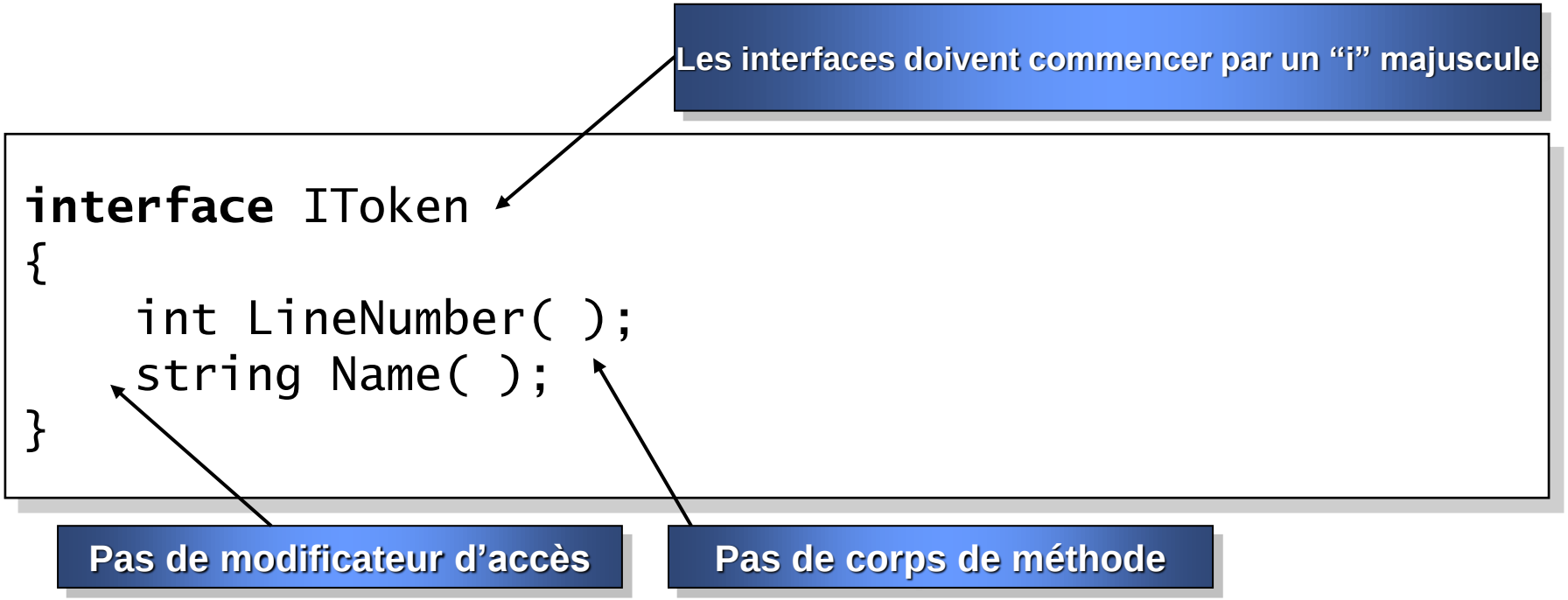
Orienté Objet :
Les interfaces en C#

Déclarer des interfaces en C#

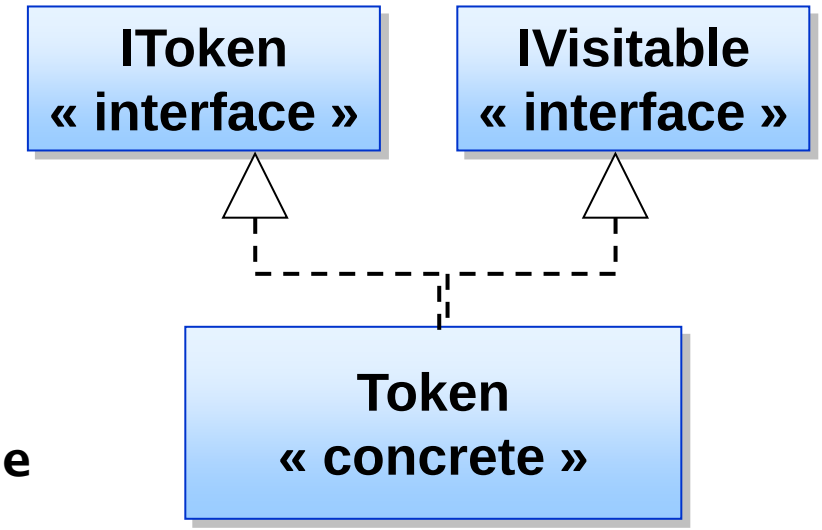
- Une interface se déclare un peu comme on déclare une classe, si ce n'est l'emploi du mot clé **interface** à la place du mot clé **class.**



Implémenter des interfaces

- Une classe **peut implémenter** de 0 à n interfaces
- Une interface **peut étendre** de 0 à n autres interfaces
- Une classe **doit** implémenter **toutes** les méthodes déclarées dans les interfaces qu'elle implémente

```
interface IToken
{
    string Name( );
}
interface IVisitable
{
    void Accept(IVisitor v);
}
class Token: IToken, IVisitable
{
    public string Name() { … }
    public void Accept(IVisitor v) { … }
}
```



Ce qu’est une interface

- Liste de méthodes dont on donne seulement la signature ;
- Représente un "contrat", ce qu'on attend d'un objet ;
- Toutes les méthodes d'une interface sont implicitement publiques et abstraites ;
- Une interface n'a pas de constructeurs ;
- Une interface ne peut avoir d’attributs champs sauf si ceux-ci sont statiques.

Exemple concret

Interface Vehicule

```
public interface IVehicule
{
        void accelerer( );
        void freiner();
}
```

On a défini ici ce qu'on attend d'un objet de type véhicule

Classe Velo

```
public class Velo  : IVehicule
{
        String marque;
        int vitesse;

        public void accelerer()
        {
                vitesse = vitesse + 10;
        }

        public void freiner()
        {
                vitesse = vitesse - 10;
        }
}
```

Classe Voiture

```
public class Voiture : IVehicule
{
        String marque;
        int vitesse;

        public void accelerer()
        {
                vitesse = vitesse + 50;
        }

        public void freiner()
        {
                vitesse = vitesse - 50;
        }
}
```

Exemple concret

- Dans cet exemple, nous avons réalisé deux implémentations de IVehicule.

  Conséquences :

  - Ces 2 objets (Vélo et Voiture) peuvent être vus comme des véhicules => **polymorphisme**
  - Partout où l'on attend un objet de type Vehicule, on peut passer un de ces deux objets
  - Par ce biais, on introduit une couche d'abstraction dans notre programmation ce qui la rend beaucoup plus flexible.

Exemple concret

- Si, par exemple, nous avions une classe Personne possédant une méthode *conduire (**Vehicule** v)*, on peut alors écrire :

```
Personne p = new Personne();
```

- Comme la méthode attend un Vehicule en argument, on peut passer tout objet implémentant cette interface.

```
p.conduire(new Auto());
p.conduire(new Velo());
```

Exemple concret

- On peut "instancier" un Vehicule par le biais de ses implémentations

```
IVehicule v = new Auto();
IVehicule t = new Velo();
```

- Dans ce cas v et t sont vus comme des Vehicules et par conséquent on ne peut appeler sur ces objets que les méthodes définies dans l'interface IVehicule.

Implémenter les méthodes d'une interface

- Les méthodes implémentées doivent être identiques aux méthodes définies dans l'interface (même signature).
- Les méthodes implémentées peuvent être ou pas déclarées virtual.

```
class Token: IToken, IVisitable
{
    public virtual string Name( )
    { ...
    }
    public void Accept()
    { ...
    }
}
```

Même niveau d'accès (Toujours public)
Même type de retour
Même nom
Mêmes paramètres

Implémenter les méthodes d'une interface de manière explicite

- Dans certains cas, il faudra utiliser le nom complet de l'interface pour résoudre des conflits de nommage (ex: deux interfaces implémentées qui définissent deux méthodes ayant la même signature)

```
class Token: IToken, IVisitable
{
    string IToken.Accept()
    { ...
    }
    void IVisitable.Accept()
    { ...
    }
}
```

- Les restrictions suivantes s'appliquent alors :
    - *On ne peut accèder aux méthodes qu'en passant par l'interface.*
    - *On ne peut déclarer la méthode comme virtual.*
    - *On ne peut pas préciser de modificateurs d'accès.*

Où sont les bugs ?

```
interface IToken
{
    string Name( );
    int LineNumber( ) { return 42; }
    private string name();
}

class Token
{
    string IToken.Name( ) { ... }
    static void Main( )
    {
        IToken t = new IToken( );
    }
}
```