ANGULAR NOTES

By: Deepa Chaurasia

https://www.linkedin.com/in/deepa-chaurasia-3704351a8

Page
Date

What is Data Binding?

→ Communication between typescript code of your component and the template (html) that the user see.

Data Binding == Communication

We get different ways of communication.

For eg → we want to output data from our typescript code in html code.
we can use String Interpolation for this

Syntax for String Interpolation

{{ data }}

you put your data that will come from ts in double curly braces

Page
Date

Eg of String Interpolation

Eg → In your ts file.

```
export class MyComponent {
  name : string = 'deepa';
}
```

In your html file

```
<p> My name is {{ name }} </p>
```

// will print My name is deepa.

⇒ Now Suppose you have a button to submit, It must be enabled only if you're admin.

In this case you'll use property binding

☆ The property [disabled] will bind to native boolean which we'll define in ts.

Eg of Property binding

In ts File.

```
export class MyComponent {
  isAdmin : boolean = true;
}
```

In html

```
<button [disabled] = "!isAdmin"> Add </button>
```

// if not admin then disable

Event Binding → It allows your component to react to user's actions such as button click, key strokes, etc.

Like property Binding use [] square brackets

Event Binding use () paranthesis

Let's say on click of button I want to show something.

In ts File

```
export class MyComponent {
  name : String ;
  onClick() {
    this.name = 'deepa';
  }
}
```

In html

```
<p> My name is {{name}} </p>
<button (click) = "onClick()" Add </button>
```

So before clicking Add, name will be blank
After clicking Add, name will be shown (i.e. deepa)

Page
Date

What are Directives?

Directives are instruction in the DOM!

Basically these are custom html attributes which tell angular to change the style or behaviour of Dom elements.

Components are building blocks of Angular applications, Similarly directives are also building block of Angular Applications.

Types of Directives

Component	Structural Directives	Attribute Directives
Directives with a template	Change layout of the elements	Change appearance or behaviour of a particular element

1. Component / Directives

* Directives must be declared in Angular Modules in the same manner as components.

* Directive in Angular is a re-usable component.

* The other 2 directives such as structual and attribute do not have templates

It is decorated with @ Component decorator

Example :- change-text directive

change-text.directive.ts

```
import { directive, ElementRef } from '@angular/core'

@Directive ({
    selector : '[changeText]'
})

export class ChangeTextDirective {
    constructor (Element : ElementRef) {
        Element.native Element.innesText =
            "Text is changed";
    }
}
```

app.component.html

<span changeText> Welcome

In above eg, there is class called ChangeTextDirective and a constructor, which takes the element of type ElementRef, which is mandatory.

The element has all the details to which change Directive is applied.

2. Structural Directives :

* Structural Directives are responsible for changing structure of DOM
* They work by adding or removing element from DOM.
* The Structural directive always starts with '*'

Three most popular Structural Directive

1) ngIf
2) ngFor
3) ngSwitch

All of them work same as If, for or switch respectively.

Except that you use it for DOM Tree in html template

Eg →

```
<div *ngIf = "info" class = "name" >
    {{ info.name }} </div>

<div *ngFor = "let movie of movies" >
    {{movie.name }} </div>
```

3) Attribute Directives

Attribute Directives manipulate DOM by changing its behaviour and appearance.

* It is used to show or hide elements or dynamically change behaviour of component.

* Built in Attributes are ngStyle, ngClass etc.

— By Deepa Chaurasia

What are Modules In Angular?

Angular doesn't scan automatically all your components, directives etc.

You have to tell Angular what component, directives, service your project have. Or you are using

Modules does that Job for you

Module tell Angular list of all components, directives or services we are using.

It is basically a place where you group together all your components, directives, services.

```
@ngModule ({
    declaration: [ AppComponent,
                   XComponent,
                   YComponent. ],
    imports: [ AppRoutingModule,
               BrowserModule ... ],
    providers: [ ], [Services goes here)
    bootstrap: [AppComponent],
    entryComponents: [ ]
})
export class AppModule { }
```

1. @NgModule - Angular analyzes ngModules to understand application and it's features.

You can't use a feature without including it in @NgModule

2. Imports - Imports are used for importing other modules like FormsModule, RoutingModule etc. and their features into our main Module.

3. Providers - It defines all services we are providing in our angular app.

All services declare in your app must be inside the Providers of APPMODULE

OR

You have one other method to import your services throughout root of application

```
@Injectable ({
    providedIn : 'root'
})
```

If you use @Injectable, You don't have to declare it in providers of Module.

Remember @Injectable ({
provided In : 'root'
})

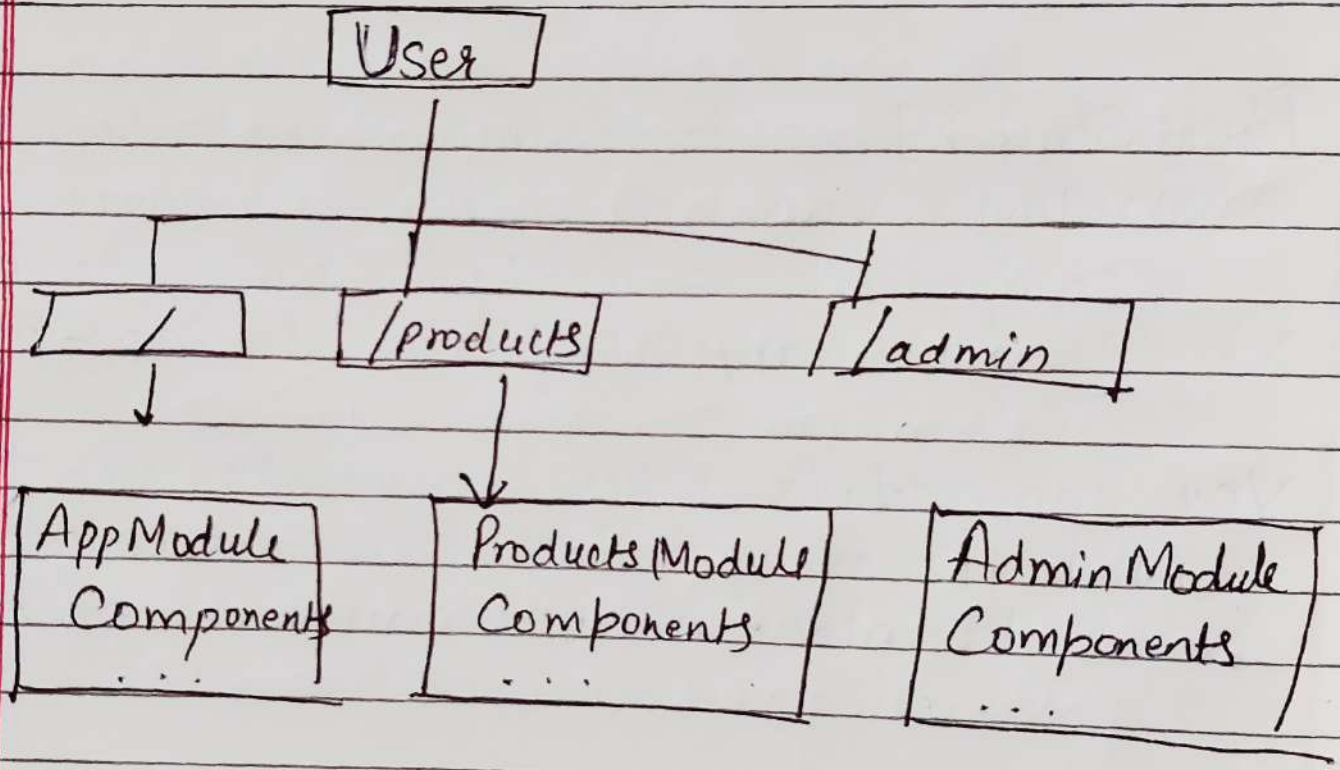
Will come above every service in Service Component.

BootStrap[] - Bootstrap is important for starting your app. It defines which component is available straight in your app at first. (i.e app-root)

You can include other components also that's why it's array. But it is not likely to add other components generally

Lazy Loading In Angular

Consider, Our application has 3 routes, Every routes has a module Associated with it that contains their respective component, services and directives.



Here you can see we have 3 routes

→ Root
→ Products
→ Admin

Now when user visit root route then we must be loading the respective modules, and then we load other modules when required.

But does it really Happen??

The answer is NO.
Angular actually load all the modules whenever we visit any page.

It's like you ordered starter, maincourse and desert

And chef is saying No I will make all starter, maincourse, desert first then start serving.

You definetely want atleast your starter, first, and ask chef to make rest when it's required.

So Lazy Loading does the same for you.

In Lazy Loading the modules are loaded only if the user hit the route of that particular Module.

Advantages of Lazy Loading In Angular

* Initially we load a smaller bundle and moduls for our root route.
* We load more modules / bundle as we hit route for following.
* So our first page loads faster, and it doesn't take much time becoz bundles are comparitively less to download now.
* Basically your application size shrinks and it becomes faster.

Good Optimization Can be achieved with Lazy Loading.

→ Notes to add

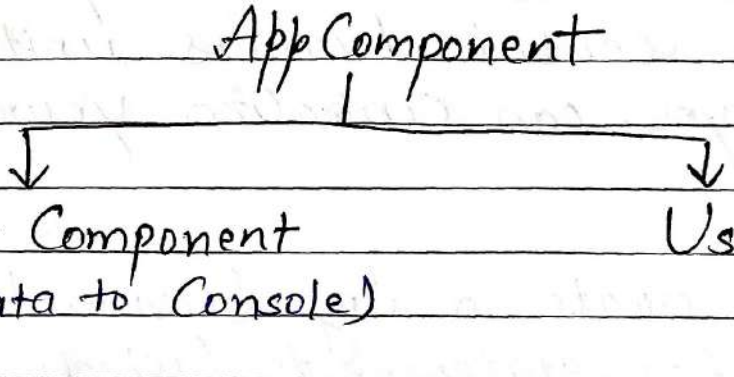
* you can read about PreLoading Strategy
* dynamic importing of Components and render using Component Factory for better Angular Build Optimization

Page
Date

Services and Dependency Injection

Angular services are ~~sfg~~ singleton object that gets instantiated only once in a lifetime of an application

Consider an application having:



We might have some methods in those components

Let's say we want to log some data in About Component and User Detail Component

Both logs are same, so basically we are duplicating code in 2 different component

Suppose in User Component, we also want to access some data, some array of users and we don't know if we will later use that in other part of application

Page
Date

Both User Cases

* Duplication of Code
* Data Storage

In both cases we, typically use Service

A Service is just another piece in your Angular App, Another class which acts as a central repository, as a central business unit where you can centralize your code

We can create a log Service to centralize our log statement (which is used in 2 components)

Service are also useful for communication between components

How to create a Service

* Simply run command
  ng g s service-name

your service will be generated

Page
Date

How do you Access a Service In your Component ?

Let's say you have made service named Logging Service

you can simply instantiate your Logging Service in component

account.component.ts

* ~~you can~~ const service = new LoggingService();
* Import the Logging Service
* service.logstatus (status);
  ↓
  your method you wanted to call from service in component.

But this is not suggested, Angular provides us more better way to Inject Service in Our Component.

Angular Dependency Injector

Angular provides us more better way to inject service in component It is Angular dependency Injector

So, we have dependency on logging Service from our component and dependency injector simply injects this dependency instance of our class automatically into our component.

We need to only Inform Angular, that we require such an instance.

How do we Inform that?

* We add constructor to the component you'll see constructor present If you have made service through command component
* Inside constructor, I can bind it to property

```
constructor (private logging: LoggingService) {
}
```

(we use this access specifier (must))

(Imp to specify type and that should be service)

(make sure to import logging Service at top)

Page
Date

Now Angular know we want logging Service But it doesn't know how to give Instance

* We need to provide a service
Add one providers property In @Component decorator.

```
@Component ({
  selector;
  templateURL:
  StyleURL:
  providers: [Logging Service] → [Here we have provided]
});
```

Hierarchical Injector

Angular dependency Injector is actually a hierarchical Injector.

It means that if we provide a service in one component, then Angular framework knows how to create instance of service for this component and important, all it's child components

All of them will receive same instance of service

Date

Hierarchical Injector

App Module → Same Instance of Service is available Application-wide

App Component → Same Instance of Service is available for all components (but not for other service)

Any other Component → Same Instance is available for component and all its child components.

It works Top to down not down to Top

Suppose If you want to use same Instance of Service in App Component to child component

Then you don't use providers in child component

Everything else stays same.

Page
Date

Injecting Services into Services

The highest level of providing service. is provide your service in providers array of App.Module.TS

If we are providing service in module, we make sure we are using same instance of service everywhere, in every component throughout app Unless we are overriding our service.

Route Guards

What is Route Guards ?

They are interface provided by angular which when implemented allow us to control the accessibility of route based on condition provided in class implementation of that Interface

Five types of Route Guards

- ★ Can Activate
- ★ Can Activate Child
- ★ Can Load
- ★ Can Deactivate
- ★ Resolve

1. Can Activate → Can Activate

Auth Guard Service
(define class AuthGuard
Implement canActivate Method)

will call everytime user navigate to this field

{path : 'user/:id', Component : HomeComponent
Can Activate : [AuthGuard]}

// Auth Guard Service

canActivate (route : ActivatedRouteSnapshot,
state : RouteStateSnapshot) :
Observable <boolean> | Promise <boolean> {

{
return true; (If navigation happen)
}

Now Use this service

2. CanActivateChild – Similar to canActivate but also prevent access to child routes of given Route.

3. CanLoad – In angular we Implement lazy loading to download only required modules using loadchildren.

If we want to prevent unauthorized user we can use CanActivate Guard.

that will do job but also download Module.

Now to control navigation as well as prevent downloading of that module we can use CanLoad Guard

canLoad (Route: Route, segments: UrlSegment[])
: Observable <boolean> | Promise <boolean>

{
  return true; (whether want to download module or not.)
}

4. canDeactivate – This guard is useful when you want to prevent user from accidentally navigating away without saving or some undone task.

canDeactivate (component: T, currentRoute: ActivatedRouteSnapshot, currentState: RouterStateSnapshot, nextState?):
Observable <boolean | UrlTree> | Promise <boolean | UrlTree> | boolean | UrlTree;

Here T is interface, that we will be using in component as well as guard

Page
Date

// CanDeactivate Service

→ export interface canComponentDeactivate {

canDeactivate : () ⇒ Observable <boolean>
| Promise <boolean> | boolean;

}

→ export class CanDeactivate implements
CanDeactivate < canDeactivate Component Deactivate>

{
CanDeactivate (component : CanComponentDeact
, currenRoute , current state, next state)

{ return component. canDeactivate();

}

}

Now Implement canComponentDeactivate
interface in component.

→ export class EditComponent implements OnInit,
CanComponentDeactivate {

canDeactivate (): Observable <boolean>
| Promise <boolean> {

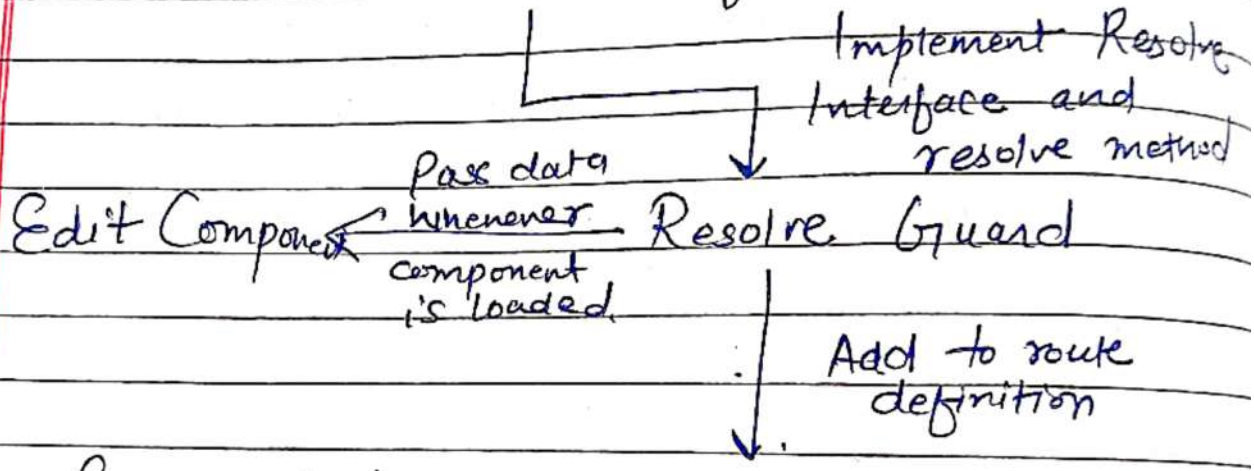
}

}

5 Resolve Guard

While data communication b/w components, Sometimes data is so heavy that is not possible to pass data through query params.

To handle this we have Resolve Guard.




```
{ path: 'id', component: EditComponent,
    resolve: { data: ResolveGuard } }

// Resolve Guard Service

resolve (route, state)
    {
        return this.fetchService.get (+ rout.
                                        params['id']
    }
```

Difference between Constructor and ngOnInit in Angular

ngOnInit	Constructor
ngOnInit is just a method on a class, it's completely a choice to use it or not.	Constructor will always be called, whether you implement it or not
ngOnInit is a place to put code we need to execute first as soon as class is Instantiated	It is used for declaration and initialization of classes and sub classes
ngOnInit is starting point to our application	It is used to set Dependency Injection
Angular calls ngOninit once it finished creating component DOM, injected all required dependencies	constructor | first ngOnchange | ngOnInit

Page
Date

Angular bootstrap process consist of 2 major stages

1) Constructing Component Tree
2) running change detection

Constructor of Component called at 1st stage, when angular constructs component tree.

All lifecycle hooks Including ngOnInit are called in 2nd stage. In change detection phase.

Usually a component Initialization logic requires either some DI providers or available input bindings or rendered DOM.

When angular constructs a component tree the root modules injector is already configured so you can inject global dependencies

When Angular Instantiates a child component class the injector for parent component is already setup. So you can inject providers defined on the parent component including parent component itself.

A constructor is the only method that is called in context of injector so if you need any dependency that's the only place to get those dependencies.

The @Input communication mechanism is processed as part of following change detection phase so Input bindings not available in constructor

Date

All about @NgModule In Angular.

An NgModule class is marked by @NgModule decorator.

@NgModule takes a meta data object

```
({ imports : [],
   declarations : [],
   providers : [].
   exports : [],
});
```

which tells components, directive and pipes belonging to the module., make some public by using export, so that other components can use them.

@NgModule can also add service providers to application dependency Injectors.

Different type of Angular Decorators

1) Class Decorators
2) Property Decorators
3) Method Decorators
4) Parameter Decorators

Class Decorators

They are top-level decorators that are used to define purpose of classes.

They tell if class is component or Module

```
@Component ( { } )
Eg → export class MyComponent {
        }
@NgModule ( { } )
    export class MyModule {
        }
```

Property Decorators

They are used to decorate specific properties inside classes.

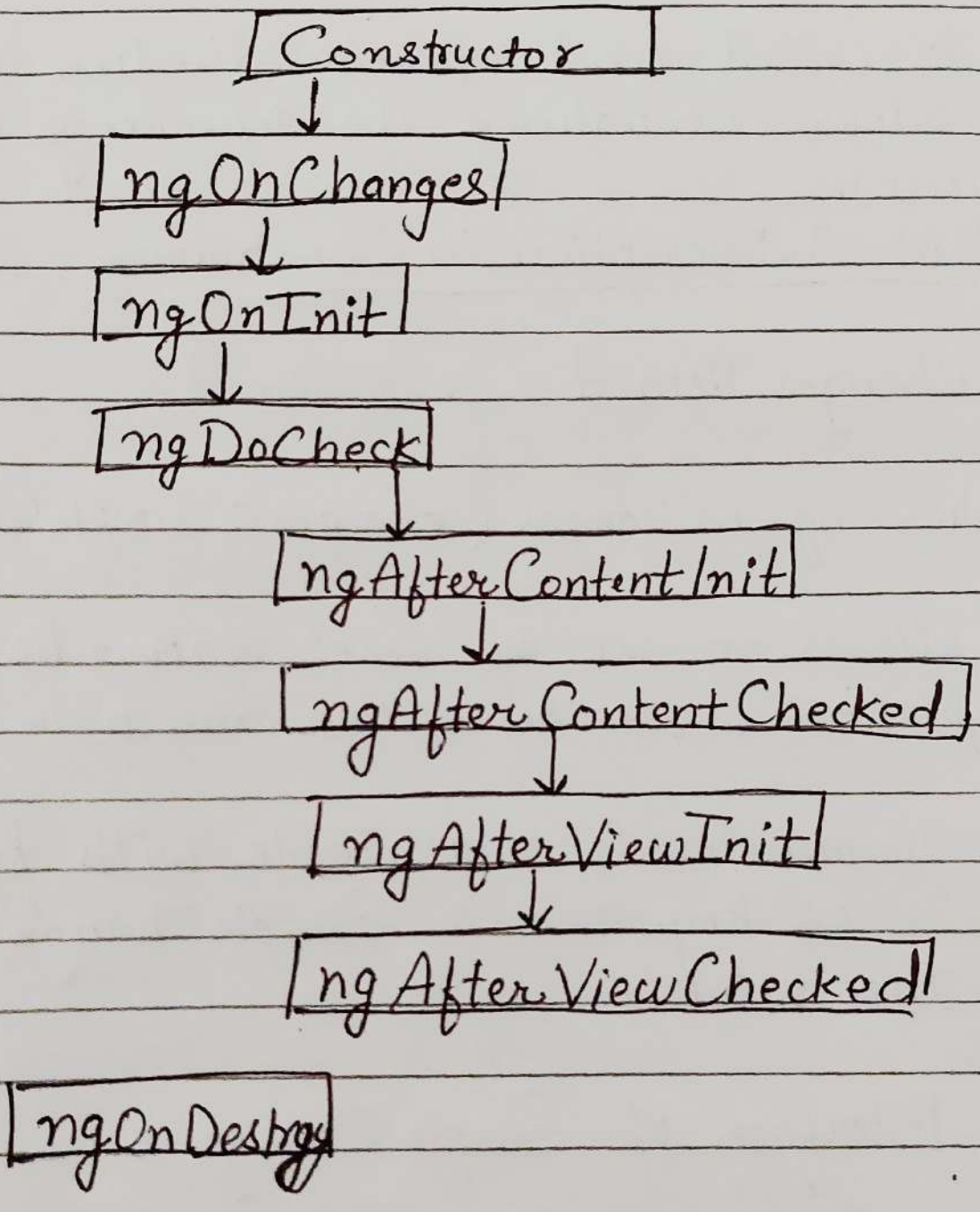
For eg- `@Input ( ) exampleProp;`

Here with decorator angular will automatically perform Input binding

Method Decorator- Used to decorate method defined inside our class with functionality.

Foreg @HostListner

Life Cycle of Angular Components



Instantiation

☆ Method : Constructor()

- It is called on Component / directive Instantiation
- It allows centralizing the dependency ~~Instantia~~ Injections.
- Avoid Subscription in Constructor

☆ On Changes Method

⇒ public ngOnChanges (changes : SimpleChanges) : void;

It allows to set or reset data-bound input properties (can be called multiple times).

The parameter is HashTable with properties name as key and a SimpleChange object as the value.

The Interface of SimpleChange is

```
export interface SimpleChange {
        previousValue : any;
        currentValue : any;
        firstValue : boolean;
}
```

Use Case → To perform light operations depending on data-bound Input properties.

☆ OnInit

⇒ public ngOnInit () : void ;

Use-case → It's called once after the first ngOnchanges().

Data-bound Input properties Initialized

It allows initializing the component/directive.

⇒ Here is a good place for Subscriptions.

☆ DoCheck Method

⇒ public ngDoCheck () : void ;

Use Case → It's called after every ngOnchanges and just after ngOnInit () first run.

In some cases, Angular can't detect changes on it's own.
For-eg — If an Input property is object and it's reference does not change, but one property does, It will not be detected by Angular.

Be careful while using this

☆ After Content Init Method

⇒ public ngAfterContentInit() : void ;

Use-case → It's called once after ngDoCheck()

If you use one of @ContentChild to get contentChild , they are initialized here.

☆ After Content Checked Method

⇒ public ngAfterContentChecked() : void ;

Use-Case → Same as ngDoCheck() but for ContentChild properties

☆ After View Init Method

⇒ public ngAfterViewInit : void ;

Use-Case → It's called only once after first ngAfterViewInit

☆ After View Checked Method

⇒ public ngAfterViewChecked() : void ;

Sames as ngDOCheck() byt for ViewChildren

☆ On Destroy Method

called Immediately before Angular destroys the component/directives.

Thankyou For Reading

Like,Share and Comment

https://www.linkedin.com/in/deepa-chaurasia-notes