static (ViewChild seulement)

- Le paramètre static dans le décourateur permet de changer le comportement de la view query
- La requète n’est effectuée qu’une seule fois, après l’initialisation du composant

Lifecycle Hooks

Cycle de vie des composants

- Considérons le composant suivant, qui prends une taille en @Input, et affiche un tableau de cette taille rempli de nombre aléatoires

```
export class TableauComponent {

  @Input({required: true})
  size!: number

  numberArray: number[] = [...Array(this.size)].map(() => Math.round(Math.random()*10))

}
```

- Un seul élément est crée dans le tableau, pourquoi ?

- Lors de l'appel au constructeur du composant, les bindings ne sont pas encore définis, this.size est donc undefinded

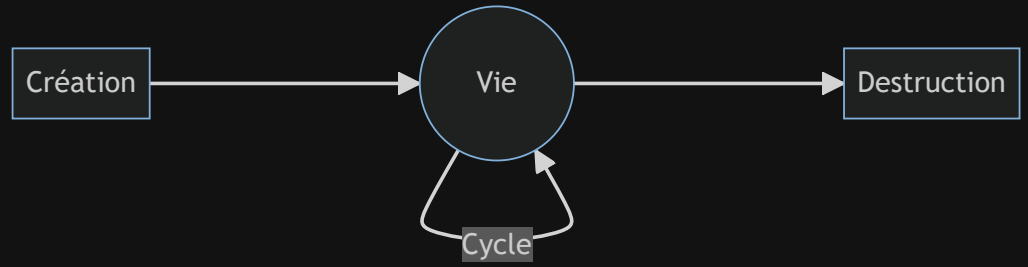
- Autre exemple : avec une View Query

```
@ViewChild(ChildComponent, {read: HighlightDirective})
childDirectiveRef!: HighlightDirective
```

- Dans cet exemple, childDirectiveRef est encore undefined dans le constructeur

```
constructor() {
  this.childDirectiveRef.hoverColor = 'orange'
}
```

Cycle de vie des composants



- Le cycle de vie d'un composant Angular comporte trois parties :
  - Création du composant
  - Vie du composant (detection cycle)
  - Destruction du composant
- Pour chaque composant, il est possible de définir des hooks, des méthodes qui seront appelées automatiquement à des moments précis du cycle de vie
- Il est également possible de définir des hooks pour des directives, pipes et services

Liste des hooks

Création

- constructor()
- ngOnInit()
- ngAfterContentInit()
- ngAfterViewInit()

Création + Vie

- ngOnChanges()
- ngDoCheck()
- ngAfterContentChecked()
- ngAfterViewChecked()

Destruction

- ngOnDestroy()

- Les hooks de vie du composant sont égalements appelés lors de la création
- A chaque hook est associé une interface

ngOnChanges(changes: SimpleChanges)

- Appelé une première fois après que les attributs @Input soient initialisés
- Puis à chaque fois qu'un attribut @Input est modifié
- Un paramètre SimpleChanges permet de récupérer les changements

```
export declare interface SimpleChanges {
    [propName: string]: SimpleChange;
}
```

```
export declare class SimpleChange {
    constructor(previousValue: any, currentValue: any, firstChange: boolean);

    previousValue: any;
    currentValue: any;
    firstChange: boolean;
    isFirstChange(): boolean;
}
```

ngOnInit()

- Appelé après que les bindings du composants aient été initialisés (et après le premier ngOnChanges)

ngDoCheck()

- Appelé à chaque detection cycle
- Peut être utilisé pour détecter des changements que ngOnChanges ne prend pas en compte, comme des mutations d'objets
- Appelé avant qu'Angular ne mette à jour le template
- ⚠ Eviter d'appeler des méthodes coûteuses

ngAfterViewInit()

- Appelé une fois après que le template ait été initialisé

ngAfterViewChecked()

- Appelé à chaque detection cycle, une fois que le template ait été mis à jour

- ⚠ Ne pas modifier la valeur des atributs interpolés dans les hooks ngAfterViewInit() et ngAfterViewChecked(), au risque d'avoir un état incohérent entre la valeur des attributs et ce qui est affiché à l'écran

ngOnDestroy()

- Appelé une fois lors de la destruction du composant
- Peut être utilisé pour unsubscribe à des Observables

Detection de changement

- La vie d’une application Angular est rythmée par les cycles de détection de changement
- A chaque cycle de détection de changement, Angular met à jour l’affichage de l’application
- Un cycle de détection de changements se déclenche lors des évènements suivants :
  - Evènements du DOM
  - setTimeout() and setInterval()
  - Requètes HTTP

Cycle de détection de changement

- Parent ngDoCheck()
- Appel des méthodes du template parent
- Parent view update
- Child ngDoCheck()
- Child view update
- Child afterViewChecked()
- Parent afterViewChecked()
- Appel des méthodes du template parent

Remarques (1/2)

- Par défaut, Angular vérifie pour chaque composant si il y a un changement d'affichage à faire
- La vérification se fait du composant racine vers les enfants
- Il est possible dans le hook ngDoCheck() de modifier des données du père vers les enfants, mais pas l'inverse

Remarques (2/2)

- Les méthodes dans le tempalte sont appellées deux fois en développement
- Angular effectue un controle pour s'assurer que rien n'ait changé après la mise à jour de l'affichage
- L'erreur NG0100: ExpressionChangedAfterItHasBeenCheckedError, est lancée par Angular en développement uniquement si une valeur change après que la détéction ait eu lieu

ChangeDetectionStrategy

- Lors d’un cycle de détection de changement, Angular ne sais pas quels composants doivent être actualisés
- La vérification se fait sur tous les composants, ce qui peut être couteux pour de grosses applications

ChangeDetectionStrategy.OnPush

- La stratégie OnPush permet de réduire le nombre de vérification à faire lors de la détection de changement
- L'option se configure au niveau des métadonnées du décorateur @Component

```
changeDetection: ChangeDetectionStrategy.OnPush
```

- Le composant ne sera vérifié uniquement si il est marqué comme dirty
- Les hooks ngDoCheck() et ngAfterViewChecked() d'un composant sont appelés, même si celui-ci n'est pas dirty
- Par contre les hooks des enfants ne sont pas appelés

ChangeDetectionStrategy.OnPush

- Un composant est marqué automatiquement à dirty avec les actions suivantes
  - Un évènement du DOM lié à ce composant à lieu
  - Une variable @Input ou @Output de ce composant change
- Lorsque un composant est marqué dirty, tous ses parents sont égalements marqués
- Il est également possible de marquer un composant dirty manuellement en injectant le service ChangeDetectorRef

```
cdr = inject(ChangeDetectorRef)
```

- Et en utilisant la méthode markForCheck()

```
this.cdr.markForCheck()
```