**Lifecycle Hooks in Angular**

Oh we're nearing the end of the section there is one thing you might have recognized before where we weren't sure what it does.

What's up with this ngOnInit() a method we have in new component's created through the CLI.

**What's it doing?**

ngOnInit() is a lifecycle hook and angular supports a couple of lifecycle hooks. Let's take a closer look. If a new component is created in angular. And of course angular is responsible for creating these components when it finds one of our selectors, for example it will instantiate a new word and of that component and add it into the DOM.

So once a new component is instantiated angular goes for a couple of different phases in this creation process and it will actually give us a chance to hook into these phases and execute some code.

We can hook into these phases by implementing some methods angular will call if they are present.

The first phase to first hook we can hook into is:

***ngOnChanges*** and this may actually be executed multiple times it's executed right at the start when a new component is created but thereafter it's also always called whenever one of our bound input properties changes.

And with that I mean properties decorated with @Input. So whenever these properties received new values.

Now the second hook is:

**ngOnInit** this method gets executed once the component has been initialized.

This does not mean that we can see it. It has not been added to the DOM yet so to say it has not been displayed yet but angular finished a basic initialization our properties can now be accessed and initialized for example.

So the object was created you could say.

And if you're interested *ngOnInit* would run after the constructor.

Then we have:

**ngDoCheck** that will also run multiple times.

Actually this method will be executed a lot because this will run whenever change detection runs. Now change detection simply used a system by which angler determines whether something changed on the template of a component or inside of a component or should say. So whether it needs to change something in the template so whether some property value changed from 1 to 2 let's say. And that property is output in the template.

Well of course angular needs to re render that part of the template and then we do check is a hook executed on every check angle or makes now important on every check.

So not just if something changed. A lot of times *ngDoCheck* will run because you clicked some button which doesn't change anything but still it's an event and on events angular has to check if something changed because how else would it know. You don't tell it. Right.

So it has to check on certain triggering events like you clicked somewhere or a time or fired or an observable was resolved and on these occasions it will check your code and *ngDoCheck* will be executed.

Now this might sound very inefficient angular does this in a very efficient way. So change direction angler works pretty great and doesn't cost a lot of performance, *ngDoCheck* is a great method to use if you want to do something on every change detection cycle like maybe manually inform angular about some change and would not be able to detect otherwise though that is a very advanced use case.

Well then we reach:

**ngAfterContentInit** this is called whenever the content which is projected via ng- content has been initialized.

So not the view of the component itself but instead you could say the view of the parent t component especially apart which will get added to our component through ng-content and *ngAfterContentInit* check is executed whenever change detection checked this content we're projecting into our component

**ngAfterViewInit** is then reached once the view of our own component has been finished initializing.

So once our view has been rendered you could say and the same goes to:

**ngAfterViewChecked** well that is called whenever our view has been checked so once we are assured that well either all changes which had to be done were displayed in the view or no changes were detected by angular.

**ngOnDestroy** and finally if you destroy a component for example if you placed ngOnInit and this then gets set to false and therefore it removes it from the DOM entry *ngOnDestroy*is called. And here's a great place to do some clean-up work because this is called right before the object itself will be destroyed by angular.

These are hooks.

Nice to see them in theory.

Let's see them in practice and the next lecture