Section 3 | Decorators |

→ used with @ sign prefixed
→ Angular classes & properties in them defined by decorators that sit on them
→ In angular everything is class, But these decorators transform them into component, module service or directive

Few decorators in Angular:

@NgModule @Component @Directive @Pipe @Injectable
@inject @Input @output @Viewchild @Contentchild
@contentChildren @HostListener @HostBinding

⇒ you can see project → src see all ts files like app-routing-module.ts, -ll-.spec.ts, app.module.ts all contains some decorator
⇒ all decorators start with @
⇒ your can hover over any decorator & you will eget info about it

☆ that module is root module

Module in angular

angular must have at least one NgModule class

→ Angular apps are modular
⇒ Angular app has own modularity ⇒ "NgModules"
⇒ modules are great way to organize & extend capability of application from external library
⇒ these exist many module in angular Ex:
FormsModule, HttpClientModule, & RouterModule
⇒ Also has extend available like angularmaterial, Ionic, AngularFire, etc

⇒ How to create new module

Command: ng generate module name-of-module

⇒ module will be created in: src → app → name-of-module folder

File name will be [name-of-module.module.ts]

file will have some prewritten code [see file]

⇒ a) ngmodule decorator ⇒ @NgModule

⇒ b) a export class naming name-of-moduleModule

⇒ For a component or module angular will create a new folder

⇒ See [app.module.ts] and try to understand the content present inside it

↳ in NgModule we have

(a) declarations
(b) imports
(c) Providers
(d) Bootstrap

[these are some properties are used to configure angular]

declarations

It is mean to register an [angular Component, Pipe, Directive] on Angular Module

→ [Component, pipe, directive] ⟹ as a whole called as [declarables]

→ declarations : [AppComponent] → It is an array

→ declaration only accept Component, Pipe, directive

It rejects
⇒ @NgModule
⇒ @Injectable
⇒ A non-angular Class → typescript class.

⇒ Declarable which is registered on another module

⇒ A declarable used inside Angular Module without adding in declarations array result in error

so you can't use a component, directive, pipe, which is not added in declarations else will throw error

⇒ ng g service user/user → service
↳ angular not create folder for service so you have pass # like this

⇒ ng g c user → folder
↑ comp
add component file to user folder

also add class

⇒ ng g cl model/user
↑ class

model/user
↑ model folder, user class

⇒ Say add component in core
ng g c core/ComponentName

⇒ If you provide path so file will added in app folder

⇒ ng g d transform d is used to create new directives

⇒ ng g p pipes/pipe → files
↑ folder

⇒ As you do lot of changes the app.module.ts file will also changes you can see it

⇒ ng serve → this will compile the your angular code & this will show errors if have any

⇒ If another declaration has a component & you try to add it in another so that will also throw

if you have a component and you not added it in declaration say app.module.ts so you can't above access that <app-component> inside app.component.html

⇒ so if we want to use a module component, that should be in declarations

Imports array in app.module.ts ↓ declarables

as we know declaration know only (directive, component & pipes)

⇒ Imports is mean of extending capability exported by some angular module in angular module

⇒ Import will store modules from which we import some information

⇒ Accept only set of angular module

⇒ Error throw if we add declrables inside it

(automatically handled nowaday)

Exports array

Accepts ⇒ ⊙ declarable which registred on angular module

⊙ Exported declarations are module's public api

⊙ (Just directly use name of that compo it is easy to us) (we have to run <app-header>)

Core folder

↳ Header navacha component the

Core.module.ts

import [header Component]

↳ this should be exported and should imported in

first

Core cha component

↓

app.module.ts → im

Core exports

good Idea

app imports

app module connected to

Core module

there should always be someone to export & main app.module.ts to import

Say folder → has Component folder.ts
folder.module.ts

core → core.module.ts ⇒ should have folderComponent in import & export
app → Should have folderComponent in import app.module.ts

⇒ the Components which are there in app.module.ts can be used inside app.component.html

⇒ same apply for pipe, directive (~~direct~~ declarables)

⇒ now we played with compon

⇒ so if we import a module so the component pipes, also easily accessible

~~exports~~ providers

- providers is set of injectable objects
- provider array is there to handle dependency

Bootstrap — initialise ~~boot~~ app component along with app module

means to bootstrap component when a model is bootstrapped

Ex: Bootstrap: [AppComponent] — { set of startup component }

It ~~sta~~ Say that whenever angular start app module app component should also be ~~in~~ initialised.

⇒ Bootstrap is only present in app.module.ts that is main module.ts file

⇒ add
<app-new></app-new> in index.html
and add to newComponent in bootstrap [then it will] work

⇒ So all content in bootstrap array can be used to write & render directly in index.html along with <app-root> <app-root>

⇒ the bootstrap array must contains a component directive, pipe service all are invalid.

Entry Component

means to register components that can dynamically loaded into view

⇒ It only contains components
⇒ these are added in entryComponent list
⇒ these component inside list is stored & compiled dynamically at runtime & used when needed.

→ this is just theory so keep in mind we will learn in more

App bootstrap working mechanism

index.js / main.ts ⇒ entrypoints and they defined in angular.json

⇒ for working browse needed which support JIT architecture

angular.json → Configure all file in project
main.ts ⇒ Create execution environment for our App to run
App module ⇒ root module

* we inject app.Component.html to index.html using <app-root></app-root>

⇒ angular runs on dynamic JIT compilation process
→ app.component.ts ⇒ links html & css with the corresponding ~~component~~ component / module.

Components

→ a angular application is basically a tree of component
→ Angular component is most basic building block of (UI) in angular app
→ angular components are subset of directives

types of directive
- → structural directive
- → component directive → has template
- → attribute directive → don't have template

→ one component can instancious per element in template
for all root component ⇒ <app-root></app-root>

→ @component (a decorator) is a class and come with addition metadata to determine how component should processed, instantiated & used in runtime
⇒ to use a component in specific html we have to add that component in that specific module file inside declarations list
⇒ Angular component runtime behaviour by implementing various lifecycle hooks

app.component.ts

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

→ you can add more stylesheet also
→ a list as you can use multiple stylesheets

& in expat you can change title which look in top of website

app.component.ts

Store selector, if you write that selector in html which template (html) to inject & which stylesheet to use for it

---

Create a Component

ng g c component-name

→ this will create a folder called ~~named~~ component-name along with 4 file

component-name.component.css → stylesheet
component-name.component.html → template to inject
component-name.component.ts
component-name.component.spec.ts → unit test file for angular

File linkage & selector

(a) component also

two compone Abc xyz

Say you type

<app-abc> </app-abc>
<app-xyz> </app-xyz>

in app.component.html

so

app root is parent of app-abc, app-xyz &

<app-root>

<app-abc></> <app-xyz></>

app xyz & app-abc are sibling

⇒ location of files inside component.html defines the heiarchy of component tag

→ see developer tools

@component in component.ts file & it's metadata

(A) selector → the idea to use html tags

if selector: "app-grs" ⇒ you can use <app-grs> in component.html file

(B) template ⇒ you can directly provide small html template instead of creating html file

(C) used when you have small code snippet in a component (used sometimes)

⇒ you can write multiline code

(D) templateUrl ⇒ path to html file of component

(E) StyleUrls ⇒ A list where you can store stylesheet, you can add multiple style sheet to single component

(F) Styles ⇒ you can directly inject styling from component.ts file directly

(G) Providers array you can add some kind of service so now this will tell component to create new service don't use existing one (Service injection done)

(H) animations ⇒ you can add animation to angular app from compant, you have to use [BrowserAnimationModule] for this then you can import in app.module.ts then we import them in user.component.ts & use it in animation file list

a large topic learn after

(I) Encapsulation: style encapsulation to be done by component

(J) changeDetection: Change detection strategy used by component

---

View Encapsulation

⊕ View Encapsulation (Styling)

⊙ Emulated : default — inherit global property & local style are private (component specific)

⇒ If you apply some styling on global, it will apply to all & if styling in local (component specific) will apply only to specific component

```
<app-root style="1">
    <app-ghansham style="2">
        <app-gham style="3">
        </app-gham>
    </app-ghansham>
</app-root>
```

⇒ so styling of this apply to themselve and stylings of app-root also apply to them

⇒ this default strategy

⇒ see html code to analyse css in developer mode

Ⓑ native → deprecated

Ⓒ ShadowDom :

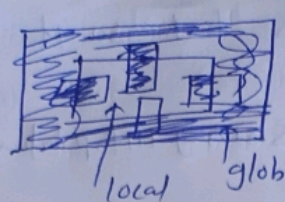
global styles will not inherited & local styles will kept private

viewEncapsulation ⇒ enum

@Component({ ... encapsulation : ViewEncapsulation.Emulated or : ViewEncapsulation.ShadowDom }) ← default

⇒ global style will not leak inside our component style

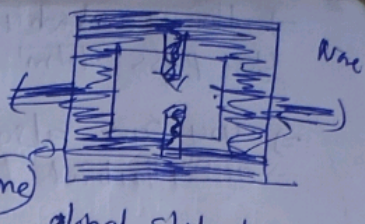
ViewEncapsulation.None ⇒ this will leak property from local to global & also global to local



Emulated
global style leak inside
local

Shadow
Dom

⇒ global style will
not leak in
local style

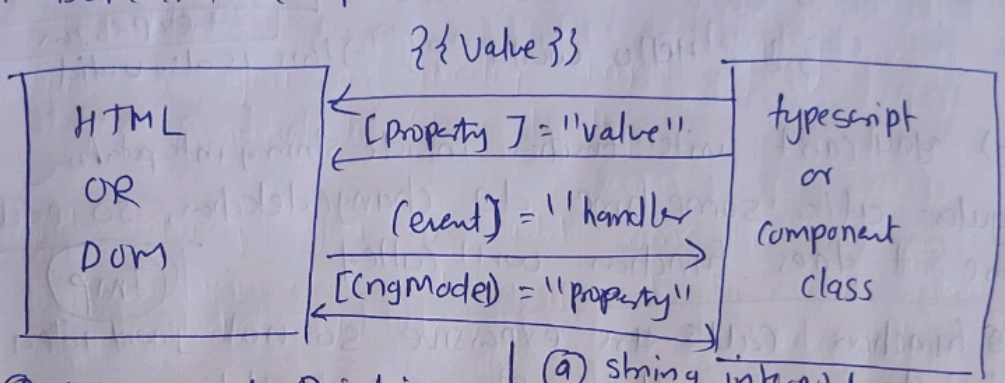


global styles leak
outside & inside
& also local style
can leak

⇒ None both local & global style will leak

ⓗ Bindings (communication)

these are used for communication betn html and typescript or DOM and component



ⓗ Component Binding

ⓐ String interpolation
ⓑ Property binding
ⓒ Event binding
ⓓ Attribute binding

ⓐ String interpolation

⇒ we embedd expression in {{ }}

use {{ some prop }} or {{ "text" }}

⇒ Angular first evaluate expression & convert into string

⇒ also can call method

{{ method() }}

Flows from component.ts file to component.html file

⇒ we can render variable, function from component.ts file to component.html file which exported

⇒ the content inside {{ }} is treated as text so if you pass html it will not render

⇒ you can also pass values using string interpolation as attribut

Ex: <p innerHTML = "{{ msg }}"

⇒ you have to pass as attribute if you want to render as html

⇒ So you can use string interpolation inside tags or also as tag attribute

Can pass value, props, function using String interpolation + expression

⇒ also you also add some stm

{{ "Hello world" }} → this is also valid

⇒ you can't write function inside string interpolation

⇒ angular after some time do changedetection, so each time it does function will called

Imp

⇒ & function calls are expensive so not good idea to do function call in string interpolation

Property binding

We send data from typescript of component to html file

⇒ so we can able to set properties of html element using property binding

say in

```
Component.ts                    → exported
m = "hello world"
```

Component.html

```
<p [innerText] = "m"></p>
```

attribute will be inside

you can put property in double quotes & [square brakets]

<a [src] = "source" > hindi
getting property value from component.ts file

⇒ you can send value also.

value

<p [innerText] = "'hello I am ghanshyam'" >

double quote & single quotes should handled carefully

⇒ you can also call function in property binding

abc(num: number)

<p [innerText] = "abc(100)" > </p>

Such like that you can able to see & provide to property of html tags

also expression

⇒ can pass value, property function using property binding.

⇒ Inside this also time exist large no change detection so multiple function calls so this is expensive

⇒ Avoid calling function here also

Event binding

html ——————→ component.ts

so event happens on dom & we so handle that event on component.ts file

we have pass event from html elem

<button (event name) = "someExpression" >

(eventname) = "some method(arg)" or

So such like that you can handle event from which send from html file & will get handled in ts file

⇒ for a event you can provide some expression or you can provide some handling function

⇒ use function only called if event triggered so like property binding or data binding functions will not get called multiple time at time of change detection

* Express cmt

(already known)

(1) Easy just like the (CSS) (JS) (rahi => agile)

(#) Component interaction Technique

→ we want to learn two way binding interaction but before that lets ~~run~~ run our eyes around component interaction

=> If component are present in parent & child heiarchy we can use input / output ~~directo~~ decorators to make that communication happen

(#) Input

→ only applicable for parent-child Reln
→ Data to be sent from parent to child
→ Use child getting data from parent so child use input decorator
→ Parent pass some property to child in template using property name

@ input

In

We send value from Parent to child with help of selector & property value

klpd = 1000

In app.component.html

```
<app-io [value] = "klpd" >
```

transfering value from app.component.ts to child

=> we have to handle it in

app-io ;component.ts and with @Input

now in ~~ap~~ io.component.ts (File)

```
@ input() klpd : number;
```

& now you can use klpd in (component.ts) file and also in

component.html

So for input

handle with @Input

write @Input in expart of child

component parent → html parent → component child → html child

[value] = "val"

Send value with property binding

⇒ such hierarchy is followed

For sending you also can use string interpolation {{}}

⇒ you can only pass boolean, number, string with string interpolation as it convert content to string

⇒ you can pass array using property binding

If you're passing multiple value so you have to use multiple input decorator abc xyz are variables from component.ts

<p [val1] = "abc" [val2] = "xyz"

So in child export

@input() a : any;

@input() b : any;

Such like you can manage multiple input for data send by parent

---

output (conv for parent child comm^n)

used to send data from child to parent

@output used for this communication

⇒ this property generally of type EventEmmiter <T>

⇒ child cell emit event which eventualy call a function binded to these properties via event binding

⇒ we have to listen event in parent's component.html & inside tag of childare

<app-io (newEvent) = "func()"> </app-io>

Format

we have to list

work flow

(A) Create output() event emitter

@output() name : eventEmitter<type> = new EventEmitter();

(B) Create a trigger in component.html of child which emit output

(C) Inside parent's component.html in child selector tag listen for that newEvent

(D) Create a event handler inside component.ts file of parent

=> Such like that you can handle communication between child ⟶ Parent

Key keep all thing in mind

Counter problem solve (see code) (Imp)

(☆) to handle event emitter in parent component you should have event listener in child's attribute

Two way binding

With this HTML and component can do two way communication

HTML DOM ⟷ [(ngModel)] = "property" ⟷ Typescript or Component

So [abc] = "property" → Property binding

(Component to html type)

(event) html ⟶ component

= "function / statement"

So two communication can done using

[(ngModel)] = "Property" → Remember the declaration

two way data binding syntax. → we send property not a handler function

two way data binding

⊙ An angular component use @Input to accept some & @output and EventEmitter to emit something

⇒ whatever this thing so we can see two way data binding use

How to implement

so in parent component we pass

<app-io [(county)] = "county"> ← Example

↑ two way binding

inside child.component.ts

while Exporting name of emitter should always be

), this name + Change

(counterChange)

emitter name must be ↑ this else this will not work

Say you want to create two way event binding for a value say (SKS)

so in child component while declaring event emitter name should be

(SksChange) else not work

& in Parent.Component.html you should pass

< app-child [(sks)] = "sks" > </app-child>

Imp

@ Keep that mind

⇒ the name of input property should be same in child.component.ts & for event emitter just add a change after it.

such like that we can use binding (Remember naming else is easy)

Attribute binding

→ html element has attribute

→ For each element browser create a javascript object & store all attribute as property

⇒ So attribute binding take place there is no element property is there to bind to

⇒ use set attribute when don't have corresponding to element property

---

⇒ with angular you can use property binding to provide value to html element

Ex: rowspan & colspan

```
td [attr.rowspan]="2" [colSpan]="3"> data </td>
```

- such like that we can provide data to html attribute using ts value

⇒ so you can fetch attribute of element using property binding

⇒ For some element you have [rowSpan] but not for all so you can use

[attr.rowspan]

↑ name like html & you can easily set value

⇒ so you can use html attribute directly or take value from component.ts file & then we can be able to manage it

* So use camelCase or use attr.attribute name in property binding to give value to some attributes of html element

Class-binding

we can use property binding to overwride css class prop.

[class] = "overridding css class property name"

or you can use → [attr.class] also

do same task

Also for toggling

to change className

div class="red" [className] = "class2" → only class will do same

[class] = "class2"

new styling will be applied

[className] = ".class2" [class] = "class2"

[attr.class] = "class2" ⇒ all work same & override previously set styles

[css.hell] = "property" → this if true class added else not.

so you can toggle it with button

So if we want to toggle class it depends on

this is good way to toggle

not so good way as we lost previous styles

Style binding

we use style & . dot operator to set style for html attribute

[style.color] = "'red'" such like that

⇒ cssclass or Ngstyle Directive is prefered though

You can use ngstyle & ngclass also

[ngStyle] = "{'background-color' : 'green'}"

[ngclass] = "{'class-Name1', 'class-Name2'}"

such like that you can use Ngclass & Ngstyle

⇓ you can pass objects to Ngclass & Ngstyle both are super helpful.

* Basically the things which you can done with html that can also done with angular

Directive

3 type of direchie are there

- Componant
  - Directive with template
- Structural directive
  - change the DOM By adding & Removing DOM Elements
- Attribute directive
  - Change appearance or behaviour of elements, components or another directives

lifecycle hook are applicable for both direchie & components

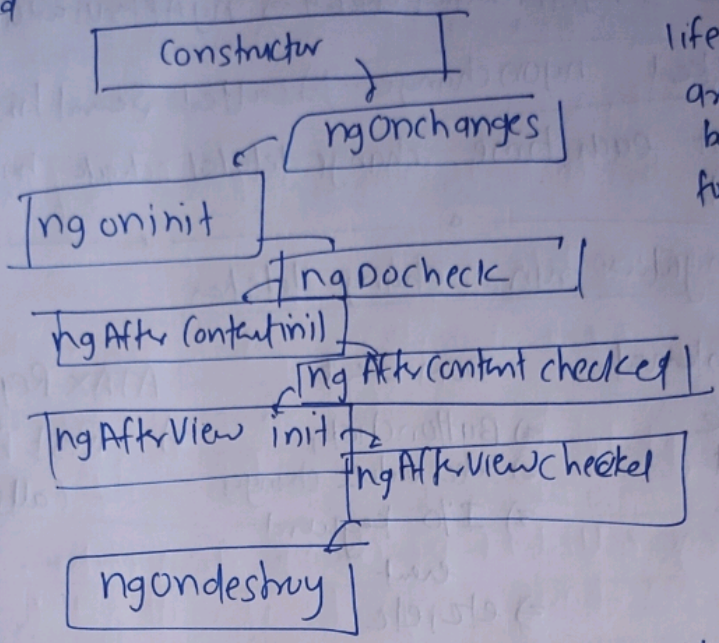
⇒ Componant & hooks have lifecycle which is manged by angular

⇒ angular provides hook which can tap in into key

For lifecycle hooks these are interfaces so if we want to implemant we have full fill it's contract such that implemant functions over it

So componant or directives are basically some class so inorder to implemnt then we have some heiarchy

while creating a component / directive several lifecyclehooks get called



lifecyclehooks are basic life based property for component or directive

⇒ these all lifecycle hook which decide behavior of component or directive

When these are called → also called again & again

lifecycle hook method		when
⚠ ngOnchanges	→	Everytime data bound @ input prop. changes
ngoninit	→	once aftr first ngOnchanges
⚠ ngDocheck	→	Everytime a change get detected
ngAftrContentinit	→	Once aftr first ~~Ngoninit~~ ngDocheck
⚠ ngAftr content checked	→	Right aftr ng Aftr Content init & everytime aftr ngDocheck
ngAftrViewinit	⇒	once aftr first ngAftrContentChecked
⚠ ngAftr view checked	⇒	Right aftr first ngAftr View init & eachtime aftr ng Aftr Content Checked
⚠ ngonDestroy	⇒	immediatly before Angular destroy directive or component

⚠ → symbol say these hooks several time

when changes detected

ngDocheck ⇒ ngAfterContentChecked ⇒ ngAfterViewChecked

Check, checked ngonchange ⇒ called Several times

logically right each time change detected check happens to follow it

How angular runs change detection

timeout / interval functions like
- ⇒ setTimeout
- ⇒ setInterval

DOM Event
- ⇒ Button clicks
- ⇒ Select value changes
- ⇒ I/O keyword event
- ⇒ etc, etc

AJAX Request
- ⇒ Rest API calls

#

⇒ ngAfterContentInit only called when angular finishes projection of content

Say child component has

<ng-content> </ng-content>

And you pass some HTML markup in opening & closing tags of child component

that is projection (injecting html in component)

ngAfterViewInit

frequency of lifecycle hooks

Constructor ⇒ only once (not part of lifecycle hook) but initiate it

called once: ngoninit, ngAfterContentInit, ngOnDestroy, ngAfterViewInit

called many times: ngOnCheck, ngDocheck, ngAfterContentChecked, ngAfterViewChecked

⇒ the lifecyclehook which contains 'name''init'' inside them you can edit them

⇒ try to avoid changing ngoncheck, ngAfterContentchecked, ngDocheck, ngAfterViewChecked

use cases of lifecycle hooks

props ⇒ properties

lifecycle hook method	use case
⚠ ngOnchanges	Act upon changes to data-bound @ input props
ngoninit	initialisse component/directive's properties
⚠ ngDocheck	Handle changes that angular won't detect
ngAfterContentInit	~~Handle~~ logic to access projected content can be written here
⚠ ngAfter ContentChecked	Respond content after projection of content
ngAfterViewInit	logic to access child's or component views
⚠ ngAfterviewchecked	Responds after angular checks component's or child's view
ngDestroy	unsubscribe observables, detach event handler, clear timeout or interval

⚠ ⇒ those who called very frequently ⇒ Preferably selected

we mostly use : ngOninit (which initalse Component or directive property)

ngAfterViewInit & ngdestroy

mostly used ⇒ 3 lifecycle hooks which we are going to manipulate

⊕ now lets use lifecycle hooks,
In all lifecyle hook we only have to learn about
ngOninit | ngAfterViewInit | ngDestroy

that's it only 3

⇒ note-item-ts note-list-item.component.ts / file

You can manipulate console output

console.log('%c String', 'color: red, font size: 30px')

in second quote you can provide style

see file very helpfull

⇒ ngonchange called if we change any value with input
→ If you not chang any valu it will not do any

P

⊕ ngOninit & ngOndestroy

⇒ see code and with ngOninit we can set some timer & ondestroy we apply some logic & can stop timer

✱ way to stop interval

clearInterval(this.interval) ⇒ this function used to stop interval

Such like that we can avoid memory leak using ~~ngDestroy~~ ngDestroy hook

⇒ lifecycles are just function you can manipulate the function very easily

⇒ also in the ngdestroy you can stop intervals, or timeout both way the ngDestroy used to avoid Memory leak

clearTimeout() ⇒ used to stop timeout
clearInterval() ⇒ used to stop interval] usefull

⇒ Also you can stop event listener using
⇒ removeEventListener() function

⇒ the life cycle function don't need to be mentioned in implement if not do it so it is fine

---

view query

<ng-content> </ng-content>

Say app.component.ts is parent

child.component.ts child so

<app-child>
--||String||--
</app-child>

So in child component you can get content inside.

app-child usage
<ng-content> </ng-content>

⇒ It is like parameter to html element which we can accept in child class using ng-content

---

Flow ⇒ Sham Component → sham-msg Component

⇒ View child using this you can access variable of child

Say a file

See sham & sham msg

Sham is parent sham-msg child

Say Sham ⇒ so we pass value from sham to shammsg and we can manipulate value from sham-msg using sham with use of view child

⇒ see code in

See resource folder

⇒ sham.component.html
⇒ sham.component.ts

sham-msg component.ts

three files only

⇒ we can only manipulate value in child in ngAfterViewChild function

View Children

⇒ ngAfterViewInit used to change content after content projection is done by ngAfterContentInit

View children
⇒ return values of all values passed by ~~child to~~ to child
⇒ output format will be Query list
you can use ~~cross~~ toArray() function to convert it into array & work on it.

and you can fetch values

So with viewchild & view children we are manipulating content which we sending to child

⇒ so now lets handle content which coming from parent

flow app component ⟶ ~~to~~ Show Component

note for all viewchild ~~view Child~~ - contentchild, contentchildren Component are used

⇒ <ng content> ⇒ use content child
↑ called as projected content

⇒ you assign some id to tags in ng

<app-root #hello> → act like id

↑ you can use it in Contentchild or viewchild

See last video of Component after some time