Angular Lifecycle Hooks

In Angular, the component lifecycle refers to a series of events that occur during the life of a component. These events give developers the opportunity to perform certain actions at different stages of the component's life, such as initializing data, making API calls, or cleaning up resources when the component is destroyed.

Below is an explanation of each Angular lifecycle hook:

1. **ngOnChanges**
   - **When it's called**: It is called when an input property of the component changes.
   - **Usage**: It allows you to capture changes in input properties and take appropriate actions.
   - **Example**:
     ```typescript
     ngOnChanges(changes: SimpleChanges): void {
         console.log(changes);
     }
     ```

2. **ngOnInit**
   - **When it's called**: It is called after the component's inputs have been initialized for the first time.
   - **Usage**: It is used for initialization tasks, such as fetching data or setting up component state.
   - **Example**:
     ```typescript
     ngOnInit(): void {
     ```

```
        console.log("ngOnInit called");
    }
    ```

3. **ngDoCheck**
    - **When it's called**: It is called after ngOnChanges and ngOnInit, and then after every change detection cycle.
    - **Usage**: It allows you to implement custom change detection and track changes.
    - **Example**:
      ```typescript
      ngDoCheck(): void {
          console.log("ngDoCheck called");
      }
      ```

4. **ngAfterContentInit**
    - **When it's called**: It is called after Angular has fully initialized the component's content (ng-content).
    - **Usage**: This hook is called once after the content of the component has been projected.
    - **Example**:
      ```typescript
      ngAfterContentInit(): void {
          console.log("ngAfterContentInit called");
      }
      ```

5. **ngAfterContentChecked**

- **When it's called**: It is called after Angular checks the component's content (ng-content) for changes.
- **Usage**: It is called after every change detection cycle on the content.
- **Example**:

```typescript
ngAfterContentChecked(): void {
    console.log("ngAfterContentChecked called");
}
```

6. **ngAfterViewInit**
- **When it's called**: It is called after the component's view (template) has been initialized.
- **Usage**: It is used for operations that require access to the component's view and its DOM elements.
- **Example**:

```typescript
ngAfterViewInit(): void {
    console.log("ngAfterViewInit called");
}
```

7. **ngAfterViewChecked**
- **When it's called**: It is called after Angular checks the component's view (template) for changes.
- **Usage**: This hook is called after each change detection cycle on the view.
- **Example**:

```typescript
```

```
ngAfterViewChecked(): void {
    console.log("ngAfterViewChecked called");
}
```

8. **ngOnDestroy**
   - **When it's called**: It is called just before the component is destroyed.
   - **Usage**: It is used to clean up resources like subscriptions, timers, or any other tasks that need to be disposed of when the component is destroyed.
   - **Example**:
     ```typescript
     ngOnDestroy(): void {
         console.log("ngOnDestroy called");
     }
     ```

Lifecycle Sequence:

1. **ngOnChanges** -> **ngOnInit** -> **ngDoCheck** -> **ngAfterContentInit** -> **ngAfterContentChecked** -> **ngAfterViewInit** -> **ngAfterViewChecked** -> **ngOnDestroy**

Example Component Implementing All Lifecycle Hooks:

```typescript
import { Component, OnInit, OnChanges, SimpleChanges, DoCheck, AfterContentInit, AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy } from '@angular/core';

@Component({
```

```
  selector: 'app-lifecycle',
  templateUrl: './lifecycle.component.html',
  styleUrls: ['./lifecycle.component.css']
})
export class LifecycleComponent implements OnInit, OnChanges, DoCheck, AfterContentInit,
AfterContentChecked, AfterViewInit, AfterViewChecked, OnDestroy {
  ngOnChanges(changes: SimpleChanges): void {
    console.log("ngOnChanges called", changes);
  }

  ngOnInit(): void {
    console.log("ngOnInit called");
  }

  ngDoCheck(): void {
    console.log("ngDoCheck called");
  }

  ngAfterContentInit(): void {
    console.log("ngAfterContentInit called");
  }

  ngAfterContentChecked(): void {
    console.log("ngAfterContentChecked called");
  }

  ngAfterViewInit(): void {
```

```
    console.log("ngAfterViewInit called");
  }

  ngAfterViewChecked(): void {
    console.log("ngAfterViewChecked called");
  }

  ngOnDestroy(): void {
    console.log("ngOnDestroy called");
  }
}
```

Conclusion:

Understanding the Angular lifecycle hooks is essential for optimizing your components' behavior, performing initialization tasks, handling changes, and cleaning up resources effectively. Each hook plays a crucial role in managing the component lifecycle efficiently.