

TYPESCRIPT · LES INTERFACES



DÉFINITION

- Une interface permet de définir une abstraction **sans écrire de classe**
- C'est une sorte de modèle, qui permettra d'écrire nos classes
- Comme les classes abstraites, une interface ne peut **être instanciée**
- On parle également de **contrat**

SYNTAXE

```
interface InterfaceName {  
    // ...  
}
```

- Contrairement aux classes, les classes définissent les membres, mais pas leur implémentation
- Les méthodes n'ont pas de corps, et on ne peut pas définir de constructeur

SYNTAXE

```
interface Dinosaur {
    taille: string;
    poids: string;
    manger(nourriture: unknown): string;
}
```

- Les propriétés et méthodes n'ont pas de portée
- Elles sont toutes implicitement publiques

SYNTAXE

```
interface Dinossaure {  
    poids: number;  
}  
  
class Trex implements Dinossaure {  
    poids: number;  
    constructor (poids: number) {  
        this.poids = poids  
    }  
}  
  
const dino: Dinossaure = new Trex(25)
```

- Une instance peut être manipulée en tant que type d'interface qu'elle implémente
- On peut implementer plusieurs interfaces dans une classe

NOTA BENE

- Lors de la transpilation, les interfaces ne sont pas traduites en JS
- JS ne permet pas l'usage d'interfaces
- Leur utilisation n'est 'utile' que lors de la compilation

DÉMONSTRATION

On refait le code, mais en mieux !

LES IMPORTS



Semifir

10

DÉFINITION

- L'intérêt des imports est de travailler avec plusieurs fichiers plutôt qu'un seul
- Le code sera plus lisible et les fichiers bien moins longs
- Implique une certaine rigueur en ce qui concerne le rangement des modules

EXPORT

- On utilisera le mot clef **export** devant la déclaration de classe pour indiquer que la classe doit être exportée
- On utilisera **import** depuis l'autre fichier pour importer la classe

SYNTAXE

Fichier 'MonInterface.ts' :

```
export interface MonInterface {  
    // code  
}
```

Fichier 'MaClasse.ts' :

```
import {MonInterface} from '../Interfaces/MonInterface'  
  
class MaClasse implements MonInterface {  
    // code  
}
```



EXAMPLE D'ARBORESCENCE TYPE

Voici un exemple d'organisation :

```
Models # Egalement appelé 'src'  
|  
|   __Collaborateur # Nom du parent le plus haut  
|  
|   |   __Interfaces # Répertoire des interfaces  
|   |   |   __Collaborateur  
|   |   |   |   Remuere  
|   |   |   |   DureeLimite  
|  
|   |   __Classes # Répertoire des classes  
|   |   |   __Cdi  
|   |   |   |   __Cdd  
|   |   |   |   __Stagiaire  
|  
|   |   __index.ts # Fichier qui servira à l'exécution
```



PRÉCISIONS

- Il n'existe pas qu'une seule manière de s'organiser
- L'essentiel est de s'y retrouver facilement
- Dans tous les cas : Respectez le cahier des charges !



DÉMONSTRATION

Faisons le ménage dans notre projet Collaborateur !

LE POLYMORPHISME



18

DÉFINITION

- Cinquième et dernier pilier de la POO
- Concept qui permet de traiter les objets de différents types de manière identique
- Il existe plusieurs méthodes, mais TypeScript n'en permet que 2 de par sa nature

POLYMORPHISME PARAMÉTRIQUE : GÉNÉRICITÉ

- Consiste à définir plusieurs surcharges d'une méthode pour chaque type qu'elle doit traiter
- C'est ce que nous avons vu dans la partie sur les **surcharges**
- La signature de la méthode est adaptée en fonction du type d'objet qu'elle accueille

20

EXAMPLE

```
1 function mafonction(a:string, b:string):string;
2 function mafonction(a:number, b:number): number;
3
4
5 function mafonction(a: any, b:any): any {
6   return a + b;
7
8 console.log(mafonction('Coucou ', 'toi !')) // Coucou toi !
9 console.log(mafonction(5, 10)) // 15
```

- On peut tout à fait appliquer la généricité dans le cadre d'un héritage

LE POLYMODORPHISME D'HÉRITAGE

- Vous avez déjà vu du polymorphisme d'héritage sans le savoir
- Nous avons utilisé ce principe lors de la dernière démonstration

22

LE CONCEPT

- Une signature de méthode abstraite est définie dans une classe abstraite ou une interface
- Elle est implémentée de manière différente dans ses enfants (redéfinition)
- Cette fois, les paramètres et leur type sont les mêmes, idem pour le retour
- La logique en revanche est différente

23

EXAMPLE (1/5)

DÉFINITION DE L'INSTANCE 'FORME'

/Interfaces/Forme.ts

```
1 export default interface Forme {
2     longueur: number;
3     largeur: number;
4     /**
5      * Calcule l'aire d'une forme à partir de sa longueur et de sa largeur
6      * @return number : Aire en cm2
7     */
8     calculerAire(): number
9 }
```

Elle dispose d'attributs et d'une fonction

calculerAire() qui retourne un nombre

24

EXEMPLE (2/5)

DÉFINITION DE LA CLASSE 'CARRE'

/Classes/Carre.ts

```
1 // Import de l'interface Forme
2 import Forme from './Interfaces/Forme'
3 export default class Carré implements Forme {
4     longueur: number;
5     largeur: number;
6     // Définition du constructeur
7     constructor(longueur: number, largeur: number) {
8         this.longueur = longueur;
9         this.largeur = largeur;
10    }
11    // Implémentation de la méthode, spécifique pour les carrés
12    calculerAire(): number {
13        return this.longueur * this.largeur
14    }
15 }
```

25

EXEMPLE (2/5)

DÉFINITION DE LA CLASSE 'CARRE'

/Classes/Carre.ts

```
1 // Import de l'interface Forme
2 import Forme from './Interfaces/Forme'
3 export default class Carré implements Forme {
4     longueur: number;
5     largeur: number;
6     // Définition du constructeur
7     constructor(longueur: number, largeur: number) {
8         this.longueur = longueur;
9         this.largeur = largeur;
10    }
11    // Implémentation de la méthode, spécifique pour les carrés
12    calculerAire(): number {
13        return this.longueur * this.largeur
14    }
15 }
```

25

EXEMPLE (2/5)

DÉFINITION DE LA CLASSE 'CARRE'

/Classes/Carre.ts

```
1 // Import de l'interface Forme
2 import Forme from './Interfaces/Forme'
3 export default class Carré implements Forme {
4     longueur: number;
5     largeur: number;
6     // Définition du constructeur
7     constructor(longueur: number, largeur: number) {
8         this.longueur = longueur;
9         this.largeur = largeur;
10    }
11    // Implémentation de la méthode, spécifique pour les carrés
12    calculerAire(): number {
13        return this.longueur * this.largeur
14    }
15 }
```

25



EXEMPLE (3/5)

DÉFINITION DE TRIANGLERECTANGLE

/Classes/TriangleRectangle.ts

```
1 // Import de l'interface :  
2 import Forme from './Interfaces/Forme'  
3  
4 export default class TriangleRectangle implements Forme {  
5     longueur: number;  
6     largeur: number;  
7  
8     constructor(longueur: number, largeur: number){  
9         this.longueur = longueur;  
10        this.largeur = largeur;  
11    }  
12    // Implémentation de la méthode, spécifique pour les triangles rectangles  
13    calculeAire(): number {  
14        return this.longueur * this.largeur / 2  
15    }  
16}
```


EXEMPLE (3/5)

DÉFINITION DE TRIANGLERECTANGLE

/Classes/TriangleRectangle.ts

```
1 // Import de l'interface :  
2 import Forme from '../Interfaces/Forme'  
3  
4 export default class TriangleRectangle implements Forme {  
5     longueur: number;  
6     largeur: number;  
7  
8     constructor(longueur: number, largeur: number){  
9         this.longueur = longueur;  
10        this.largeur = largeur;  
11    }  
12    // Implémentation de la méthode, spécifique pour les triangles rectangles  
13    calculeAire(): number {  
14        return this.longueur * this.largeur / 2  
15    }  
16}
```


EXEMPLE (3/5)

DÉFINITION DE TRIANGLERECTANGLE

/Classes/TriangleRectangle.ts

```
1 // Import de l'interface
2 import Forme from '../Interfaces/Forme'
3
4 export default class TriangleRectangle implements Forme {
5   longueur: number;
6   largeur: number;
7
8   constructor(longueur: number, largeur: number) {
9     this.longueur = longueur;
10    this.largeur = largeur;
11  }
12  // Implémentation de la méthode, spécifique pour les triangles rectangles
13  calculerAire(): number {
14    return this.longueur * this.largeur / 2
15  }
16 }
```


EXAMPLE (4/5)

TESTS

index.ts :

```
1 // Interfaces:  
2 import Forme from './Interfaces/Forme'  
3 // Classes :  
4 import Carré from './Classes/Carré'  
5 import TriangleRectangle from './Classes/TriangleRectangle'  
6 // Instanciation :  
7 const carré: Forme = new Carré(5, 5)  
8 const triangle: Forme = new TriangleRectangle(5, 5)  
9 // Création d'une liste de Formes  
10 const formes: Forme[] = [carré, triangle]  
11 // Affichage de la méthode polymorphée :  
12 formes.forEach((forme) => {  
13   console.log(`Je suis de type ${forme.constructor.name} et mon aire est  
14 })
```



EXAMPLE (4/5)

TESTS

index.ts :

```
1 // Interfaces:  
2 import Forme from './Interfaces/Forme'  
3 // Classes :  
4 import Carré from './Classes/Carré'  
5 import TriangleRectangle from './Classes/TriangleRectangle'  
6 // Instanciation :  
7 const carré: Forme = new Carré(5, 5)  
8 const triangle: Forme = new TriangleRectangle(5, 5)  
9 // Crédit d'une liste de Formes  
10 const formes: Forme[] = [carré, triangle]  
11 // Affichage de la méthode polymorphée :  
12 formes.forEach((forme) => {  
13   console.log(`Je suis de type ${forme.constructor.name} et mon aire est  
14 })
```



EXAMPLE (4/5)

TESTS

index.ts :

```
1 // Interfaces:  
2 import Forme from './Interfaces/Forme'  
3 // Classes :  
4 import Carré from './Classes/Carré'  
5 import TriangleRectangle from './Classes/TriangleRectangle'  
6 // Instanciation :  
7 const carré: Forme = new Carré(5, 5)  
8 const triangle: Forme = new TriangleRectangle(5, 5)  
9 // Création d'une liste de Formes  
10 const formes: Forme[] = [carré, triangle]  
11 // Affichage de la méthode polymorphée :  
12 formes.forEach((forme) => {  
13   console.log(`Je suis de type ${forme.constructor.name} et mon aire est  
14 })
```



EXAMPLE (4/5)

TESTS

index.ts :

```
1 // Interfaces:  
2 import Forme from './Interfaces/Forme'  
3 // Classes :  
4 import Carré from './Classes/Carré'  
5 import TriangleRectangle from './Classes/TriangleRectangle'  
6 // Instanciation :  
7 const carré: Forme = new Carré(5, 5)  
8 const triangle: Forme = new TriangleRectangle(5, 5)  
9 // Création d'une liste de Formes  
10 const formes: Forme[] = [carré, triangle]  
11 // Affichage de la méthode polymorphée :  
12 formes.forEach((forme) => {  
13   console.log(`Je suis de type ${forme.constructor.name} et mon aire est  
14 })
```



EXAMPLE (4/5)

TESTS

index.ts :

```
1 // Interfaces:  
2 import Forme from './Interfaces/Forme'  
3 // Classes :  
4 import Carré from './Classes/Carré'  
5 import TriangleRectangle from './Classes/TriangleRectangle'  
6 // Instanciation :  
7 const carré: Forme = new Carré(5, 5)  
8 const triangle: Forme = new TriangleRectangle(5, 5)  
9 // Crédit d'une liste de Formes  
10 const formes: Forme[] = [carré, triangle]  
11 // Affichage de la méthode polymorphée :  
12 formes.forEach((forme) => {  
13   console.log(`Je suis de type ${forme.constructor.name} et mon aire est  
14 })
```

27

EXAMPLE (5/5)

RETOUR

- > Je suis de type Carré et mon aire est égale à 25 cm².
- > Je suis de type TriangleRectangle et mon aire est égale à 12.5 cm².

- Il s'agit bien là de polymorphisme
- Une fonction qui change de formes selon le type

BONNES PRATIQUES



30

NOMENCLATURE

- Vous verrez régulièrement cette notation sur les interfaces :

```
interface IMonInterface {  
}
```

```
class CMaClasse {  
}
```

```
abstract class AMaClasseAbstraite {  
}
```

A NE PAS UTILISER

- Ces notations sont historiques et reflètent une époque où il était difficile de faire la différence entre classe et interface du point de vue du code
- Elle est aujourd'hui considérée comme mauvaise pratique car peut entraîner dérives et abus

EXAMPLE DE MAUVAISE PRATIQUE

```
interface IPerson {  
}  
  
class CPerson {  
}
```

- Souvent, cette pratique est utilisée par souci 'pratique'
- C'est en effet moins difficile d'ajouter un 'I' ou un 'C' que de trouver un nom représentatif

33

EXPLICATIONS

- On comprendra rapidement qui est interface ou classe,
- Par contre, on peut facilement tomber dans des dérives abstraites et des abus
- Résultat : On ne comprend pas à quoi sert quoi

NB

- Certains langages (comme le C#) ont besoin de déclarer les interfaces avec 'I'
- C'est un cas particulier et propre au langage.

EXERCICE !

Réalisez l'exercice 5

LA SUITE

Par ici !



37