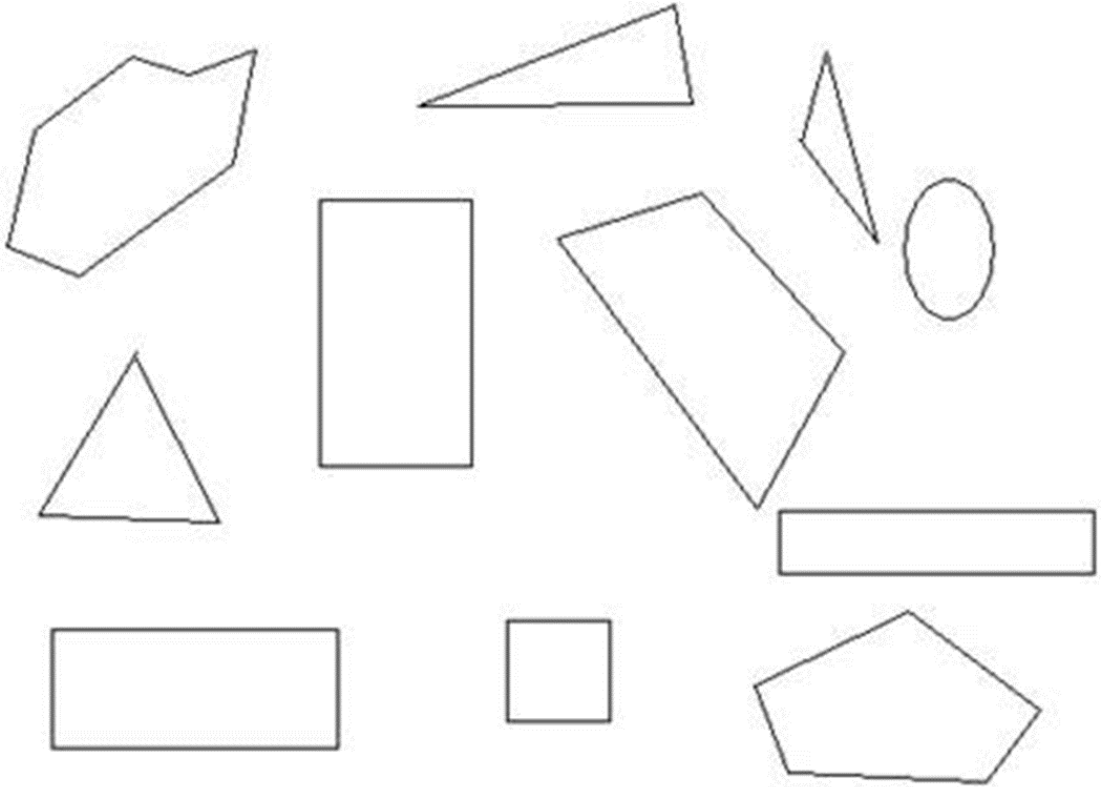
# Знайомство з ООП. Поглиблене вивчення мови C#

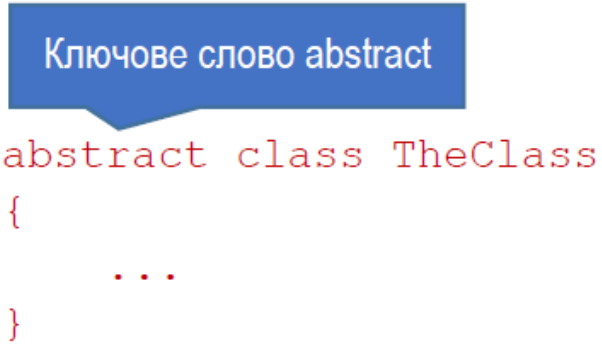
# Тема 3. Абстрактні класи та інтерфейси

# Геометричні фігури мають площу



# Абстрактні класи

- Абстрактні класи – це класи, призначені виключно для спадкування їх іншими класами. Створити екземпляр такого класу неможливо.
- В абстрактному класі описують (але не реалізують) найзагальніші характеристики об'єкта, які повинні бути уточнені шляхом конкретної їх реалізації у похідному класі.
- Абстрактний клас оголошують з модифікатором abstract:



# Абстрактні класи

Якщо в класі є хоча б один абстрактний метод чи властивість, такий клас оголошується як абстрактний. Для цього використовують слово abstract.

```
abstract class [ім’я_класу]  {
   //тіло
}
```

Такий клас має певні особливості:

- не можна створювати екземпляри (об'єкти) абстрактного класу;
- абстрактний клас може містити як абстрактні методи/властивості, так і звичайні;
- у класі спадкоємці повинні бути реалізовані всі абстрактні методи і властивості, оголошені в базовому класі.

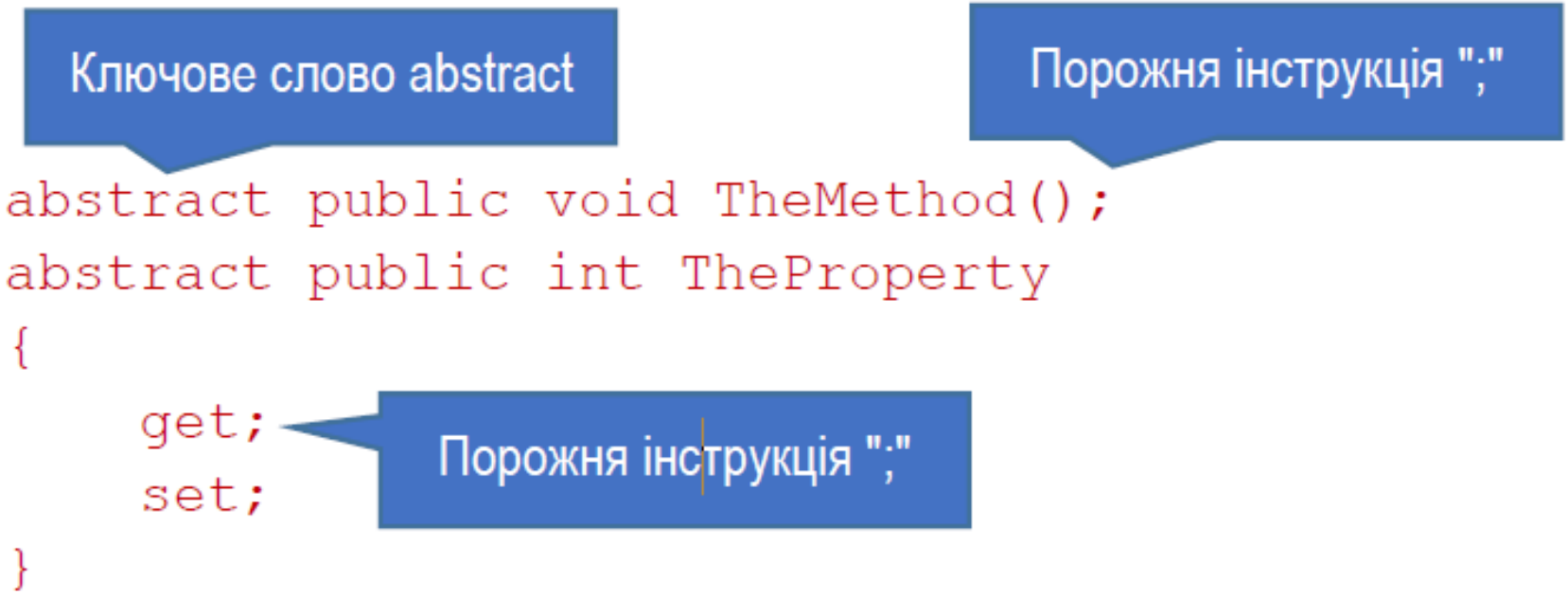
# Приклад абстрактного класу «Фігура»

```
abstract class Shape
{
    7 references
    public abstract double Area();
}
```

# Абстрактні елементи

- Абстрактний елемент класу – це елемент, призначений для заміщення у похідному класі. Абстрактними можуть бути методи та інші функціональні елементи (властивості, події, індексатори). Елементи даних класу (поля класу) не можуть бути абстрактними.
- Абстрактний елемент позначають модифікатором abstract. Він не повинен мати реалізації: його код представляють однією порожньою інструкцією (крапкою з комою):

# Абстрактні елементи



# Абстрактні елементи

- Абстрактні елементи класу можна оголошувати тільки в абстрактному класі.
- Хоча вони і мають бути заміщені в похідному класі з використанням директиви override, але при їх оголошенні не можна використовувати модифікатор virtual.
- Віртуальні елементи, на відміну від абстрактних, повинні мати код реалізації.
- У похідному класі віртуальні елементи можуть бути заміщеними, а абстрактні – повинні бути заміщеними.

# Абстрактні методи та властивості

- Модифікатор abstract в оголошенні методу чи властивості дозволяє вказати, що цей метод чи властивість не містять реалізації.
- Абстрактні методи надають такі можливості:
  - Абстрактний метод неявно є віртуальним методом.
  - Оголошення абстрактних методів допускаються лише у абстрактних класах.
  - Оскільки оголошення абстрактного методу не надає фактичної реалізації, тіло методу відсутнє, а оголошення методу закінчується крапкою з комою і фігурних дужок після назви методу немає.
  - При оголошенні абстрактного методу не можна використовувати модифікатори static або virtual.

# Абстрактні методи

## Абстрактний метод

[модифікатор доступу] abstract [тип] [ім'я методу] ([аргументи]);

Реалізація абстрактного методу в класі спадкоємця відбувається так само, як і перевизначення методу – за допомогою ключового слова **override**:

```
[модифікатор доступу] override [тип] [ім'я методу] ([аргументи])
{
  // реалізація методу
}
```

```
abstract class Shape
{
    5 references
    public abstract double Area();
}
```

Абстрактний метод

```
4 references
class Rectangle:Shape
{
    double a, b;
    2 references
    public Rectangle(double a, double b)
    {
        this.a = a;
        this.b = b;
    }

    5 references
    public override double Area()
    {
        return a * b;
    }
}
```

Перевизначення абстрактного методу

# Абстрактні класи, методи та властивості

## Абстрактні властивості

Створення абстрактних властивостей не сильно відрізняється від методів:

[модифікатор доступу] abstract [тип] [ім'я властивості] { get; set; }

Реалізація в класі-спадкоємці:

```
[модифікатор доступу] override [тип] [ім'я властивості]
{
  get { тіло аксессора get }
  set { тіло аксессора set }
}
```

```
// Abstract class
abstract class BaseClass
{
    protected int _x = 100;
    protected int _y = 150;

    // Abstract method
    public abstract void AbstractMethod();

    // Abstract properties
    public abstract int X { get; }
    public abstract int Y { get; }
}
```

```
class DerivedClass : BaseClass
{
    public override void AbstractMethod()
    {
        _x++;
        _y++;
    }

    public override int X   // overriding property
    {
        get
        {
            return _x + 10;
        }
    }

    public override int Y   // overriding property
    {
        get
        {
            return _y + 10;
        }
    }
```

Перевизначення абстрактного методу

Перевизначення абстрактної властивості

Перевизначення абстрактної властивості

# Інтерфейси

- Інтерфейс – це посилальний тип, який задає множину функціональних елементів, але не реалізовує їх. Інтерфейс можуть реалізовувати (впроваджувати) класи та структури. Семантично інтерфейси схожі на абстрактні класи, але жоден метод інтерфейсу не може мати реалізації.
- Інтерфейс показує, що клас повинен робити, але не визначає, як саме. При впровадженні інтерфейсу у клас мають бути реалізовані всі методи інтерфейсу, але кожен клас може реалізувати їх по-іншому.

# Різниця між абстрактними класами та інтерфейсами


FIAT


TOYOTA


HONDA


Ferrari


NISSAN

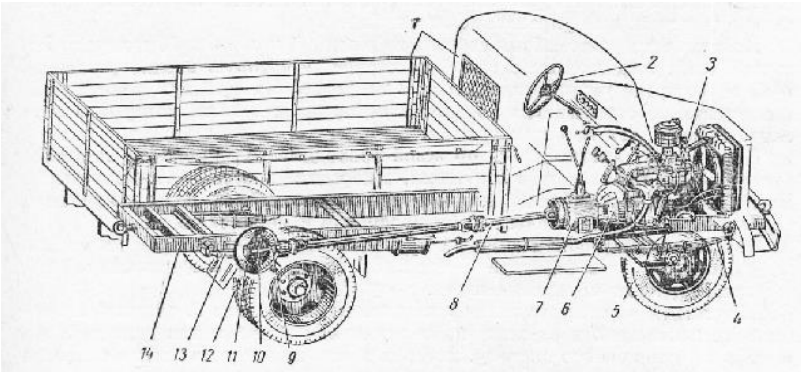
# Абстрактне визначення автомобіля

- Автомобіль – це технічний транспортний засіб, що має як мінімум 4 колеса, двигун, що приводить у рух ці колеса та кермо, що дозволяє керувати рухом автомобіля.

# Абстрактний автомобіль

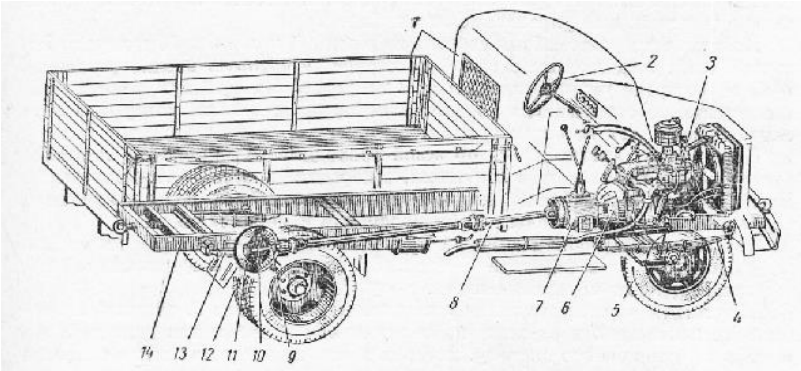
# Автомобіль

# Вантажний автомобіль

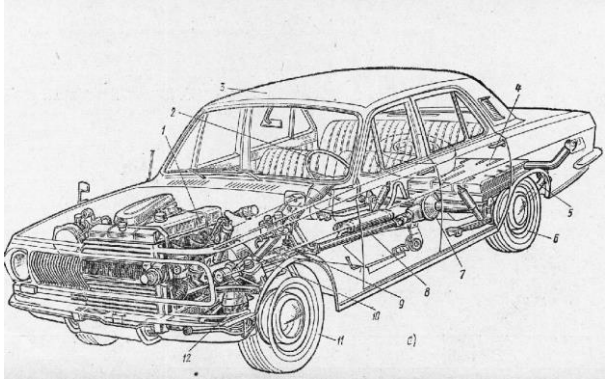


# Автомобіль

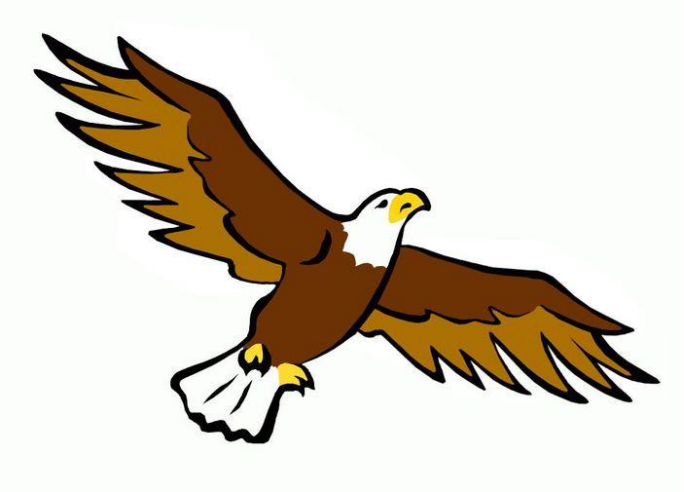
## Вантажний автомобіль



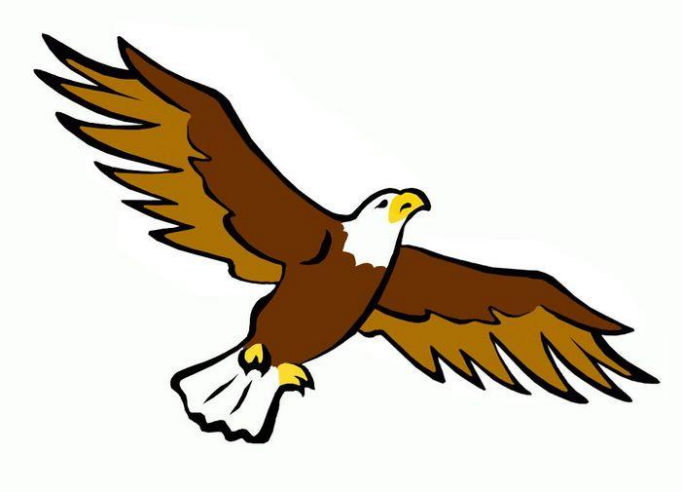
## Легковий автомобіль



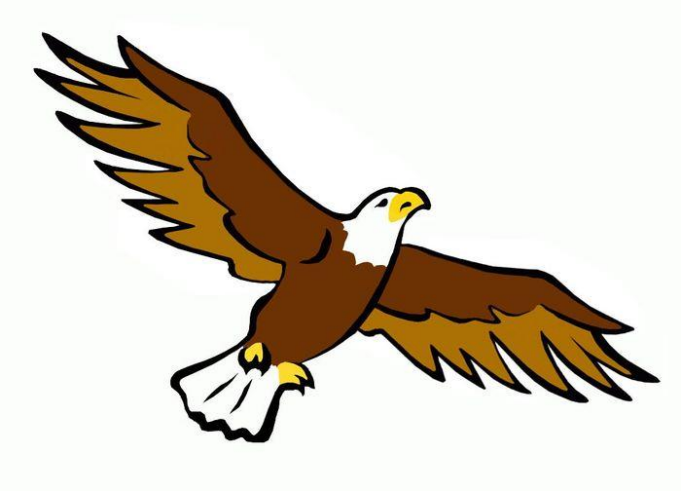
# Об’єкти, які можуть літати



# Об’єкти, які можуть літати



# Об’єкти, які можуть літати



# Основна принципова відмінність між абстрактними класами та інтерфейсами

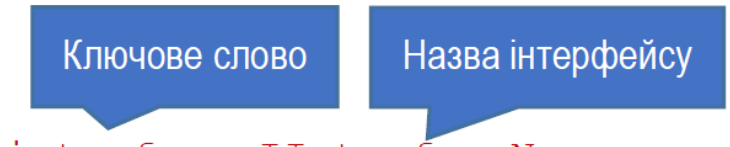
Інтерфейс говорить про поведінку класу і змушує відповідати на запитання "що я роблю?", а абстрактний клас визначає клас-спадкоємець і змушує його відповідати на питання "хто я?".

# Інтерфейси

- **Інтерфейси** – це ще один інструмент реалізації поліморфізму в C#. Інтерфейс являє собою набір методів (властивостей, подій, індексатори), реалізацію яких має забезпечити клас, який реалізує інтерфейс.
- Інтерфейс користувача може містити тільки сигнатури (ім'я і типи параметрів) своїх членів. Інтерфейс не може містити конструктори, поля, константи, статичні члени (це обмеження змінилося, починаючи з версії C# 8.0).
- Створювати об'єкти інтерфейсу неможливо.

# Інтерфейси

Імена інтерфейсам прийнято давати, починаючи з префіксу «I», щоб відразу відрізняти де клас, а де інтерфейс.
Всередині інтерфейсу оголошуються сигнатури його членів, модифікатори доступу вказувати не потрібно. Загальний синтаксис оголошення інтерфейсу такий:

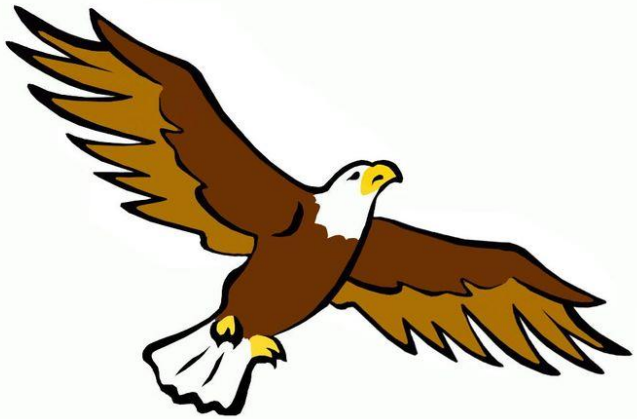



```
interface IInterfaceName
{
    // Заголовки функціональних елементів інтерфейсу
}
```

# Інтерфейси

- Клас, який впроваджує інтерфейс, обов'язково повинен реалізувати всі елементи інтерфейсу. При цьому код для реалізації елементів інтерфейсу може як реалізуватися самим класом, так і спадкуватися від базових класів.
- Щоб впровадити інтерфейс у клас, слід при оголошенні класу вказати назву інтерфейсу в переліку базових класів:

# Об’єкти, які можуть літати



# Інтерфейс IFlyable (літаючий об’єкт)

```
interface IFlyable
{
    void Fly();
}
```

Інтерфейс IFlyable (літаючий об’єкт) містить оголошення методу Fly().
Цей метод потрібно обов’язково реалізувати в усіх класах,
які будуть реалізовувати цей інтерфейс.

Класи реалізують інтерфейс IFlyable, але кожний клас має власну реалізацію методу Fly()

```
class Rocket : IFlyable
{
    public void Fly()
    {
        Console.WriteLine("Ракета літає, бо має реактивний двигун");
    }
}

class Eagle : IFlyable
{
    public void Fly()
    {
        Console.WriteLine("Орел літає, бо має крила");
    }
}

class Balloon : IFlyable
{
    public void Fly()
    {
        Console.WriteLine("Повітряна куля літає, бо легша за повітря");
    }
}
```

# Інтерфейси. Множинне успадкування

В C# клас може успадковувати лише один клас, але реалізувати відразу кілька інтерфейсів. Це і є головною відмінністю використання інтерфейсів і абстрактних класів.

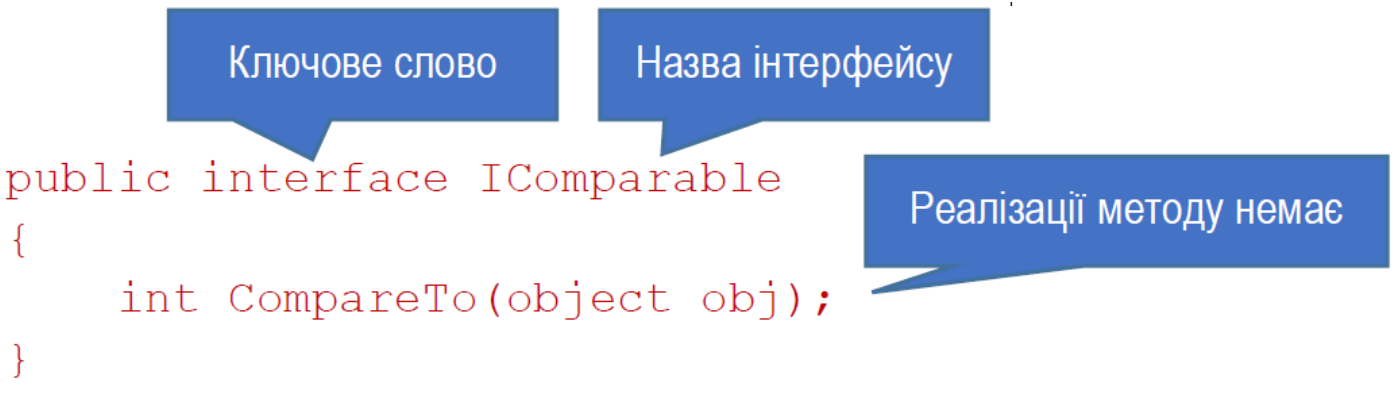
Якщо клас, у який впроваджують інтерфейс, є похідним, то його базовий клас у списку успадкованих класів повинен бути першим. Після назви базового класу через кому можна вказати довільну кількість інтерфейсів (у той же час базовий клас може бути тільки один):


```
class TheClass : TheBaseClass, IInterface1, IInterface2
{ ... }
```

# Стандартні інтерфейси

Бібліотека C# містить набір інтерфейсів, які можна використовувати у своїх програмах. Один із них – інтерфейс IComparable, – реалізує метод, що дозволяє порівнювати екземпляри будь-якого класу між собою. Цей інтерфейс  використовується методом Sort класу Array, який представляє масив. Метод Sort класу Array залежить від інтерфейсу, який називається IComparable.

Інтерфейс IComparable оголошено так:



# Стандартний інтерфейс IComparable

Щоб метод Sort міг сортувати екземпляри певного класу, цей клас повинен реалізувати інтерфейс IComparable. Для цього при оголошенні класу назву інтерфейсу слід вказати в переліку базових класів, а також забезпечити реалізацію методу CompareTo.

Метод CompareTo має повертати такі значення:

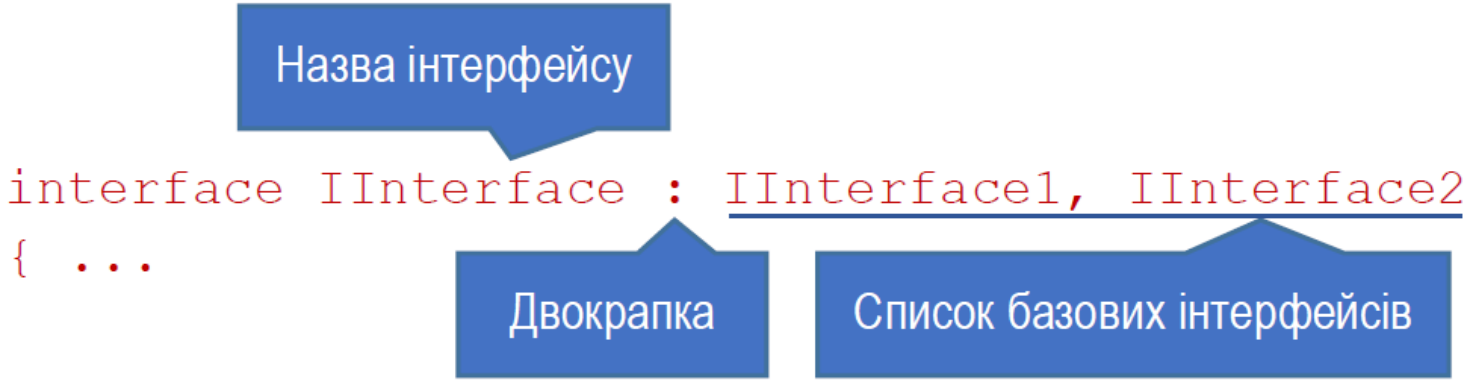
- Від'ємне значення, якщо у впорядкованому (відсортованому) списку поточний об'єкт повинен бути розташований перед об'єктом, заданим параметром object.
- Додатне значення, якщо поточний об'єкт повинен бути розташований після об'єкта object.
- Нуль, якщо обидва об'єкти при впорядкуванні займають одну і ту ж позицію.

Приклад реалізації інтерфейсу IComparable у класі Student

```
class Student : IComparable<Student>
{
    public string Name { get; set; }
    public int Mark { get; set; }
    public Student(string name, int mark)
    {
        Name = name;
        Mark = mark;
    }
    public int CompareTo(Student otherStudent)
    {
        if (Mark > otherStudent.Mark) return 1;
        else
        if (Mark < otherStudent.Mark) return -1;
        else
            return 0;
    }
}
```

# Спадкування інтерфейсів

Інтерфейс, як і клас, може спадкувати структуру іншого інтерфейсу. Таким чином, на основі одних інтерфейсів можна створювати інші інтерфейси. Щоб вказати, що інтерфейс успадковує інші інтерфейси, імена базових інтерфейсів слід вказати так само, як при спадкуванні класів – в окремому переліку:



# Особливості оголошення інтерфейсів в C# 8.0

Починаючи з C# 8.0, член інтерфейсу може мати реалізацією за замовчуванням. Члени з тілом дозволяють інтерфейсу надавати реалізацію за замовчуванням для класів та структур, які не надають реалізацію з перевизначенням. Крім того, починаючи з C# 8.0 інтерфейс може включати:

- Константи
- Оператори
- Статичний конструктор
- Вкладені типи
- Статичні поля, методи, властивості, індексатори та події
- Оголошення членів із використанням синтаксису явної реалізації інтерфейсу.
- Явні модифікатори доступу (за замовчуванням — public).

# Домашнє завдання

Створити абстрактний базовий клас планета (Planet), та похідні класи Земля (Earh) та Місяць (Moon).

Клас Planet має абстрактні методи ReportAboutMovement() (повідомити навколо якого небесного тіла рухається планета) та ReportAboutLife() (повідомити про наявність життя на планеті). Кожний з похідних класів перевизначає методи ReportAboutMovement(), ReportAboutLife() базового класу.

Використати принцип поліморфізму для виведення текстових повідомлень щодо руху та наявності життя на Землі та Місяці. Для цього:

1. Створити масив об’єктів типу Planet

2. Створити по одному об’єкту класів Earh та Moon і додати їх до масиву.

3. За допомогою оператору циклу foreach для кожного елементу масиву через посилання на тип Planet викликати методи ReportAboutMovement(), ReportAboutLife().