Angular 8 : Online Class

Class – 5

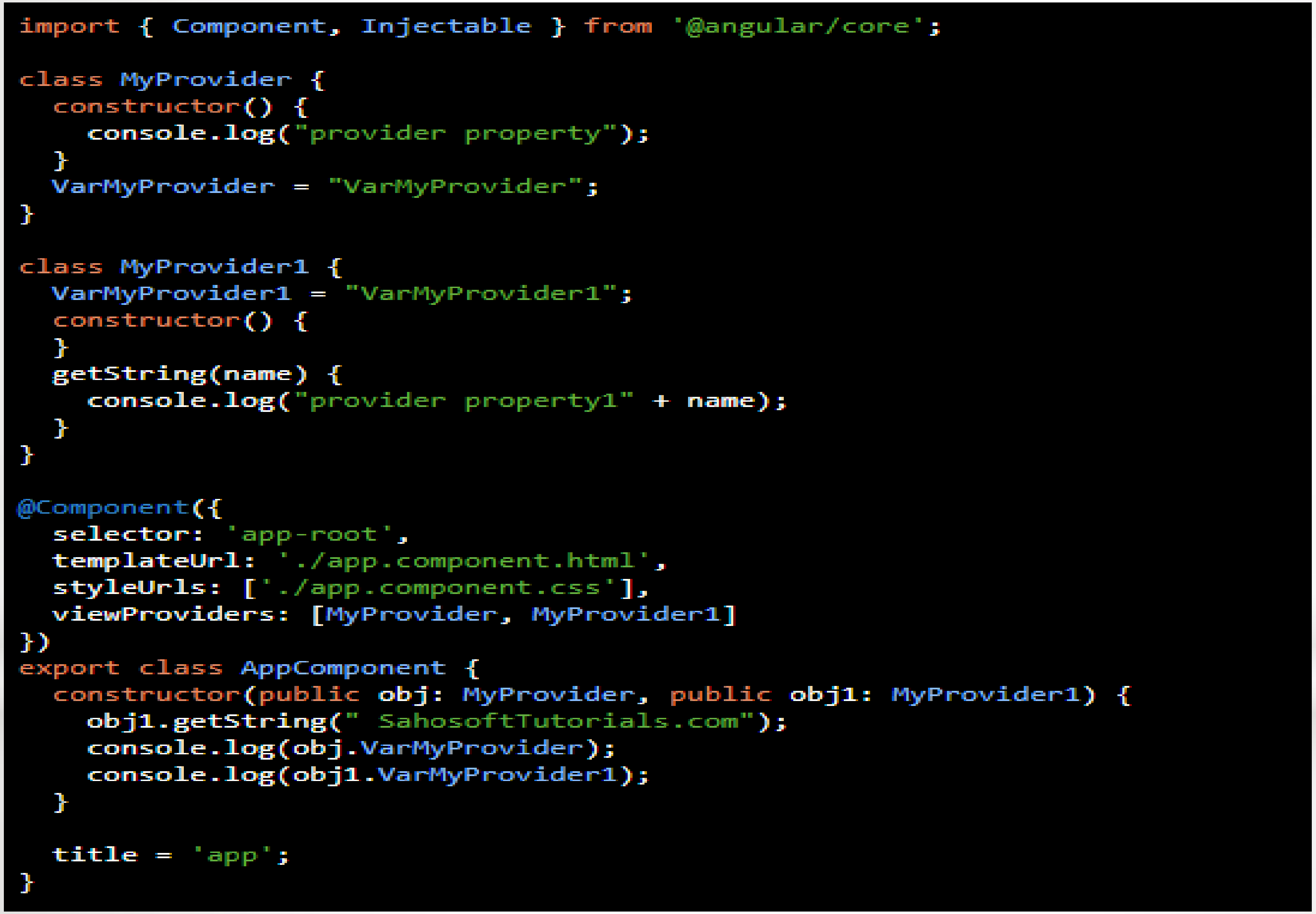
By: Sahosoft Solutions

Presented by : Chandan Kumar

viewProvider

We know that the decorator functions of @Component take object and this object contains many properties.

The viewProviders property allows us to make providers available only for the component’s view.
When we want to use a class in our component that is defined outside the @Component () decorator function, then, first of all, we need to inject this class into our component, and we can achieve this with the help of the "viewProvider" property of a component.



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

class MyProvider {
  constructor() {
    console.log("provider property");
  }
  VarMyProvider = "VarMyProvider";
}

class MyProvider1 {
  VarMyProvider1 = "VarMyProvider1";
  constructor() {
  }
  getString(name) {
    console.log("provider property1" + name);
  }
}

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  viewProviders: [MyProvider, MyProvider1]
})
export class AppComponent {
  constructor(public obj: MyProvider, public obj1: MyProvider1) {
    obj1.getString(" SahosoftTutorials.com");
    console.log(obj.VarMyProvider);
    console.log(obj1.VarMyProvider1);
  }

  title = 'app';
}
```

moduleId

it is used to resolve relative paths for your stylesheets and templates.Module ID of the module that contains the component. We had to be able to resolve relative URLs for templates styles. In Dart, this can be determined automatically and it is not necessary to configure it. In CommonJS or SystemJS, this can always be set in module.id.

Google has added a new SystemJS plugin (systemjs-angular-loader.js) to our recommended SystemJS configuration. This plugin dynamically converts "component-related" paths into templateUrl and styleUrls into "absolute paths".
It is strongly recommended to write only paths related to the components. This is the only form of URL discussed in these documents. You no longer need to write @Component ({moduleId: module.id})

moduleId

moduleId?: string

moduleId parameter inside the **@Component** annotation takes a **string** value which is;

"The module id of the module that contains the component.“

CommonJS usage: module.id,
SystemJS usage: __moduleName

Interpolation

Before we get to know interpolation, we need to know the data Binding in Angular. Let's start with the data Binding.

The data binding is a powerful feature of Angular, that allows us to communicate between the component and its view. The data binding can be a one-way data binding [angular interpolation / string interpolation, linking properties, event linking] or a two-way data binding.
In the one-way data binding, the model value is inserted into an HTML element (DOM) and the model cannot be updated from the view. In the two-way binding, the automatic synchronization of data occurs between the Model and the View (each time the Model changes, it will be reflected in the View and vice versa)

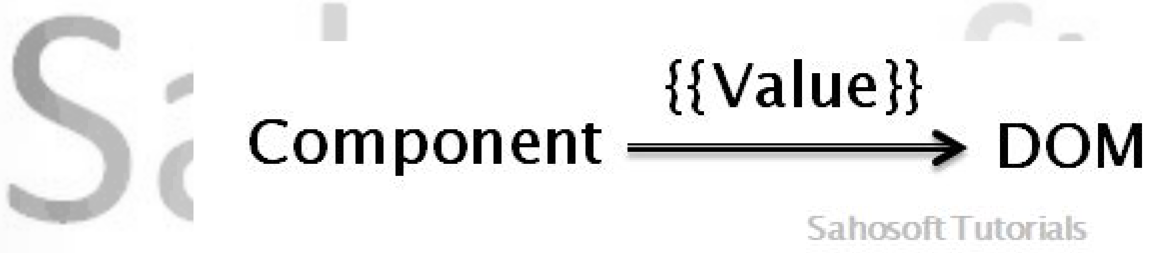
Interpolation

Angular has four types of Data binding

1. **Interpolation / String Interpolation** (one-way data binding)
2. **Property Binding** (one-way data binding)
3. **Event Binding** (one-way data binding)
4. **Two-Way Binding**

Interpolation

Interpolation is a technique that allows the user to bind a value to an element of the user interface. Interpolation binds data in one-way. This means that when you change the value of the bound field by interpolation, it is also updated on the page. The field value cannot be changed. An object of the component class is used as a data context for the component template. Therefore, the value to be bound in the view must be assigned to a field in the component class.


{{Value}}
Component
DOM
Sahosoft Tutorials