esprit
Se former autrement

Manipulation de composants

Année universitaire

2020-2021

CONFÉRENCE DES GRANDES ÉCOLES

cdio
CONCEIVE DESIGN IMPLEMENT OPERATE

AUF Agence UNIVERSITAIRE DE LA FRANCOPHONIE
IFGU

United Nations Educational, Scientific and Cultural Organization
UNESCO Chair "Project-based learning" ESPRIT School of engineering, Tunisia

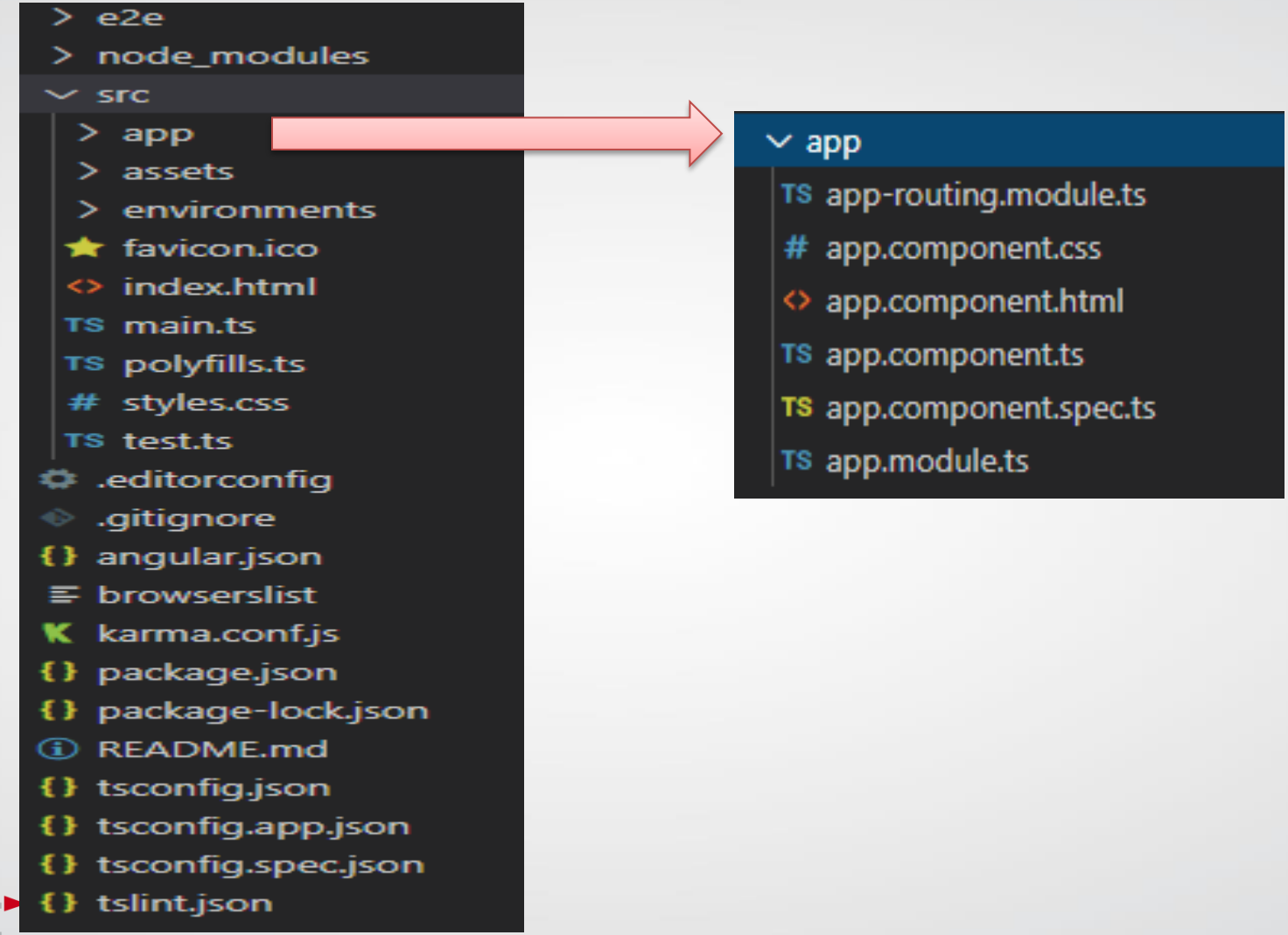
Plan

- Application Angular
- Modules/NgModules
- Composant
- Template - Vue
- Template Syntaxe
- Cycle de vie d’un composant

Application **Angular - Présentation**

- Une application Angular est modulaire.
- Elle possède au moins un module appelé « module racine » ou « root module »
- Elle peut contenir d'autres modules à part le module racine.
- Par convention, le module racine est appelé « **AppModule** » et se trouve dans un fichier appelé « **app.module.ts** »

Application Angular – Structure (1/3)



Application Angular – Structure (2/3)

- **e2e** : Tout ce qui va concerner les tests end to end (tests d'intégration)
- **node_modules** : contient les dépendances installés avec npm
- **src**: les différents éléments de l'application (composants, services, …)
- **src/app** : le code de l'application
- **package.json** : fichier déclarant les dépendances NPM tirées lors de l'installation du projet et nécessaire à la compilation et les tests

Application **Angular – Structure (3/3)**

- **tslint.json** : fichier définissant les règles de codage TypeScript.
- **tsconfig.json:** un fichier de définition TypeScrit
- **src/assets** : où placer tous les *assets* tels que les images.
- **src/environments** : on retrouve les différents fichiers de configuration spécifiques aux environnements d'exécution.
- …….

Modules/NgModules

- Un module est un bloc de code qui sert à encapsuler des fonctionnalités similaires une application.
- Le système de modularité dans Angular est appelé **NgModules**.
- Un module peut être exporté sous forme de classe.
- La classe qui décrit le module Angular est une classe décorée par @NgModule.

Exemple: FormsModule, HttpClientModule, RouterModule

Modules - Exemples

Exemple1: Module racine

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { AppComponent } from './app.component';
import { HelloWorldComponent } from './hello-world/hello-world.component';
@NgModule({
  declarations: [
    AppComponent,
    HelloWorldComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Exemple2: autre Module

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';

import { ProjetRoutingModule } from './projet-routing.module';
import { CreateProjectComponent } from './create-project/create-project.component';

@NgModule({
  imports: [
    CommonModule,
    ProjetRoutingModule
  ],
  declarations: [CreateProjectComponent]
})
export class ProjetModule { }
```

NgModule

- NgModule = classe importé de angular/core et marké aprés le décorateur @NgModule et sa métadata
- @NgModule : décorateur qui permet de définir la classe en tant que NgModule
- Métadata ou métadonnée est un objet qui décrit le comportement d'un élément qui est dans notre cas le module

NgModule - métadata

- **Declarations**: les composants – directives – pipes utilisés par ce module
- **Imports**: les modules internes ou externes utilisés dans ce module
- **Providers**: Les services utilisés
- **Bootstrap**: déclare la vue principale de l'application (celle du composant racine). Seul le *root NgModule* modifie cette propriétés.

Composant – Définition (1/4)

- Un composant est un élément réutilisable, indépendant et responsable d'une seule action métier.
- Un module peut contenir un ou plusieurs composants.
- Une application a au moins un composant racine

Composant – Définition (2/4)

Composant – Définition (3/4)

• Un composant crée est décrit par défaut par quatre fichiers :

Fichier	Rôle
nomComp.component.ts	Classe décrit le métier du composant
nomComp.component.html	Fichier template du composant
nomComp.component.css	Fichier de style du composant
nomComp.component.spec.ts	Fichier de test du composant

• Le composant racine est décrit par les fichiers app.component.css|html|ts|spec.ts

Composant – Définition (4/4)

- Un composant définit de(s) vue(s) et utilise de(s) service(s) pour réaliser un traitement.
- Le métier du composant est décrit dans la classe du composant qui peut être exportée, définit par le décorateur @component et décrite par un objet « métadata »

Composant - Métadata

- **selector** : définit une instance du composant dans le fichier HTML. Exemple: <app-root></app-root>
- **templateUrl**: l'url du fichier HTML associé au composant
- **styleUrls**: l'url du fichier CSS associé au template du composant
- **providers**: les services dont le composant a besoin pour son fonctionnement

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers:  [ TestService ]})
```

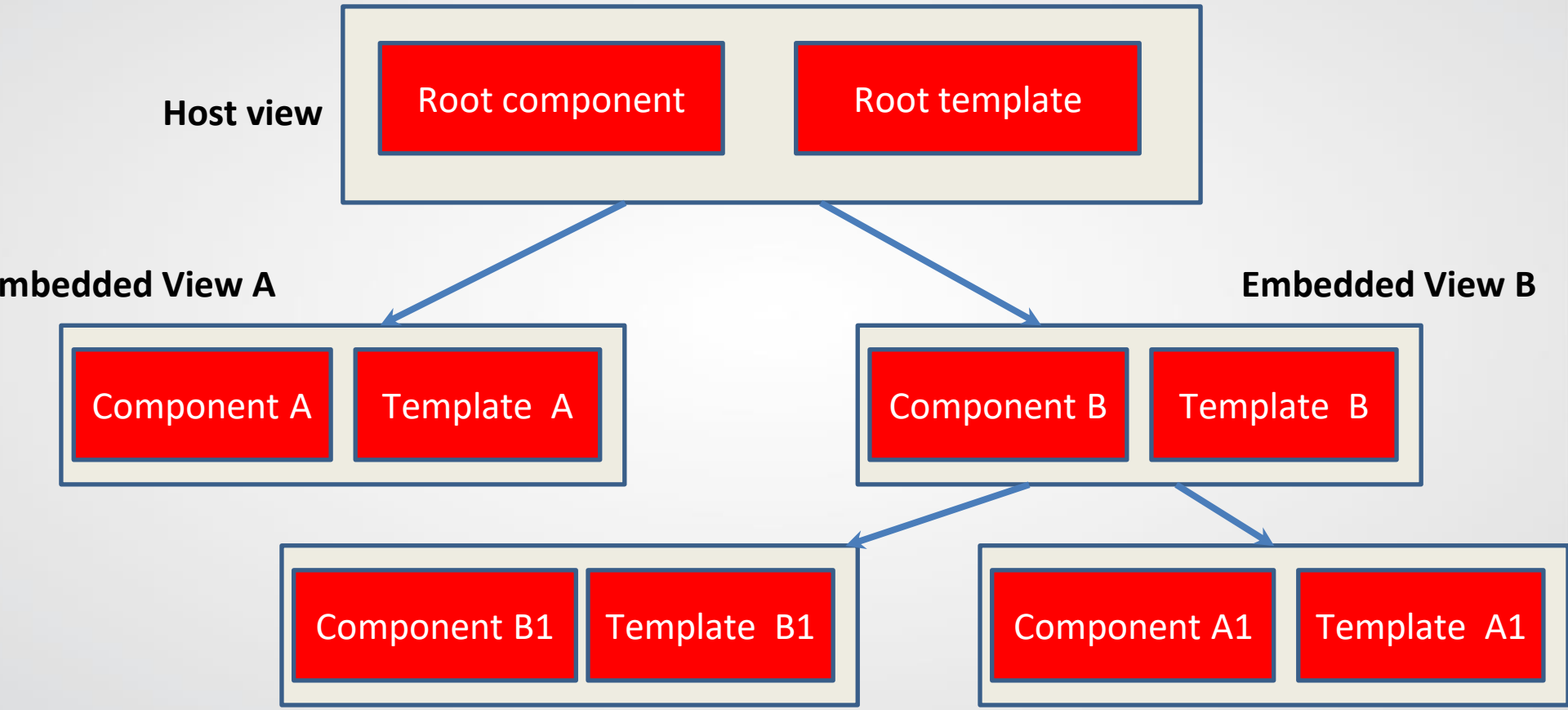
métadata

Template – Vue (1/2)

- Une vue est définit par le composant et son template.
- Les vues sont oragnisées d'une façon hiérarchique afin de pouvoir modifier ou cacher des sections de pages ou des pages entières.
- Le template attaché directement au composant est la vue hôte de ce composant « host view »

Template – Vue (2/2)

Un composant peut définir une hiérarchie de vues.



Template Syntaxe

Un template est un fichier HTML contenant:

1. La syntaxe régulière d'HTML (,<h1>,<p>,…)
2. Interpolations et expressions
3. Des éléments Angular tel que :
   - Les composants
   - Les directives
   - Les pipes
   - Le databinding

Template Syntaxe – Interpolation

- L'interpolation permet d'évaluer une expression

{{ expression_template }}

- Angular évalue l'expression ensuite convertit le résultat en chaine de caractères. Exemples:

{{ title }} : title est une propriété du composant

{{ 1 + 2 }} : le résultat affiché est 3

{{ 1+ getVal() }} : le résultat affiché est la somme de 1 avec e résultat de la méthode getval()

: le résultat affiché est l'image associée au chemin indiqué par src_image.

Template Syntaxe – Expression (1/5)

- Dans une expression, les opérateurs sont similaires à ceux de JavaScript.
- Vous ne pouvez pas utiliser les expressions JavaScript qui favorisent des effets secondaires tel que: les affectations (=, + =, - =, ...), les opérateurs(typeof, instanceof, etc), chaînage d'expressions avec (;ou,), les opérateurs d'incrémentation et de décrémentation ++ et - ainsi que certains des opérateurs ES2015 +

Template Syntaxe – Expression (2/5)

- Pas de support pour les opérteurs « bitwise » tel que | et &
- Nouveaux symboles tel que:

1. Pipe operator (|): sert à transformer une expression

```
{{title | uppercase}}
```

1. Safe navigation operator (?) : protège contre les valeurs nulles et non définies.

Il fonctionne parfaitement avec de longs chemins de propriété tels que? .B? .C? .D.

Template Syntaxe – Expression (3/5)

- Pas de support pour les opérateurs « bitwise » tel que | et &.
- Nouveaux symboles tel que:

1. Pipe operator (|): sert à transformer une expression avant l'afficher **{{title | uppercase}}**
2. Safe navigation operator (?) : protège contre les valeurs nulles et non définies.

Bonjour {{ item?.name}}.
Si item est null :
le résultat affiché est Bonjour.

Bonjour {{ item.name}}
Si item est null : TypeError: Cannot read property 'name' of null.

Template Syntaxe – Expression (4/5)

3. L'opérateur d'assertion non null (!) prend une variable typée et en supprime les types non définis et nuls.

- Cet opérateur prend effet si la propriété strictNullChecks est active

```
function myFunc(maybeString: string | undefined | null) {
const onlyString: string = maybeString; //compilation error: string | undefined | null is not assignable to string
const ignoreUndefinedAndNull: string = maybeString!; //no problem
}
```

Template Syntaxe – Expression (5/5)

Au niveau du template, le contexte d'une expression peut être :

- Une propriété du composant

```
{{title}}
[src]="imageUrl"
```

- Une variable d'entrée de template

customers: Propriété du composant

```
<ul> <li *ngFor="let customer of customers">{{customer.name}}</li> </ul>
```

- Une variable référence de template

```
Nom: <input name="nom"  #val> : {{val.value}}
```

=> **val** réfère à l'objet input identifié par **#val** ayant plusieurs attributs y compris value.

esprit Se former autrement

Template Syntaxe – Directives (1/3)

- **Une directive** est un décorateur qui marque une classe comme directive.
- Une directive applique une logique aux éléments du DOM.
- Il existe trois types de directives:
  1. Composant
  2. Directives structurelles
  3. Directives attributs

Template Syntaxe – Directives (2/3)

1. Directives structurelles: elles changent le DOM en ajoutant et retirant des éléments au template (NgIF, NgForOf, NgSwitch)

```
<ul> <li *ngFor="let customer of customers">{{customer.name}}</li> </ul>
```

- Customer1
- Customer2
- Customer 3

! Autant de puces que le nombre d'éléments dans la liste customers

Template Syntaxe – Directives (3/3)

2. Directive attribut: change l'apparence et l'attitude d'un élément DOM, composant ou autre directive tels que : NgStyle, NgClass, NgModel, etc

```
<element [ngClass]="string|array|object">
</element>
```

```
<p [ngClass]="'first second'">… </p>
<p [ngClass]="['first', 'second']">… </p>
<p [ngClass]="{'first': true, 'second': true}">…
</p>
```

```
<element [ngStyle]="objectExpression">
</element>
```

```
<p [ngStyle]="{'font-style': italic">… </p>
```

esprit Se former autrement

Template Syntaxe - Pipes

- Les pipes transforment les données avant de les afficher.
- L'opérateur pipe passe le résultat d'une expression sur la gauche à une fonction pipe sur la droite tels que: DatePipe, UpperCasePipe, LowerCasePipe, ….

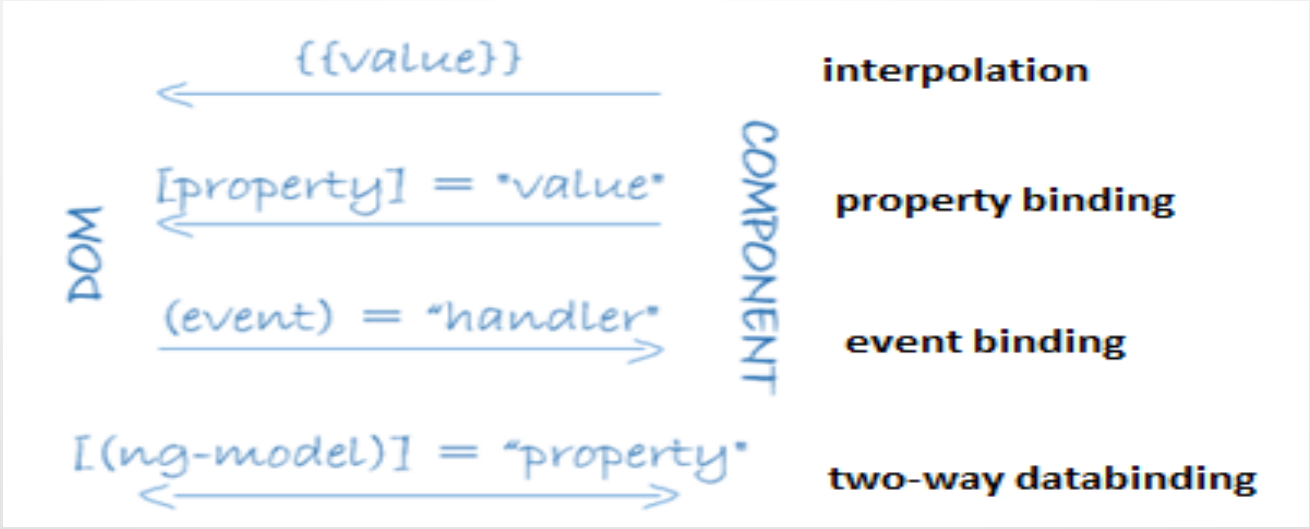
```
{{item.helpDate | date:'longDate'}}
```

- Vous pouvez chaîner des expressions via plusieurs pipes

```
{{title | uppercase | lowercase}}
```

Template Syntaxe - Data binding (1/6)

- Le databinding est un mécanisme de coordination entre le composant et le template dans un seul sens ou dans les deux sens.
- Il existe 4 formes de databinding



Template Syntaxe - Data binding (2/6)

Soit un composant A1 comme suit:

a1.component.ts

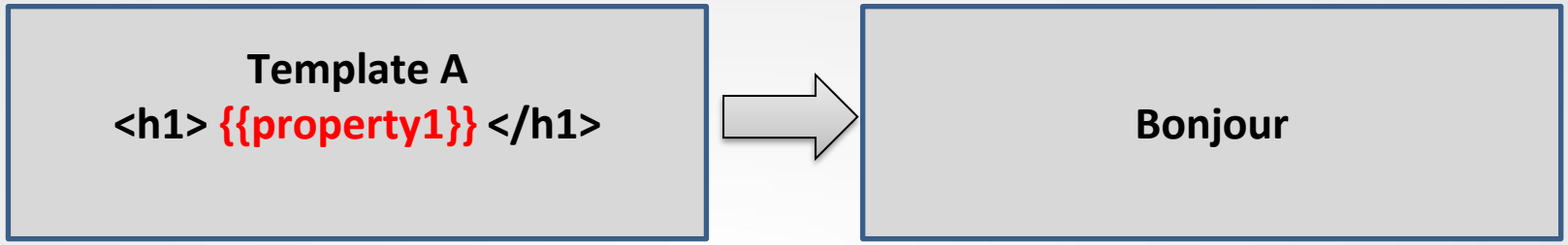
```
@component{
....
}
Export class A1{
property1="Bonjour";
property2="../fleur.png";

methodeA(){…}
}
```

a1.component.html

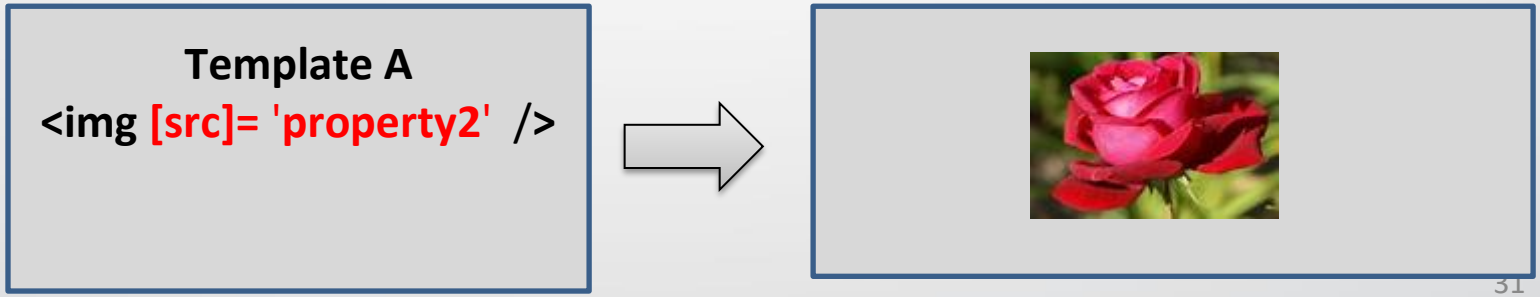
Template A

Template Syntaxe - Data binding (3/6)

L'interpolation permet d'afficher la valeur d'une propriété d'un composant au niveau du template



Property binding permet de modifier la valeur d'une propriété d'un élément du DOM par la valeur d'une propriété du composant



Template Syntaxe - Data binding (4/6)

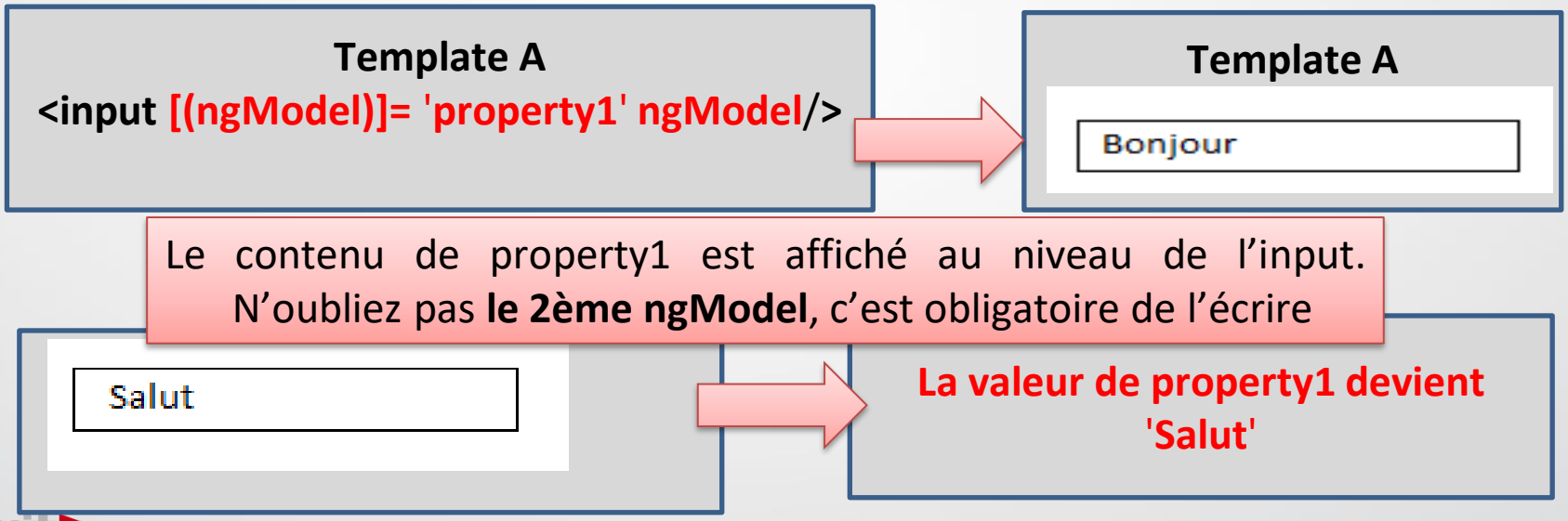
Event binding permet d'appeler une méthode du composant suite à une action faite par l'utilisateur au niveau du template

Template A

```
<button (click)="methodeA()"> envoyer</button>
```

Template Syntaxe - Data binding (5/6)

Two-way data binding: permet de récupérer une valeur à partir du template et l'envoyer vers une propriété du composant et vis versa. Ceci se fait grâce à la directive NgModel.



Template Syntaxe - Data binding (6/6)

Pour utiliser la directive **NgModel**, il faut importer le module **FormsModule** depuis **@angular/forms** et le déclarer dans la liste des **imports** du module racine **AppModule**.

```
import { NgModule } from '@angular/core';
import { FormsModule } from '@angular/forms';
....
@NgModule({
  declarations: [...],
  imports: [...., FormsModule,...],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Cycle de vie d'un composant (1/6)

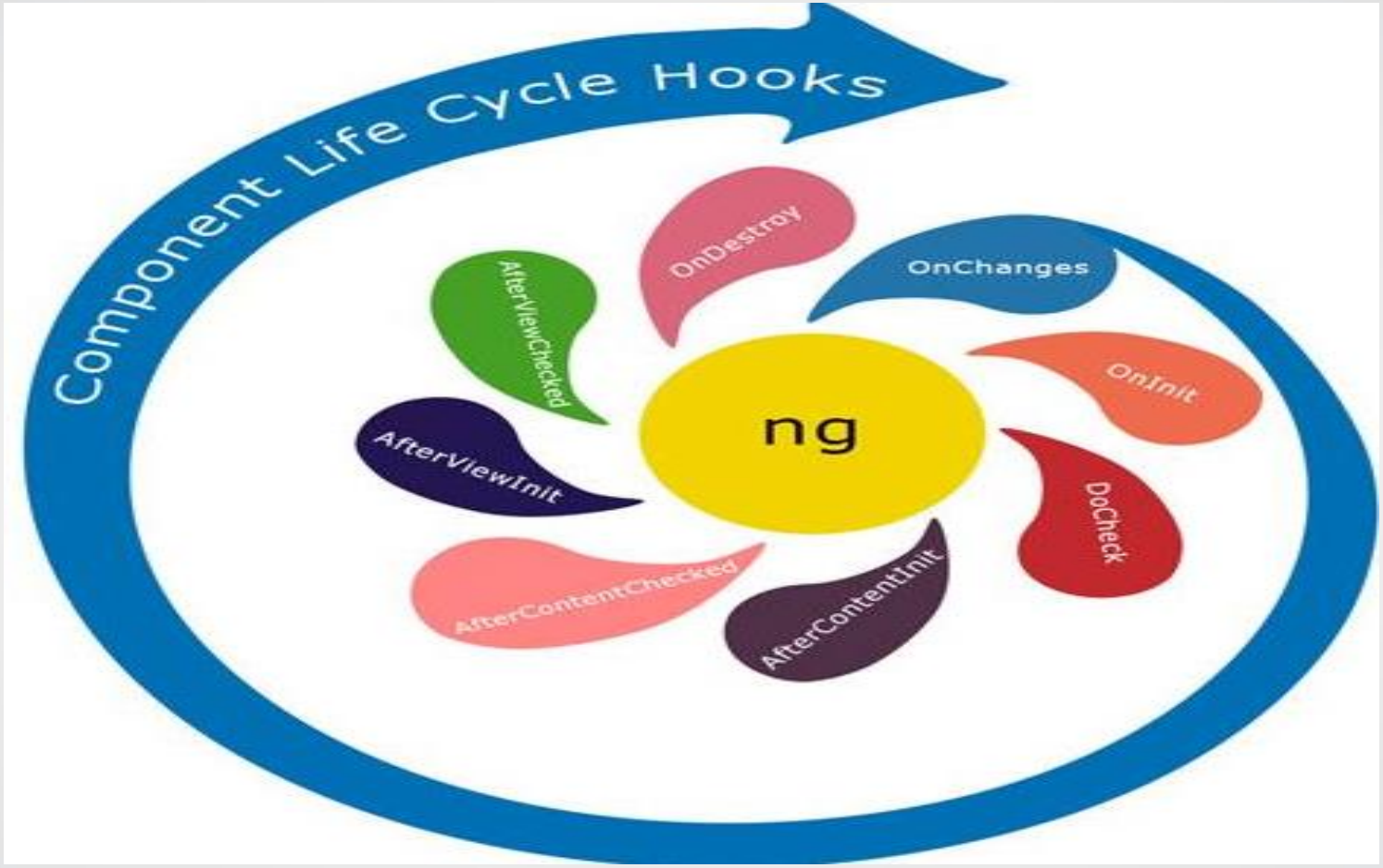
Un composant passe par plusieurs phases depuis sa création jusqu'à sa destruction : cycle de vie

Angular maintient et suit ces différentes phases en utilisant des méthodes appelées « hooks ».

On peut alors à chaque phase implémenter une logique.

Ces méthodes se trouvent dans des interfaces dans la librairie « @angular/core »

Cycle de vie d'un composant (2/6)



Cycle de vie d’un composant (3/6)

RQ: le constructeur d’un composant n’est pas un hook mais il fait partie du cycle de vie d’un composant : sa création

Il est logiquement appelé en premier, et c'est à ce moment que les dépendances (services) sont injectées dans le composant par Angular.

Cycle de vie d'un composant (4/6)

Méthode/hook	Rôle
ngOnChanges	Appelé lorsqu'une propriété input est définie ou modifiée de l'extérieur. L'état des modifications sur ces propriétés est fourni en paramètre
ngOnInit	Appelé une seule fois après le 1er appel du hook ngOnChanges(). Permet de réaliser l'initialisation du composant, qu'elle soit lourde ou asynchrone (on ne touche pas au constructeur pour ça)
ngDoCheck	Appelé après chaque détection de changements
ngAfterContentInit	Appelé une fois que le contenu externe est projeté dans le composant (transclusion)

Cycle de vie d'un composant (5/6)

Méthode	Rôle
ngAfterContentChecked	Appelé chaque fois qu'une vérification du contenu externe (transclusion) est faite
ngAfterViewInit	Appelé dès lors que la vue du composant ainsi que celle de ses enfants sont initialisés
ngAfterViewChecked	Appelé après chaque vérification des vues du composant et des vues des composants enfants.
ngOnDestroy	Appelé juste avant que le composant soit détruit par Angular. Il permet alors de réaliser le nettoyage adéquat de son composant. C'est ici qu'on veut se désabonner des Observables ainsi que des events handlers sur lesquels le composant s'est abonné.

Cycle de vie d'un composant (6/6)

Les méthodes **ngAfterContentInit**, **ngAfterContentChecked**,**ngAfterViewInit** et **ngAfterViewChecked** sont exclusives aux composants, tandis que toutes les autres le sont aussi pour les directives.

Références

https://angular.io/