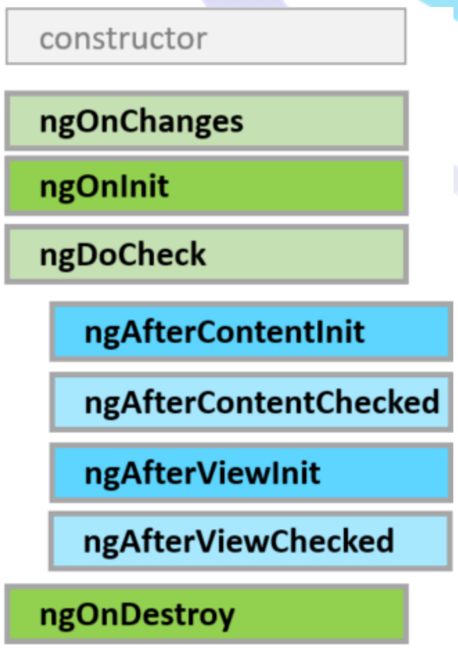
Angular Lifecycle Hooks

- Each Angular version goes through various phases in its lifecycle. Components have a critical job in Angular.
- For the smoothness in the development process, Angular manages all its components. Like any natural lifecycle, each component has its lifecycle events like – birth, life events, and death.
- Components are the primary building block for any Angular version.
- So it becomes utmost important to understand them to understand the processing steps of the lifecycle of components, then only it can be implemented in the development of an application.
- In Angular is each, and every component in it has a lifecycle, and every stage of a lifecycle goes from initialization to destruction.
- A component lifecycle goes typically through eight different stages.
- When an Angular component gets initialized, it creates and presents its root components.
- Which then designed and produced its heirs. For all of the components that gets loaded during the development of the application, it keeps checking when the data binding properties are getting changed and updated.
- When the component is not utilized anymore, it approaches the death phase which is then decimated and is expelled from the DOM.
- Sometimes you may have to write some additional codes as these events take place.

Lifecycle hook overview

The events in the life of a component are also referred to as "lifecycle hooks."
Lifecycle hooks are callback method that Angular raises when a positive event happens in the lifecycle of a component. There are 8 distinct kinds of hooks in the lifecycle of a component or directive.



ngOnchanges
This event gets executed as and when the input control gets renewed inside the component.
A changed data map is received by the Angular containing the present and previous position of the data-bound property encased in a simple change.
Using this lifecycle hook a Parent component can easily communicate with its child component if the property decorator exposes @InputDecorator of the child component.
Developers use this hook to discover the details about the input property that has been changed and how it got changed.

Properties:

- It can be utilized practically in all the components that have input.
- Gets invoked whenever the input value gets changed.

It gets the initial call to get raised before ngOnInit.

ngOnInit

When Angular has completed the creation and introduction of components this callback is invoked, it also gets initialized as Angular displays data-bound properties.
This event gets its call only after ngOnChanges event and after the constructor.
With this hook, you can initialize logic to your component.
As this hook gets initialization after ngOnChanges that means all the properties ngOnInit can use all its properties.
Any of the child directive properties cannot be used before this hack gets triggered.

Properties:

- This hook initializes data for a component.
- After setting the input values, this hook gets its call.
- This hook is added by default by Angular CLI to all the components.
- It is called only for once.

ngDoCheck

This hook gets invoked whenever there is a vitality to review the input property of a component or directive. we can even use this call back for our logic check.
In short, through this hook, you can do custom check for your logic that you want to implement in the component.
This hook comes on demand instantly after ngOnInit, and this hook has its duty of execution even if there is no change in the property of a component.
This hook arises to rescue if Angular miscarries to detect any change in the input property.

Properties:

- Run by Angular to detect any changes.
- Called for change detection.

ngAfterContentInit

ngAfterContentInit becomes a demand next to ngDoCheck when every content of the components gets introduced and checked for the first time.
This method is implemented as soon as Angular makes any content projection within a component view.
This method is also called when the properties get clearly demarcated as ContentChild and ContentChildren and are fully initialized.

Properties:

- After ngDoCheck it is called initially.
- It does its work by initializing the content.

ngAfterContentChecked

This hook method accomplishes its work by investigating the modification in the content of the component using Angular change detection apparatus, and it still performs its task even if there is not at all any modification.
It gets its call after ngAftercontentInit and also gets executed after every execution od ngDoCheck. It plays a big role in the initialization of the child component.

Properties:

- This method waits for ngContentInit to finish its execution to get started.
- Executed after all ngDocheck.

ngAfterViewInit

This lifecycle method gets its call after ngAfterContentChecked and finds its use only on components.
This is very much similar to ngAfterContentInit, and it gets invoked only after all the component view and its child view.

Properties:

- After the initialization of view, it gets its call only for once.

ngAfterViewChecked

This Angular lifecycle method gets triggered subsequently as it checks component's view and child view.
This method gets its call after ngAfterViewInit and then for every ngAfterContentChecked method.
Like many other lifecycle hooks discussed above it is also applicable for components only.
When something is awaited from the child component, this component can be helpful.

Properties:

- After the checking and initialization are done, this gets its called.
- After every ngAfterContentChecked method finishes its job, this method starts its work.

ngOnDestroy

This lifecycle hook gets its call after Angular destroys all the components or directives. This is the place where we can use our clean up logic and unsubscribe from all observable and detach from event handlers, by doing so we can prevent memory leakage.

Properties:

- Gets its call just before components get removed from DOM.