Component Lifecycle Hooks

Angular manages the lifecycle of each component. This lifecycle includes events such as:

The creation of the component
The rendering of the component's appearance (HTML)
The creation and rendering of the component's children
Incremental checks when data-bound property values are changed
Destruction of the component and it's subsequent removal from the DOM>
In a common web application, you will often need to trigger your custom logic when one or more of these events occur. To support this scenario, Angular offers a suite of lifecycle hooks. These hooks are special methods where you can add your custom code to run at these specific lifecycle moments.

Component lifecycle Methods:

This table is sourced from http://angular.io

Hook	Purpose	Components	Directives
ngOnInit	Initialize the directive/component after Angular initializes the data-bound input properties.	✓	✓
ngOnChanges	Respond after Angular sets a data-bound input property. The method receives a changes object of current and previous values.	✓	✓
ngDoCheck	Detect and act upon changes that Angular can or won't detect on its own. Called every change detection run.	✓	✓
ngOnDestroy	Cleanup just before Angular destroys the directive/component. Unsubscribe observables and detach event handlers to avoid memory leaks.	✓	✓
ngAfterContentInit	After Angular projects external content into its view.	✓	
ngAfterContentChecked	After Angular checks the bindings of the external content that it projected into its view.	✓	
ngAfterViewInit	After Angular creates the component's view(s).	✓	
ngAfterViewChecked	After Angular checks the bindings of the component's view(s).	✓	

Lifecycle sequence

Once Angular creates a new component, the lifecycle hooks are invoked in a specific sequence:

This table is also sourced from http://angular.io

Hook	Timing
ngOnChanges	before ngOnInit and when a data-bound input property value changes.
ngOnInit	after the first ngOnChanges.
ngDoCheck	during every Angular change detection cycle.
ngAfterContentInit	after projecting content into the component.
ngAfterContentChecked	after every check of projected component content.
ngAfterViewInit	after initializing the component's views and child views.
ngAfterViewChecked	after every check of the component's views and child views.
ngOnDestroy	just before Angular destroys the directive/component.

Hands-On: Handling Lifecycle Events using Hooks

Objective: In this guided exercise, you will create a method to invoke logic based on an Angular component lifecycle event.

Pre-requisite Prior to completing this lab, you should have already installed all pre-requisite tooling including: Visual Studio Code, Node Package Manager (NPM), TypeScript Compiler, Yeoman, and the start-angular Yeoman Generator. If you require instructions on how to install these tools, you should go back and complete the Setup: Installing Angular Development Tools exercise.

Setup

1. On your local machine, open Visual Studio Code.
2. Go to the File menu and select the Open Folder option.
3. Create a new folder for this exercise and select this folder in the dialog.

Bootstrapping Your Environment

1. In Visual Studio Code, right click the Explorer and select the Open in Terminal option.
2. Run the Angular generator using yeoman:
   ```
   yo start-angular --nochild --blank
   ```
3. Close your terminal.

Implement the Lifecycle Interfaces

1. Within the app folder, open the app.component.ts file.
2. Update the existing import statement by also importing the OnInit, AfterContentInit and AfterViewInit members (interfaces) from the @angular/core library module:

```
import {Component, OnInit, AfterContentInit, AfterViewInit} from '@angular/core';
```

3. Update the existing AppComponent class by implementing the OnInit, AfterContentInit and AfterViewInit interfaces:

```
export class AppComponent implements OnInit, AfterContentInit, AfterViewInit {
}
```

4. Within the AppComponent class, add a property named messages of type string[] and initialize it to an empty array:

```
export class AppComponent implements OnInit, AfterContentInit, AfterViewInit {
  messages : string[] = [];
}
```

5. To implement the OnInit interface, create a new method named ngOnInit:

```
export class AppComponent implements OnInit, AfterContentInit, AfterViewInit {
  messages : string[] = [];
  public ngOnInit() {
  }
}
```

6. Within the ngOnInit method, add a string to the messages array with the value OnInit:

```
public ngOnInit() {
  this.messages.push('OnInit');
}
```

7. To implement the AfterContentInit interface, create a new method named ngAfterContentInit:

```
export class AppComponent implements OnInit, AfterContentInit, AfterViewInit {
  messages : string[] = [];
  public ngOnInit() { .. }
  public ngAfterContentInit() {
  }
}
```

8. Within the ngAfterContentInit method, add a string to the messages array with the value AfterContentInit:

```
public ngAfterContentInit() {
  this.messages.push('AfterContentInit');
}
```

9. To implement the AfterViewInit interface, create a new method named ngAfterViewInit:

```
export class AppComponent implements OnInit, AfterContentInit, AfterViewInit {
  messages : string[] = [];
  public ngOnInit() { .. }
```

```
    public ngAfterContentInit() { ... }
    public ngAfterViewInit() {
    }
}
```

10. Within the ngAfterViewInit method, add a string to the messages array with the value AfterViewInit:

```
public ngAfterViewInit() {
    this.messages.push('AfterViewInit');
}
```

Add a View For Your Component's Messages

1. Within the app/views folder, open the app.component.html file.
2. Add a new ol element afer the h1 element:

```
<h1>Demo App</h1>
<ol>
</ol>
```

Within the ol element, add a li element:

```
<ol>
      <li>
      </li>
</ol>
```

3. Update the li element to use the ngFor looping syntax to create a for loop over the messages property using a message variable for each item:

```
<li *ngFor="let message of messages">
</li>
```

4. Update the content of the li element by binding it to the message variable within the loop:

```
<li *ngFor="let message of messages">
    {{message}}
</li>
```

Debugging Your Solution

1. Locate and select the Debug: Start Debugging command using the Command Palette again. The web page will now open in your default browser.
2. View your web application.
3. Observe the order of the lifecycle events.
4. Close your open browser and Visual Studio Code.

Result: At the end of this guided exercise, you used a method to invoke logic when an Angular component is initialized and after it's content/views are initialized.