтерфейсі можна оголошувати тільки методи, та шаблон властивості з методами get і set, що забезпечують доступ до поля. Методи доступу до полів, успадкованих від інтерфейсу, можна реалізувати як відкриті так і як закриті.

```
using System;
namespace IFieldsConsole
{
  interface IFields
```

```
  {
    string Name { get; set; }
    int Age { get; }
  }

  // Клас, що успадковує інтерфейс IFields
  // Має поля Name і Age
  // доступ до поля Name відкритий клієнтам класу
  // доступ до поля Age закритий і відкритий з перейменуванням!
 class TwoFields : IFields
  {
    string name;
    int age;
    public TwoFields()
    {
    name = "Captain Custo";
    age = 37;
    }
    public TwoFields(string name, int age)
    {
    this.name = name;
    this.age = age;
    }
    public string Name
    {
    get { return name; }
    set { name = value; }
    }
    int IFields.Age
    {
    get { return age+30; }
    }
    public int GetAge()
    {
    return age-10;
    }
 }
class Program
 {
  static void Main(string[] args)
 {
   Console.WriteLine("Робота з об'єктом класу TwoFields!");
   TwoFields twofields = new TwoFields();

   // twofields.Age  - відсутній, тому .GetAge()!
  Console.WriteLine("Ім'я: {0}, Вік: {1}",twofields.Name, twofields.GetAge());

  twofields.Name = "Captain Nemo";
  Console.WriteLine("Робота з інтерфейсним об'єктом  IFields!");
  IFields ifields = (IFields)twofields;
  Console.WriteLine("Ім'я: {0}, Вік: {1}", ifields.Name, ifields.Age);
  Console.ReadLine();
 }
 }
}
```

Властивості інтерфейсів та їх успадкування

До інтерфейсу можуть бути використані специфікатори new, public, protected, internal і private. Специфікатор new застосовується для вкладених інтерфейсів і має такий самий зміст, як і відповідний модифікатор методу класу.

Інші специфікатори управляють видимістю інтерфейсу. За замовчунням інтерфейс доступний лише із збірки, в якій він описаний (internal).

За допомогою ключового слово **new** у інтерфейсі-нащадку вказують елементи, які приховують відповідний елемент базового інтерфейсу (аналогічній як і для класів).
Можна звертатися до інтерфейсу як через об'єкт класу, так і через об'єкт типу інтерфейсу. Клас успадковує всі методи свого предка, у тому числі ті, які реалізували інтерфейси. Він може перевизначити ці методи за допомогою специфікатора **new**, але звертатися до них можна буде лише через об'єкт класу. Якщо використовувати для звернення посилання на інтерфейс, викликається неперевизначена версія:
Приклад.

```
class Base : IBase
    {
    public void A() { ... }
    }

    class Derived : Base
    {
    new public void A() { ... }
    }

    Derived d = new Derived();
    d.A();        // викликається Derived.A();
    IBase id = d;
    id.A();         // викликається Base.A();
```

Звернення до методу класу через об'єкт типу інтерфейс

Об’єктам типу інтерфейс можна присвоїти посилання на об'єкти різних класів, що підтримують цей інтерфейс.

Приклад. Є метод з параметром типу інтерфейс IAction. На місце цього параметра можна передавати будь-який об'єкт, что реалізує інтерфейс:

```
    static void Act( IAction A )
    {
       A.Draw();
    }
    static void Main()
    {
      Monster holum = new Monster( 50, 50, “Голум" );
      Act(holum);
     }
```

Явне задання імені інтерфейсу

Ім’я вказується перед реалізованим елементом. Специфікатори доступу не вказуються. Можна звертатися - лише через об'єкт типу інтерфейс:

Приклад.

```
class Monster : IAction
{
  int IAction.Power { get { return ammo * health; } }
  void IAction.Draw() { Console.WriteLine("Тут був " + name); }
}

IAction Actor = new Monster(10, 10, “Голум" );
Actor.Draw(); // звертання через об’єкт типу інтерфейса

// Monster holum = new Monster( 50, 50, "Голум" );
// holum.Draw();   помилка!
```

Відповідні методи не входять в інтерфейс класу. Таким спсобом “приховують” елементи інтерфейсу непотрібні кінцевому користувачеві і уникають конфліктів при множинному успадкуванні.

Приклад. Нехай клас **Monster** підтримує два інтерфейси: один для управління об'єктами, а інший для тестування:

```
interface ITest
{
  void Draw();
}

interface IAction
{
  void Draw();
  int Attack(int a); …
}
class Monster : IAction, ITest
{
  void ITest.Draw()
  {
    Console.WriteLine("Testing " + name);
  }
  void IAction.Draw()
  {
    Console.WriteLine("Testing " + name);
  }
}

//Операція перетворення типу
Monster Vasyl = new Monster(50, 50, "Василь");
((ITest) Vasia).Draw();   // результат: Тут був Василь
((IAction) Vasia).Draw();   // результат: Testing Василь
```

Операції is і as

is - перевірка, чи підтримує об'єкт підтримує інтерфейс.

true - можна перетворити до заданого типу, **false** - інакше.

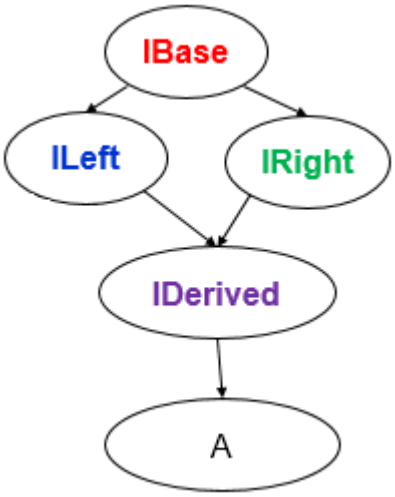
```
if ( об’єкт is тип )
{
  // “претворити”  об’єкт до типу
  // дії с перетвореним об’єктом
}
```

as – **перетворення** до **типу**, якщо неможливе, повертає ***null***

```
static void Act(object A)
{
  IAction Actor = A as IAction;
  if (Actor != null) Actor.Draw();
}
```

Застосовуються як до інтерфейсів, так і до класів.

Приклад. Перевизначення методу




```
interface IBase
{
  void F(int i);
}
interface ILeft : IBase
{
  new void F(int i);  /* Перевизначення методу F */
}
interface IRight : IBase
{
  void G();
}
interface IDerived : ILeft, IRight { }

class A
{
  void Test(IDerived d)
  {
    d.F(1);     // Викликається ILeft.F
    ((IBase)d).F(1);  // Викликається IBase.F
    ((ILeft)d).F(1);  // Викликається ILeft.F
    ((IRight)d).F(1); // Викликається IBase.F
  }
}
```

Метод **F** із інтерфейса **IBase** приховується інтерфейсом **ILeft**, хоча в ланцюжку **Iderived** — **Iright** — **IBase** він не перевизначається.

Практичні завдання

1. Закінчита приклад 2 (Монстри).
2. Побудувати ієрархію класів: **Повітряний транспорт**, літак, вертоліт, транспортний літак з використанням інтерфейсів з методами та властивостями. Під час опису властивостей використовувати синтаксис C# 6.0