

## Hooking into Component Life Cycle

- A component instance has a life cycle.
- Life cycle indicates various phases a component follows from start to end.
- The life cycle starts when **Angular instantiates the component**.
- The life cycle continues with **Change Detection, content projection & clean up**.
- The life cycle ends when Angular destroys the component instance and removes from DOM.
- Component **creates, updates and destroys instances**.
- All these phases of a component are maintained by a sequence of events.
- These events are controlled with a set of methods known as “Hook Method”.
- Often know as “**Life Cycle Hooks**”. [Life cycle methods]
- The various life cycle methods used by component

Hook Method	Purpose
ngOnChanges()	<ul style="list-style-type: none"><li>- Angular sets the value.</li><li>- Binds the values to any property DOM property.</li></ul>

	<ul style="list-style-type: none"> <li>- It gets notified with the changes in values by using “SimpleChanges” object.</li> <li>- It gets the previous value and current value.</li> <li>- It includes loading component and handling any action (Event) performed in application.</li> <li>- It loads both child and parent components,</li> <li>- <b>Managers the property and event binding in Angular.</b></li> </ul>
ngOnInit()	<ul style="list-style-type: none"> <li>- It will be called after the first “ngOnChanges()”.</li> <li>- Initialize the directive or component after Angular first displays the data-bound properties and sets.</li> <li>- Initialize memory for transporting data</li> </ul>

	<p>across components and bind to parent and child properties.</p> <ul style="list-style-type: none"> <li>- <b>Memory is initialized to store values and transport across components.</b></li> <li>- <b>Only initialize memory for values.</b></li> </ul>
ngDoCheck()	<ul style="list-style-type: none"> <li>- Called immediately after “ngOnChanges()” or every “<b>change detection</b>” and also immediately after “ngOnInit()”</li> <li>- It can detect and act upon the changes that Angular can’t or won’t detect implicitly.</li> <li>- Some changes can’t be detected implicitly.</li> <li>- Then “ngDoCheck()” event is responsible for detecting the changes manually by using “<b>custom</b></li> </ul>

	<p><b>events”</b> and update the changes by using “@Input()” and “@Output()”.</p> <ul style="list-style-type: none"> <li>- <b>Transporting data between components</b></li> </ul>
ngAfterContentInit()	<ul style="list-style-type: none"> <li>- It is called after “ngOnInit()”</li> <li>- Loads the content into component view (.html)</li> <li>- It binds the content into dynamic angular components like “ng-container, ng-template”</li> </ul>
ngAfterContentChecked()	<ul style="list-style-type: none"> <li>- Called after “ngAfterContentInit()”</li> <li>- It is also called after every “ngDoCheck()”</li> <li>- This is responsible for “<b>Content Projection</b>”.</li> <li>- <i>It is a way to import HTML content from outside the component and insert that content into component template</i></li> </ul>

	<p><i>at specific location.</i></p> <ul style="list-style-type: none"> <li>- After binding data to view.</li> <li>- <b><i>It brings the content from external component or child component and renders into the current component.</i></b></li> <li>- @ViewChild() [if..then]</li> </ul>
ngAfterViewInit()	<ul style="list-style-type: none"> <li>- Called after “ngAfterContentChecked()”</li> <li>- Fire up after the initialization of memory for component views and its child views.</li> <li>- It responds to view changes.</li> <li>- It identifies the changes in parent or child view and updates the content.</li> <li>- Manages input of data from parent to child and output of</li> </ul>

	<p>data from child to parent.</p> <ul style="list-style-type: none"> <li>- Tracking of changes in data and transporting the changes to update. [parent and child]</li> </ul>
ngAfterViewChecked()	<ul style="list-style-type: none"> <li>- It is called after “ngAfterViewInit()”</li> <li>- It renders the final content into parent and child view.</li> </ul>
ngOnDestroy()	<ul style="list-style-type: none"> <li>- Clean up the memory before destroying the component.</li> <li>- Unsubscribe to methods.</li> <li>- Detach the events.</li> <li>- Destroying memory allocated for component.</li> <li>- <b>It is required to handle memory leaks.</b></li> </ul>

## Change Detection

- It is managed under “ngOnChanges()”
- Sets a value to property.

- If no value defined into property then no change detected.

Syntax:

```
public username;           // No Change
```

Detected

```
public username = "John";  // Change
```

Detection

- Binds the value to HTML element property.

Syntax:

```
<input type="text" [(ngModel)]="username">
```

- If there is a value defined into property and that is bound to element property then Change Detected.
- Detect the changes in value.
- Update the change to Model.
- “ngModel” with property and event binding [Two-Way-Binding] is one live examples of “Change Detection”. [(ngModel)]
- Model is “Single-Source-Of-Truth”.
- “**SimpleChanges**” is the base that identifies the changes by accessing
  - CurrentValue
  - PreviousValue
- SimpleChanges object identifies the changes and update the changes.
- SimpleChange object comprises of following properties

- previousValue:any
- currentValue:any
- firstChange:boolean

Ex:

- **Add following components**

> ng g c displayvalue

> ng g c sendvalue

**Displayvalue.component.ts**

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

```
@Component({  
  selector: 'app-displayvalue',  
  templateUrl: './displayvalue.component.html',  
  styleUrls: ['./displayvalue.component.css']  
})
```

```
export class DisplayvalueComponent implements  
OnChanges {
```

```
  @Input() public userName;  
  public currentValue;
```



```
public previousValue;
```

```
public msg;
```

```
constructor() { }
```

```
ngOnChanges(changes: SimpleChanges){
```

```
  for(var property in changes) {
```

```
    let change = changes[property];
```

```
    this.currentValue = change.currentValue;
```

```
    this.previousValue = change.previousValue;
```

```
  }
```

```
  if(this.currentValue===this.previousValue){
```

```
    this.msg = 'No Change Detected';
```

```
  } else {
```

```
    this.msg = 'Change Detected';
```

```
  }
```

```
}
```

```
}
```

**Displayvalue.component.html**

**<div>**

```
<h2>Child Component</h2>
Hello ! {{userName}}
<h2>{{msg}}</h2>
<dl>
  <dt>Previous Value</dt>
  <dd>{{previousValue}}</dd>
  <dt>Current Value</dt>
  <dd>{{currentValue}}</dd>
</dl>
</div>
```

### **Sendvalue.component.ts**

```
export class SendvalueComponent{
  public userName;
}
```

### **Sendvalue.component.html**

```
<div class="container-fluid">
  <h2>Parent Component</h2>
  <div>
    <label>User Name</label>
    <div>
```

```
        <input type="text" [(ngModel)]= "userName">
    </div>
</div>
<app-displayvalue [userName]= "userName" ></app-
displayvalue>
</div>
```

## Content Projection

- It is a way to import HTML content from outside the component and insert that content into component template at specific location.
- “ngAfterContentChecked()” is the life cycle hook responsible for “Content Projection”.
- The external templates are accessed and displayed at specific location by using an object of type **“TemplateRef”**
- A “TemplateRef” object is responsible for rendering external template into component at specific location.
- TemplateRef can handle multiple content.
- Which content to display in view is dynamically managed by “@ViewChild()”
- It can display one content before change and another content after change.

## **Angular Components Library – Angular Materials**