**What are Custom Directives ?**

Similar to the **built in** directives, we can also create our own such directives, also known as***custom directives.*** Such directives can be used to change the appearance of the components as we need.

In simple words, **Custom Directives** can be considered as components without a template, and it directly uses the HTML element to which it is applied. **Custom directives** can alter the appearance and bring in visual changes to the element, without wrapping them into a new component.

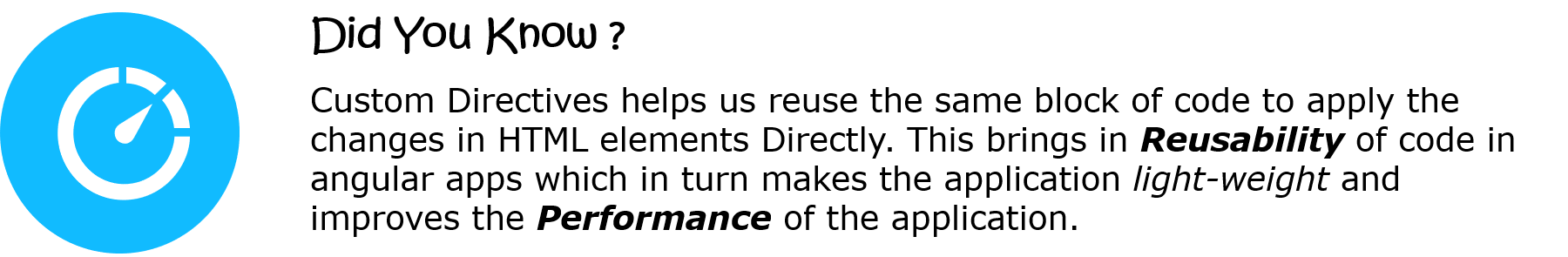
**Types of Custom Directives:**

The way we have different types of Built in directives i.e. **Structural**and **Attribute.**Similarly we can have different types of custom directives as well:

1. **Custom Attribute Directives**
2. **Custom Structural Directives**

**When can you use Custom Directives ?**

* Custom directives can be used for accessing and manipulating the DOM directly
* It can also be used for some reusable functionality which needs to be applied on other parts of the application as well
* We can use custom directives to define custom validations for the template driven forms



We can use the angular cli command as given below to generate a custom directive:

1. ng generate directive custom-directive-name

Let's create a custom directive with name **button:**

1. ng generate directive button

This creates **two**files in the application folder i.e. **button.directive.ts**and **button.directive.spec.ts**for directive implementation and directive test cases respectively.

Once the directive is created, it would also be added to the corresponding module **declarations array**as show below:

**app.module.ts**

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { AppRoutingModule } from './app-routing.module';
4. import { AppComponent } from './app.component';
5. import { ButtonDirective } from './button.directive';
6. @NgModule({
7. declarations: [
8. AppComponent,
9. ButtonDirective
10. ],
11. imports: [
12. BrowserModule,
13. AppRoutingModule
14. ],
15. providers: [],
16. bootstrap: [AppComponent]
17. })
18. export class AppModule { }

**Line 11: ButtonDirective**is the name of the created custom directive.

The created **button.directive.ts** file will have default code as below:

1. import { Directive } from '@angular/core'
2. @Directive({
3. selector: '[appButton]',
4. })
5. export class ButtonDirective {
6. constructor() {}
7. }

Where,

**Line 1: Directive** is imported from @angular/core library

**Line 3:** **@Directive** is the decorator that signifies the **ButtonDirective** class as a **Directive**

**Line 4:**The @Directive decorator has a single property i.e.**selector,** which should be used to use this custom directive with the HTML elements in angular components.

Interestingly, the selector is enclosed within **square brackets ([]),**and hence it is called as Attribute selector which helps angular to identify all the elements having the attribute named **appButton**

**Note:-**By default the selector name will be the directive nameprefixed with **app,**but we can always change this to any other value if needed

Next we will see how to implement custom attribute and structural directives.

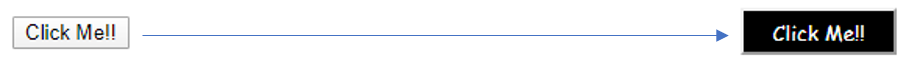
We can create both types of custom directives i.e. **custom attribute directives**and **custom structural directives**. Let us first start with a simple example of **custom attribute directives.**

To access and manipulate the HTML element in the angular components, we need to follow the following basic steps in any custom attribute directive:

1. import **ElementRef**module from **@angular/core**
2. Inject **ElementRef** into the directive class constructor to create an instance of the same
3. Use the **ElementRef**instance to access and manipulate the HTML elements

We had already created a custom directive with selector **appButton**. Let us see how we can use this directive to do changes in the appearance of a button element in angular component.

We need to use the **appButton**directive to change the appearance of a **button element**present in app component as shown below:



To do this, we need to write the following code in **button.directive.ts** file.

1. import { Directive, ElementRef } from '@angular/core';
2. @Directive({
3. selector: '[appButton]'
4. })
5. export class ButtonDirective {
6. constructor(private elementRef: ElementRef) {
7. this.elementRef.nativeElement.style.backgroundColor = 'black';
8. this.elementRef.nativeElement.style.color = 'white';
9. this.elementRef.nativeElement.style.fontFamily = 'cursive';
10. this.elementRef.nativeElement.style.width = '100px';
11. this.elementRef.nativeElement.style.height = '30px';
12. }
13. }

**Line 1:** **ElementRef**is imported from @angular/core. It helps us to directly access the DOM elements and make changes to it

**Line 8: ElementRef**is injected into the constructor to create its instance which will then be used to access the DOM elements

The **app.component.html** file should be modified as below:

1. <button appButton>Click Me!!</button>

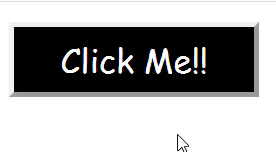
**Note:-**The directive should be added as an attribute as shown in the above HTML code, to modify the appearance of that HTML element

Next we will see how to listen to events to change the appearance using custom directives.

In the previous example, we changed the appearance of the button by hard coding the styling parameters i.e. color, background-color, etc. We can also change the styling of the element based on the event that occurs on the element.

We can Listen to the events and do changes based on the event using the module HostListener of @angular/core. Lets see how to use the **HostListener**.

**Problem Statement -**Create a button with label ***Click Me!!***, change the color of the button to **black**on **mouseover**and and to **orange** when mouse leaves the button, with orange being the default color on load of the component.



**Steps:-**To achieve the above functionality, follow the given steps

1. Add the import **HostListener**from **@angular/core** in **button.directive.ts**
   1. import { Directive, ElementRef, HostListener } from '@angular/core';
2. Use **HostListener** as a decorator to listen to different events that occur on the DOM elements
   1. @HostListener('mouseenter') onMouseEnter() {
   2. this.elementRef.nativeElement.style.backgroundColor = 'orange';
   3. }

Where ***mouseenter***is the event and ***onMouseEnter***is the function that gets executed whenever **mouseenter**event occurs on the DOM element having **appButton**directive.

The complete code to be written in **button.directive.ts** is:

1. import { Directive, ElementRef, HostListener } from '@angular/core';
2. @Directive({
3. selector: '[appButton]'
4. })
5. export class ButtonDirective {
6. constructor(private elementRef: ElementRef) {
7. this.elementRef.nativeElement.style.backgroundColor = 'black';
8. this.elementRef.nativeElement.style.color = 'white';
9. this.elementRef.nativeElement.style.fontFamily = 'cursive';
10. this.elementRef.nativeElement.style.width = '100px';
11. this.elementRef.nativeElement.style.height = '30px';
12. }
13. @HostListener('mouseenter') onMouseEnter() {
14. this.elementRef.nativeElement.style.backgroundColor = 'orange';
15. }
16. @HostListener('mouseleave') onMouseLeave() {
17. this.elementRef.nativeElement.style.backgroundColor = 'black';
18. }
19. }

**app.component.html** should have the following code having a button with **appButton**directive

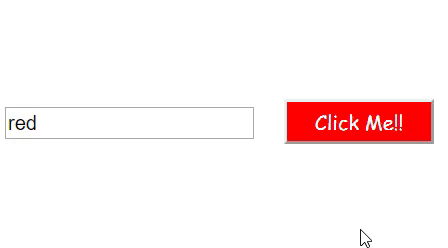
1. <button appButton>Click Me!!</button>

Next we will see how to pass data from the component to the directive.

In the previous two examples, we have already seen how to change the appearance of the element and how to listen to the events occuring on the elements. Still the styling was hardcoded. This looks to be a bit inflexible. We can allow the developer to set the color while using the directive on the DOM elements.

To bind the component with the custom directive properties we need to use @Input imported from @angular/core

**Problem Statement:-**Create an**input field**and a**button**with custom directive **appButton.**The background color of the button should change to the color entered in the input field when the user hovers over the button.



**Steps:-**To achieve the above functionality, follow the given steps

1. Add the import **Input**from **@angular/core** in **button.directive.ts**
   1. import { Directive, ElementRef, HostListener, Input } from '@angular/core';
2. Define a property named **buttonColor**in **button.directive.ts** file as shown below that takes the data from component where the **appButton**directive is used:
   1. @Input() buttonColor:string;
3. Pass the value using **property binding**in the button element of **app.component.html**as shown,
   1. <button appButton [buttonColor]='color'>Click Me!!</button>

The complete code to be written in **button.directive.ts** is:

1. import { Directive, ElementRef, HostListener, Input } from '@angular/core';
2. @Directive({
3. selector: '[appButton]'
4. })
5. export class ButtonDirective {
6. @Input() buttonColor:string;
7. constructor(private elementRef: ElementRef) { }
8. ngOnInit(): void {
9. this.elementRef.nativeElement.style.backgroundColor = this.buttonColor;
10. this.elementRef.nativeElement.style.color = 'white';
11. this.elementRef.nativeElement.style.fontFamily = 'cursive';
12. this.elementRef.nativeElement.style.width = '100px';
13. this.elementRef.nativeElement.style.height = '30px';
14. }
15. @HostListener('mouseenter') onMouseEnter() {
16. this.elementRef.nativeElement.style.backgroundColor = this.buttonColor;
17. }
18. @HostListener('mouseleave') onMouseLeave() {
19. this.elementRef.nativeElement.style.backgroundColor = 'black';
20. }
21. }

**app.component.html** should have the following code having a button with **appButton**directive:

1. <input type="text" [(ngModel)]="color">
2. <button appButton [buttonColor]='color'>Click Me!!</button>

**app.component.ts** should have the following code to set the **default**value of **color**to **red**:

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'app-root',
4. templateUrl: './app.component.html',
5. styleUrls: ['./app.component.css']
6. })
7. export class AppComponent {
8. color = 'red';
10. }