HOOKS

Q.1) What is Angular Component Lifecycle hooks ?

- The Angular lifecycle hooks are nothing but callback functions.
- This angular invokes when a specific event occurs during the component lifecycle.
- A Component in Angular has a lifecycle.
- A no of different phases it goes through from insert to destroy.

Q.2) The Order of Execution of Life Cycle Hooks ?

- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit
- ngAfterContentChecked
- ngAfterViewInit
- ngAfterViewChecked
- ngOnDestroy

Q.3) What is Change Detection Cycle ?

- Change detection is the mechanism by which angular keeps the template in sync with the component. *(dokyavarun gel right ?)*
- Consider this code in template file.
- ** Hello{{**name**}} **
- Angular update the DOM whenever the value of name changes.
- And this change process known as change detection cycle.

Q.4) How does angular know when the value of the name changes ?

- It happened because of running a change detection cycle on every event that may result in a change.
- During the change detection cycle, angular checks every bound property in the template with that of the component class..
- If it detects any changes, it updates the DOM.

Q.5) Now let's going to know what happened when we create any component.

- The lifecycle of component begins when Angular creates the component class.
- The first method that invokes is class Constructor.
- Constructor is not a life cycle hook nor is it specific to Angular.
- Constructor is a JavaScript feature.
- It is a method that is invokes when a class is created.

HOOKS

Q.6) Different usage of Constructor in Angular.

- Angular makes use of Constructor to inject dependencies.
- Constructor used to create and initializes an object instance of a class.
- Subclassing. → super(); //call the constructor of the parent class.
- Logging and Debugging.
- **constructor() {** console.log('MyComponent constructor called.'**); }**

Q.7) ngOnChanges

- It's Invoked every time there is a change in @input property of the component.
- When there is a change in a @input decorator that ngOnChages Hook will call automatically.
- It works only when there is a @Input decorator and
- This hook is not raised if change detection does not detect any changes.
- It is only hook that is work with parameter.
- We know @input decorator we use to send data from parent component to child.
- And we need to declare @input decorator in child component.

- Ex. In Child component we declares a property

```
1
2 @Input() message:string
3
```

- Now the parent comp. send the data to the child comp. using property binding.

```
1
2 <app-child [message]="message">
3 </app-child>
4
```

The change detector uses the === strict equality operator for detecting changes. Here for objects, the hook is fired only if the references are changes.

Q.8) ngOnInit

- Its Invoked when given component has been initialized.
- It raised after the ngOnChanges hook.
- This hook is fired **only once** and immediately after its creation.
- It's perfect place to add any initialization logic for your component.
- Here we can access every @input property of the component.
- We can use them in HTTP get requests to get the data from the back-end server.
- What is meaning of component has been initialized.
- It means that certain piece of code is executed once a specific part of software component is ready to start working.
- So, initialize means like preparing something and
- Invoked means like starting when it's ready.

HOOKS

Q.9) ngDoCheck

- The angular invokes the ngDoCheck hook event during every change detection cycle.
- This hook is invoked even if there is no change in any of the properties.
- Angular invoke it after the ngOnChanges and ngOnInit hooks.
- We can use ngDoCheck for custom change detection logic.
- Whenever angular fails to detect the changes made to input properties.
- This hook is convenient to detect the changes.

Q.10) ngAfterContentInit

- ngAfterContentInit called after the content has been projected into the view.
- It's called after the Components projected content has been fully initialized.
- This hook is also raised even if there is the no content to project.
- The angular components can include the ng-content element,
- Which acts as a placeholder for the content from parent.
- It's useful when you want to work with content projected into component.

Q.11) ngAfterContentChecked

- It's call after every change detection.
- Called every time the projected content has been checked.
- If there is no content projection but still it will execute.
- This hook is very similar to ngAfterContentInit hook.
- Both are called after the external content is initialized, checked and update.
- Only the difference is →
- NgAfterContentChecked is raised after every change detection cycle.
- ngAfterContentInit is raised during the first change detection cycle.

Q.12) ngAfterViewInit

- Called after the components view and child view has been initialized.
- Angular also updates the properties decorated with the ViewChild & ViewChildren properties before raising this hook.
- The hook is called during the first change detection cycle, where angular initializes the view for the first time.
- At this point, all the lifecycle hook methods and change detections of all child components and directives are processes and component is entirely ready.

Q.13) ngAfterViewChecked.

- Called every time the view and child view has been checked.
- This hooks is fired during every change detection cycle.
- Both are called after all the child components & directives are initialized and updated.
- Only the difference is →
- ngAfterViewChecked is raised after every change detection cycle.
- ngAfterViewInit is raised during the first change detection cycle.

HOOKS

Q.14) ngOnDestroy.

- This hook is called just before the Components/Directives instance is destroy by angular.
- We can perform any clean-up logic for the components here.
- This is where you would like to unsubscribe Observables and detach to avoid memory leaks.

Q.15) what is content Projection ?

- Content projection is a pattern in which you insert, or project, the content you want to use inside another component.
- Is way to pass the HTML content from the parent component to child component.
- @input decorator having some limitations that's why we use Content Projection.
- If we want to pass html complete element to child component then use <ng-content></ng-content>

Q.16) What is ng-content ?

- If we want to pass html complete element to child component then use <ng-content>.
- The <ng-content> tag acts as a placeholder.

Q.17) Can we have a multiple ng-content?

- Yes we have, but for that we need to use selector.

Q.18) How to use Lifecycle Hooks?

- Import Hook Interfaces.
- Declare that component/Directive Implement lifecycle hook interface.
- Create the Hook Method.

Q.19) Is it Mandatory to Implements OnInit, OnChanges, DoCheck, etc..?

- No. Without implementation we can use LifeCycle Hooks.

Q.20) What is Advantage to Implement Interface?

- For Debugging and Code Readability Make easy.

Q.22) What is the problem of Memory Leakages?

- A memory leak is a type of resource leak caused by poor management of memory allocation in a way that is not cleared from the memory when is not needed anymore.

Q.23) Difference between Constructor and ngOnInit ?

- Constructor is executed when the class is instantiated.
- It is a feature of JavaScript and Angular does not have control over it.
- ngOnInit is called when the Angular has initialized the component with all input Properties.

HOOKS

Q.24) Perform simple Lifecycle hooks?

typescript Copy code

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

@Component({
  selector: 'app-root',
  template: `
    <app-user [userName]="user"></app-user>
  `
})
export class AppComponent {
  user = 'John Doe';
}
```

typescript Copy code

```typescript
import { Component, Input, OnInit, OnChanges, OnDestroy } from '@angular/co

@Component({
  selector: 'app-user',
  template: `
    <h2>User Profile</h2>
    <p>{{ userName }}</p>
    <button (click)="editName()">Edit Name</button>
  `
})
export class UserComponent implements OnInit, OnChanges, OnDestroy {
  @Input() userName: string;

  constructor() {
    console.log('Constructor called');
  }

  ngOnInit() {
    console.log('ngOnInit called');
  }

  ngOnChanges() {
    console.log('ngOnChanges called');
  }

  ngOnDestroy() {
    console.log('ngOnDestroy called');
  }

  editName() {
    this.userName = prompt('Enter new name:');
  }
}
```