In Angular, **ViewChild** is a decorator used to access child components from a parent component. This allows you to interact with and manipulate the child component's properties and methods.

**Key Concepts:**

* **Access Child Components:** ViewChild provides a way to access child components within the parent component's class.
* **Template Reference Variables:** You use a template reference variable (e.g., #childComponent) to give a name to the child component in the parent component's template.
* **Lifecycle Hooks:** ViewChild typically works in conjunction with lifecycle hooks like AfterViewInit or ngAfterViewChecked to ensure that the child component has been fully initialized before you can access it.

**Example:**

**Parent Component (parent.component.html)**

HTML

<app-child #myChild></app-child>

**Parent Component (parent.component.ts)**

TypeScript

import { Component, ViewChild, AfterViewInit } from '@angular/core';

import { ChildComponent } from './child.component';

@Component({

selector: 'app-parent',

template: `

<app-child #myChild></app-child>

`

})

export class ParentComponent implements AfterViewInit {

@ViewChild('myChild') childComponent: ChildComponent;

ngAfterViewInit() {

if (this.childComponent) {

// Access child component properties or methods

this.childComponent.childMethod();

}

}

}

**Child Component (child.component.ts)**

TypeScript

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

@Component({

selector: 'app-child',

template: `

<p>Child Component</p>

`

})

export class ChildComponent {

childMethod() {

console.log('Child method called');

}

}

**Key Considerations:**

* Use ViewChild with caution, as it can increase component coupling.
* Consider using alternative approaches like services or state management solutions for more complex interactions between components.
* Always use ViewChild in conjunction with appropriate lifecycle hooks (AfterViewInit, ngAfterViewChecked) to ensure that the child component is fully initialized before accessing it.

**In Angular 19:**

* Consider using **standalone components** and **dependency injection** as alternative approaches to accessing child components. This can lead to more modular and testable code.

By effectively using ViewChild, you can establish communication and interactions between parent and child components in your Angular applications.