Angular : introduction

Achref El Mouelhi

Docteur de l'université d'Aix-Marseille
Chercheur en programmation par contrainte (IA)
Ingénieur en génie logiciel

elmouelhi.achref@gmail.com

Plan

Angular

Angular

- Framework **JavaScript** (ou **TypeScript**)
- Open-source
- Présenté par **Google** en 2009
- Permettant de créer des applications web et mobiles
  - Front-End
  - Single page

Angular respecte l'architecture **MVVM**

- MVVM : Model-View-ViewModel
- **Model** : représenté généralement par une classe référencée par la couche d'accès aux données (classe ou interface **TypeScript**).
- **View**
  - contenant la disposition et l'apparence de ce qu'un utilisateur voit à l'écran,
  - recevant l'interaction avec l'utilisateur : clic, saisie, survol...
- **ViewModel**
  - remplaçant du contrôleur dans l'architecture **MVC**,
  - connecté à la vue par le **data binding**,
  - représenté dans **Angular** par un fichier `*.component.ts`.

Angular

Angular utilise :

- les composants web
- l'injection de dépendance
- le DOM Virtuel
- le change detection

Angular

Injection de dépendance ?

- concept connu en programmation orientée objet.
- utilisé par plusieurs frameworks Back-End (**Spring**, **Symfony**...).
- consistant à utiliser des classes sans faire de l'instanciation statique.

Angular

DOM Virtuel ?

- introduit par la librairie **React**.
- une représentation en mémoire du DOM physique.
- les modifications se font sur ce DOM virtuel ensuite **Angular** s'occupe de les synchroniser vers le DOM physique.

Angular

Change detection ?

- réalisé par la librairie **zone.js** (qu'on peut trouver dans `node_modules`).
- utilisé pour synchroniser la vue avec le composant.
- chaque composant dispose d'un change detector qui surveille en particulier les expressions utilisées dans l'interpolation ou le propriété binding.
- un changement de la valeur retournée par l'expression $\Rightarrow$ mise à jour de la vue.

Quelques outils utilisés par **Angular**

- **npm** (**n**ode **p**ackage **m**anager) : le gestionnaire de paquets par défaut pour une application **JavaScript**.
- **angular-cli** **c**ommand **l**ine **i**nterface : outil proposé par **Google** pour faciliter la création et la construction d'une application **Angular** en exécutant directement des commandes.
- **webpack** : bundler JavaScript
  - construit le graphe de dépendances.
  - regroupe des ressources de même nature (`.js` ou `.css`...) dans un ou plusieurs bundles.
  - fonctionne avec un système de module : un fichier **JS** est un module, un fichier **CSS** est un module...
- **Ivy** : moteur de compilation et de rendu utilisé partiellement dans **Angular** 8, intégralement depuis la version 9. Il permet d'accélérer la compilation et d'avoir une meilleure lisibilité des messages d'erreur.

Angular

Quels langages utilise **Angular** ?

- **HTML** pour les vues.
- **CSS** pour les styles.
- **TypeScript** pour les scripts depuis la version 2.

Les différentes versions d'**Angular**

- **Angular** 1 (ou **AngularJS**) présenté en 2009 : utilisant **JavaScript**
- **Angular** 2 présenté en octobre 2014 : remplacement du **JavaScript** par **TypeScript**
- **Angular** 4 présenté en décembre 2016
- **Angular** 5 présenté en novembre 2017
- **Angular** 6 présenté en mai 2018
- **Angular** 7 présenté en octobre 2018
- **Angular** 8 présenté en mai 2019
- **Angular** 9 présenté en février 2020
- **Angular** 10 présenté en juin 2020
- **Angular** 11 présenté en novembre 2021

Les différentes versions d'**Angular**

- **Angular** 1 (ou **AngularJS**) présenté en 2009 : utilisant **JavaScript**
- **Angular** 2 présenté en octobre 2014 : remplacement du **JavaScript** par **TypeScript**
- **Angular** 4 présenté en décembre 2016
- **Angular** 5 présenté en novembre 2017
- **Angular** 6 présenté en mai 2018
- **Angular** 7 présenté en octobre 2018
- **Angular** 8 présenté en mai 2019
- **Angular** 9 présenté en février 2020
- **Angular** 10 présenté en juin 2020
- **Angular** 11 présenté en novembre 2021

Quelques sites Web réalisés avec **Angular**

`https://www.madewithangular.com/`

Angular

Étapes

- Télécharger et installer **NodeJS** (Dernière version stable LTS)
- Démarrer une console
- Lancer la commande `npm install -g @angular/cli` : permet d'installer la dernière version disponible d'**Angular**
- Redémarrer la console
- Pour vérifier l'installation, exécuter la commande `ng version`
- Pour explorer la liste des commandes **Angular** disponibles, exécuter la commande `ng`

Angular

Étapes

- Télécharger et installer **NodeJS** (Dernière version stable LTS)
- Démarrer une console
- Lancer la commande `npm install -g @angular/cli` : permet d'installer la dernière version disponible d'**Angular**
- Redémarrer la console
- Pour vérifier l'installation, exécuter la commande `ng version`
- Pour explorer la liste des commandes **Angular** disponibles, exécuter la commande `ng`

Quelques sites Web réalisés avec **Angular**

Pour préciser une version d'**Angular** (par exemple 6.1.0) au moment de l'installation, on lance la commande

```
npm install -g @angular/cli@6.1.0
```

Angular

Quel IDE (Environnement de développement intégré) ?

- **Visual Studio Code** (À ne pas confondre avec Visual Studio)
- Eclipse
- ...

Angular

Quel IDE (Environnement de développement intégré) ?

- **Visual Studio Code** (À ne pas confondre avec Visual Studio)
- Eclipse
- ...

Visual Studio Code (ou VSC) , pourquoi ?

- Gratuit.
- Offrant la possibilité d'intégrer des éditeurs de texte connus (comme **Sublime Text**, **Atom**...).
- Pouvant s'étendre selon le langage de programmation.
- Recommandé par **Microsoft** (créateur de **TypeScript**) pour les projets **Angular**.

Angular

Quelques recommandations pour **VSC**

- Pour activer la sauvegarde automatique : aller dans `File > AutoSave`.
- Pour bien indenter son code : sélectionner tout `CTRL a` ensuite `ALT Shift f`.

Angular

Quelques recommandations pour **VSC**

- Pour activer la sauvegarde automatique : aller dans `File > AutoSave`.
- Pour bien indenter son code : sélectionner tout `CTRL a` ensuite `ALT Shift f`.

Extension VSC pour les templates **Angular**

Angular Language Service

Angular

Étapes

- Lancer la commande `ng new angular` et répondre aux questions suivantes (depuis la version 7) :
  - Do you want to enforce stricter type checking and stricter bundle budgets in the workspace ? ? (**Yes**)
  - Would you like to add Angular routing ? (**Yes**)
  - Which stylesheet format would you like to use ? (**CSS**)
- Se déplacer dans le projet `cd angular`
- Lancer le projet `ng serve --open`

Angular

Arborescence d'un projet **Angular**

- `e2e` : contenant les fichiers permettant de tester l'application
- `node_modules` : contenant les fichiers nécessaires de la librairie nodeJS pour un projet **Angular**
- `src` : contenant les fichiers sources de l'application
- `package.json` : contenant l'ensemble de dépendance de l'application
- `.angular-cli.json` (ou `angular.json` depuis la version 6) : contenant les données concernant la configuration du projet (l'emplacement des fichiers de démarrage...)
- `tslint.json` : contenant les données sur les règles à respecter par le développeur (nombre de caractères max par ligne, l'emplacement des espaces...)
- `tsconfig.json` : contenant les données de configuration de **TypeScript**

Angular

Que contient `src` ?

- `assets` : l'unique dossier accessible aux visiteurs et contenant les images, les sons...
- `index.html` : l'unique fichier **HTML** d'une application **Angular**
- `styles.css` : la feuille de style commune de tous les composants web de l'application
- `favicon.ico` : le logo d'**Angular**
- `app` : contient initialement les 5 fichiers du module principal
  - `app.module.ts` : la classe correspondante au module principal
  - `app.component.ts` : la classe associé au composant web
  - `app.component.html` : le fichier contenant le code **HTML** associé au composant web
  - `app.component.css` : le fichier contenant le code CSS associé au composant web
  - `app.component.spec.ts` : le fichier de test du composant web

Angular

Que contient `src` ?

- `assets` : l'unique dossier accessible aux visiteurs et contenant les images, les sons...
- `index.html` : l'unique fichier **HTML** d'une application **Angular**
- `styles.css` : la feuille de style commune de tous les composants web de l'application
- `favicon.ico` : le logo d'**Angular**
- `app` : contient initialement les 5 fichiers du module principal
  - `app.module.ts` : la classe correspondante au module principal
  - `app.component.ts` : la classe associé au composant web
  - `app.component.html` : le fichier contenant le code **HTML** associé au composant web
  - `app.component.css` : le fichier contenant le code CSS associé au composant web
  - `app.component.spec.ts` : le fichier de test du composant web

Pour créer un projet sans les fichiers de test (`.spec.ts`), utiliser la commande `ng new ProjectName --skip-tests=true`

Angular

Contenu d'`index.html`

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>ProjectName</title>

  <!-- cette balise va permettre d'assurer le routage -->
  <base href="/">

  <meta name="viewport" content="width=device-width, initial-
     scale=1">
  <link rel="icon" type="image/x-icon" href="favicon.ico">
</head>
<body>

  <!-- le composant web introduit dans le module principal -->
  <app-root></app-root>
</body>
</html>
```

Contenu de `app.module.ts`

```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Contenu de `app.module.ts`

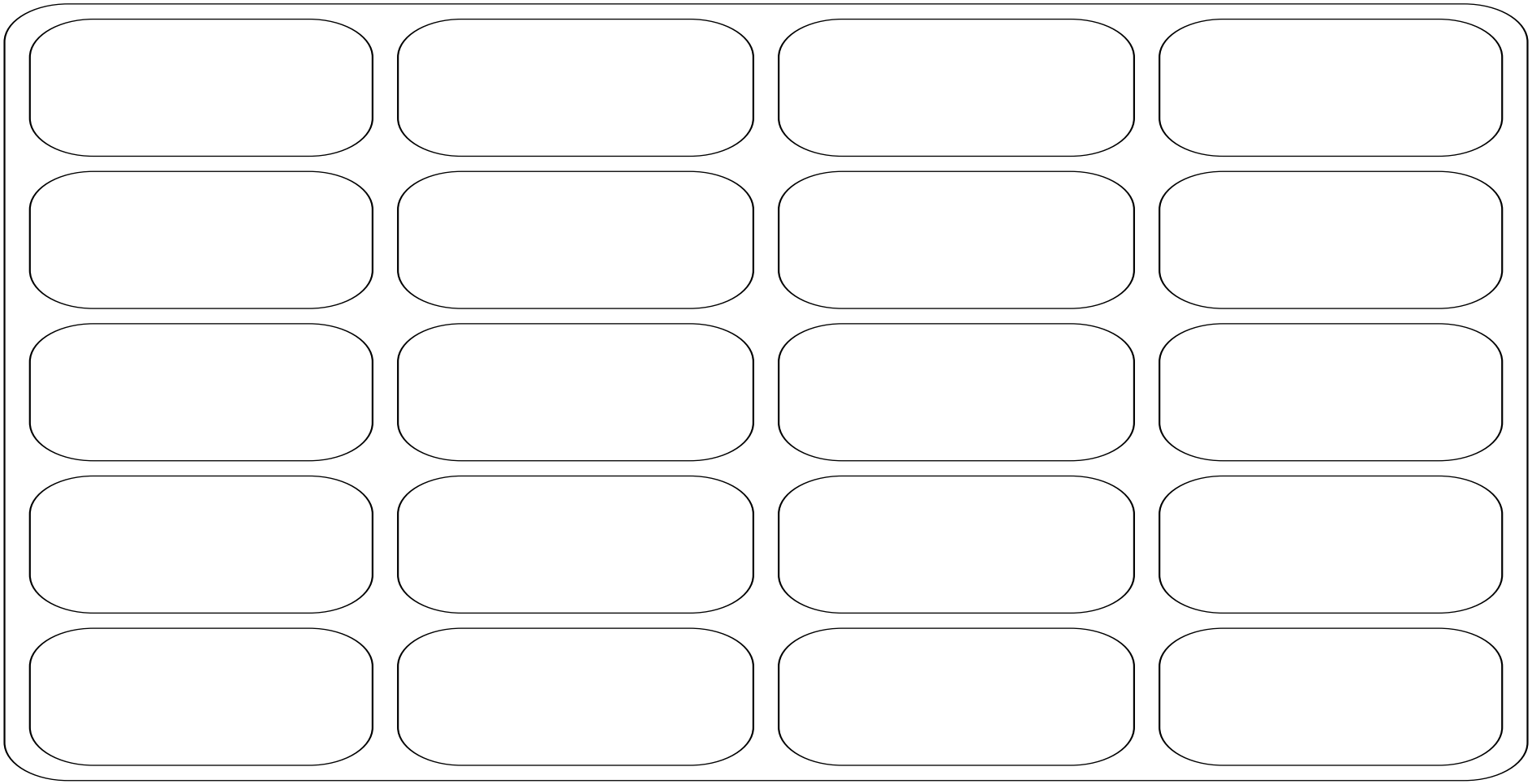
```
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Explication

- `@NgModule` : pour déclarer cette classe comme module
- `declarations` : dans cette section, on déclare les composants de ce module
- `imports` : dans cette section, on déclare les modules nécessaires pour le module
- `providers` : dans cette section, on déclare les services qui seront utilisés dans le module
- `bootstrap` : dans cette section, on déclare le composant principal de ce module

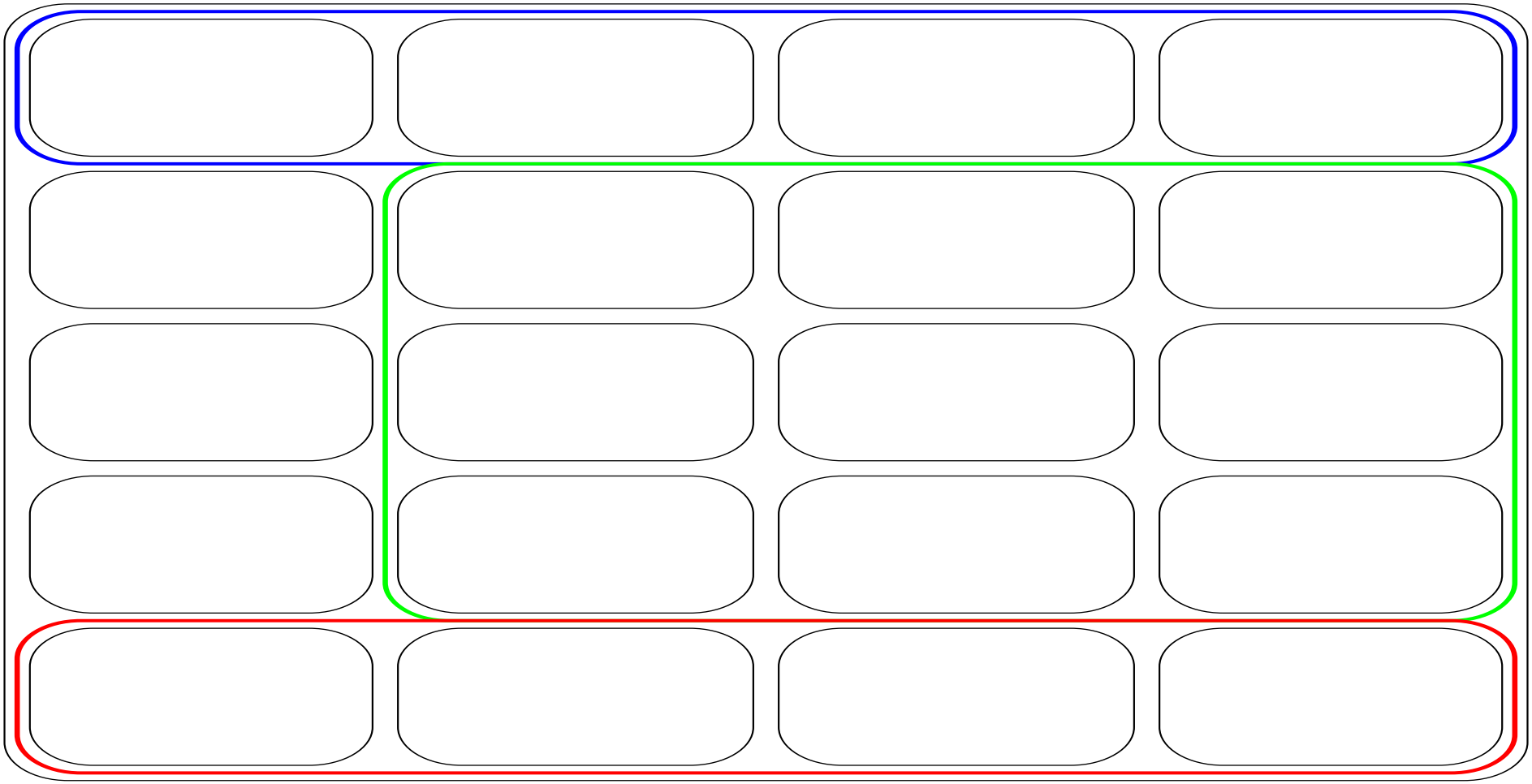
Angular

Angular : les composants



Angular

Angular : les modules



Le fichier `app.component.ts`

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
}
```

Le fichier `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  ...
</div>
```

Le fichier `app.component.ts`

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
}
```

Le fichier `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  ...
</div>
```

Explication

- `@Component` : pour déclarer cette classe comme composant web
- `selector` : pour définir le nom de balise correspondant à ce composant web
- `templateUrl` : pour indiquer le fichier contenant le code **HTML** correspondant au composant web
- `styleUrls` : pour indiquer le(s) fichier(s) contenant le code CSS correspondant au composant web

Le fichier `app.component.ts`

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
}
```

Le fichier `app.component.html`

```
<div style="text-align:center">
  <h1>
    Welcome to {{ title }}!
  </h1>
  ...
</div>
```

Explication

- `@Component` : pour déclarer cette classe comme composant web
- `selector` : pour définir le nom de balise correspondant à ce composant web
- `templateUrl` : pour indiquer le fichier contenant le code **HTML** correspondant au composant web
- `styleUrls` : pour indiquer le(s) fichier(s) contenant le code CSS correspondant au composant web

{{ Interpolation }}

`Welcome to {{ title }}!` : `title` sera remplacé par sa valeur dans `app.component.ts`

Angular

Objectif : afficher les attributs et les valeurs sous forme d'une liste HTML

- Créer un répertoire `classes` dans `app` où on va placer toutes les classes
- Créer une première classe `Personne` ayant les attributs `num`, `nom` et `prénom`
- Créer un objet de cette classe dans `app.component.ts`
- Afficher les valeurs de cet objet sous forme de liste dans `app.component.html`

Angular

On peut aussi faire

```
ng generate class classes/personne
```

Angular

On peut aussi faire

```
ng generate class classes/personne
```

Pour générer une classe sans le fichier de test (`.spec.ts`)

```
ng generate class classes/personne --skipTests=true
```

Angular

On peut aussi faire

```
ng generate class classes/personne
```

Pour générer une classe sans le fichier de test (`.spec.ts`)

```
ng generate class classes/personne --skipTests=true
```

Remarque

L'option `--skipTests=true` peut être utilisée avec la commande `generate` quel que soit l'élément généré.

Contenu de `personne.ts`

```
export class Personne {
    constructor(private _num?: number, private _nom?: string, private
       _prenom?: string) { }

    get num() {
        return this._num;
    }
    set num(_num: number) {
        this._num = _num;
    }
    get nom() {
        return this._nom;
    }
    set nom(_nom: string) {
        this._nom = _nom;
    }
    get prenom() {
        return this._prenom;
    }
    set prenom(_prenom: string) {
        this._prenom = _prenom;
    }
}
```

Angular

Créer un objet de type `Personne` dans `app.component.ts`

```
import { Component } from '@angular/core';
import { Personne } from './classes/personne';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  personne: Personne = new Personne(100, 'Wick', '
     John');
  constructor() {
  }
}
```

Angular

Afficher l'objet `personne` **dans** `app.component.html`

```
<ul>
  <li> Nom : {{ personne.nom }} </li>
  <li> Prénom : {{ personne.prenom }} </li>
</ul>
```

Angular

L'écriture suivante

```
{{ personne }}
```

Angular

L'écriture suivante

```
{{ personne }}
```

affiche

```
[object Object]
```

Angular

L'écriture suivante

```
{{ personne }}
```

affiche

```
[object Object]
```

On peut utiliser un pipe pour affiche un objet au format JSON

```
{{ personne | json }}
```

Angular

L'écriture suivante

```
{{ personne }}
```

affiche

```
[object Object]
```

On peut utiliser un pipe pour affiche un objet au format JSON

```
{{ personne | json }}
```

Le résultat est

```
{ "_num": 100, "_nom": "Wick", "_prenom": "John" }
```

Angular

Créer un tableau d'entiers `tab` dans `app.component.ts`

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  personne: Personne = new Personne(100, 'Wick', '
     John');
  tab: number[] = [2, 3, 5, 8];
  constructor() {
  }
}
```

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab[1] }} </li>
  <li> {{ tab[2] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab[1] }} </li>
  <li> {{ tab[2] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Afficher le tableau `tab` dans `app.component.html`

```
<ul>
  <li> {{ tab[0] }} </li>
  <li> {{ tab[1] }} </li>
  <li> {{ tab[2] }} </li>
  <li> {{ tab[3] }} </li>
</ul>
```

Remarques

- Ce code est trop répétitif
- Et si on ne connaissait pas le nombre d'éléments
- Ou si on ne voulait pas afficher tous les éléments

Solution

- utiliser les directives (section suivante)

Angular

Ajouter une méthode `direBonjour()` **dans** `app.component.ts`

```
import { Component } from '@angular/core';
import { Personne } from './classes/personne';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'cours-angular';
  personne: Personne = new Personne('Wick', 'John');
  tab: number[] = [2, 3, 5, 8];
  constructor() {
  }
  direBonjour(): string {
    return 'bonjour Angular';
  }
}
```

Angular

Appeler la méthode `direBonjour()` **dans** `app.component.html`

```
<p>{{ direBonjour() }}</p>
```

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

On peut aussi faire le one way binding en utilisant la propriété JavaScript `textContent`

```
<p [textContent]=title></p>
```

Angular

Pour afficher le contenu de l'attribut `title` dans le template, on peut utiliser l'interpolation

```
<p> {{ title }} </p>
```

On peut aussi faire le one way binding en utilisant la propriété JavaScript `textContent`

```
<p [textContent]=title></p>
```

`[ one way binding ]` ou `[ property binding ]`

`[property] = 'value'` : permet de remplacer `value` par sa valeur dans la classe `app.component.ts`

Angular

Plusieurs directives possibles

- `*ngFor`
- `*ngIf`
- `*ngSwitch`
- `ngStyle`
- `ngClass`

Angular

Plusieurs directives possibles

- `*ngFor`
- `*ngIf`
- `*ngSwitch`
- `ngStyle`
- `ngClass`

Ces directives s'utilisent conjointement avec les composants web suivant

- `ng-container`
- `ng-template`

Angular

*ngFor

- permet de répéter un traitement (affichage d'un élément HTML)
- s'utilise comme un attribut de balise et sa valeur est une instruction itérative **TypeScript**

Angular

Afficher le tableau `tab` en utilisant `*ngFor`

```
<ul>
  <li *ngFor="let elt of tab">
    {{ elt }}
  </li>
</ul>
```

Angular

Et pour avoir l'indice de l'itération courante

```
<ul>
  <li *ngFor="let elt of tab; let i = index">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

Angular

Et pour avoir l'indice de l'itération courante

```
<ul>
  <li *ngFor="let elt of tab; let i = index">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

ou

```
<ul>
  <li *ngFor="let elt of tab; index as i">
    {{ i }} : {{ elt }}
  </li>
</ul>
```

Angular

On peut aussi utiliser `first` pour savoir si l'élément courant est le premier de la liste

```
<ul>
  <li *ngFor="let elt of tab; let i = index; let isFirst =
     first">
    {{ i }} : {{ elt }} : {{ isFirst }}
  </li>
</ul>
```

Angular

On peut aussi utiliser `first` pour savoir si l'élément courant est le premier de la liste

```
<ul>
  <li *ngFor="let elt of tab; let i = index; let isFirst =
     first">
    {{ i }} : {{ elt }} : {{ isFirst }}
  </li>
</ul>
```

Autres paramètres possible

- `last` : retourne `true` si l'élément courant est le dernier de la liste, `false` sinon.
- `even` : retourne `true` si l'indice de l'élément courant est pair, `false` sinon.
- `odd` : retourne `true` si l'indice de l'élément courant est impair, `false` sinon.

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
personnes: Array<Personne> = [
    new Personne(100, 'Wick', 'John'),
    new Personne(101, 'Abruzzi', 'John'),
    new Personne(102, 'Marley', 'Bob'),
    new Personne(103, 'Segal', 'Steven')
];
```

Angular

Considérons la liste suivante (à déclarer dans `app.component.ts`)

```
personnes: Array<Personne> = [
    new Personne(100, 'Wick', 'John'),
    new Personne(101, 'Abruzzi', 'John'),
    new Personne(102, 'Marley', 'Bob'),
    new Personne(103, 'Segal', 'Steven')
];
```

Exercice

Écrire un script **Angular** qui permet d'afficher dans une liste **HTML** les nom et prénom de chaque personne de la liste `personnes` .

Angular

Pour tester puis afficher si le premier élément est impair

```
<ul>
  <li *ngIf="tab[0] % 2 != 0">
    {{ tab[0] }} est impair
  </li>
</ul>
```

Angular

Exercice

Écrire un code **HTML**, en utilisant les directives **Angular**, qui permet d'afficher le premier élément du tableau (`tab`) ainsi que sa parité (pair ou impair).

Angular

Une première solution avec `*ngIf` et `else`

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; else sinon">
    {{ tab[0] }} est impair
  </li>
  <ng-template #sinon>
     <li>
       {{ tab[0] }} est pair
     </li>
  </ng-template>
</ul>
```

`ng-template` n'apparaîtra pas dans le DOM de la page.

Angular

Une deuxième solution avec `*ngIf`, `then` et `else`

```
<ul>
  <li *ngIf="tab[0] % 2 != 0; then si else sinon">
    Ce code ne sera jamais pris en compte
  </li>
  <ng-template #si>
    <li>
      {{ tab[0] }} est impair
    </li>
  </ng-template>
  <ng-template #sinon>
    <li>
      {{ tab[0] }} est pair
    </li>
  </ng-template>
</ul>
```

Angular

Exercice

Écrire un code **HTML**, en utilisant les directives **Angular**, qui permet d'afficher sous forme d'une liste chaque élément du tableau précédent (`tab`) ainsi que sa parité.

Angular

Solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li *ngIf="elt % 2 != 0; else sinon">
      {{ elt }} est impair
    </li>
    <ng-template #sinon>
      <li>
        {{ elt }} est pair
      </li>
    </ng-template>
  </ng-container>
</ul>
```

`ng-container` n'apparaîtra pas dans le DOM de la page.

Angular

Exercice

Écrire un code **HTML**, en utilisant les directives **Angular**, qui permet d'afficher sous forme d'une liste chaque élément du tableau `moyennes = [18, 5, 11, 15]` avec un message défini ainsi :

- Si la valeur est comprise entre 0 et 9 : échec
- Si elle est entre 10 et 13 : moyen
- Si elle est entre 14 et 16 : bien
- Sinon : très bien

Angular

Exemple avec `ngSwitch`

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <ng-container [ngSwitch]="elt">
      <li *ngSwitchCase="1">
        {{ elt }} = un
      </li>
      <li *ngSwitchCase="2">
        {{ elt }} = deux
      </li>
      <li *ngSwitchCase="3">
        {{ elt }} = trois
      </li>
      <li *ngSwitchDefault>
        {{ elt }} : autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```

Pour comparer avec une chaîne de caractère, il faut ajouter les simples quotes (par exemple : `*ngSwitchCase="'bonjour'"`.

Angular

Refaire cet exercice avec `ngSwitch`

Écrire un code **HTML**, en utilisant les directives **Angular**, qui permet d'afficher sous forme d'une liste chaque élément du tableau `moyennes` avec un message défini ainsi :

- Si la valeur est comprise entre 0 et 9 : échec
- Si elle est entre 10 et 13 : moyen
- Si elle est entre 14 et 16 : bien
- Sinon : très bien

Angular

Solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <ng-container [ngSwitch]="true">
      <li *ngSwitchCase="elt >= 0 && elt < 10">
        {{ elt }} : échec
      </li>
      <li *ngSwitchCase="elt >= 10 && elt < 13">
        {{ elt }} : moyen
      </li>
      <li *ngSwitchCase="elt >= 13 && elt < 15">
        {{ elt }} : bien
      </li>
      <li *ngSwitchCase="elt >= 15">
        {{ elt }} : très bien
      </li>
      <li *ngSwitchDefault>
        autre
      </li>
    </ng-container>
  </ng-container>
</ul>
```

Angular

Remarques

- `*ngFor` nous permet d'itérer sur un tableau, mais non pas sur un objet.
- Il est possible d'itérer sur un objet après avoir défini un pipe qui convertit virtuellement un objet en tableau.

Angular

ngStyle

- permet de modifier le style d'un élément **HTML**.
- s'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe.

Angular

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nom = 'wick';
  couleur = 'blue';
  // le contenu précédent
}
```

Angular

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  nom = 'wick';
  couleur = 'blue';
  // le contenu précédent
}
```

Pour afficher le contenu de l'attribut `nom` dans le template avec une couleur de fond rouge

```
<p [textContent]=nom [ngStyle]="{backgroundColor: 'red'}"></p>
```

Angular

Pour utiliser une couleur définie dans un attribut de la classe associée

```
<p [textContent]=nom [ngStyle]="{backgroundColor:
  couleur}"></p>
```

Angular

Il est possible de définir des méthodes dans `app.component.ts` qui gèrent le style d'un élement HTML

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent

  getColor(): string {
    return 'white';
  }
  getBackgroundColor(): string {
    return 'red';
  }
}
```

Angular

Pour afficher le contenu de l'attribut `nom` dans le template avec une couleur de fond rouge

```
<p [ngStyle]="{color: getColor(), backgroundColor:
   getBackgroundColor()}">
   {{ nom }}
</p>
```

Angular

ngClass

- permet d'attribuer de nouvelles classes d'un élément **HTML**.
- s'utilise conjointement avec le **property binding** pour récupérer des valeurs définies dans la classe ou dans la feuille de style.

Angular

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent
}
```

Angular

Considérons le contenu suivant de `app.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  // le contenu précédent
}
```

On définit deux classes rouge et bleu dans `app.component.css`

```
.rouge {
    color: red;
}
.bleu {
    color: blue;
}
```

Angular

Pour associer la classe `rouge` à la balise `<p>`

```
<p [ngClass]="{'rouge': true}">
  {{ nom }}
</p>
```

Angular

Pour associer la classe `rouge` à la balise `<p>`

```
<p [ngClass]="{'rouge': true}">
  {{ nom }}
</p>
```

On peut aussi appeler une méthode dans la directive `ngClass`

```
<p [ngClass]="{'rouge': afficherEnRouge()}">
  {{ nom }}
</p>
```

Angular

Pour associer la classe `rouge` à la balise `<p>`

```
<p [ngClass]="{'rouge': true}">
  {{ nom }}
</p>
```

On peut aussi appeler une méthode dans la directive `ngClass`

```
<p [ngClass]="{'rouge': afficherEnRouge()}">
  {{ nom }}
</p>
```

Définissons la méthode dans `app.component.ts`

```
afficherEnRouge(): boolean {
    return this.couleur === 'blue' ? true : false;
}
```

Angular

Ainsi, on peut faire aussi un affichage conditionnel

```
<p [ngClass]="{'rouge': nom == 'wick'}">
  {{ nom }}
</p>
```

Angular

Ainsi, on peut faire aussi un affichage conditionnel

```
<p [ngClass]="{'rouge': nom == 'wick'}">
  {{ nom }}
</p>
```

Ou encore (le paragraphe sera affiché en rouge si `nom` contient `wick`, en bleu sinon)

```
<p [ngClass]="{'rouge': nom == 'wick', 'bleu': nom
   != 'wick'}">
  {{ nom }}
</p>
```

Angular

On peut aussi utiliser les expressions ternaires

```
<p [ngClass]="nom == 'wick' ? 'rouge' : 'bleu'">
  {{ nom }}
</p>
```

Angular

Exercice

Utiliser `ngClass` dans un code **HTML** permettant d'afficher en bleu les éléments pairs du tableau précédent (`tab`) et en rouge les éléments impairs.

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': elt % 2 != 0, 'bleu': elt % 2 ==
       0}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="{'rouge': elt % 2 != 0, 'bleu': elt % 2 ==
       0}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Deuxième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li [ngClass]="elt % 2 == 0 ? 'bleu' : 'rouge'">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li *ngIf="elt%2==0" [ngClass]="{'rouge': !estPair(
      elt), 'bleu': estPair(elt)}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let elt of tab">
    <li *ngIf="elt%2==0" [ngClass]="{'rouge': !estPair(
      elt), 'bleu': estPair(elt)}">
      {{ elt }}
    </li>
  </ng-container>
</ul>
```

Dans `app.component.ts`**, on définit la méthode** `estPair`

```
estPair(elt: number): boolean {
    return elt % 2 === 0;
}
```

Angular

Considérons la liste `personnes` précédente

```
personnes: Array<Personne> = [
    new Personne(100, 'Wick', 'John'),
    new Personne(101, 'Abruzzi', 'John'),
    new Personne(102, 'Marley', 'Bob'),
    new Personne(103, 'Segal', 'Steven')
];
```

Angular

Considérons la liste `personnes` précédente

```
personnes: Array<Personne> = [
    new Personne(100, 'Wick', 'John'),
    new Personne(101, 'Abruzzi', 'John'),
    new Personne(102, 'Marley', 'Bob'),
    new Personne(103, 'Segal', 'Steven')
];
```

Exercice

Écrire un script **Angular** qui permet d'afficher dans une liste **HTML** les éléments de la liste `personnes` (on affiche que les nom et prénom). Les éléments d'indice pair seront affichés en rouge, les impairs en bleu.

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of personnes; let isEven = even
     ; let isOdd = odd">
    <li [ngClass]="{'rouge': isEven,'bleu': isOdd}">
      {{ elt.nom }} {{ elt.prenom }}
    </li>
  </ng-container>
</ul>
```

Angular

Première solution

```
<ul>
  <ng-container *ngFor="let elt of personnes; let isEven = even
     ; let isOdd = odd">
    <li [ngClass]="{'rouge': isEven,'bleu': isOdd}">
      {{ elt.nom }} {{ elt.prenom }}
    </li>
  </ng-container>
</ul>
```

Deuxième solution

```
<ul>
  <ng-container *ngFor="let elt of personnes; let i = index;">
    <li [ngClass]="i % 2 != 0 ? 'rouge' : 'bleu'">
      {{ elt.nom + " " + elt.prenom }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let personne of personnes">
    <li [ngStyle]="{color: getNextColor()}">
      {{ personne.prenom }} {{ personne. nom }}
    </li>
  </ng-container>
</ul>
```

Angular

Troisième solution

```
<ul>
  <ng-container *ngFor="let personne of personnes">
    <li [ngStyle]="{color: getNextColor()}">
      {{ personne.prenom }} {{ personne. nom }}
    </li>
  </ng-container>
</ul>
```

Dans `app.component.ts`**, on définit la méthode** `getNextColor`

```
couleur = 'blue';
getNextColor(): string {
    this.couleur = this.couleur === 'red' ? 'blue' : 'red';
    return this.couleur;
}
```

Angular

Pour désinstaller Angular

```
npm uninstall -g @angular/cli
```

Angular

Pour désinstaller Angular

```
npm uninstall -g @angular/cli
```

Pour vider le cache

```
# À partir de la version 5 de NPM
npm cache verify

# Avant la version 5
npm cache clean
```

Angular

Pour mettre à jour la version d'Angular

```
ng update @angular/cli @angular/core
```

Angular

Pour mettre à jour la version d'Angular

```
ng update @angular/cli @angular/core
```

Pour mettre à jour tous les paquets définis dans `Package.json`

```
ng update @angular/cli @angular/core --all=true
```