LIFECYCLE HOOKS

Angular calls lifecycle hook methods on directives and components as it creates, changes, and destroys them.

Component Lifecycle

A Component has a lifecycle managed by Angular itself. Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.

Angular offers **Lifecycle hooks** that give us visibility into these key moments and the ability to act when they occur.

We cover these hooks in this chapter and demonstrate how they work in code.

Live Example

The Lifecycle Hooks

Directive and component instances have a lifecycle as Angular creates, updates, and destroys them.

Developers can tap into key moments in that lifecycle by implementing one or more of the "Lifecycle Hook" interfaces, all of them available in the `angular2/core` library.

Here is the complete lifecycle hook interface inventory:

- `OnInit`
- `OnDestroy`
- `DoCheck`
- `OnChanges`
- `AfterContentInit`
- `AfterContentChecked`
- `AfterViewInit`
- `AfterViewChecked`

No directive or component will implement all of them and some of them only make sense for components.

Each interface has a single hook method whose name is the interface name prefixed with `ng`. For example, the `OnInit` interface has a hook method named `ngOnInit`.

Angular calls these hook methods in the following order:

- `ngOnChanges` - called when an input or output binding value changes
- `ngOnInit` - after the first `ngOnChanges`

- `ngDoCheck` - developer's custom change detection
- `ngAfterContentInit` - after component content initialized
- `ngAfterContentChecked` - after every check of component content
- `ngAfterViewInit` - after component's view(s) are initialized
- `ngAfterViewChecked` - after every check of a component's view(s)
- `ngOnDestroy` - just before the directive is destroyed.

The live example demonstrates these hooks.

Peek-a-boo

The `PeekABooComponent` demonstrates all of the hooks in the same component.

> Except for `DoCheck`. If our component superseded regular Angular change detection with its own change detection processing we would also add a `ngDoCheck` method. We would **not** implement `ngOnChanges`. We write either `ngOnChanges` or `ngDoCheck`, not both.
>
> Custom change detection and `ngDoCheck` are on our documentation backlog.

Peek-a-boo is a demo. We'd rarely if ever implement all interfaces like this in real life.

We look forward to explaining the Peek-a-boo example and the other lifecycle hook examples in an update to this chapter. Meanwhile, please enjoy poking around in the code.

Interface optional?

The lifecycle interfaces are optional. We recommend adding them to benefit from TypeScript's strong typing and editor tooling.

But they disappear from the transpiled JavaScript. Angular can't see them at runtime. And they are useless to someone developing in a language without interfaces (such as pure JavaScript).

Fortunately, they aren't necessary. We don't have to add the lifecycle hook interfaces to our directives and components to benefit from the hooks themselves.

Angular instead inspects our directive and component classes and calls the hook methods *if they are defined*. Angular will find and call methods like `ngOnInit()`, with or without the interfaces.

> **Next Step**
>
> Npm Packages