Custom Attribute Directives

Here directive is just a class with decorator **@Directive** and this allows us to specify the directive's selector.

- Using custom attribute directive, we can change the color, back-ground, font-size etc., of the HTML host element by using **ElementRef.**
- **Renderer** class is a built-in service that provides an abstraction for UI rendering manipulations.

Let's create a custom attribute directive "**ShowAlert**"

1. Create a folder and name it as **CustomDirective: /src/app/CustomDirective**
2. Now create a typescript file to create your own attribute directive under the associated folder created. And then **import Directive, ElementRef, HostListener, Renderer from @angular/core library.**
3. And provide your selector name, here I've named it as **ShowAlert,** and create a class that make this directive visible to the component.

File: **showalert.directive.ts**

```
import { Directive, ElementRef, HostListener, Renderer } from '@angular/core';
@Directive({
  selector: '[showalert]'
})
export class ShowAlertDirective {}
```

4. Now define constructor for and define the style and events using renderer.

You can set styles using

Renderer.setElementStyle(rendererElement:any, styleName: string, styleValue: string)

(or)

ElementRef.nativeElement.style

Create events using

Renderer.listen(parentElement: any, name: string, callBack: function)

showalert.directive.ts

```
export class ShowAlertDirective{
constructor(private el: ElementRef, private renderer: Renderer) {
    renderer.setElementStyle(el.nativeElement, "cursor", "pointer");
    renderer.setElementStyle(el.nativeElement, "color", "blue");

    renderer.listen(el.nativeElement, "mouseover", function () {
      renderer.setElementStyle(el.nativeElement, "color", "red");
       //renderer.setElementStyle(el.nativeElement, "font-size", "24px");
```

```
        el.nativeElement.style.fontSize = '24px';
      });
      renderer.listen(el.nativeElement, "mouseout", function () {
        renderer.setElementStyle(el.nativeElement, "color", "blue");
        renderer.setElementStyle(el.nativeElement, "font-size", "16px");
      });
      renderer.listen(el.nativeElement, "click", function () {
        alert("hello world");
      });
  }
}
```

5. Create your component

File: **app.component.ts**

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

@Component({
  selector: 'my-app',
  template: `<div showalert color="blue"> Hover Me!! Click Me!!</div>`
})
export class AppComponent {}
```

6. Create **NgModule** and declare the attribute directive class with AppComponent

```
import { ShowAlertDirective } from './showalert.directive';

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

Output: Default color will be blue, on mouseover font size get increased and changes its color to red, on mouseout will change color to blue and decrease its font size.

Using HostListener

@HostListener is a decorator for the Callback event handler methods to listen to the events that are raised on the host element. When an event get fired on the host element corresponding method will get called.

Syntax:

```
@HostListener("event") methodname( ) {
//your code
}
```

From the ShowAlert example

Edit File: showalert.directive.ts

```
import { Directive, ElementRef, Renderer, HostListener, Input, OnInit } from '@angular/core';
@Directive({
  selector: '[ShowAlert]'
})
export class ShowAlertDirective implements OnInit {
  constructor(private el: ElementRef, private renderer: Renderer) {}
  @Input() color: string;
  ngOnInit(): any {
    this.el.nativeElement.style.cursor = "pointer";
    this.changeColor(this.color);
  }
  @HostListener("click") onclick() {
    alert("hello world");
  }
  @HostListener("mouseover") onmouseover() {
    this.changeSize(24);
    this.changeColor("red");
  }
  @HostListener("mouseout") onmouseout() {
    this.changeSize(16);
    this.changeColor("blue");
  }
  private changeSize(size: number) {
    this.renderer.setElementStyle(this.el.nativeElement, "font-size", size+"px");
  }
```

```
  private changeColor(color: string) {
    this.el.nativeElement.style.color = color;
  }
}
```

Using HostBinding

- To bind the input properties of the host element from directive we use **HostBinding.**
- This uses decorator **@HostBinding** for linking an internal property to an input property on host element. So if any changes are done to internal properties that will reflect on the input property as well.

Form the previous example let us bind color to the host element using **@HostBinding,** import **HostBinding from @angular/core**

Showalert.directive.ts

```
import {. . . . , HostBinding ,. . . . . .} from '@angular/core';
export class ShowAlertDirective implements OnInit {
  constructor(private el: ElementRef, private renderer: Renderer) { }
  @Input() color: string;
  ngOnInit(): any {
    this.el.nativeElement.style.cursor = "pointer";
    this.color = "blue";
  }
  @HostListener("click") onclick() {
    alert("hello world");
  }
  @HostListener("mouseover") onmouseover() {
    this.changeSize(24);
    this.color = "red";
  }
  @HostListener("mouseout") onmouseout() {
    this.changeSize(16);
    this.color = "blue";
  }
  @HostBinding('style.color') get AnyNameIsOK() {
    return this.color;
  }
  private changeSize(size: number) {
```

```
    this.renderer.setElementStyle(this.el.nativeElement, "font-size", size+"px");
  }
}
```

Example2: Custom Validators

Till now we have used **HTML5 built-in validators,** now let us create our own custom validators which can be added to input elements as directive.

Example: That take input with first character uppercase

Step1: Implement Validation Factory

Validation Factory return validation function and to get access to the field value we need to pass the **AbstractControl** as parameter to the validation function.

Step1: Create a reusable directive

Now our validation function is ready, now we need to make use of this validation as an attribute to form elements, to achieve that we need to create a **Directive.**

File: upper.validator.ts

```
import { Directive } from '@angular/core'
import { AbstractControl, ValidatorFn, Validator, FormControl, NG_VALIDATORS } from '@angular/forms';

function validateUpperCaseFactory(): ValidatorFn {
  return (ctrl: AbstractControl) => {
    let isValid = ctrl.value.charAt(0).toUpperCase() === ctrl.value.charAt(0);
    if (isValid) {
      return null;
    }
    else {
      return {
        upper: {
          valid: false
        }
      };
    }
  }
}
@Directive({
```

```
  selector: '[upper][ngModel]',
  providers: [
    {
      provide: NG_VALIDATORS,
      useExisting: UpperCaseValidator,
      multi: true
    }
  ]
})
export class UpperCaseValidator implements Validator {
  validator: ValidatorFn;
  constructor() {
    this.validator = validateUpperCaseFactory();
  }
  validate(ctrl: FormControl) {
    return this.validator(ctrl);
  }
}
```

Step2: Import custom validator in the component

File: **myform.component.ts**

```
import { Component, OnInit } from '@angular/core';
import { FormGroup, FormControl, Validators, FormBuilder } from '@angular/forms';
import { UpperCaseValidator } from './upper.validator';

@Component({
  selector: 'my-form,
  templateUrl: `./myformtemplate.html`,
})
export class MyFormComponent implements OnInit {
  constructor(private fb: FormBuilder) { }
  myform: FormGroup;
  ngOnInit() {
    this.myform = this.fb.group({
      myname: ['', UpperCaseValidator]
```

```
    });
}
```

Step4: Implement the custom validator to the form element

File: **myformtemplate.html**

```
<form [formGroup]="myform" novalidate>
<div class="form-group">
        <label>Enter Name</label>
        <input type="text" class="form-control" formControlName="myname" upper />
        <div class="form-control-feedback" *ngIf="myform.controls['myname'].errors &&
(myform.controls['myname'].dirty || myform.controls['myname'].touched)">
          <p *ngIf="myform.controls['myname'].errors.required">Name is required</p>
          <p *ngIf="myform.controls['myname'].errors.upper">Name should start with an uppercase</p>
        </div>
      </div>
      <pre>{{myform.value | json}}</pre>
</form>
```

File: app.module.ts

```
import { FormsModule, ReactiveFormsModule } from '@angular/forms';
import { MyFormComponent } from "./myform.component"

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