ngOnChanges

The `ngOnChanges` lifecycle hook is called when one or more bound input properties of a component change. It provides information about the previous and current values of the changed properties. This hook is particularly useful when you need to respond to changes in input properties and perform specific actions accordingly.

Here's a detailed explanation and example of how to use `ngOnChanges`:

Example:

Let's say you have a component with an input property `data`, and you want to log the changes whenever the value of `data` changes.

```
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-example',
  template: '<p>{{ data }}</p>',
})
export class ExampleComponent implements OnChanges {
  @Input() data: string;

  ngOnChanges(changes: SimpleChanges) {
    // Called when input properties change

    // Check if 'data' property changed
    if (changes.data) {
      const previousValue = changes.data.previousValue;
      const currentValue = changes.data.currentValue;

      console.log(`'data' property changed from ${previousValue} to
${currentValue}`);
    }
  }
}
```

In this example:

- The component has an `@Input` property called `data`.
- The `ngOnChanges` method is implemented to handle changes to input properties.
- The `SimpleChanges` parameter contains information about changed properties, and you can check if the `data` property changed using `changes.data`.
- If the `data` property changed, you can access its `previousValue` and `currentValue`.

Usage:

Now, let's use this component in another component:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <app-example [data]="message"></app-example>
    <button (click)="updateData()">Update Data</button>
  `,
})
export class AppComponent {
  message = 'Hello, Angular!';

  updateData() {
    this.message = 'Updated Message';
  }
}
```

In this example:

- The `AppComponent` uses the `ExampleComponent` and binds the `message` property to its `data` input.
- There is a button that triggers the `updateData` method when clicked, changing the value of `message`.

When you run this application, you'll see that the `ngOnChanges` method in `ExampleComponent` is called whenever the value of `data` changes, and it logs information about the changes.

This is a simple example, but `ngOnChanges` is powerful when you need to respond to changes in input properties and perform actions based on those changes in your Angular components.

Example

When you change the data from a parent component and pass it to a child component using an input binding, Angular triggers the `ngOnChanges` lifecycle hook in the child component. This hook is specifically designed to handle changes in input properties.

Here's an explanation of the process:

1. **Parent Component:**
    - The parent component has a property, let's call it `parentData`.
    - This property is bound to an input property of the child component.

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-parent',
  template: `
    <app-child [childData]="parentData"></app-child>
    <button (click)="updateData()">Update Parent Data</button>
  `,
})
export class ParentComponent {
  parentData = 'Initial Data';

  updateData() {
    this.parentData = 'Updated Data';
  }
}
```

2. **Child Component:**

   - The child component has an input property, let's call it `childData`.
   - The `ngOnChanges` hook is implemented to handle changes in the input property.

```
import { Component, Input, OnChanges, SimpleChanges } from
'@angular/core';

@Component({
  selector: 'app-child',
  template: `
    <p>Child Data: {{ childData }}</p>
  `,
})
export class ChildComponent implements OnChanges {
  @Input() childData: string;

  ngOnChanges(changes: SimpleChanges) {
    if (changes.childData) {
      const previousValue = changes.childData.previousValue;
      const currentValue = changes.childData.currentValue;

      console.log(`'childData' property changed from ${previousValue} to
${currentValue}`);
    }
  }
}
```

3. **Scenario:**

   - Initially, the parent component sets `parentData` to 'Initial Data'.

- The child component receives this value through the input property `childData`.

```
<!-- Initial State -->
<p>Child Data: Initial Data</p>
```

- When you click the "Update Parent Data" button, the parent component updates `parentData` to 'Updated Data'.
- Angular detects the change and triggers change detection.

```
<!-- Updated State -->
<p>Child Data: Updated Data</p>
```

- As a result of the change, Angular calls the `ngOnChanges` hook in the child component.

```
ngOnChanges(changes: SimpleChanges) {
  if (changes.childData) {
    const previousValue = changes.childData.previousValue;
    const currentValue = changes.childData.currentValue;

    console.log(`'childData' property changed from ${previousValue} to
${currentValue}`);
  }
}
```

- The `ngOnChanges` hook logs the changes in the console, showing the previous and current values of `childData`.

In summary, `ngOnChanges` is triggered whenever an input property changes, allowing the child component to respond dynamically to those changes. This is a fundamental mechanism for communication between parent and child components in Angular.