Cours Interfaces

Programme du cours

Activité	Durée
Lecture du document	20min

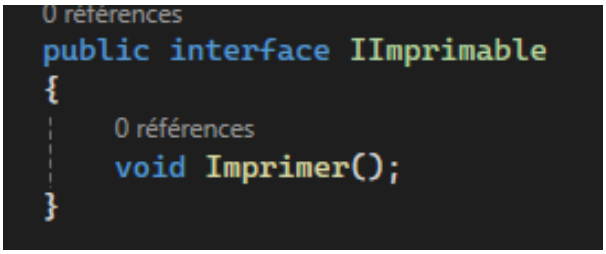
Objectif

- Ce qu'est une interface en C#
- Quelle est la différence fondamentale entre une interface et une classe abstraite
- Comment et quand utiliser une interface dans vos conceptions
- Implémenter une classe qui hérite d'une classe abstraite et implémente une ou plusieurs interfaces

Définition

Une **interface** est un **contrat** que s'engage à respecter une classe. Elle **déclare** des membres (méthodes et propriétés) **sans fournir d'implémentation.** (Dans les nouvelles versions de C#, il est possible de fournir une implémentation, mais ce n'est pas recommandé).

Une interface sert à définir ce qu'une classe peut faire par exemple :



- Une interface commence toujours par la lettre **"I"** par convention.
- Les méthodes sont **sans corps** (pas d'accolades, pas de code).
- Une **classe qui implémente une interface** s'engage à implémenter **toutes** ses méthodes.

Classe Abstraite vs Interface

	Classe Abstraite	**Interface**
Possède du code ?	Oui	Non (jusqu'à C# 8)
Variable privée autorisés ?	Oui	Non
Constructeurs ?	Oui	Non
Héritage multiple ?	Non (1 seule classe de base)	Oui (plusieurs interfaces possibles)
Relation hiérarchique ?	Oui, forte (type parent/enfant)	Non

Exemples

Prenons par exemple l'interface IImprimable, toute les classes qui hérite de IImprimable doivent implémenter l'a méthode Imprimer

```
0 références
public interface IImprimable
{
    0 références
    void Imprimer();
}
```

Créons une classe abstraite Document

```
3 références
public abstract class Document
{
    2 références
    public string Titre { get; set; }

    1 référence
    public Document(string titre)
    {
        Titre = titre;
    }

    1 référence
    public abstract void Afficher();
}
```

Si on veut maintenant créer une classe facture qui est considéré comme un document et est aussi imprimable

```
public class Facture : Document, IImprimable
{
    2 références
    public decimal Montant { get; set; }

    0 références
    public Facture(string titre, decimal montant) : base(titre)
    {
        Montant = montant;
    }

    1 référence
    public override void Afficher()
    {
        Console.WriteLine($"Facture : {Titre} - {Montant} CHF");
    }

    1 référence
    public void Imprimer()
    {
        // Logic pour imprimer la facture
    }
}
```

On peut créer d'autres classe imprimable mais qui ne sont pas des documents

```
4 références
public class Photo : IImprimable
{
    1 référence
    public void Imprimer()
    {
        // Logic pour imprimer la photo
    }
}

0 références
public class PageWeb : IImprimable
{
    1 référence
    public void Imprimer()
    {
        // Logic pour imprimer une page web
    }
}
```

A l'utilisation, on peut par exemple, utiliser du polymorphisme pour ajouter tous les elements imprimable dans une liste

```
List<IImprimable> elementAImprimer = new List<IImprimable>();

Facture facture = new Facture("Carte crédit", 100);
Photo photo = new Photo();
PageWeb pageWeb = new PageWeb();

foreach(IImprimable element in elementAImprimer)
{
    element.Imprimer();
}
```

Maintenant, on voudrait que des classes soient transformable en Json, on peut créer l'interface ISerialisable

```
2 références
public interface ISerialisable
{
    2 références
    string ConvertirEnJson();
}
```

N’importe quelle classe peut implémenter l’interface ISerialisable y compris notre classe facture

```
0 références
public class Utilisateur : ISerialisable
{
    1 référence
    public string Nom { get; set; }
    1 référence
    public string Email { get; set; }

    1 référence
    public string ConvertirEnJson()
    {
        return $"{{ \"nom\": \"{Nom}\", \"email\": \"{Email}\" }}";
    }
}
```

```
public class Facture : Document, IImprimable, ISerialisable
{

    ... même implémentation qu'avant

    1 référence
    public string ConvertirEnJson()
    {
        return $"{{ \"titre\": \"{Titre}\", \"montant\": {Montant} }}";
    }
}
```

Quand utiliser une Interface ?

Utiliser une interface quand :

- Vous voulez **définir un comportement commun** à plusieurs classes **non liées entre elles**
- Vous avez besoin d’**héritage multiple** (plusieurs comportements à combiner)
- Vous écrivez du **code générique ou modulaire**

Ne pas utiliser une interface quand :

- Vous avez **du code partagé à centraliser**
- Vous avez besoin de **champs, constructeurs ou logique commune**
- Vos classes ont une **relation hiérarchique forte** (parent/enfant, comme Animal et Chien)

En résumer

Une **classe abstraite** définit ce qu’une classe est.

Une **interface** définit ce qu’une classe peut faire.