Angular (Part – 5)

PROF. P. M. JADAV
ASSOCIATE PROFESSOR
COMPUTER ENGINEERING DEPARTMENT
FACULTY OF TECHNOLOGY
DHARMSINH DESAI UNIVERSITY, NADIAD

Content

- Forms
- Common Foundation
- Forms Validation
- Template-driven Forms
- Model-driven Forms (Reactive Forms)

Forms

Forms are used to

- login
- place an order
- book a flight ticket
- schedule a meeting etc.

Forms

1. Template Driven Forms
   - validation and binding are all setup in a declarative way at the level of the template
   - directives like required, ngModel, NgForm, maxlength are used
   - are asynchronous
   - Difficult to unit test
   - Uses FormsModule
   - Used while designing simple forms or in simple application

Forms

2. Model Driven Forms (Reactive Forms)

- Processing of the form data is done in the model (component class)
- value and validity updates are always synchronous and under your control
- More scalable, reusable and testable
- Uses ReactiveFormsModule
- Used while designing complex forms or forms are used heavily

Reactive Vs. Template-Driven Forms

	REACTIVE	TEMPLATE-DRIVEN
Setup (form model)	More explicit, created in component class	Less explicit, created by directives
Data model	Structured	Unstructured
Predictability	Synchronous	Asynchronous
Form validation	Functions	Directives
Mutability	Immutable	Mutable
Scalability	Low-level API access	Abstraction on top of APIs

Common Foundation

- FormControl tracks the value and validation status of an individual form control
- FormGroup tracks values and status for a collection of form controls
- FormArray tracks the same values and status for an array of form controls (an alternative to [FormGroup] for managing any number of unnamed controls)
- ControlValueAccessor creates a bridge between Angular FormControl instances and native DOM elements

Form Model Setup

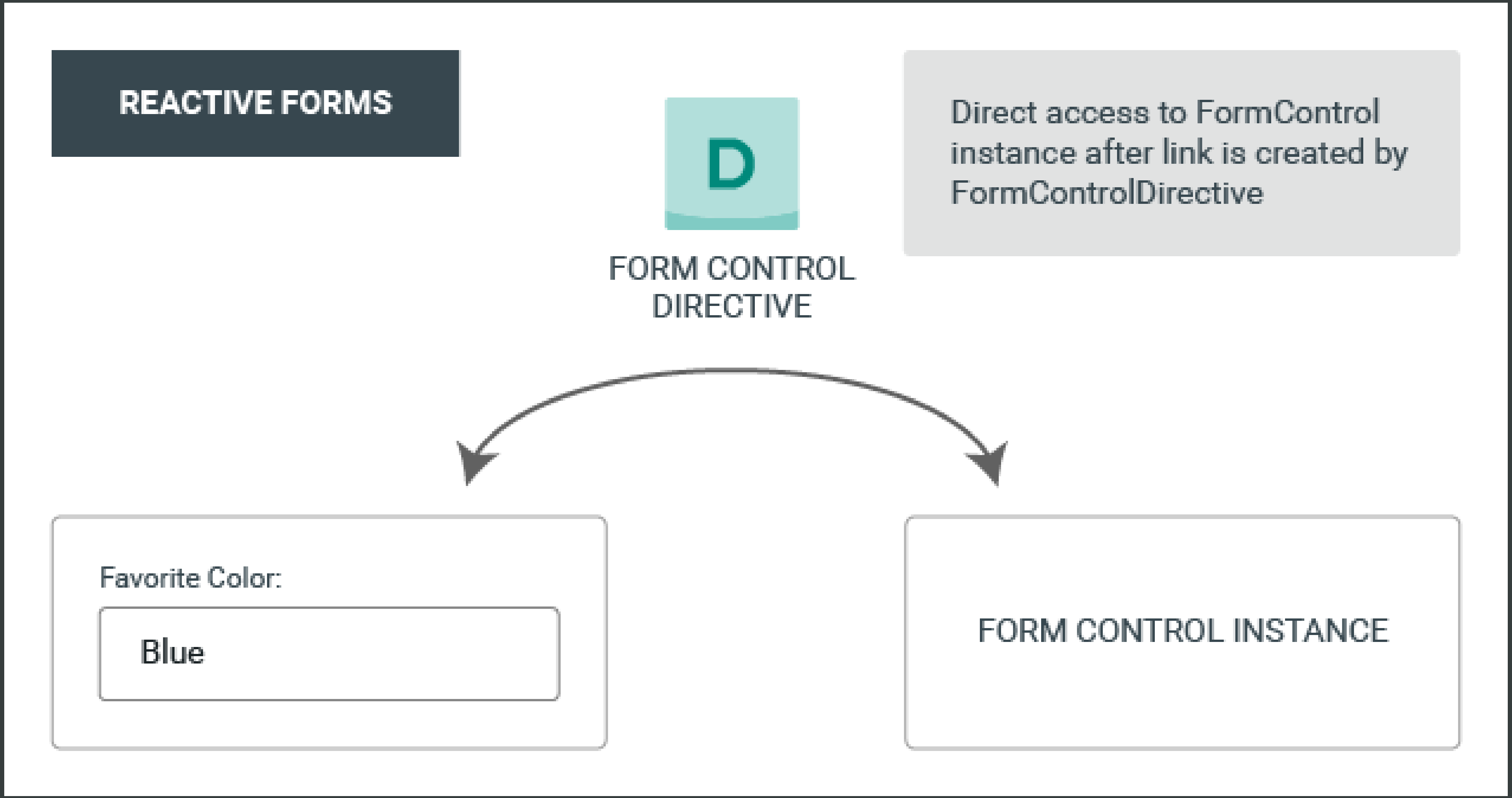
- Reactive and template-driven forms both use a form model
  - ✓ to track value changes between Angular forms and form input elements

Setup in Reactive Forms

```
Favourite Color :
<input type="text" [formControl]="color">
```

```
color = new FormControl('')
```

Source - FormControl



Direct access to FormControl instance after link is created by FormControlDirective

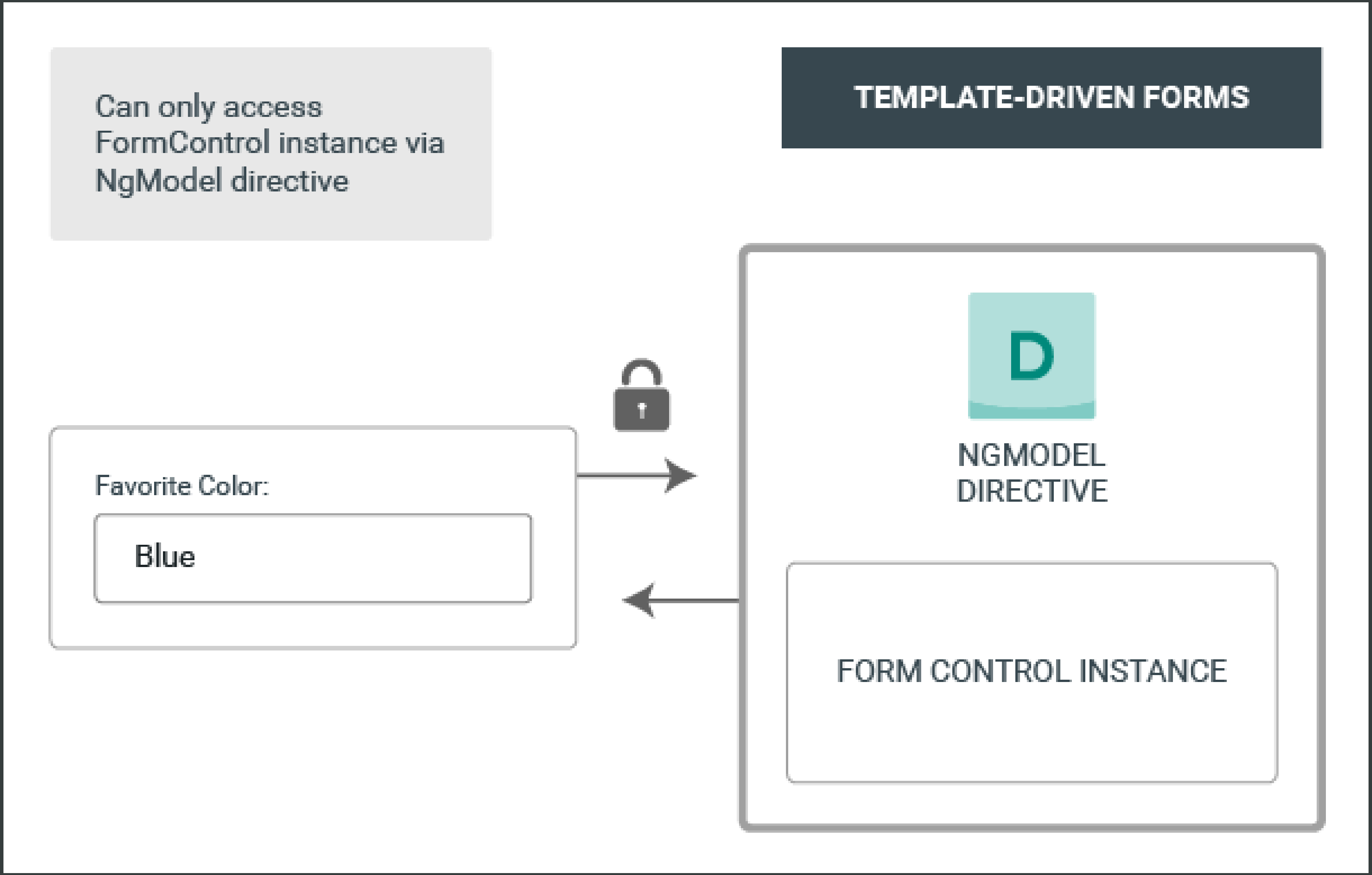
Setup in Template-driven Forms

Creates and manages FormControl instance

```
Favourite Color:
<input type="text" [(ngModel)]="color">
```

```
color = 'blue';
```

Source - FormControl



Form Validation

- Angular has
  - Built-in validators
  - Supports custom validators
- Reactive Forms
  - Functions used for custom validation
- Template-driven Forms
  - uses template directive for custom validation

Form Validation

- Angular has built-in validators such as
  - mandatory field
  - minlength
  - maxlength, and
  - pattern
- These are to be accessed using the Validators module

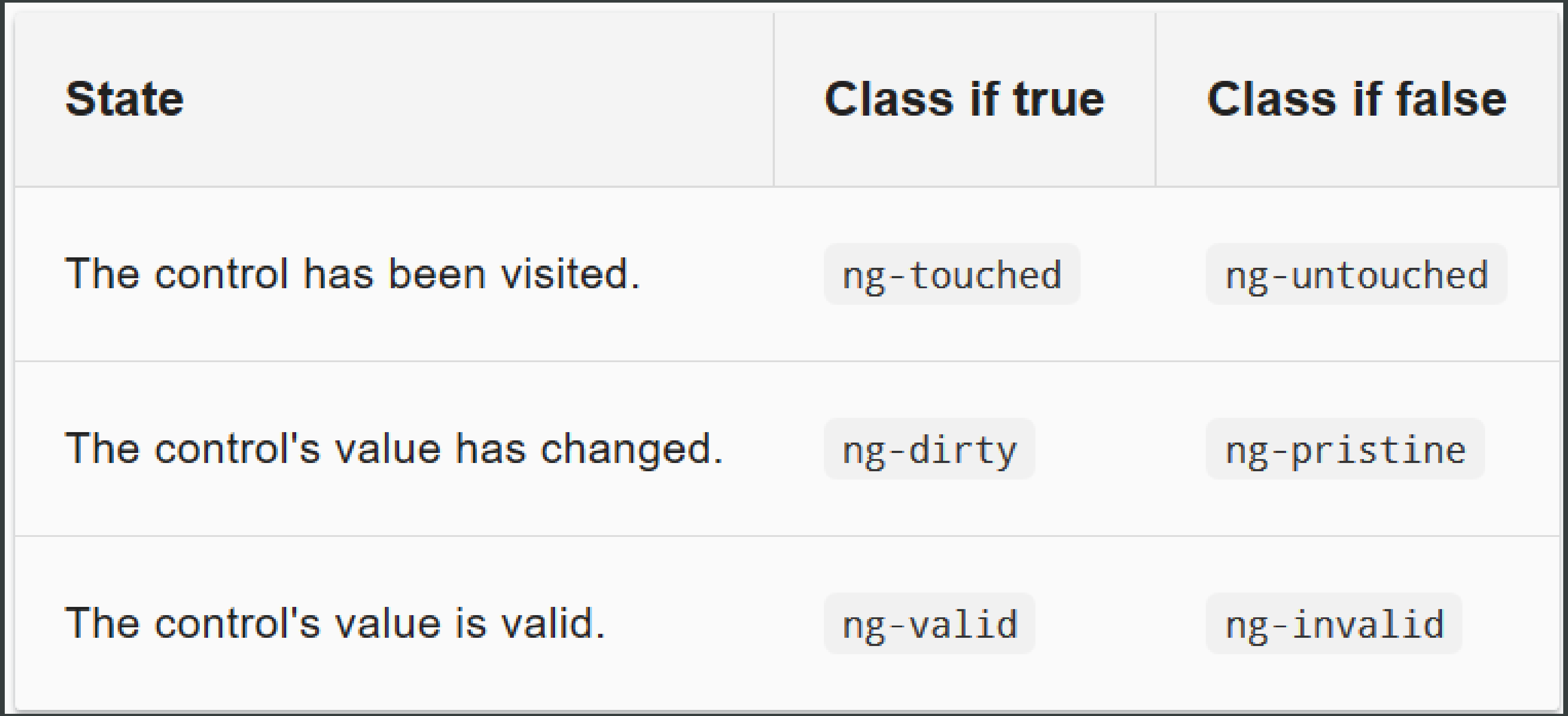
Template Driven Validation

- Add the same validation attributes as you would with native HTML form validation
- Angular uses directives to match these attributes with validator functions in the framework
- Every time the value of a form control changes, Angular runs validation and generates either a list of validation errors, which results in an INVALID status, or null, which results in a VALID status
- You can then inspect the control's state by exporting ngModel to a local template variable

Track control state and validity with *ngModel*

- Using ngModel in a form gives you more than just two-way data binding
- It also tells you if the user
  - touched the control
  - if the value changed, or
  - if the value became invalid
- The NgModel directive doesn't just track state; it updates the control with special Angular CSS classes that reflect the state
- You can leverage those class names to change the appearance of the control

Track control state and validity with *ngModel*



State	Class if true	Class if false
The control has been visited.	`ng-touched`	`ng-untouched`
The control's value has changed.	`ng-dirty`	`ng-pristine`
The control's value is valid.	`ng-valid`	`ng-invalid`

Template Driven Forms

Create a new Component:

```
ng  g  c  temp-form
```

app.module.ts

```
import { FormsModule} from '@angular/forms';
import { TempFormComponent } from './temp-form/temp-form.component';
@NgModule({
        declarations: [
                AppComponent, TempFormComponent
        ],
        imports: [
                BrowserModule, FormsModule,
        ],
        bootstrap: [AppComponent]
})
export class AppModule { }
```

temp-form.component.html

```
<form (ngSubmit)="onSubmit()" #regForm="ngForm">
    <input    type="text"    name="name"
              required
              minlength="4"
              [(ngModel)]="name"
              #name_ctl="ngModel“
    >
```

temp-form.component.html

```
    <div *ngIf = "name_ctl.invalid &&
                (name_ctl.dirty || name_ctl.touched)">
        <div *ngIf = "name_ctl.errors.required">
                Name is required.
        </div>
        <div *ngIf = "name_ctl.errors.minlength">
                Name must be at least 4 characters long.
        </div>
    </div>
    <button type="submit" [disabled]="!regForm.form.valid">
        Submit
    </button>
</form>
```

temp-form.component.ts

```
export class TempFormComponent  {
        name : string

        onSubmit() {
                console.log(name)
        }
}
```

Model Driven Forms

Create a new Component:

```
ng  g  c  react-form
```

app.module.ts

```
import {ReactiveFormsModule} from '@angular/forms';
import { ReactFormComponent } from './react-form/react-form.component';


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

react-form.component.html

```
<form [formGroup] = "profileForm" (ngSubmit) = "onSubmit()">
        <input type="text" formControlName="firstName"  >
        <div    *ngIf = "fname.invalid &&
                          (fname.dirty || fname.touched)">
                <div *ngIf="fname.errors.required">
                        Name is required.               </div>
        </div>
        <button     type="submit"
                    [disabled]="!profileForm.valid" >Submit
        </button>
</form>
```

react-form.component.ts

```
export class ReactFormComponent implements OnInit {
        profileForm : FormGroup

        ngOnInit() {
                this.profileForm = new FormGroup({
                        firstName: new FormControl('', Validators.required),
                });
        }
        get fname() { return this.profileForm.get('firstName')}

        onSubmit() {    console.log(this.profileForm.value); }
}
```

react-form.component.ts (Using FormBuilder)

```
export class ReactFormComponent implements OnInit {
        registerForm : FormGroup
        constructor(private formBuilder: FormBuilder) { }
        ngOnInit() {
                this.registerForm = this.formBuilder.group({
                        firstName: ['', Validators.required],
                        email: ['', [Validators.required, Validators.email]],
                })
        }
        get f() { return this.registerForm.controls; }
        onSubmit() {    console.log(this. registerForm.value);          }
}
```

References

- https://angular.io/docs