Chapitre 1 Classe, interface et Type

Créez un dossier **chap1** nous allons découvrir les notions de classe et d'interface.

Écrivez le code qui suit dans un fichier product.ts, il définit une classe avec attributs et setter/getter. Notez que les attributs sont privés et protégés, nous précisons également le type string et number. Le code ne « compilera » pas si vous vous trompez dans l'assignation du type pour ces variables. Cela permet de sécuriser le code au niveau des types (essentielle).

```
// définition de la classe
class Product {
    private _name: string; // privé dans la classe actuelle
    protected _ref: number = 1000; // protégé dans la classe actuelle et classe fille

    constructor(name: string) {
        this.name = name; // setter assigne une valeur à l'attribut _name
    }

    // setter
    set name(name: string) {
        this._name = name;
    }

    // getter afficher une valeur dans le code courant
    get name(): string {
        return this._name;
    }
}

// instance de la classe
let bike = new Product('Super Bike');

console.log(bike.name); // affichera Super Bike
```

Transcompilé le code en JS, tapez à la racine de ce dossier le code suivant, nous précisons la cible (norme JS, les setter et getter ne sont supportés qu'à partir du JS ES5) :

```
tsc product.ts --target es5
```

Exécutez le code avec la ligne de code suivante, vous devriez voir « Super Bike » s'afficher en console.

```
node product.js
```

Remarque vous pouvez aller voir le code dans le fichier js (product.js) généré par TypeScript, c'est du JS ES5.

Exercice 1

Récupérez les sources et ajoutez les setter et getter pour l'identifiant _ref, puis affichez ces valeurs en console, essayez le code par exemple changer le nom de votre produit et affichez le.

Exercice 2

Définissez dans un fichier une classe Music et Guitar comme suit dans un fichier guitar.ts :

```
class Music {

    protected _instrument: string;

    play(): string {
        return "play music";
    }
}

// classe étendue
class Guitar extends Music{

}
```

Vous allez préciser dans la classe Guitar sans créer de setter le nom de l'instrument à l'aide de l'attribut _instrument. Puis affichez le nom de l'instrument dans le script courant à l'aide d'un getter que vous écrivez dans l'une des deux classes, la plus appropriée.

Transformez la classe Music **en classe abstraite** (dans ce cas on ne pourra plus faire d'instance de la classe Music directement dans le script courant. Cette classe devient alors **non instanciable ou abstraite**. Voyez le contrat makeSound définit dans la classe Music. Vous devez maintenant implémenter le code qui manque à la classe Guitar pour que celle-ci compile.

Remarques : une classe abstraite permet de définir un/des contrat(s), méthode(s) abstraite(s) et du code utile dans la classe elle-même. Si au moins une méthode est abstraite dans votre classe alors elle doit être définie « abstraite », sinon votre code ne compilera pas.

```
abstract class Music {
    protected _instrument: string;
    abstract makeSound():string;

    play(): string {
        return "play music";
    }
}

// classe étendue
class Guitar extends Music{…
}
```

Pensez à compiler avec la bonne cible JS au minimum ES5, et exécutez le code avec node.

Exercice 3

Soit l'interface **Duck** suivante la classe **Thing** l'implémente. Complétez le code pour que tout fonctionne correctement (compile). Il ne faut pas préciser la visibilité des membres d'une interface, par définition ils sont tous publiques.

Testez votre code en affichant en console le message : « Nage comme un canard ».

```
// définition de l'interface <=> contrat
interface Duck{
    name : string ;
    swim(): string;
}

class Thing implements Duck{

}
```

Une interface ou une classe permet de définir également des types pour d'autre variable. Voyez l'exemple suivant, dans les deux cas vous définissez des Array d'un certain type.

```
class Recipe {
    name: string;
    star?: number; // attribut facultatif
}

let recipes: Recipe[] = []; // pour le type notation équivalente Array<Recipe>

interface Bike{
    name : string;
    type? : string;
}

let bikes: Bike[] = [];
```

Exercice 4.1

Soit la définition suivante ajoutez quelques recettes dans la variable recipes ci-dessous. Et affichez ces valeurs en console.

```
class Recipe {
    name: string;
    star?: number; // attribut facultatif
}

let recipes: Recipe[] = []; // pour le type notation équivalente Array<Recipe>
```

Exercice 4.2

Créez une variable « notes », précisez le type de cette variable dans sa définition (Array de « number »). Ajoutez quelques valeurs dans ce tableau et affichez en console le contenu de celle-ci.