الـ Abstract Class والـ Interface

1. إيه هو الـ Abstract Class؟

الـ Abstract Class زي "القالب الناقص" اللي بيحدد هيكل للكلاسات التانية، بس مش بتقدر تنشئ منه Object مباشرة. يعني، هو كلاس عادي بس بيكون فيه بعض الـ Methods أو الـ Properties مجردة (Abstract) يعني مش معمولها Implementation، وبتسيب تنفيذها للكلاسات اللي هتورث منه.

- **مميزات الـ Abstract Class**

بيسمح بكتابة Abstract Methods بدون Body، ودي لازم تنفذها في الكلاسات الوارثة.

كمان بتقدر تضيف Methods عادية بتنفيذ كامل، عشان تشارك الكود بين الكلاسات الوارثة.

يدعم الـ Properties، Fields، و Constructors.

مثال: لو عايز تصمم نظام للحيوانات، الـ Animal هيبقي الـ Abstract Class هيحدد دالة مجردة زي MakeSound()، وكل حيوان (زي Dog أو Cat) هينفذها بطريقته.

```
public abstract class Animal {
    public abstract void MakeSound(); // دالة مجردة

    public void Eat() { // دالة عادية
        Console.WriteLine("The animal is eating.");
    }
}

public class Dog : Animal {
    public override void MakeSound() {
        Console.WriteLine("Woof!");
```

```
    }
}

// استخدام: Dog dog = new Dog(); dog.MakeSound(); // Woof!
```

هنا، مش تقدر تقول `Animal animal = new Animal();` لأنه abstract.

2. إيه هو الـ Interface؟

الـ Interface زي "العقد" اللي بيحدد مجموعة من الـ Methods أو الـ Properties اللي لازم ينفذها أي كلاس يطبقه. هو مش كلاس، بس بيحدد "إيه اللي لازم يتعمل" بدون أي تنفيذ (في الإصدارات قبل C# 8.0). في لغات زي C#، الـ Interface بيبدأ بحرف I (زي IAnimal).

🔹 مميزات الـ Interface

كل الدوال فيه مجردة افتراضيًا (قبل C# 8.0)، ومفيش تنفيذ.

من C# 8.0 وما فوق، بتقدر تضيف Default Implementations لبعض الدوال.

مش بيسمح بـ Fields أو Constructors، بس يدعم Properties.

كلاس واحد بيقدر يطبق أكتر من interface فبقي متاح ليك تعمل Multiple Inheritance.

مثال: لو عايز تضمن إن كل Object يقدر "يطير"، هتعمل interface اسمه IFlyable جواه Method اسمها Fly()، وأي حاجة زي Bird أو Plane هتطبقه.

```
public interface IAnimal {
    void MakeSound(); // مجردة تلقائيًا
}

public class Cat : IAnimal {
```

```
    public void MakeSound() {
        Console.WriteLine("Meow!");
    }
}

// استخدام: IAnimal cat = new Cat(); cat.MakeSound(); // Meow!
```

هنا، تقدر تقول `IAnimal animal = new Cat();` بس مش هتقدر تنشئ instance من الـ Interface نفسه.

3. الفروقات الرئيسية بين الـ Abstract Class والـ Interface

الفرق مش بس في الاسم، ده في التصميم والاستخدام. خلينا نقارن بجدول بسيط عشان الوضوح:

الميزة	Abstract Class	Interface
التنفيذ	بيسمح ب Methods مجردة وعادية بتنفيذ كامل	قبل C# 8.0: كل الدوال مجردة، بعد C# 8.0: يدعم default implementations
الوراثة	كلاس واحد بس يقدر يورث من abstract class واحد	كلاس بيقدر يطبق أكتر من interface
الخصائص	يدعم Fields، Properties، Constructors	مش بيدعم Fields أو Constructors (بس Properties بيدعم)

الوصول	يمكن يكون protected أو internal	كل حاجة public تلقائيًا
الاستخدام	لما يكون في كود مشترك بين كلاسات مرتبطة	لما عايز تضمن سلوك معين بدون علاقة وراثة
الإنشاء	مش تقدر تنشئ instance مباشرة	مش تقدر تنشئ instance أبدًا

استخدم Abstract Class لما تكون الكلاسات "مرتبطة" (زي عائلة حيوانات)، و Interface لما تكون "سلوكيات" مستقلة (زي الطيران أو السباحة).

4. الفروقات بين الـ Interface قبل وبعد C# 8.0

في الإصدارات القديمة من C# (قبل 8.0)، الـ Interface كان مجرد "عقد" صارم، يعني كل الـ Methods لازم تكون مجردة وما فيش أي تنفيذ. لكن من C# 8.0 وطالع، حصل تحديث كبير غيّر طريقة استخدام الـ Interfaces. خلينا نشوف الفروقات:

- **قبل C# 8.0**:

- كل الـ Methods في الـ Interface لازم تكون مجردة (بدون Body).
- الكلاس اللي بيطبق الـ Interface لازم ينفذ كل الـ Methods بنفسه.
- ما كانش في دعم لـ Default Implementations، فلو عايز تضيف Method جديدة للـ Interface، لازم تعدل كل الكلاسات اللي بتطبقه.

🔹 **من C# 8.0 وما فوق:**

- بقى ممكن تضيف Default Implementations للـ Methods ، يعني تقدر تكتب جسم (Body) للدالة داخل الـ Interface.
- ده بيسمح بإضافة دوال جديدة من غير ما تكسر الكلاسات القديمة (Backward Compatibility).
- كمان بقى يدعم الـ Static Members والـ Constants.
- مثال:

```
public interface IAnimal {
    void MakeSound(); // دالة مجردة

    void Sleep() { // Default Implementation
        Console.WriteLine("The animal is sleeping.");
    }
}

public class Cat : IAnimal {
    public void MakeSound() {
        Console.WriteLine("Meow!");
    }
    // مش لازم تنفذ Sleep() لأنها ليها تنفيذ افتراضي
}
```

🔹 **ليه التغيير ده مهم؟:**

- **مرونة أكبر**: تقدر تضيف ميزات جديدة للـ Interface بدون ما تعدل كل الكلاسات.
- **تشابه مع Abstract Class**: الـ Default Implementations خلّت الـ Interface أقرب للـ Abstract Class، بس لسه أخف وأكثر مرونة.

5. استخدامات عملية للـ Abstract Class والـ Interface

الاثنين مش نظرية بس، دول بيحلوا مشاكل حقيقية في البرامج الكبيرة.

- **استخدامات الـ Abstract Class**:

- **تصميم هيكل مشترك**: في ألعاب الفيديو، في الغالب بيبقي في Abstract Class اسمه Character بيحدد دوال زي ()Move و ()Attack ، وكل شخصية (زي Warrior أو Mage) تنفذها بطريقتها، مع مشاركة دوال عادية زي ()Die.
- **حماية الكود**: بيمنع إنشاء كائنات غير مكتملة، ويفرض تنفيذ معين.
- **مثال عملي**: في تطبيق بنكي، كلاس Account كـ Abstract Class بيحدد ()CalculateInterest، وكل نوع حساب (Savings أو Current) يحسب الفائدة بشكل مختلف.

- **استخدامات الـ Interface**:

- **المرونة في التصميم**: في تطبيقات الويب،ILogger كـ interface بيحدد ()LogError، وتقدر تنفذه بطرق مختلفة (زي FileLogger أو DatabaseLogger) بدون تغيير الكود الرئيسي.
- **Multiple Inheritance**: كلاس زي Drone بيطبق IFlayable و ICamera، عشان يطير ويصور في نفس الوقت.
- **مثال عملي**: في Dependency Injection (زي في ASP.NET)، بتستخدم Interfaces عشان تبدل التنفيذات بسهولة، زي تبديل Database من SQL لـ MongoDB بدون إعادة كتابة الكود.

⚙️ **Tips مهمة للتعامل معاهم:**

✔️ استخدم Abstract Class للكلاسات اللي هتبقي (Is-A relationship)، زي Dog is Animal.

✔️ استخدم Interface للسلوكيات اللي هتبقي بتعمل حاجة معينة (Can-Do)، زي Bird can Fly.

✔️ لو بتستخدم C# 8.0+، استغل Default Implementations في الـ Interface عشان توسّع الكود بسهولة.

✔️ جرب في مشروعك: ابدأ بـ Interface لو عايز مرونة، وانتقل لـ Abstract Class لو محتاج كود مشترك.