Unit V – Introduction to Client-side JS Framework – Part II

Agenda

- Components and Its Life Cycle
- Data Binding
- Forms in Angular
  - Template Driven Forms
  - Model Driven Forms or Reactive Forms
  - Custom Validators
- Dependency Injection
- Services
- RxJS Observables
- HTTP Service
- Routing

Components and Its Life Cycle

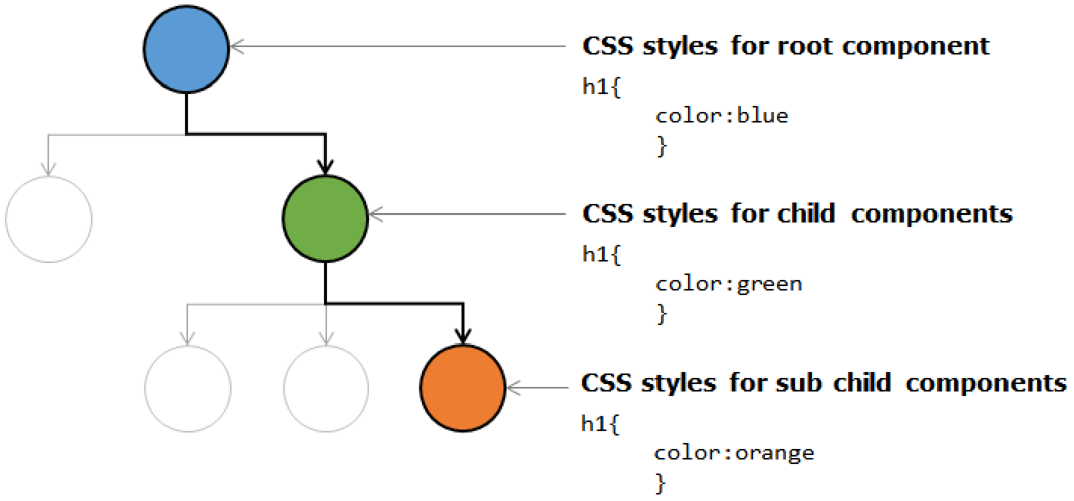
Infosys® | Education, Training and Assessment

Nested Components

- Nested component is a component which is loaded into another component
- The component where we load another component is called as container component/parent component

Shadow DOM

- Shadow DOM is a web components standard by W3C.
- It **enables encapsulation for DOM tree and styles**. It means that shadow DOM hides DOM logic behind other elements and also confines styles only for that component.


CSS styles for root component
CSS styles for child components
CSS styles for sub child components

```
h1{
        color:blue
        }
```

```
h1{
        color:green
        }
```

```
h1{
        color:orange
        }
```

Refer to official site for Demos: http://angular.io

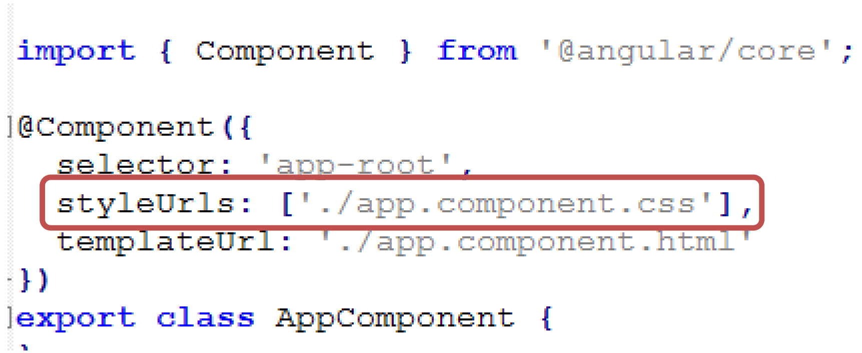
Component Styling

- Component styling can be done in 3 different ways:
  - **styleUrls metadata** - external style sheet is added to component.ts file
  - **styles metadata** – Internal style added to component.ts
  - **Inline style into template** - <style> <style> tag added in component.html file

styleUrls metadata

- Example:

app.component.ts



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

@Component({
  selector: 'app-root',
  styleUrls: ['./app.component.css'],
  templateUrl: './app.component.html'
})
export class AppComponent {
```

app.component.css

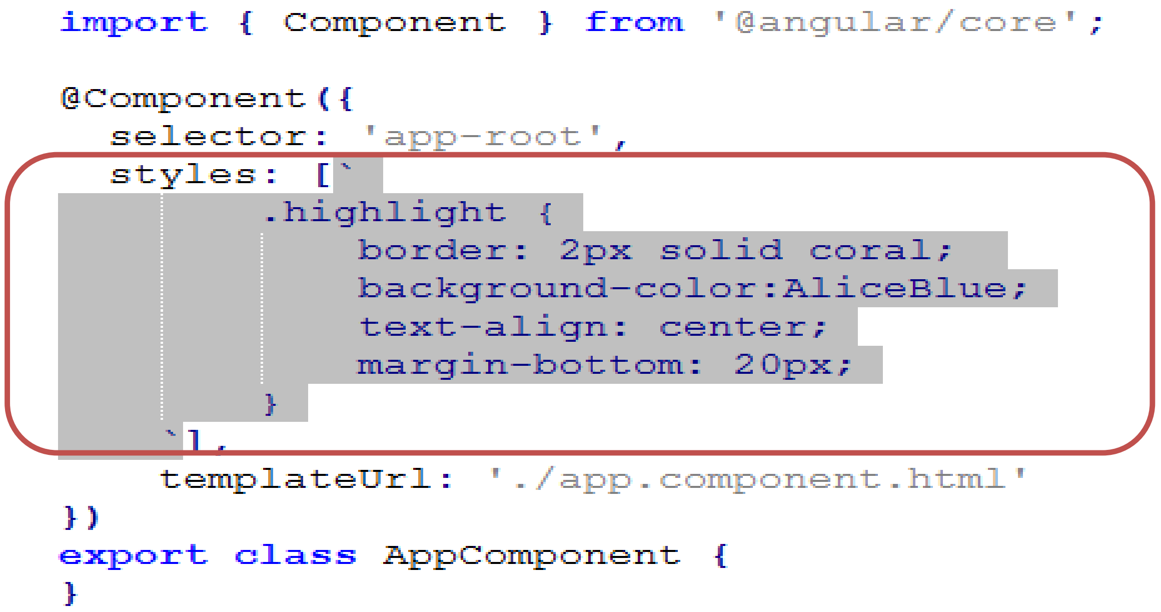
```
.highlight {
        border: 2px solid coral;
        background-color: aliceblue;
        text-align: center;
        margin-bottom: 20px;
}
```

Refer to official site for Demos: http://angular.io

styles metadata

- Example:

app.component.ts



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

@Component({
  selector: 'app-root',
  styles: [`
        .highlight {
            border: 2px solid coral;
            background-color:AliceBlue;
            text-align: center;
            margin-bottom: 20px;
        }
    `],
    templateUrl: './app.component.html'
})
export class AppComponent {
}
```

Refer to official site for Demos: http://angular.io

Inline style

- Example

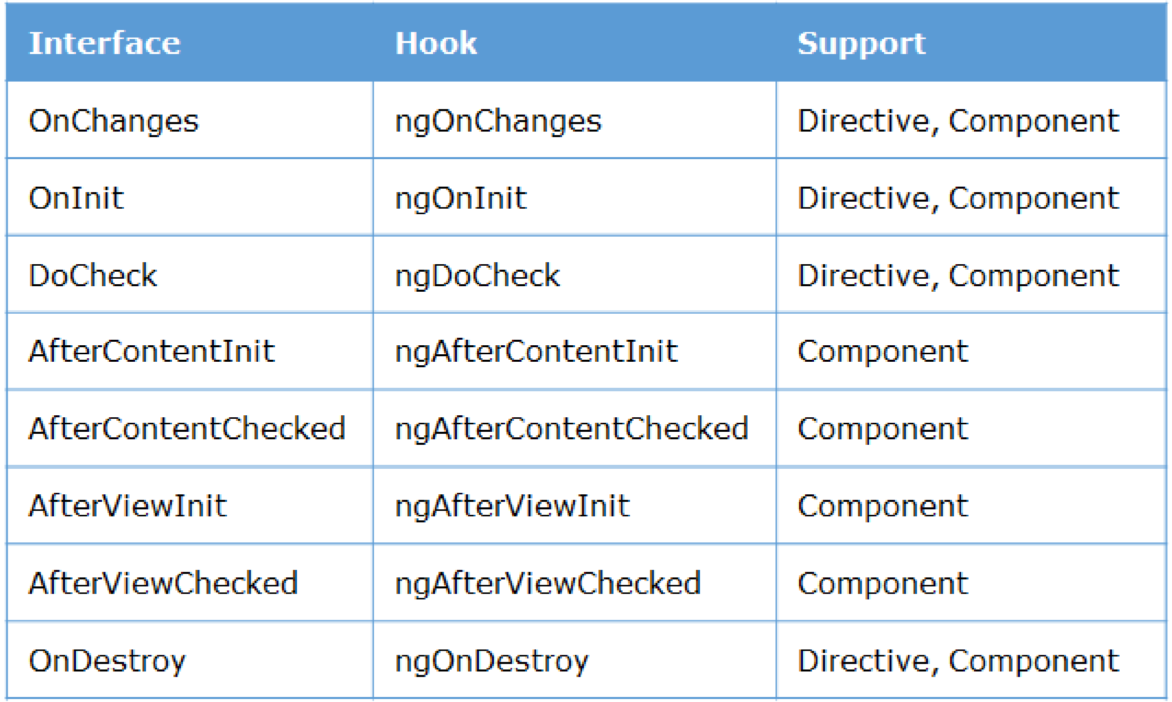
app.component.html

```
<style>
 .highlight {
     border: 2px solid coral;
     background-color: aliceblue;
     text-align: center;
     margin-bottom: 20px;
 }
</style>
<div class="highlight">
    Container Component
</div>
<app-child></app-child>
```

Refer to official site for Demos: http://angular.io

Component Life Cycle(1/2)

- A component has a life cycle managed by Angular which contains creating a component, rendering it, creating and rendering its child components, checks when its data bound properties change and destroys it before removing it from the DOM



Interface	Hook	Support
OnChanges	ngOnChanges	Directive, Component
OnInit	ngOnInit	Directive, Component
DoCheck	ngDoCheck	Directive, Component
AfterContentInit	ngAfterContentInit	Component
AfterContentChecked	ngAfterContentChecked	Component
AfterViewInit	ngAfterViewInit	Component
AfterViewChecked	ngAfterViewChecked	Component
OnDestroy	ngOnDestroy	Directive, Component

Component Life Cycle(2/2) - Lifecycle Hooks

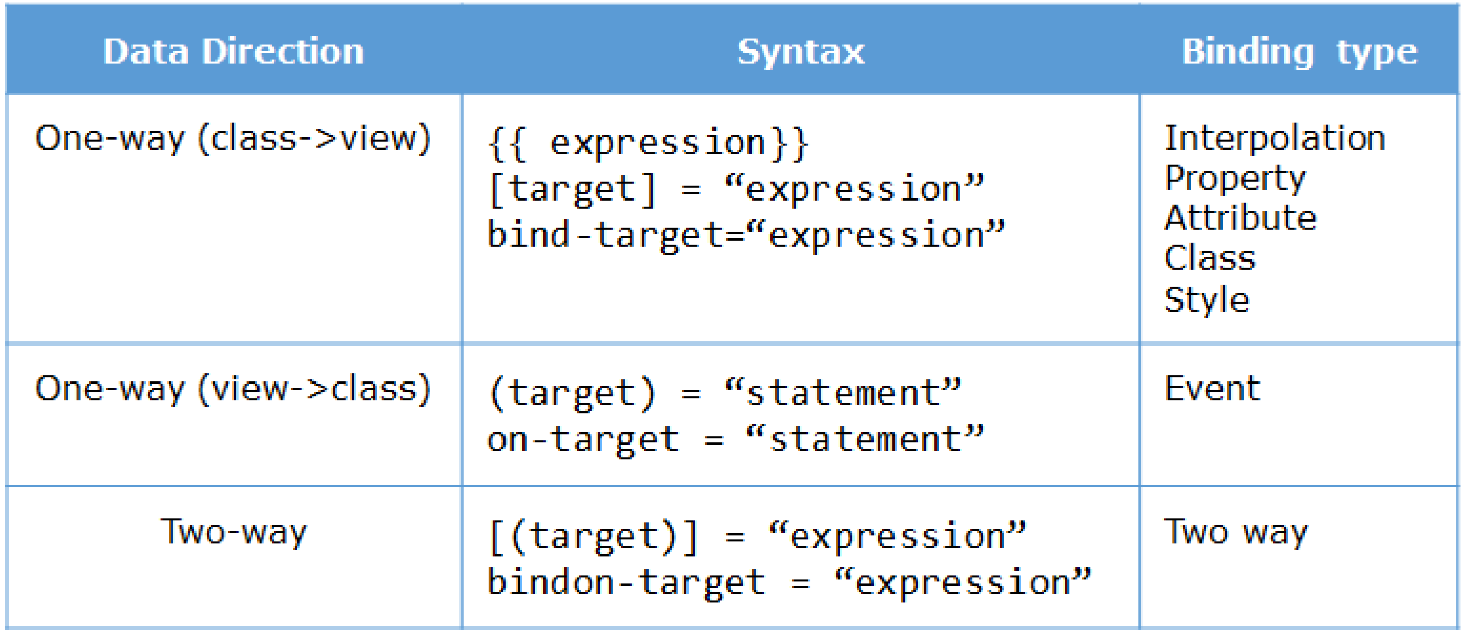
- **ngOnChanges** – It will be invoked when Angular sets data bound input property i.e., property attached with @Input(). This will be invoked whenever input property changes its value
- **ngOnInit** – It will be invoked when Angular initializes the directive or component
- **ngDoCheck** - It will be invoked for every change detection in the application
- **ngAfterContentInit** – It will be invoked after Angular projects content into its view
- **ngAfterContentChecked** – It will be invoked after Angular checks the bindings of the content it projected into its view
- **ngAfterViewInit** – It will be invoked after Angular creates component's views
- **ngAfterViewChecked** – It will be invoked after Angular checks the bindings of the component's views
- **ngOnDestroy** – It will be invoked before Angular destroys directive or component

Data Binding

Infosys® | Education, Training and Assessment

Data Binding

- DataBinding is a **mechanism used to coordinate between what users see on the screen and the data in the class**



Data Direction	Syntax	Binding type

One-way (class->view)	`{{ expression}}`
	`[target] = “expression”`
	`bind-target=“expression”`
		Interpolation
		Property
		Attribute
		Class
		Style

One-way (view->class)	`(target) = “statement”`
	`on-target = “statement”`
		Event

Two-way	`[(target)] = “expression”`
	`bindon-target = “expression”`
		Two way

One-way Data Binding

- **Property Binding**

```
<img [src] = ‘imageUrl’ />
or
<img bind-src = ‘imageUrl’ />
```

- **Attribute Binding** - Attribute binding can be used to set the value of an attribute directly.

```
<td colspan = “{{ 2+3 }}”>Hello</td>
```

- **Style Binding** - [style.styleproperty]

```
<button [style.color] = "isValid ? 'blue' : 'red' ">Hello</button>
```

- **Event Binding**

```
<button (click) ="onSubmit(username.value,password.value)">Login</button>
        Or
<button on-click = "onSubmit(username.value,password.value)">Login</button>
```

PUBLIC

Refer to official site for Demos: http://angular.io

Two Way Data Binding

- It is a mechanism where if model property value changes, it updates the element to which the property is binded to and vice versa. It uses [()] (banana in a box)

[(ngModel)]

- eg., <input [(ngModel)] = "course.courseName">

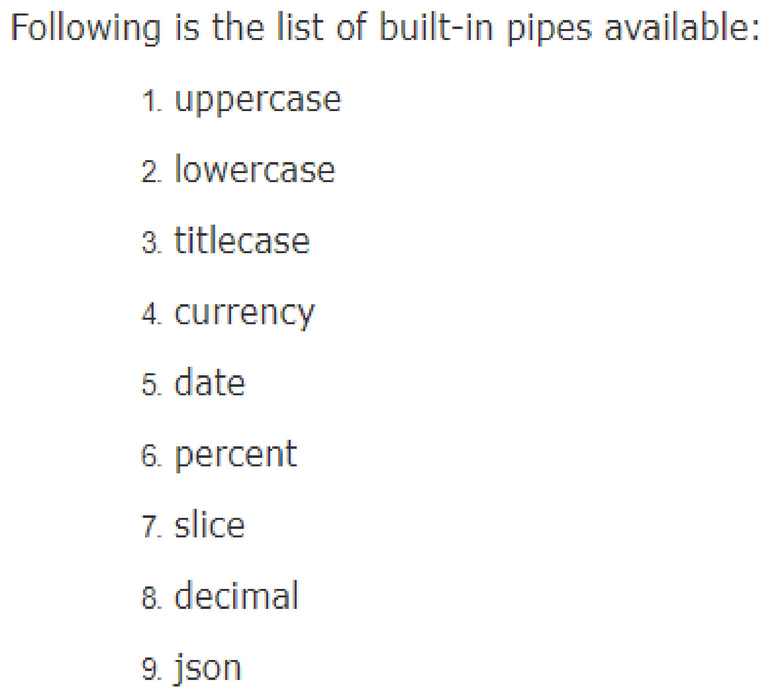
```
export class AppComponent{
    course=[
        {“courseName”:”HTML5”, “Duration”: “3 days”},
        {“courseName”:”TypeScript”, “Duration”: “3 days”}
]
}
```

Refer to official site for Demos: http://angular.io

Pipes

Built-in Pipes (1/2)

- Pipes are used to format the data before displaying it to the user. A pipe takes data as input and transforms it into the desired output.
- {{ expression | pipe }}
- Example : {{ “Infosys” | uppercase }}



Following is the list of built-in pipes available:

1. uppercase
2. lowercase
3. titlecase
4. currency
5. date
6. percent
7. slice
8. decimal
9. json

Built-in Pipes (2/2)

app.component.ts

```
export class AppComponent {
  title = 'Angular 4 Project!';
  todaydate = new Date();
  jsonval = {name:'Rox', age:'25', address:{a1:'Mumbai', a2:'Karnataka'}};
  months = ["Jan", "Feb", "Mar", "April", "May", "Jun",
         "July", "Aug", "Sept", "Oct", "Nov", "Dec"];
}
```

- **Case**:
  - {{title | uppercase}}
  - {{title | lowercase}}
- **Currency**
  - {{6589.23 | currency:"USD"}}
  - {{6589.23 | currency:"USD":true}}
- **Date**
  - {{todaydate | date:'d/M/y'}}
  - {{todaydate | date:'shortTime'}}
- **Decimal**
  - {{ 454.78787814 | number: '3.4-4' }}
- **Percent**:
  - {{00.54565 | percent}}
- **Slice**
  - {{months | slice:2:6}}
- **json**
  - {{ jsonval | json }}

Refer to official site for Demos: http://angular.io

Forms in Angular

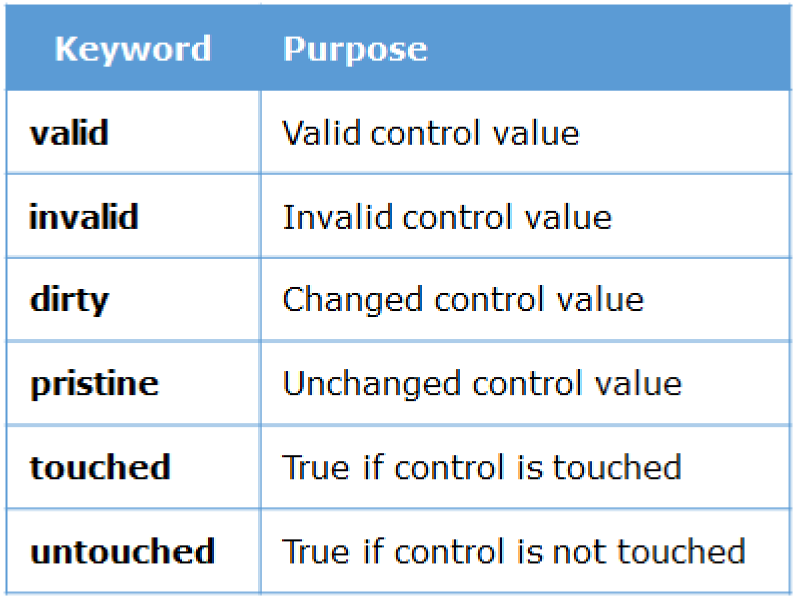
Infosys® | Education, Training and Assessment

Forms in Angular(1/3)

- Forms are the crucial part of web applications through which we get majority of our data input from users.
- We can create two different types of forms in Angular
  - **Template driven Forms**
    - Used to create small to medium sized forms
    - Entire form validation is done in HTML templates
  - **Model driven Forms or Reactive Forms**
    - Used to create large size forms
    - Entire form validation is done in Component class using **FormBuilder** and **Validators** classes

Forms in Angular(2/3)

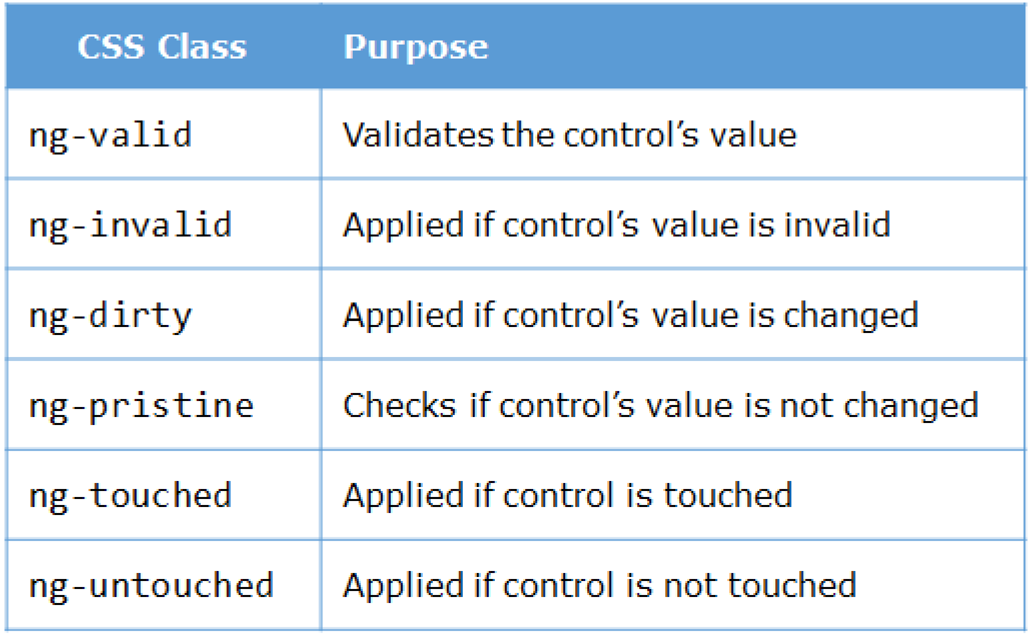
Form properties used in HTML Templates



Keyword	Purpose
valid	Valid control value
invalid	Invalid control value
dirty	Changed control value
pristine	Unchanged control value
touched	True if control is touched
untouched	True if control is not touched

Forms in Angular(3/3)

CSS properties used in HTML Templates



CSS Class	Purpose
`ng-valid`	Validates the control's value
`ng-invalid`	Applied if control's value is invalid
`ng-dirty`	Applied if control's value is changed
`ng-pristine`	Checks if control's value is not changed
`ng-touched`	Applied if control is touched
`ng-untouched`	Applied if control is not touched

Template Driven Forms

- With a template driven form, most of the work is done in the template

```
 <form (ngSubmit)="onSubmit()" #courseForm="ngForm">
<div class="form-group">
 <label for="id">Course Id</label>
 <input type="text" class="form-control" required
[(ngModel)]="course.courseId" name="id" #id="ngModel">
 <div [hidden]="id.valid || id.pristine" class="alert alert-
danger"> Course Id is required  </div>
 </div>
```

app.component.html

PUBLIC

Refer to official site for Demos: http://angular.io

Model Driven or Reactive Forms(1/2)

- **Example**

```
<form [formGroup]="registerForm">
  <div class="form-group">
 <label>First Name</label>
 <input type="text" class="form-control"
formControlName="firstName">
<p *ngIf="registerForm.controls.firstName.errors"
class="alert alert-danger">This field is required!</p>
</div>
```

app.component.html

Refer to official site for Demos: http://angular.io

Model Driven or Reactive Forms(2/2)

- **Example**

app.component.ts

```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
      ..
      export class RegistrationFormComponent implements OnInit {

        registerForm: FormGroup;

        constructor(private formBuilder: FormBuilder) { }

        ngOnInit() {
          this.registerForm = this.formBuilder.group({
            firstName: ['', Validators.required],
            lastName: ['', Validators.required],
            address: this.formBuilder.group({
              street: [],
              zip: [],
              city: []
            })
          });
        }

      }
```

Custom Validation

- **Validators** class in Angular has all build-in validators and for any customized validation, we need to create our own validations.

app.component.ts

```
export class AppComponent {
  title = 'Angular 4 Project!';
  todaydate;  componentproperty;
  emailid;    formdata;

ngOnInit() {
    this.formdata = new FormGroup({
      emailid: new FormControl("", Validators.compose([
        Validators.required,
        Validators.pattern("[^ @]*@[^ @]*")
      ])),
      passwd: new FormControl("", this.passwordvalidation)
    });
  }
```

```
  passwordvalidation(formcontrol) {
    if (formcontrol.value.length <'; 5) {
      return {"passwd" : true};
    }
  }
  onClickSubmit(data) {this.emailid = data.emailid;}
}
```

PUBLIC

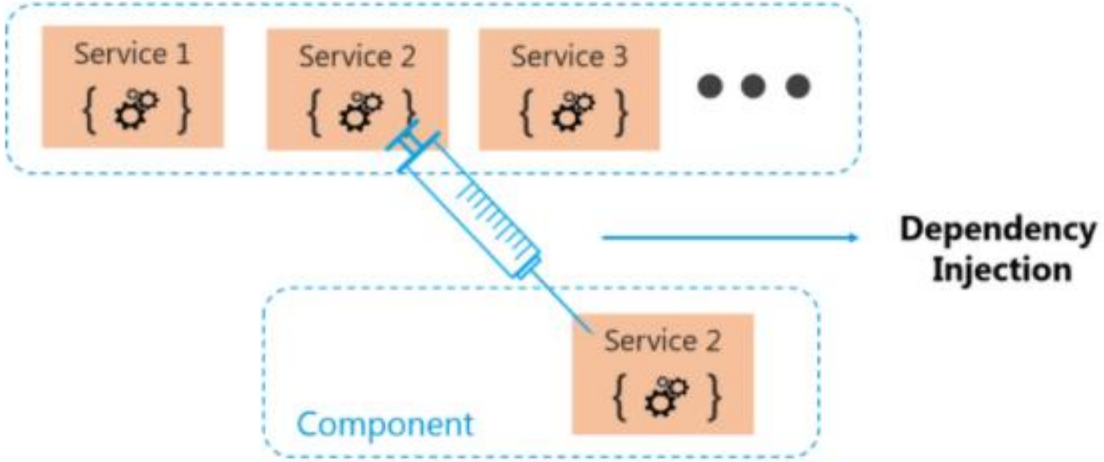
Refer to official site for Demos: http://angular.io

Dependency Injection

Infosys® | Education, Training and Assessment

Dependency Injection

- Dependency Injection
  - Dependency Injection (DI) is a mechanism where the required resources will be injected into the code automatically.
  - Angular comes with an in-built dependency injection subsystem.



Services

Infosys® | Education, Training and Assessment

Services

- A service in Angular is a class which contains some functionality that can be reused across the application. A service is a singleton object.
- Angular services are a mechanism of abstracting shared code and functionality throughout the application.

```
@Injectable()
class MyService
{
}
```

- Create a class with @Injectable() decorator.
- @Injectable() decorator makes the class injectable into application components.

PUBLIC

Create a Service and Inject

- **Create a Service**
  - ng generate service <servicename>
- **Add the service to app.module.ts**
  - Providers: [MyService]
- **Injecting a Service into Component class**
  - providers: [MyService]
  - Or
  - constructor(private service: MyService){ }

Refer to official site for Demos: http://angular.io

RxJS Observables

RxJS (Reactive Extensions for JavaScript)

- RxJS is an incredible tool for reactive programming to retrieve multiple data asynchronously
- An Observable is just a function, with a few special characteristics.
  - **“observer”** : an object with “next”, “error” and “complete” methods on it
  - **subscribe():** will invoke Observable to receive data from observer

Syntax: .subscribe(next, error, complete)

.next() - Observer calls when a new value was successfully captured

.error() - Observer calls when an error occurs

.complete() – when no more values to be sent or no longer interested in the values and to unsubscribe

.map() - returns a new Observable

.filter() - return Observable with the filtered results

RxJS Operators

Example 1:

```
        const one$ = new Observable(observer => {
                observer.next(1);
                observer.next(2);
                observer.complete();
        });
one$.subscribe({ next: value => console.log(value),
});
 // 1 2
```

Example 2:

```
var source =Rx.Observable.range(1,5);

Var subscription=source.subscribe(
x=> console.log(“onNext:%s”,x),     //map()
e=> console.log(“onError:%s”,e),   //error()
()=> console.log(“onComplete”)     //complete
);
```

Source : https://xgrommx.github.io/rx-book/index.html

HTTP Service

Http Service (1/4)

- Angular comes with its own library to make calls to server.
- When we make calls to an external server, we want our user to continue to be able to interact with the page. That is, we don't want our page to freeze until the HTTP request returns from the external server. So, all HTTP requests are asynchronous.
- The preferred method of dealing with asynchronous code in Angular is using Observables.
- We need to import **HttpModule** in the module class to make http service available to the entire module. Import Http service class into a component's constructor. We can make use of http methods like get, post, put and delete.

Http Service (2/4)

Syntax : this.http.get(url)

- Let assume we have a **data.json** file under assets folder with below code:

```
[
{ “regNo": “1001", “studentName": “Rahul" },
{ “regNo": “1002", “studentName": “Nithesh" },
]
```

Http Service (3/4)

- Create a **student.service.ts** file - **ng generate service student**

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { HttpErrorResponse, HttpResponse } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';
import { ErrorObservable } from 'rxjs/observable/ErrorObservable';
import { catchError, retry } from 'rxjs/operators';

  @Injectable()
  export class StudentService {
      dataUrl = 'assets/data.json';
      constructor(private http: HttpClient) { }
      getStudent() {
          return this.http.get(this.dataUrl);
      } }
```

Http Service (4/4)

- Call getStudent() from StudentService into a Component and collect the JSON data using subscribe() in RXJS

app.Component.ts:

- Inject the StudentService in Component Constructor.

```
var data="";
getData() {
        this.studentService.getStudent() .subscribe(data =>{ this.data = data;},      // Returns data from JSON file
        error => this.error = error
        );       }
```

app.component.html

```
<ul>
<li *ngFor="let n of data">{{n.regNo}} – {{n.studentName}}</li>
</ul>
```

Routing

Routing(1/4)

- Routing is navigation between multiple views in a single page
- Routing allows us to express some aspects of the application’s state in the URL. We can build the full application without ever changing the URL.

Why Routing?

- Maintain the state of the application
- Implement modular applications
- Implement the application based on the roles

<base href=“/”>

Routing(2/4)

- **RouterModule and Router** class is used for routing in angular

app.module.ts

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { RouterModule } from '@angular/router';
import { AppComponent } from './app.component';

@NgModule({
imports: [BrowserModule, RouterModule],
bootstrap: [AppComponent],
declarations: [AppComponent],
})
export class AppModule {}
```

Routing(3/4)

- Create an **app.routing.ts** file to set the routing path and its corresponding components:

```
import { Routes, RouterModule } from '@angular/router';
const appRoutes: Routes = [
    { path: 'welcome', component: WelcomeComponent },
    { path: 'placeOrder', component: PlaceOrderComponent },
    { path: 'bill/:bill', component: BillComponent },
    { path: '', redirectTo: '/welcome', pathMatch: 'full' }
];
@NgModule({
    imports: [RouterModule.forRoot(appRoutes)],
    exports: [RouterModule]
})
export class AppRoutingModule { }
```

Routing(4/4)

- User [routerLink] to change path in the URL and load the required component based on the path.

app.component.html

```
<ul class="nav navbar-nav">
    <li> <a [routerLink]="['/placeOrder']">Click Here to Place an Order</a> </li>
</ul>
```

http://localhost:4200/placeOrder

```
<router-outlet></router-outlet>
```

Router-outlet is a container where components will get loaded dynamically based on router path

Summary

- You are now knowledgeable on:
  - Implementation of Forms in Angular
  - Work of Dependency Injection
  - Reusability nature of Services
  - Asynchronous Data retrieval using RxJS Observables
  - Ajax call using HTTP
  - Routing

Reference

- Links:
  - https://angular.io/tutorial
  - https://www.tutorialspoint.com/angular4/index.htm
  - https://coursetro.com/courses/12/Learn-Angular-4-from-Scratch
  - https://www.udemy.com/learn-angular-from-scratch/
  - https://programmingwithmosh.com/angular/angular-4-tutorial/
  - https://www.techiediaries.com/angular-4-tutorial/

- Videos:
  - Angular 4 Tutorial for Beginners: Learn Angular 4 from Scratch - https://www.youtube.com/watch?v=k5E2AVpwsko

Thank You

● ● ●

Infosys
be more