1. **@ViewChild() using Component**

@ViewChild() can be used for component communication. A component will get instance of another component inside it using @ViewChild(). In this way parent component will be able to access the properties and methods of child component. The child component selector will be used in parent component HTML template. Now let us discuss example.   
  
**1. Number example**  
  
In this example we will increase and decrease the counter using two buttons. The counter will be from a child component. Increase and decrease button will be inside parent component. We will communicate parent and child components using @ViewChild() decorator.   
First of all we will create a component where we will write methods to increase and decrease counter. 

**number.component.ts**

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

@Component({

selector: 'app-number',

template: '<b>{{message}}</b>'

})

export class NumberComponent {

message:string ='';

count:number = 0;

increaseByOne() {

this.count = this.count + 1;

this.message = "Counter: " + this.count;

}

decreaseByOne() {

this.count = this.count - 1;

this.message = "Counter: " + this.count;

}

}

Now we will create the instance of NumberComponent in our parent component using @ViewChild(). 

**number-parent.component.ts**

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

import { NumberComponent } from './number.component';

@Component({

selector: 'app-number-parent',

templateUrl: './number-parent.component.html'

})

export class NumberParentComponent {

@ViewChild(NumberComponent)

private numberComponent: NumberComponent;

increase() {

this.numberComponent.increaseByOne();

}

decrease() {

this.numberComponent.decreaseByOne();

}

}

We will observe that we are able to access the methods of NumberComponent in NumberParentComponent. We will use selector of NumberComponent in HTML template of NumberParentComponent.

**number-parent.component.html**

<h3>@ViewChild using Component</h3>

Number Example:

<button type="button" (click)="increase()">Increase</button>

<button type="button" (click)="decrease()">Decrease</button>

<br/>

<br/>

<app-number></app-number>

As usual both components NumberComponent and NumberParentComponent need to be configured in @NgModule in application module.   
Find the print screen.



**2. Stopwatch example**  
  
In this example we will create a simple stopwatch. We will create two components, one of which will contain the functionality of stopwatch and second component will instantiate first component using @ViewChild() decorator. Find the component that will contain the functionality of simple stopwatch. 

**stopwatch.component.ts**

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

@Component({

selector: 'app-stopwatch',

template: '<h2>{{counter}}</h2>'

})

export class StopwatchComponent {

shouldRun:boolean = false;

counter:number = 0;

start() {

this.shouldRun = true;

let interval = setInterval(() =>

{

if(this.shouldRun === false){

clearInterval(interval);

}

this.counter = this.counter + 1;

}, 1000);

}

stop() {

this.shouldRun = false;

}

}

Now we will instantiate the above component using @ViewChild() in the following component.

**stopwatch-parent.component.ts**

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

import { StopwatchComponent } from './stopwatch.component';

@Component({

selector: 'app-stopwatch-parent',

templateUrl: './stopwatch-parent.component.html'

})

export class StopwatchParentComponent {

@ViewChild(StopwatchComponent)

private stopwatchComponent: StopwatchComponent;

startStopwatch() {

this.stopwatchComponent.start();

}

stopStopwatch() {

this.stopwatchComponent.stop();

}

}

We observe that we are able to access the methods of StopwatchComponent in StopwatchParentComponent. Now use the selector of StopwatchComponent in the HTML template of StopwatchParentComponent.

**stopwatch-parent.component.html**

<h3>@ViewChild using Component</h3>

Stopwatch Example:

<button type="button" (click)="startStopwatch()">Start</button>

<button type="button" (click)="stopStopwatch()">Stop</button>

<br/>

<app-stopwatch></app-stopwatch>

Find the print screen.



1. **@ViewChild() using Directive**

@ViewChild() can instantiate a directive within a component and in this way the component will be able to access the directive methods. Here we will create a directive that will contain the methods to change the text color of the host element of DOM layout. Find the directive.

**cpcolor.directive.ts**

import { Directive, ElementRef, AfterViewInit } from '@angular/core';

@Directive({

selector: '[cpColor]'

})

export class CpColorDirective implements AfterViewInit{

constructor(private elRef: ElementRef) {

}

ngAfterViewInit() {

this.elRef.nativeElement.style.color = 'red';

}

change(changedColor: String) {

this.elRef.nativeElement.style.color = changedColor;

}

}

AfterViewInit: It is the lifecycle hook that is called after a component view has been fully initialized. To use AfterViewInit, our class will implement it and override its method ngAfterViewInit().   
  
Now create the component, that will instantiate CpColorDirective and access its methods.

**cpcolor-parent.component.ts**

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

import { CpColorDirective } from './cpcolor.directive';

@Component({

selector: 'app-cpcolor-parent',

templateUrl: './cpcolor-parent.component.html'

})

export class CpColorParentComponent {

@ViewChild(CpColorDirective)

private cpColorDirective: CpColorDirective;

changeColor(color: string) {

this.cpColorDirective.change(color);

}

}

In the HTML template of component a host element will use directive.

**cpcolor-parent.component.html**

<h3>@ViewChild using Directive</h3>

Color Example:

<p cpColor>Change my Color </p>

<div>

Change Color:

<input type="radio" name="rad" (click)= "changeColor('green')"> Green

<input type="radio" name="rad" (click)= "changeColor('cyan')"> Cyan

<input type="radio" name="rad" (click)= "changeColor('blue')"> Blue

</div>

As usual both component and directive need to be configured in @NgModule in application module.   
Find the print screen.



1. **@ViewChild() with Template Variable using ElementRef to access Native Element**

@ViewChild() can instantiate ElementRef corresponding to a given template reference variable. The template variable name will be passed in @ViewChild() as an argument. In this way component will be able to change the appearance and behavior of element of given template variable. Find the HTML template.

**cptheme.component.html**

<h3>@ViewChild with Template Variable using ElementRef to access Native Element </h3>

<div>

Name: <input type="text" #name> <br/>

City: <input type="text" #city>

</div>

In the above HTML template, we have two input boxes and their template reference variables are #name and #city. We will instantiate corresponding ElementRef using @ViewChild() as given below in the component.

**cptheme.component.ts**

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

@Component({

selector: 'app-cptheme',

templateUrl: './cptheme.component.html'

})

export class CpThemeComponent implements AfterViewInit {

@ViewChild('name')

private elName : ElementRef;

@ViewChild('city')

private elCity : ElementRef;

ngAfterViewInit() {

this.elName.nativeElement.style.backgroundColor = 'cyan';

this.elName.nativeElement.style.color = 'red';

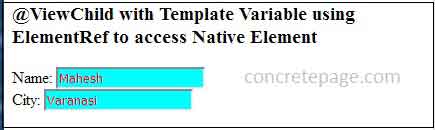
this.elCity.nativeElement.style.backgroundColor = 'cyan';

this.elCity.nativeElement.style.color = 'red';

}

}

Find the print screen.



**@ContentChild and @ContentChildren**

<https://www.concretepage.com/angular-2/angular-2-4-contentchild-and-contentchildren-example#download>

As the name suggests, @ContentChild and @ContentChildren queries will return directives existing inside the <ng-content></ng-content> element of your view, whereas @ViewChild and @ViewChildren only look at elements that are on your view template directly.

**TestvcComponent.component.ts**

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

@Component({

selector: 'alert',

//templateUrl: './testvc.component.html',

template: `<h1 (click)="alert()">{{type}}</h1>`,

styleUrls: ['./testvc.component.css']

})

export class TestvcComponent implements OnInit {

constructor() { }

ngOnInit() {

}

message: string;

@Input() type: string = "success";

alert() {

console.log("alert");

}

**TestvcParentComponent.Component.ts**

import { Component, OnInit, ViewChildren, QueryList } from '@angular/core';

import { TestvcComponent } from '../../app/testvc/testvc.component';

@Component({

selector: 'app-testvc-parent',

//templateUrl: './testvc-parent.component.html',

styleUrls: ['./testvc-parent.component.css'],

template: `

<alert></alert>

<alert [type]="type1"></alert>

<alert [type]="type1"></alert>

`,

})

export class TestvcParentComponent implements OnInit {

constructor() { }

public type1: string = "hello";

ngOnInit() {

}

@ViewChildren(TestvcComponent) alerts: QueryList<TestvcComponent>

ngAfterViewInit() {

this.alerts.forEach(

alertInstance => {

if (alertInstance.type == "success") {

alert("Success");

}

console.log(alertInstance)

}

);

}

}

@ContentChild gives the first element or directive matching the selector from the content DOM. If new child element replaces the old one matching the selector in content DOM, then property will also be updated.

@ContentChildren is used to get QueryList of elements or directives from the content DOM. When there is change in content DOM, data in QueryList will also change. If child elements are added, we will get those new elements in QueryList. If child elements are removed, then those elements will be removed from the QueryList.

Below statement will give only direct children

@ContentChildren(BookDirective) topBooks: QueryList<BookDirective>

**descendants**: This is Boolean value. When it is **true** then direct children and other descendants will also be included. If the value is **false** then only direct children will be included. descendants is used as follows.

@ContentChildren(BookDirective, {descendants: true}) allBooks: QueryList<BookDirective>

The default value of descendants is **false**.

Using AfterContentInit

AfterContentInit is a lifecycle hook that is called after directive content is fully initialized. It has a method ngAfterContentInit(). This method runs after angular loads external content into the component view. This method runs once after first ngDoCheck() method. Contents queried by @ContentChild and @ContentChildren are set before ngAfterContentInit() is called. AfterContentInit is used as given below.

@ContentChildren(BookDirective) books: QueryList<BookDirective>

ngAfterContentInit() {

this.books.forEach(booksInstance => console.log(booksInstance))

}

Example 1: @ContentChild and @ContentChildren using Directive

Find the example of @ContentChild and @ContentChildren decorators using directive. First we will create a directive with selector as element name.   
**book.directive.ts**

import { Directive, Input } from '@angular/core';

@Directive({

selector: 'book'

})

export class BookDirective {

@Input() bookId: string;

@Input() bookName: string;

}

In the above directive we have used two @Input() properties. <book> element can be used in any component. Now create a component to use @ContentChild decorator to query element of type <book>.   
**writer.component.ts**

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

import { BookDirective } from './book.directive';

@Component({

selector: 'writer',

template: `

Name: {{writerName}}

<br/>Latest Book: {{book?.bookId}} - {{book?.bookName}}

`

})

export class WriterComponent {

@ContentChild(BookDirective) book: BookDirective;

writerName = 'Mahesh';

}

Find the code snippet of person.component.html to use <writer> and <book> elements.

<writer>

<book bookId="1" bookName="Java 8 Tutorials" \*ngIf="latestBook"></book>

<book bookId="2" bookName="Learning Angular 4" \*ngIf="!latestBook"></book>

</writer>

<br/><button (click)="onChangeBook()">Change Book</button>

Find the code snippet of person.component.ts.

latestBook = true;

onChangeBook() {

this.latestBook = (this.latestBook === true)? false : true;

}

Now we will create the example of @ContentChildren using directive.   
**favourite-books.component.ts**

import { Component, ContentChildren, QueryList } from '@angular/core';

import { BookDirective } from './book.directive';

@Component({

selector: 'favourite-books',

template: `

<b>Top Favourite Books</b>

<ng-template ngFor let-book [ngForOf]= "topBooks">

<br/>{{book.bookId}} - {{book.bookName}}

</ng-template>

<br/><b>All Favorite Books</b>

<ng-template ngFor let-book [ngForOf]= "allBooks">

<br/>{{book.bookId}} - {{book.bookName}}

</ng-template>

`

})

export class FavouriteBooksComponent {

@ContentChildren(BookDirective) topBooks: QueryList<BookDirective>

@ContentChildren(BookDirective, {descendants: true}) allBooks: QueryList<BookDirective>

}

In the above component we are using @ContentChildren two times, one with default descendants and second with descendants with **true** value. Find the code snippet of person.component.html to use <favourite-books> and <book>elements.

<favourite-books>

<book bookId="1" bookName="Hibernate 4 Tutorials"></book>

<book bookId="2" bookName="Spring Boot Tutorials"></book>

<favourite-books>

<book bookId="3" bookName="Learning JavaScript"></book>

</favourite-books>

<favourite-books \*ngIf="showAllBook">

<book bookId="4" bookName="Thymeleaf Tutorials"></book>

<book bookId="5" bookName="Android Tutorials"></book>

</favourite-books>

</favourite-books>

<br/><button (click)="onShowAllBooks()" >

<label \*ngIf="!showAllBook">Show More</label>

<label \*ngIf="showAllBook">Show Less</label>

</button>

Find the code snippet of person.component.ts.

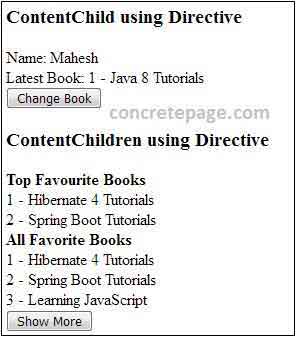
showAllBook = false;

onShowAllBooks() {

this.showAllBook = (this.showAllBook === true)? false : true;

}

Now find the print screen of the output of @ContentChild and @ContentChildren decorators using directive.



Example 2: @ContentChild and @ContentChildren using Component

Find the example of @ContentChild and @ContentChildren decorators using component. Here for child element we will create a component instead of directive.   
**city.component.ts**

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

@Component({

selector: 'city',

template: ``

})

export class CityComponent {

@Input() cityId: string;

@Input() cityName: string;

}

In the above component, we have used two @Input() properties. Now find the component that will use @ContentChild.   
**address.component.ts**

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

import { CityComponent } from './city.component';

@Component({

selector: 'address',

template: `

<b>{{title}}</b>

<br/>City: {{city?.cityId}} - {{city?.cityName}}

`

})

export class AddressComponent {

@ContentChild(CityComponent) city: CityComponent;

title = 'Address';

}

Find the code snippet of person.component.html to use <address> and <city> element.

<address>

<city cityId="1" cityName="Varanasi" \*ngIf="homeTown"></city>

<city cityId="2" cityName="Noida" \*ngIf="!homeTown"></city>

</address>

<br/><button (click)="onChangeCity()">Change City</button>

Find the code snippet of person.component.ts.

homeTown = true;

onChangeCity() {

this.homeTown = (this.homeTown === true)? false : true;

}

Now find the example for @ContentChildren using component.   
**favourite-cities.component.ts**

import { Component, ContentChildren, QueryList } from '@angular/core';

import { CityComponent } from './city.component';

@Component({

selector: 'favourite-cities',

template: `

<b>Top Favourite Cities</b>

<ng-template ngFor let-city [ngForOf]= "topCities">

<br/>{{city.cityId}} - {{city.cityName}}

</ng-template>

<br/><b>All Favourite Cities</b>

<ng-template ngFor let-city [ngForOf]= "allCities">

<br/>{{city.cityId}} - {{city.cityName}}

</ng-template>

`

})

export class FavouriteCitiesComponent {

@ContentChildren(CityComponent) topCities: QueryList<CityComponent>

@ContentChildren(CityComponent, {descendants: true}) allCities: QueryList<CityComponent>

}

Find the code snippet of person.component.html to use <favourite-cities> and <city> element.

<favourite-cities>

<city cityId="1" cityName="Noida"></city>

<city cityId="2" cityName="Mumbai"></city>

<favourite-cities>

<city cityId="3" cityName="Gurugram"></city>

</favourite-cities>

<favourite-cities \*ngIf="showAllCity">

<city cityId="4" cityName="New Delhi"></city>

<city cityId="5" cityName="Bengaluru"></city>

</favourite-cities>

</favourite-cities>

<br/><button (click)="onShowAllCities()" >

<label \*ngIf="!showAllCity">Show More</label>

<label \*ngIf="showAllCity">Show Less</label>

</button>

Find the code snippet of person.component.ts.

showAllCity = false;

onShowAllCities() {

this.showAllCity = (this.showAllCity === true)? false : true;

}

Now find the print screen of the output of @ContentChild and @ContentChildren decorators using component.



Example 3: @ContentChild and @ContentChildren using ElementRef

Find the example of @ContentChild and @ContentChildren decorators using ElementRef. First find the component that will use @ContentChild with ElementRef.   
**friend.component.ts**

import { Component, ContentChild, ElementRef, AfterContentInit } from '@angular/core';

@Component({

selector: 'friend',

template: `

Friend Name: {{friendName}}

`

})

export class FriendComponent implements AfterContentInit {

@ContentChild('name') nameRef: ElementRef;

get friendName(): String {

return this.nameRef.nativeElement.innerHTML;

}

ngAfterContentInit() {

console.log(this.friendName);

}

}

In the above component **name** inside @ContentChild('name') is the local template variable of a HTML element. Find the code snippet of person.component.html to use <friend> with a <div> element.

<friend>

<div #name \*ngIf="bestFriend">Mahesh</div>

<div #name \*ngIf="!bestFriend">Krishna</div>

</friend>

<br/><button (click)="onChangeFriend()">Change Friend</button>

Find the code snippet of person.component.ts.

bestFriend = true;

onChangeFriend() {

this.bestFriend = (this.bestFriend === true)? false : true;

}

Now find the example for @ContentChildren with ElementRef.   
**favourite-friends.component.ts**

import { Component, ContentChildren, QueryList, ElementRef, AfterContentInit } from '@angular/core';

@Component({

selector: 'favourite-friends',

template: `

<b>All Favourite Friends</b>

<br/> {{allFriends}}

`

})

export class FavouriteFriendsComponent implements AfterContentInit {

@ContentChildren('name') allFriendsRef: QueryList<ElementRef> ;

get allFriends(): string {

return this.allFriendsRef ? this.allFriendsRef.map(f =>f.nativeElement.innerHTML).join(', ') : '';

}

ngAfterContentInit() {

console.log(this.allFriends);

}

}

In the above component **name** inside @ContentChildren('name') is the local template variable of a HTML element. Find the code snippet of person.component.html to use <favourite-friends> with <div> element.

<favourite-friends>

<div #name>Mohit</div>

<div #name>Anup</div>

<div #name \*ngIf="showAllFriend">Nilesh</div>

<div #name \*ngIf="showAllFriend">Sravan</div>

</favourite-friends>

<br/><button (click)="onShowAllFriends()" >

<label \*ngIf="!showAllFriend">Show More</label>

<label \*ngIf="showAllFriend">Show Less</label>

</button>

Find the code snippet of person.component.ts.

showAllFriend = false;

onShowAllFriends() {

this.showAllFriend = (this.showAllFriend === true)? false : true;

}

Now find the print screen of the output of @ContentChild and @ContentChildren decorators using ElementRef.

