PARTIE 0: COMPLÉMENTS TYPESCRIPT

APPORTS DE TYPESCRIPT

Rappel: typage statique, gage de maintenabilité (documentation du code) et de développement simplifié (détection d'erreurs, autocomplétion dans l'IDE).

SOMMAIRE

- Javascript/EcmaScript: la programmation orientée objet (POO) est possible en javascript.
- Les compléments apportés par TypeScript pour la POO.

Pour tester le code javascript: la console du navigateur Chrome.

Pour tester en ligne le code typescript: typescript playground.

JAVASCRIPT

- tout est objet: les instances de classes, les objets littéraux, les fonctions, les tableaux … même les types primitifs sont parfois enveloppés dans un wrapper object (exemple: str.toUpperCase()).
- la création d'objets se fait par prototypage.
- les classes sont du sucre syntaxique introduit avec ES6.

LES CLASSES JAVASCRIPT (1/4)

```
class Voiture {
    // définition facultative des propriétés
    // publiques par défaut
    couleur

    // les méthodes sans mot-clé function
    // this fait référence à l'objet
    constructor(couleur){ this.couleur = couleur; }
    toString(){
        return `voiture de couleur ${this.couleur}`;
    }
}
const vBleue = new Voiture('bleu');
console.log(vBleue); // affiche l'objet
console.log(vBleue.toString()); // "voiture de couleur bleu"
console.log(`${vBleue}`); // "voiture de couleur bleu"
console.log(vBleue.couleur); // "bleu"
```

LES CLASSES JAVASCRIPT (2/4)

```
class Personne {
    #nom; // # convention private
    #age;
    constructor(nom, age) { this.#nom = nom; this.#age = age}
    get nom() {  //getter
      console.log('passage par le getter');
      return this.#nom; }
    set nom(value) { //setter
      console.log('passage par le setter');
      this.#nom = value; } }
let charlie = new Personne("Charlie", 12);
console.log(charlie.nom); // "Charlie"
charlie.nom = "Charles";
console.log(charlie.nom); // "Charles"
console.log(charlie.age); // undefined
```

LES CLASSES JAVASCRIPT (3/4)

```
1  class Animal {
2    _nom;  // _ convention protected
3    _age = 0; // valeur par défaut
4    constructor(nom, age) {
5      this._nom = nom;
6      if(age) this._age = age;
7    }
8    sePresente() {
9      console.log(`${this._nom} a ${this._age} ans.`);
10   }
11 }
12 class Chien extends Animal {
13   constructor(nom, age) {
14     // appel du constructeur de la classe de base
15     super(nom, age);
16   }
17   sePresente() { // surcharge de méthode
18     super.sePresente();
19     console.log(`${this._nom} aboie.`);
```

LES CLASSES JAVASCRIPT (3/4)

```
 7    }
 8    sePresente() {
 9      console.log(`${this._nom} a ${this._age} ans.`);
10    }
11  }
12  class Chien extends Animal {
13    constructor(nom, age) {
14      // appel du constructeur de la classe de base
15      super(nom, age);
16    }
17    sePresente() { // surcharge de méthode
18      super.sePresente();
19      console.log(`${this._nom} aboie.`);
20    }
21  }
22  const c = new Chien('milou', 2);
23  c.sePresente(); // =>  milou a 2 ans. milou aboie.
24  const a = new Animal('animal');
25  a.sePresente(); // => animal a 0 ans.
```

LES CLASSES JAVASCRIPT (3/4)

```
 7    }
 8    sePresente() {
 9      console.log(`${this._nom} a ${this._age} ans.`);
10    }
11  }
12  class Chien extends Animal {
13    constructor(nom, age) {
14      // appel du constructeur de la classe de base
15      super(nom, age);
16    }
17    sePresente() { // surcharge de méthode
18      super.sePresente();
19      console.log(`${this._nom} aboie.`);
20    }
21  }
22  const c = new Chien('milou', 2);
23  c.sePresente(); // =>  milou a 2 ans. milou aboie.
24  const a = new Animal('animal');
25  a.sePresente(); // => animal a 0 ans.
```

LES CLASSES JAVASCRIPT (4/4)

Avec les modules EcmaScript, les classes peuvent être exportées, comme les fonctions, les variables et les constantes.

```
export class Animal { … }
```

et pour l'importer pour l'utiliser dans d'autres modules/classes

```
import { Animal } from "./animal.js"
```

OU

```
export default class { … } /* au max un export default par fichier */
```

et on nomme la classe importée par défaut quand on l'importe

```
import Animal from "./animal.js"
```

Laisser autant que possible votre IDE gérer les imports.

APPORTS DE TYPESCRIPT POUR LA POO

- prise en charge native des concepts de la POO tels que les classes, **les interfaces** et l'héritage.
- **les décorateurs** permettent d'ajouter un comportement aux classes.

TYPESCRIPT (1/4)

- les propriétés dans les classes sont publiques par défaut mais peuvent être déclarées avec les mots-clés public, private ou protected.
- les propriétés peuvent être initialisées. Le constructeur n'est pas obligatoire.

TYPESCRIPT (2/4)

- les interfaces permettent de définir la structure que doit avoir un objet (= contrat sur la structure) qu'il s'agisse d'une instance de classe ou d'un objet littéral. Les noms des interfaces sont utilisés comme des types.
- une classe peut implémenter plusieurs interfaces.

TYPESCRIPT (3/4)

```typescript
interface Personne {
    readonly nom: string;
    readonly prenom: string;
}
class Salarie implements Personne {
    private _nom: string;
    private _prenom: string;
    constructor(prenom: string, nom: string) {
        this._nom = nom;
        this._prenom = prenom;
    }
    get nom(): string { return this._nom; }
    get prenom(): string { return this._prenom; }
}

function nomComplet(p: Personne) {
  return `${p.prenom} ${p.nom}`;
}

```

le _ reste uniquement nécessaire si il y a des getters/setters

TYPESCRIPT (3/4)

```
 8      constructor(prenom: string, nom: string) {
 9          this._nom = nom;
10          this._prenom = prenom;
11      }
12      get nom(): string { return this._nom; }
13      get prenom(): string { return this._prenom; }
14  }
15
16  function nomComplet(p: Personne) {
17    return `${p.prenom} ${p.nom}`;
18  }
19
20  const evelyne = new Salarie('Evelyne', 'Dupont');
21
22  // "Evelyne Dupont"
23  console.log(nomComplet(evelyne));
24
25  // "Tom Pouce"
26  console.log(nomComplet({prenom: 'Tom', nom: 'Pouce'}));
```

le _ reste uniquement nécessaire si il y a des getters/setters

TYPESCRIPT (4/4)

```
class Salarie implements Personne {
    private _nom: string;
    private _prenom: string;
    constructor(prenom: string, nom: string) {
        this._nom = nom;
        this._prenom = prenom;
    }
    get nom(): string { return this._nom; }
    get prenom(): string { return this._prenom; }
}
```

peut être raccourci en:

```
class Salarie implements Personne {
    constructor(private _prenom: string, private _nom: string) { }
    get nom(): string { return this._nom; }
    get prenom(): string { return this._prenom; }
}
```

EXERCICE PRATIQUE

Partie 0 - Exercice 1: TypeScript

ALLER PLUS LOIN: RÉFÉRENCES

- site
  - les chapitres gratuits de l'ebook de Ninja Squad sur Angular dont le chapitre TypeScript
- livres
  - Typescript Notions fondamentales de Félix Billon et Sylvain Pontoreau (Editions ENI - 2019), disponible à la bibliothèque universitaire.