Angular

AJ

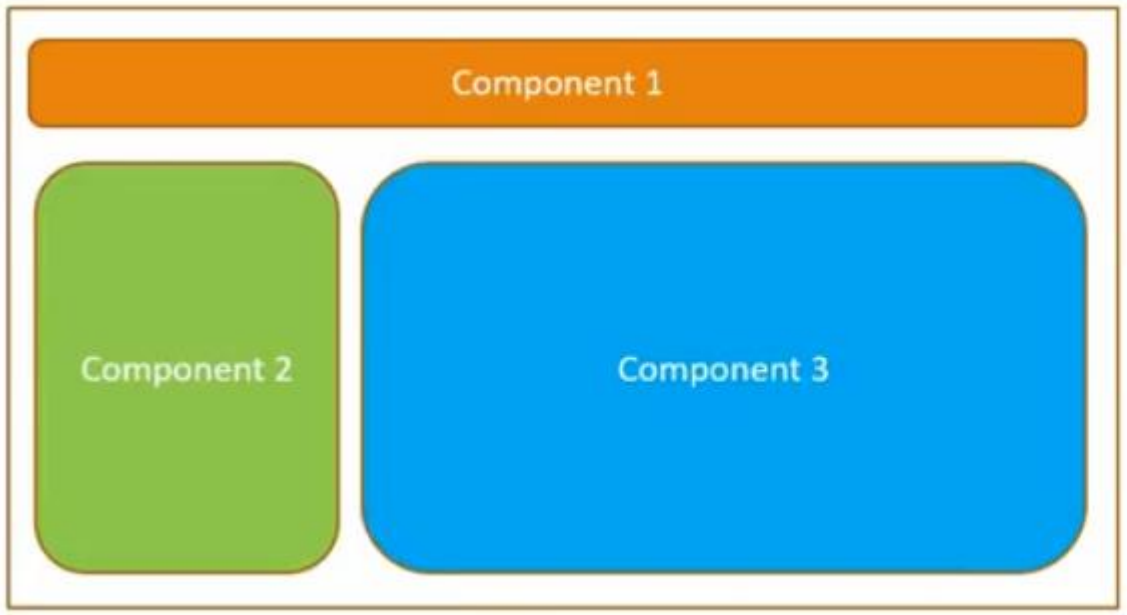
Angular Components

Components are the key features of Angular. The whole application is built by using different components.
The core idea behind Angular is to build components. They make your complex application into reusable parts which you can reuse very easily.

Creating component with CLI
Syntax
1. ng generate component component_name
Or
2. ng g c component_name

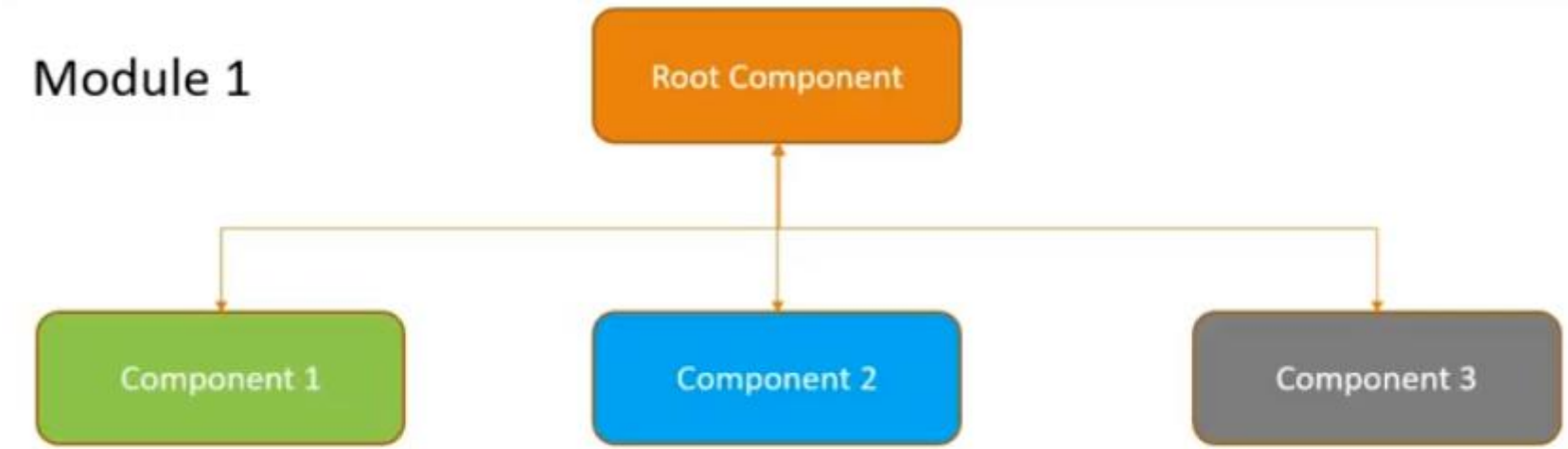
Open Command prompt and stop **ng serve** command if it is running on the browser.

Type **ng generate component server2** to create a new component named server2.
You can also use a shortcut **ng g c server2** to do the same task.



Root Component

AppComponent



Angular app – one or more modules

Module – One or more components and services

Components – HTML + Class

Services – Business logic

Modules interact and ultimately render the view in the browser

Component

View

HTML

Code

TypeScript

Data &
Methods

Information

Decorator

Different types of Component Selectors in Angular

There are mainly three ways to use the selector in HTML:

1. Selector can directly be used by typing element-name directly as a legacy selector:

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

This type of selector can access directly by typing the selector name inside the <> brackets as:

```
 <app-element></app-element>
```

2. The selector can be used as attribute selector by put the selector into square brackets:

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

In this, we have changed our selector to be an attribute. To access this type of attribute selector we have to put this as an attribute inside a div or any other element as:

```
<div app-element></div>
```

Different types of Component Selectors in Angular

3. The selector can also be select by class just like in CSS by putting a dot in the beginning:

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

In this, we can select by class as:

```
<div class="app-element"></div>
```

Note: Selecting by ID and by all pseudo selectors such as hover and so on won't work as they are not supported by Angular.

Interpolation

Angular interpolation is used display a component property in the respective view template with double curly braces syntax. We can display all kind of properties data into view e.g. string, number, date, arrays, list or map.

Data binding consist of one way data binding and two way data binding. Interpolation is used for one way data binding. Interpolation moves data in one direction from our components to HTML elements.

Angular Interpolation Syntax
The property name to be displayed in the view template should enclosed in *double curly braces* also known as **moustache syntax**. i.e.

Interpolation Syntax

```
class AppComponent
{
    propertyName: string;
    object: DomainObject;
}

{{ propertyName }}

{{ object.propertyName }}
```

Angular automatically pulls the value of the propertyName and object.propertyName from the component and inserts those values into the browser. Angular updates the display when these properties change.

Angular Interpolation Usages

1. Display simple properties – Interpolation can be used to display and evaluate strings into the text between HTML element tags and within attribute assignments.

```
Display simple properties example

<h1>Greetings {{ name }}! </h1>

<h4><img src="{{ backgroundImgUrl }}" style="height:40px"><
```

<h1>Greetings {{ name }}! </h1>

<h4><img src="{{ backgroundImgUrl }}" style="height:40px"></h4>

Angular Interpolation Usages

2. Evaluate arithmetic expressions – Another usage of interpolation is to evaluate arithmetic expressions present within the curly braces.

```
Arithmetic expressions example

<h6>{{3 + 5}}</h6>   //outputs 8 on HTML browser
```

Angular Interpolation Usages

3. Invoke methods and display return values – We can also invoke/call methods on hosting component views within interpolation expressions.

greet.component.ts

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

@Component({
selector: 'app-greet',
template: `
    <h1>Greetings {{ name }}! </h1>
    <h2>Have a good {{ getTime() }}!</h2>
`,
styleUrls: ['./greet.component.css']
})
export class GreetComponent implements OnInit {

name: string = "John Doe";

getTime(): string {
    return 'morning';
}

}
```


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


@Component({
selector: 'app-greet',
template: `
    <h1>Greetings {{ name }}! </h1>
    <h2>Have a good {{ getTime() }}!</h2>
`,
styleUrls: ['./greet.component.css']
})
export class GreetComponent implements OnInit {


name: string = "John Doe";


getTime(): string {
    return 'morning';
}


}
```

Angular Interpolation Usages

4. Display array items – We can use interpolation along with **ngFor** directive to display an array of items.

DomainObject.ts

```
export class DomainObject
{
  constructor(public id: number, public name: string) {
    //code
  }
}
```

Either we use inline template or separate HTML file for component view, the template data bindings have the same access to the component's properties.

app.component.ts

```
import { DomainObject } from './domain';

@Component({
  selector: 'app-root',
  template: `
  <h1>{{title}}</h1>
  <h2>The name is : {{domainObjectItem.name}}</h2>
  <p>Data Items:</p>
  <ul>
    <li *ngFor="let d of domainObjects">
      {{ d.name }}
      </li>
  </ul>
`
})
export class AppComponent
{
  title = 'App Title';

  domainObjects = [
    new DomainObject(1, 'A'),
    new DomainObject(2, 'B'),
    new DomainObject(3, 'C'),
    new DomainObject(4, 'D')
  ];

  domainObjectItem = this.domainObjects[0];
}
```

Angular Interpolation Example

Let's create a new component using ***@angular/cli*** by following command.

```
Create angular template

//with inline template using '-it' flag

ng generate component greet -it
```

Above command will generate 'greet.component.ts' with inline template. Let's add properties *name* and *time* to the greet component like shown below:

Angular Interpolation Example

Angular automatically pulls the value of the *name* and *time* properties from the ‘greet’ component and inserts those values into the browser. Angular updates the display when these properties change.

greet.component.ts

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

@Component({
selector: 'app-greet',
template: `
<h1>Greetings {{name}}! </h1>
<h2>Have a good {{time}}!</h2>
`,
styleUrls: ['./greet.component.css']
})
export class GreetComponent implements OnInit {

name: string = "John Doe";
time: string = "morning";

}
```

Difference between Interpolation and Property Binding

Interpolation is a special syntax that Angular converts into property binding (pair of square bracket). It's a convenient alternative to property binding.

Another major difference is that to set an element property to a **non-string data value**, we must use property binding.

In this example, `OK` button will be disabled or enabled based on the value of `'isDisabled'`. on the other side, `Cancel` button will always be disabled irrespective of the property value.

```
Property binding example - works

export class AppComponent {
    isDisabled: boolean = true;
}

<button [disabled]='isDisabled'>OK</button>          //Data binding

<button disabled='{{isDisabled}}'>Cancel</button>    //Interpolation
```