Understanding Angular life cycle hooks

In an Angular application, the components have their whole *lifecycle* managed by Angular, from creation to destruction. And it provides us access to *lifecycle hooks*, which allows us to act in key moments during the component's *lifecycle*. A component has a lifecycle managed by Angular.

*Angular creates and renders components along with their children, checks when their data-bound properties change, and destroys them before removing them from the DOM.

*Angular offers lifecycle hooks that provide visibility into these key life moments and the ability to act when they occur.

*Component instances have a lifecycle as Angular creates, updates, and destroys them

In order to use those *hooks,* we have to tell Angular we want to implement the desired *hook interface.*

Here,the important life cycle hooks and the interfaces which declared them.

Method	Interface
ngOnChanges()	OnChanges interface
ngOnInit()	OnInit interface
ngOnDestroy()	OnDestroy interface

ngOnChanges()

This method is called **once** on component's creation and then **every time** changes are detected in one of the component's *input* properties. It receives a *SimpleChanges* object as a parameter, which contains information regarding which of the *input* properties has changed

Ex:

```
@Input()  data:any;
export class MyComponent implements OnChanges {

                        ngOnChanges(changes: SimpleChanges) {

                                console.log(changes.data.previoudValue);
                                console.log(changes.data.currentValue);

                        }}
```

It is very useful if you need to handle any specific logic in the component based on the received *input* property.

ngOnInit()

This method is called only **once** during the component lifecycle, after the first *ngOnChanges* call. within this method, you can have access not only to *data-bound properties* but also the *component's input properties*.

Ex:

```
export class MyComponent implements OnInit {
        ngOnInit() {
                // Insert Logic Here!
                }
        }
```

This can be used to write component initialization logic.

ngOnDestroy

Lastly, this method is called only **once** during the component's lifecycle, right before Angular destroy it. Here is where you should inform the rest of your application that the component is being destroyed

Ex:

```
export class MyComponent implements OnDestroy {
        ngOnDestroy() {
                // Insert Logic Here!
                }
        }
```