Angular

IV Lecture

KindGeek

KindGeek School

services && dependency injection

Service examples

```
export class Logger {
  log(msg: any)   { console.log(msg); }
  error(msg: any) { console.error(msg); }
  warn(msg: any)  { console.warn(msg); }
}
```

```
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) => {
      this.logger.log(`Fetched ${heroes.length} heroes.` );
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

KindGeek School

Dependency injection (DI)

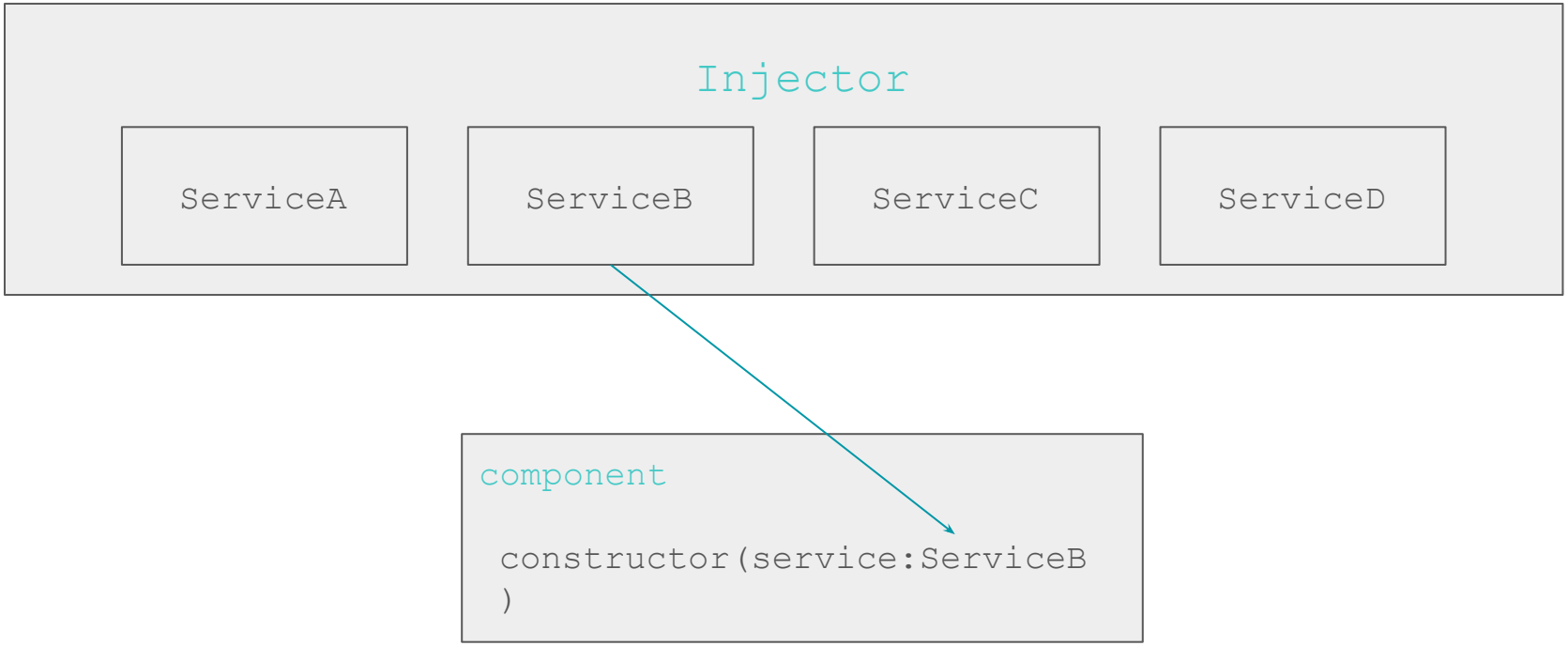
- The injector is the main mechanism. Angular creates an application-wide injector for you during the bootstrap process, and additional injectors as needed. You don't have to create injectors.
- An injector creates dependencies, and maintains a container of dependency instances that it reuses if possible.
- A provider is an object that tells an injector how to obtain or create a dependency.

```
@Injectbale()
export class HeroService {
  private heroes: Hero[] = [];

  constructor(
    private backend: BackendService,
    private logger: Logger) { }

  getHeroes() {
    this.backend.getAll(Hero).then( (heroes: Hero[]) =>
{
      this.logger.log(`Fetched ${heroes.length}` );
      this.heroes.push(...heroes); // fill cache
    });
    return this.heroes;
  }
}
```

KindGeek School



KindGeek School

Providing services

```
@Injectable({
 providedIn: 'root',
})
```

```
@NgModule({
  providers: [
  BackendService,
  Logger
 ],
 ...
})
```

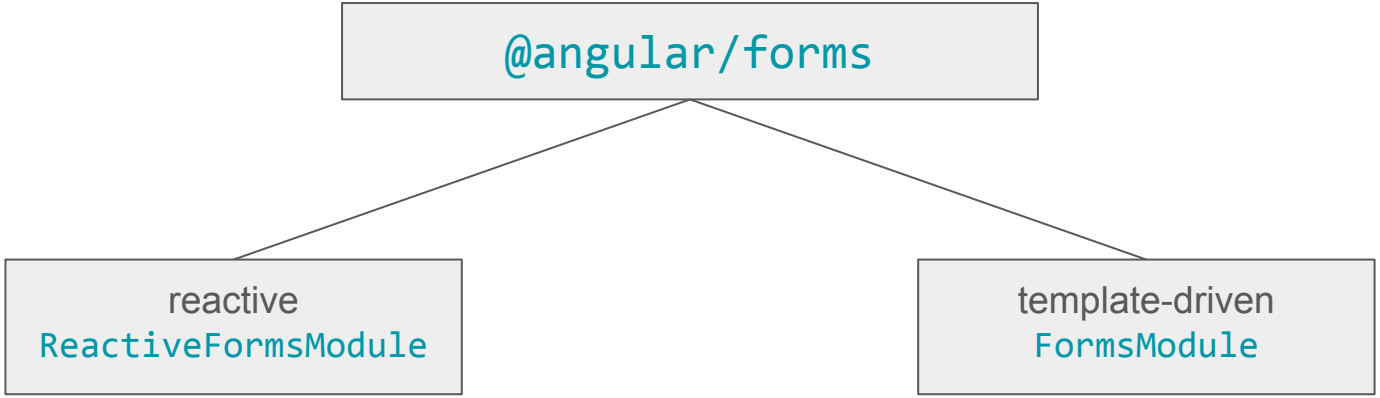
```
@Component({
  selector:     'app-hero-list',
  templateUrl: './hero-list.component.html',
  providers:   [ HeroService ]
})
```

KindGeek School

Handling Forms in Angular Apps

KindGeek School

Forms


@angular/forms
reactive
ReactiveFormsModule
template-driven
FormsModule

Reactive forms

- Reactive style
- Testing
- Validation
- Update always synchronous and under your control
- Immutability

KindGeek School

Template-driven forms

- ngModel
- mutable data model
- asynchronous

KindGeek School

Essential form classes

`AbstractControl`	AbstractControl is the abstract base class for the three concrete form control classes; FormControl, FormGroup, and FormArray. It provides their common behaviors and properties.
`FormControl`	FormControl tracks the value and validity status of an individual form control. It corresponds to an HTML form control such as an <input> or <select>.
`FormGroup`	FormGroup tracks the value and validity state of a group of AbstractControl instances. The group's properties include its child controls. The top-level form in your component is a FormGroup.
`FormArray`	FormArray tracks the value and validity state of a numerically indexed array of AbstractControl instances.

KindGeek School

Setup

- Import the ReactiveFormsModule
- Create a reactive forms component
- Create the template
- Add data to form

Import the ReactiveFormsModule

```
@NgModule({
  imports: [
    BrowserModule,
    FormsModule,
    ReactiveFormsModule,
    CommonModule
  ],
   bootstrap: [AppComponent]
})
```

KindGeek School

Create a reactive forms component

`FormBuilder`

```
public profileForm: FormGroup = this.fb.group({
  firstName: ['', Validators.required],
  lastName: ['', Validators.required],
  homeAddress: this.adressFormService.adreessFormGenerate,
  workAddress: this.adressFormService.adreessFormGenerate,
  phones: this.fb.array([]),
  color: ['', Validators.required]
});

constructor(@Inject(FormBuilder) private fb: FormBuilder) { }
```

```
public get adreessFormGenerate(): FormGroup {
    return this._fb.group({
      street: ['', Validators.required],
      city: ['', Validators.compose([this.validateCity])],
      state: ['', Validators.required],
      zip: ['', Validators.required]
    });
  }
```

KindGeek School

Create the template

```
<form [formGroup]="profileForm">
    <fieldset>
      <legend>Profile form</legend>
      <fieldset>
        <legend>Personal data</legend>
        <label>First name</label>
        <input type="text" formControlName="firstName" placeholder="First name">
        <label>Last name</label>
        <input type="text" formControlName="lastName" placeholder="Last name">
      </fieldset>
      <app-adress-form [legend]="'Work address'" [form]="profileForm.get('workAddress')">
      </app-adress-form>
      <app-color-picker formControlName="color"></app-color-picker>
    </fieldset>
</form>
```

KindGeek School

Add data to form

- patchValue()
- setValue()

```
this.profileService.getData()
     .subscribe(data => {
       this.profileForm.patchValue(data);
});
```

KindGeek School

Validators

```
public colorForm: FormGroup = this.fb.group({
    rgb: this.fb.group({
      red: [0, Validators.compose([Validators.min(0), Validators.max(255)])],
      green: [0, Validators.compose([Validators.min(0), Validators.max(255)])],
      blue: [0, Validators.compose([Validators.min(0), Validators.max(255)])]
    }),
    hsl: this.fb.group({
      hue: [0, Validators.compose([Validators.min(0), Validators.max(360)])],
      saturation: [0, Validators.compose([Validators.min(0), Validators.max(100)])],
      lightness: [0, Validators.compose([Validators.min(0), Validators.max(100)])],
    })
  });
```

KindGeek School

```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors | null
  static requiredTrue(control: AbstractControl): ValidationErrors | null
  static email(control: AbstractControl): ValidationErrors | null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string | RegExp): ValidatorFn
  static nullValidator(c: AbstractControl): ValidationErrors | null
  static compose(validators: (ValidatorFn | null | undefined)[] | null): ValidatorFn | null
  static composeAsync(validators: (AsyncValidatorFn | null)[]): AsyncValidatorFn | null
}

interface ValidatorFn {
  (c: AbstractControl): ValidationErrors | null
}

type ValidationErrors = {
    [key: string]: any;
};
```

KindGeek School

Custom validators

```
public get adreessFormGenerate(): FormGroup {
   return this._fb.group({
     street: ['', Validators.required],
     city: ['', Validators.compose([this.validateCity])],
     state: ['', Validators.required],
     zip: ['', Validators.required]
   });
 }

 public validateCity(formControl: AbstractControl): ValidationErrors | null {
   return ['London', 'Paris', 'Lviv', 'Tokyo', 'New-York', 'Ternopil']
     .map(item => item.toLowerCase()).includes(formControl.value.toLowerCase().trim()) ? null :
     { noCity: true };
 }
```

KindGeek School

FormArray

- initialize the FormArray
- display the FormArray
- add a new item to FormArray

KindGeek School

initialize the FormArray

```
public profileForm: FormGroup = this.fb.group({
   phones: this.fb.array([]),
 });
```

```
this.profileForm.setControl('phones', this.fb.array(data.phones));
```

KindGeek School

Display the FormArray

```
<div formArrayName="phones">
     <div *ngFor="let phone of phonesArr.controls; let i=index">
          <input type="text" [formControlName]="i">
     </div>
</div>

get phonesArr(): FormArray {
     return this.profileForm.get('phones') as FormArray;
}
```

KindGeek School

Add a new item to FormArray

```
public addPhone() {
    const phoneArr = this.profileForm.get('phones') as FormArray;
    phoneArr.push(new FormControl(''));
}
```

KindGeek School

Pipe

KindGeek School

Using pipes

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

@Component({
  selector: 'app-hero-birthday',
  template: `<p>The hero's birthday is {{ birthday | date }}</p>`
})
export class HeroBirthdayComponent {
  birthday = new Date(1988, 3, 15); // April 15, 1988
}
```

KindGeek School

Pipes list

P AsyncPipe	P CurrencyPipe	P DatePipe
P DecimalPipe	P DeprecatedCurrencyPipe	P DeprecatedDatePipe
P DeprecatedDecimalPipe	P DeprecatedPercentPipe	P I18nPluralPipe
P I18nSelectPipe	P JsonPipe	P KeyValuePipe
P LowerCasePipe	P PercentPipe	P SlicePipe
P TitleCasePipe	P UpperCasePipe	

https://angular.io/api?type=pipe

KindGeek School

Parameterizing a pipe

```
<p>The hero's birthday is {{ birthday | date:"MM/dd/yy" }}   </p>


template: `
  <p>The hero's birthday is {{ birthday | date:format }}</p>
  <button (click)="toggleFormat()">Toggle Format</button>
`
```

KindGeek School

Custom pipes

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'exponentialStrength' })
export class ExponentialStrengthPipe implements PipeTransform {
  transform(value: number, exponent: string): number {
    let exp = parseFloat(exponent);
    return Math.pow(value, isNaN(exp) ? 1 : exp);
  }
}
```

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

@Component({
  selector: 'app-power-booster',
  template: `
    <h2>Power Booster</h2>
    <p>Super power boost: {{2 | exponentialStrength: 10}}</p>
  `
})
export class PowerBoosterComponent { }
```

- A pipe is a class decorated with pipe metadata.
- The pipe class implements the PipeTransform interface's transform method that accepts an input value followed by optional parameters and returns the transformed value.
- There will be one additional argument to the transform method for each parameter passed to the pipe. Your pipe has one such parameter: the exponent.
- To tell Angular that this is a pipe, you apply the @Pipe decorator, which you import from the core Angular library.
- The @Pipe decorator allows you to define the pipe name that you'll use within template expressions. It must be a valid JavaScript identifier. Your pipe's name is exponentialStrength.

KindGeek School

Home task

1. Service
   Create service for share data.
2. ReactiveForm
   `{firstName:'John', lastName:'Doe',`
   `age:56,children:[{firstName:'John Jr.',lastName:'Doe'},....]}`
   firstName, lastName validation - first letter uppercase,
   age validation - min:0 max:120
3. Pipe
   a. Create pipe in order to calculate the circle square. Input data radius of circle.
   b. Create pipe for transforming HEX to RGB color.

KindGeek School

https://angular.io/guide/architecture-services
https://en.wikipedia.org/wiki/Dependency_injection

https://angular.io/guide/forms-overview
https://angular.io/guide/reactive-forms
https://angular.io/guide/form-validation

https://angular.io/guide/pipes