**Component Lifecycle**

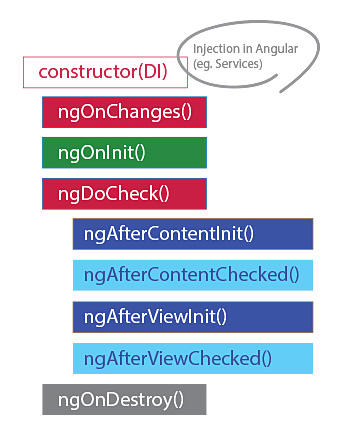
A component instance has a lifecycle that starts when Angular instantiates the component class and renders the component view along with its child views. The lifecycle continues with change detection, as Angular checks to see when data-bound properties change, and updates both the view and the component instance as needed. The lifecycle ends when Angular destroys the component instance and removes its rendered template from the DOM. Directives have a similar lifecycle, as Angular creates, updates, and destroys instances in the course of execution.

**Common Lifecycle hooks for both Directive and Component**

1. ngOnChanges()
2. ngOnInit()
3. ngDoCheck()
4. ngOnDestroy()

Component has other four lifecycle hooks also ,

1. ngAfterContentInit()
2. ngAfterContentChecked()
3. ngAfterViewInit()
4. ngAfterViewChecked()



1. ngOnChanges() :

Respond when Angular sets or resets data-bound input properties. The method receives a [SimpleChanges](https://angular.io/api/core/SimpleChanges) object of current and previous property values.

Note that this happens very frequently, so any operation you perform here impacts performance significantly.

Timing :

Called before ngOnInit() (if the component has bound inputs) and whenever one or more data-bound input properties change.

Note that if your component has no inputs or you use it without providing any inputs, the framework will not call ngOnChanges().

Ex:

import { Input, SimpleChange, SimpleChanges } from '@angular/core';

import { Component, OnChanges } from '@angular/core';

@Component({

  selector: 'app-ng-on-changes',

  templateUrl: './ng-on-changes.component.html',

  styleUrls: ['./ng-on-changes.component.css']

})

export class NgOnChangesComponent implements OnChanges{

  @Input('name') name:any;

  @Input('age') age:any;

  constructor() { }

  ngOnChanges(changes:SimpleChanges){

    // const change1:SimpleChange=changes.name;

    // const change2:SimpleChange=changes.age;

    // console.log("ngOnChanges()")

    // console.log(`Previous Value::${change1.previousValue}....Current Value::${change1.currentValue}`);

    // console.log(`Previous Value::${change2.previousValue}....Current Value::${change2.currentValue}`);

    for (const propName in changes) {

      const chng = changes[propName];

      const cur  = JSON.stringify(chng.currentValue);

      const prev = JSON.stringify(chng.previousValue);

      console.log(`${propName}: currentValue = ${cur}, previousValue = ${prev}`);

  }

}

}

Parent html

----------

<div class="row">

  <input class="col-md-5 offset-1 border-danger" type="text" [(ngModel)]="name">

  <input class="col-md-5 border border-info" type="number" [(ngModel)]="age">

</div>

<app-ng-on-changes [name]="name" [age]="age"></app-ng-on-changes>

<router-outlet></router-outlet>

1. ngOnInit()

Initialize the directive or component after Angular first displays the data-bound properties and sets the directive or component's input properties.

Timing:

Called once, after the first ngOnChanges(). ngOnInit() is still called even when ngOnChanges() is not (which is the case when there are no template-bound inputs).

Ex:

ngOnInit(): void {

   console.log("ngOnInit()")

  }

## **Difference between constructor and ngOnInit in Angular**

* We use *constructor*() for all the initialization of class members or variables.
* The *constructor*() should only be used to initialize class members and shouldn’t do other “work”.
* You can use *constructor*() to setup Dependency Injection so that you can Initialize class members.
* *ngOnInit*() is a better place to write other initialization tasks that we need to execute once the class is instantiated.
* For example, if you want to load data from Database in your HTML template view. Then such code should be written in *ngOnInit*() method.

## **Conclusion**

* *Constructor*() should be used only for initialization of class members.
* *ngOnInit*() method should be used for other initialization tasks which need to be executed right after class is instantiated.

1. ngDoCheck()

Detect and act upon changes that Angular can't or won't detect on its own.

To monitor changes that occur where ngOnChanges() won't catch them, you can implement your own change check

Timing:

Called immediately after ngOnChanges() on every change detection run, and immediately after ngOnInit() on the first run.

Ex:

ngDoCheck(): void {

    console.log("ngDoCheck()");

  }

4.ngAfterContentInit()

Respond after Angular projects external content into the component's view, or into the view that a directive is in.

Timing:

Called once after the first ngDoCheck().

Ex:

<app-life-cycle-methods [name]="name" [age]="age" >

 <h1 #childTemp>{{salary}}</h1>

</app-life-cycle-methods>

@ContentChild('childTemp',{static:true}) salary:any;

ngAfterContentInit(): void {

    console.log("ngAfterContentInit()"+this.salary);

  }

5.ngAfterContentChecked()

Respond after Angular checks the content projected into the directive or component.

Timing:

Called after ngAfterContentInit() and every subsequent ngDoCheck().

Ex:

 ngAfterContentChecked(): void {

    console.log("ngAfterContentChecked()  "+this.salary);

  }

6.ngAfterViewInit()

Respond after Angular initializes the component's views and child views, or the view that contains the directive.

Timing:

Called once after the first ngAfterContentChecked().

Ex:

ngAfterViewInit(): void {

    console.log("ngAfterViewInit()  "+this.salary+"....."+this.company.nativeElement.value);

  }

7.ngAfterViewChecked()

Respond after Angular checks the component's views and child views, or the view that contains the directive.

Timing:

Called after the ngAfterViewInit() and every subsequent ngAfterContentChecked().

Ex:

ngAfterViewChecked(): void {

    console.log("ngAfterViewChecked()  "+this.salary+"....."+this.company.nativeElement.value);

  }

8.ngOnDestroy()

Cleanup just before Angular destroys the directive or component. Unsubscribe Observables and detach event handlers to avoid memory leaks.

Timing:

Called immediately before Angular destroys the directive or component.

Ex:

ngOnDestroy(){

     console.log("ngOnDestroy()  "+this.salary+"....."+this.company.nati veElement.value);

  }

import { AfterContentChecked, AfterContentInit, AfterViewChecked, AfterViewInit, ContentChild, DoCheck, ElementRef, Input, OnDestroy, OnInit, SimpleChange, SimpleChanges, ViewChild } from '@angular/core';

import { Component, OnChanges } from '@angular/core';

@Component({

  selector: 'app-life-cycle-methods',

  templateUrl: './life-cycle-methods.component.html',

  styleUrls: ['./life-cycle-methods.component.css'],

})

export class LifeCycleMethodsComponent implements OnChanges, OnInit , DoCheck,AfterContentInit,AfterContentChecked,AfterViewInit,AfterViewChecked,OnDestroy{

  @Input('name') name: any;

  @Input('age') age: any;

  @ContentChild('childTemp',{static:false}) salary:any;

  @ViewChild('company',{static:true}) company:ElementRef=new ElementRef(name);

  constructor() {

    console.log("Constructor()")

  }

  ngOnChanges(changes: SimpleChanges) {

    // const change1:SimpleChange=changes.name;

    // const change2:SimpleChange=changes.age;

    // console.log(`Previous Value::${change1.previousValue}....Current Value::${change1.currentValue}`);

    // console.log(`Previous Value::${change2.previousValue}....Current Value::${change2.currentValue}`);

    console.log("ngOnChanges()")

    for (const propName in changes) {

      const chng = changes[propName];

      const cur = JSON.stringify(chng.currentValue);

      const prev = JSON.stringify(chng.previousValue);

      console.log(

        `${propName}: currentValue = ${cur}, previousValue = ${prev}`

      );

    }

  }

  ngOnInit(): void {

   console.log("ngOnInit()  "+this.salary+"....."+this.company.nativeElement.value);

  }

  ngDoCheck(): void {

    console.log("ngDoCheck()....."+this.company.nativeElement.value);

  }

  ngAfterContentInit(): void {

    console.log("ngAfterContentInit()  "+this.salary+"....."+this.company.nativeElement.value);

  }

  ngAfterContentChecked(): void {

    console.log("ngAfterContentChecked()  "+this.salary+"....."+this.company.nativeElement.value);

  }

  ngAfterViewInit(): void {

    console.log("ngAfterViewInit()  "+this.salary+"....."+this.company.nativeElement.value);

  }

  ngAfterViewChecked(): void {

    console.log("ngAfterViewChecked()  "+this.salary+"....."+this.company.nativeElement.value);

  }

  ngOnDestroy(){

    console.log("ngOnDestroy()  "+this.salary+"....."+this.company.nativeElement.value);

  }

}

App.component.html

--------------------

<div class="row">

  <input class="col-3  offset-1 border-danger" type="text" [(ngModel)]="name">

  <input class="col-3  offset-1 border border-info" type="number" [(ngModel)]="age">

  <input class="col-3  offset-1 border border-info" type="number" [(ngModel)]="salary">

</div>

<app-life-cycle-methods [name]="name" [age]="age" >

 <h1 #childTemp>{{salary}}</h1>

</app-life-cycle-methods>

<router-outlet></router-outlet>

**Flex Layout**