# **ANGULAR**

* npm – node package manager
* npm install -g @angular/cli
* ng new my-dream-app
* package.json – all dependencies
* node\_modules – all dependencies installed

Component:

* always has template – html
* possibly css file
* typescript file – definitions of component
* it is a separated reusable part of the page,
* it has its own logic

Data binding:

* in component you have title = app
* in html template you have <h1> Hi this is my {{ title }} </h1>
* Result: Hi this is my app

App-root

* Your own selector defined in component as

selector: ‘app-root’

result : <app-root> </app-root>

Directive:

* ngModel

Everything starts in main.ts

main.ts > app.module.ts > app.component.ts > app.component.html

Angular is a JS framework changing your DOM (HTML) at runtime!

Decorater: @component - Enhance your classes, elements

Meta data for component @component ({ })

Module: Bundle different pieces – component - into packages

**CLI commands**

Vytvorenie componentu:

* ng generate component name\_of\_component = ng g c name\_of\_component

/ spec file is usually used for testing /

Do not forget to update app.module after creating component!

templateUrl: app.component.html

template: <h2> </h2>

styleUrls: [app.component.css]

styles: [ {h3{ color : blue }} ]

As for selector in @component there are some choices:

selector: ‘app-root’ – element

‘[app-root]’ - attribute

‘.app-root’ – class

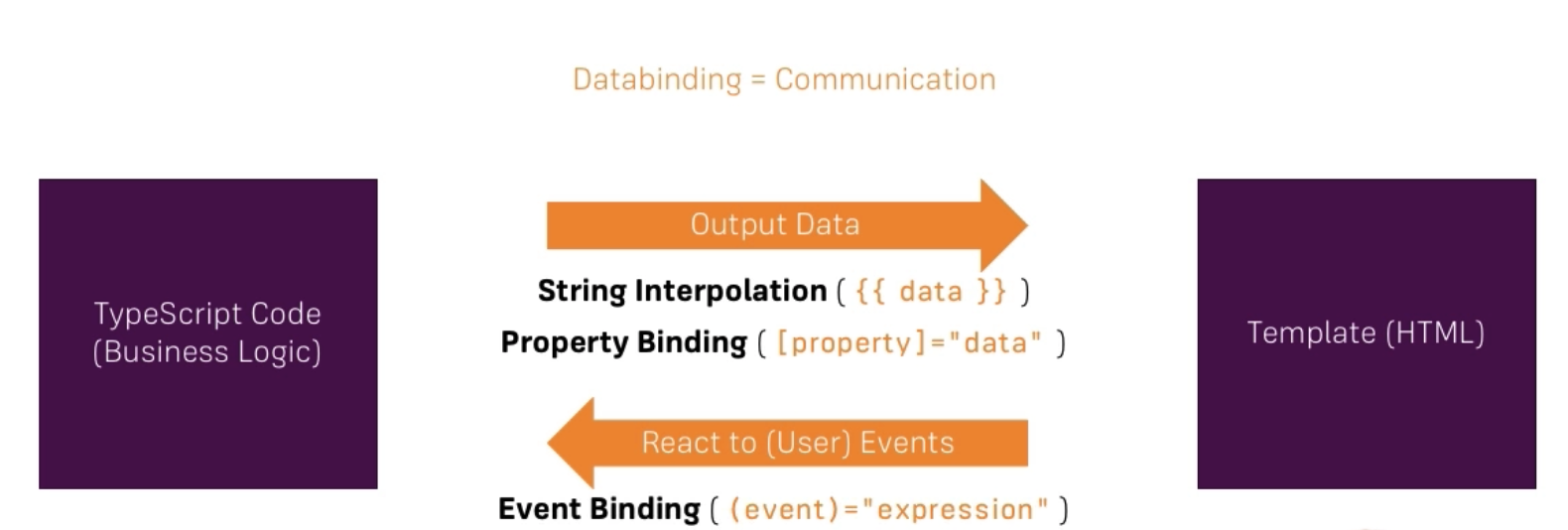
**Databinding** – communication between typescript code and template(html)

**Output Data**:

* string interpolation ( {{ data }} )
* property binding ( [property] = ”data” )

React to (user) Events **(Event binding)**

* (event) = “expression”





**String interpolation:**

You can use:

* {{ ‘server’ }} – normal string
* {{ serverId }} – data binding – defined in component.ts
* Has to resolve string in the end but number can be easily converted to a string so you can use number in {{ }}
* Methods

You cannot use:

* Multiple line expression
* For, if, …

**Property binding:**

**[disabled] =** we want to dynamically bind some property

**Why use angular?**

It is easy to interact with DOM to change something in runtime

**Property binding vs string interpolation:**

If you want to output something in your template print the test – string interpolation

If you want to change some property – property binding

**Event binding**

(click) = “method\_from\_component.ts()”

How do you know to which Properties or Events of HTML Elements you may bind? You can basically bind to all Properties and Events - a good idea is to console.log() the element you're interested in to see which properties and events it offers.

Important: For events, you don't bind to onclick but only to click (=> (click)).

The MDN (Mozilla Developer Network) offers nice lists of all properties and events of the element you're interested in. Googling for YOUR\_ELEMENT properties or YOUR\_ELEMENT events should yield nice results.

onUpdateServerName($event):

* $ = Data emitted with that event, capture the data

**Two way data binding:**

Important: For Two-Way-Binding (covered in the next lecture) to work, you need to enable the ngModel directive. This is done by adding the FormsModule to the imports[] array in the AppModule.

You then also need to add the import from @angular/forms in the app.module.ts file:

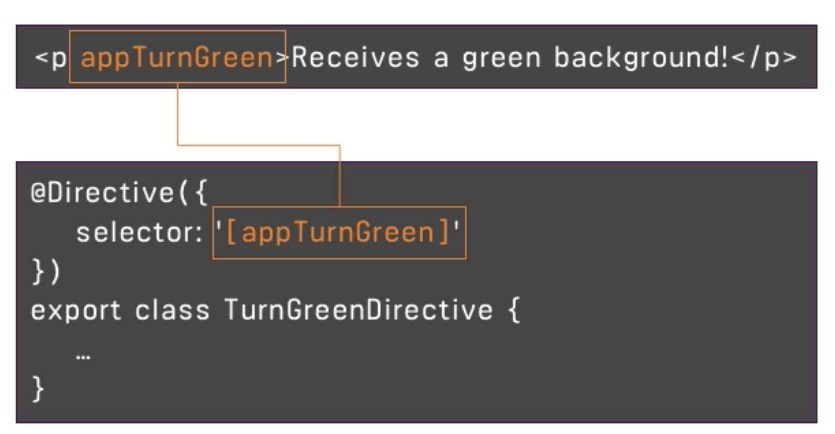
import { FormsModule } from '@angular/forms';

ngModel = directive

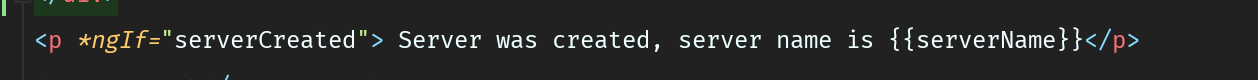
<input [(ngModel)] = “serverName”>

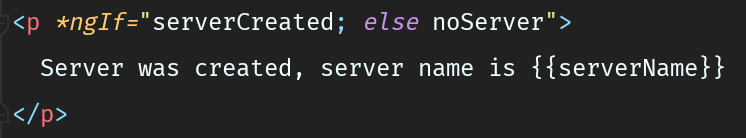
**Directives**

* Directives are instructions in the DOM
* Example: *<p appTurnGreen> Receives a green background! </p>*

**

**ngIf Output data conditionally** (ngIf is directive)

****



Styling element dynamically with **ngStyle**

***Unlike structural directives, attribute directives don’t add or remove elements. They only change the element they were placed on.***

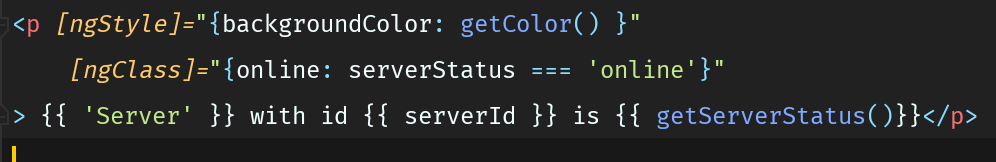


* We are binding to a property of the directive

You can use:

* <p [ngStyle]=”{ backgroundColor: red }”>
* <p [ngStyle]=”{ ‘background-color’: red”>

Applying CSS classes dynamically with ngClass



styles: [`

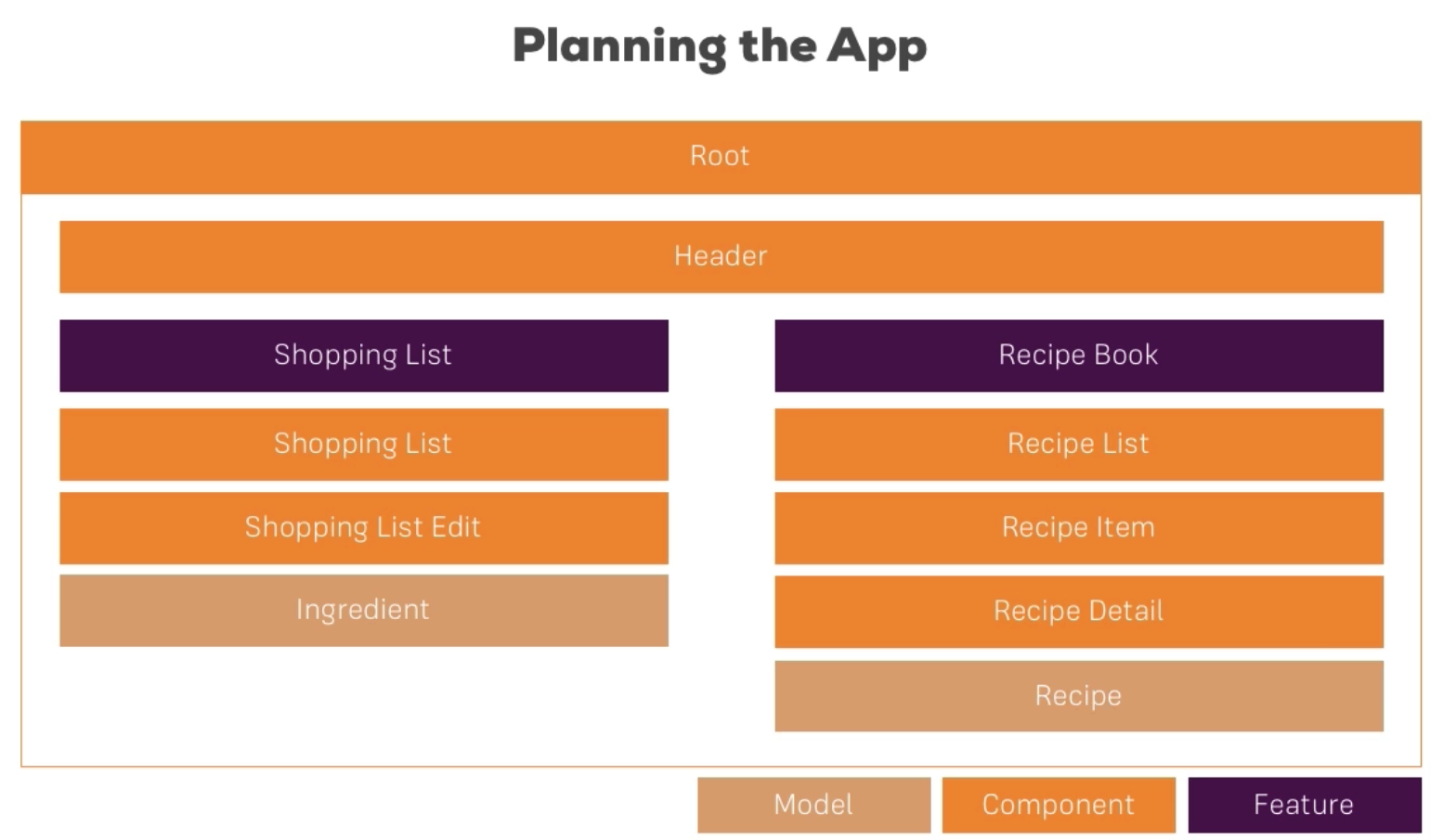
.online{

color: white;}`]}

How to make toggling paragraph - is displayed or not

<button  
 *class=*"btn btn-primary"  
 *(click)=*"displayed = !displayed"  
>  
 Display details  
</button>  
<p *\*ngIf=*"displayed"> Secret password = tuna</p>

Project:

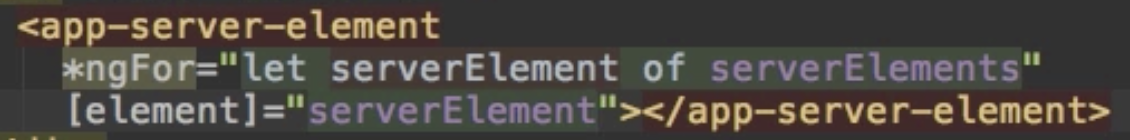


You can use:

src="{{recipe.imagePath}}" or [src]="recipe.imagePath"

\*ngFor=”let recipe of recipes”

**All properties of components are by default only accessible inside component not from outside. You have to be explicit which properties you want to expose**. You need to add decorator **@Input()**



Output nastavuješ vlastný event

A input keď si nastavuješ vlastnú property

How to pass data from a component down to another component which was implemented there - @Input()

* Make property bindable from outside from the parent component using this component

Something change in some component and we want to inform parent component. @Output()

* Allows parent component using this component to listen to events which your created through event emitter

**Services**

* solution in specific usecases – communication between components

View encapsulation:

* after adding style to element

Shadow DOM

* not supported by all browsers
* each element has own shadow DOM

@Component({  
 encapsulation: ViewEncapsulation.Emulated   
})

ViewEncapsulation can have emulated – by default, None- vypne sa zobrazovanie atributov pridavanych angularom -shadow DOM, and Native – uses shadow DOM technology, same result as emulated

**View Encapsulation Types**

Angular comes with view encapsulation built in, which enables us to use Shadow DOM or even emulate it. There are three view encapsulation types:

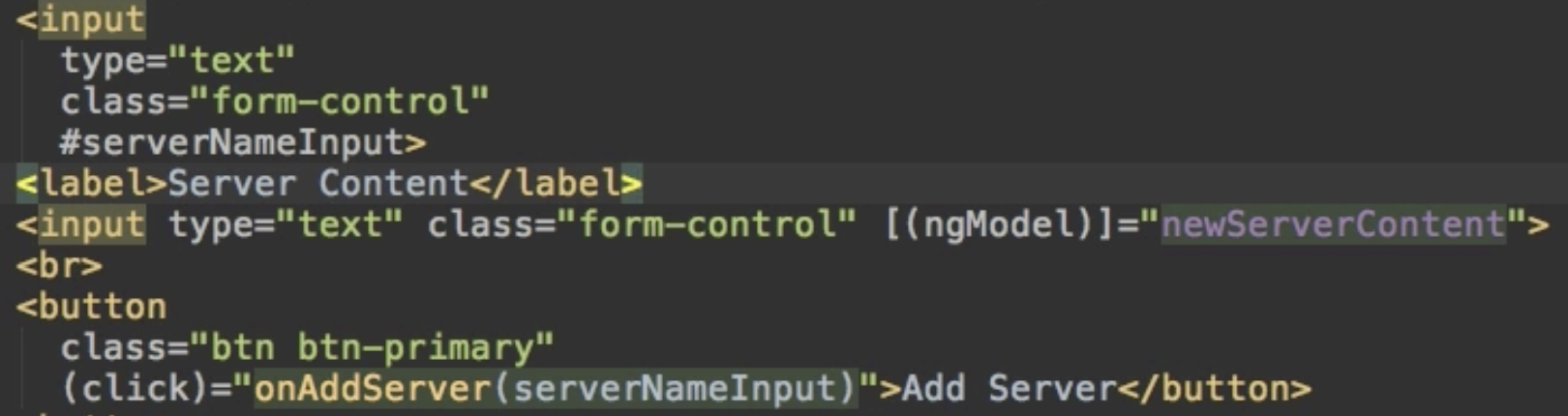
ViewEncapsulation.None - No Shadow DOM at all. Therefore, also no style encapsulation.

ViewEncapsulation.Emulated - No Shadow DOM but style encapsulation emulation.

ViewEncapsulation.Native - Native Shadow DOM with all it’s goodness.

Using local referencies in templates:

* Local reference can be places on any html element not only input element
* - #servername – reference of element
* You can use only in this template



Getting access to the Template & DOM with @ViewChild:

* You have to add local reference #serverConentInput in html template
* Then: @ViewChild(‘serverConentInput’) serverContentInput: ElementRef in component
* And you can use it by: this.serverContentInput.nativeElement.value
* You should not change element like this

ngContent:

* <ng-content>
* Check content of element

**Component Lifecycle**

* **ngOnInit** – lifecycle hook – called one when the component is initialized , run after the constructor, when object is created
* **ngOnChanges** – called after a bound input property changes – everytime when something chang, called before ngoninit
  + **Changes**: firstChange if it is after load page, previousValue
* **ngDoCheck** – called with each event – so not only after changes but also after some click, called during every change detection run, something change inside component, change something in template, value for example, every check that angular do
* **ngAfterContentInit** – Called after content – ng-content – has been projected into view, when view of parent component is initiliazed
* **ngAfterContentChecked –** called every time the projected content has been checked, change detection
* **ngAfterViewInit –** when view our own component has been finished initializing, view was rendered, is called after ngOnInit and ngDoCheck, and is called only once
* **ngAfterViewChecked –** Called every time the view (and child views) have been checked once we are sure all changes were done or non changes were detect by angular, after docheck- after each change detection cycle
* **ngOnDestroy –** Called once the component is about to be destroyed

**ContentChild**

# Navigation with ngIf

**app.component.html:**

<app-recipes \*ngIf="loadedFeature === 'recipe'" ></app-recipes>

<app-shopping-list \*ngIf="loadedFeature === 'shopping-list'"></app-shopping-list>

**app.component.ts**

loadedFeature = 'recipe';

onNavigate(feature: string) {

this.loadedFeature = feature;

}

**header.component.ts**

@Output() featureSelected = new EventEmitter<string>();

onSelect(feature: string) {

this.featureSelected.emit(feature);

}

**header.component.html**

<li><a href="#" (click)="onSelect('recipe')">Recipes</a></li>

<li><a href="#" (click)="onSelect('shopping-list')">Shopping list</a></li>

# Understanding Directives

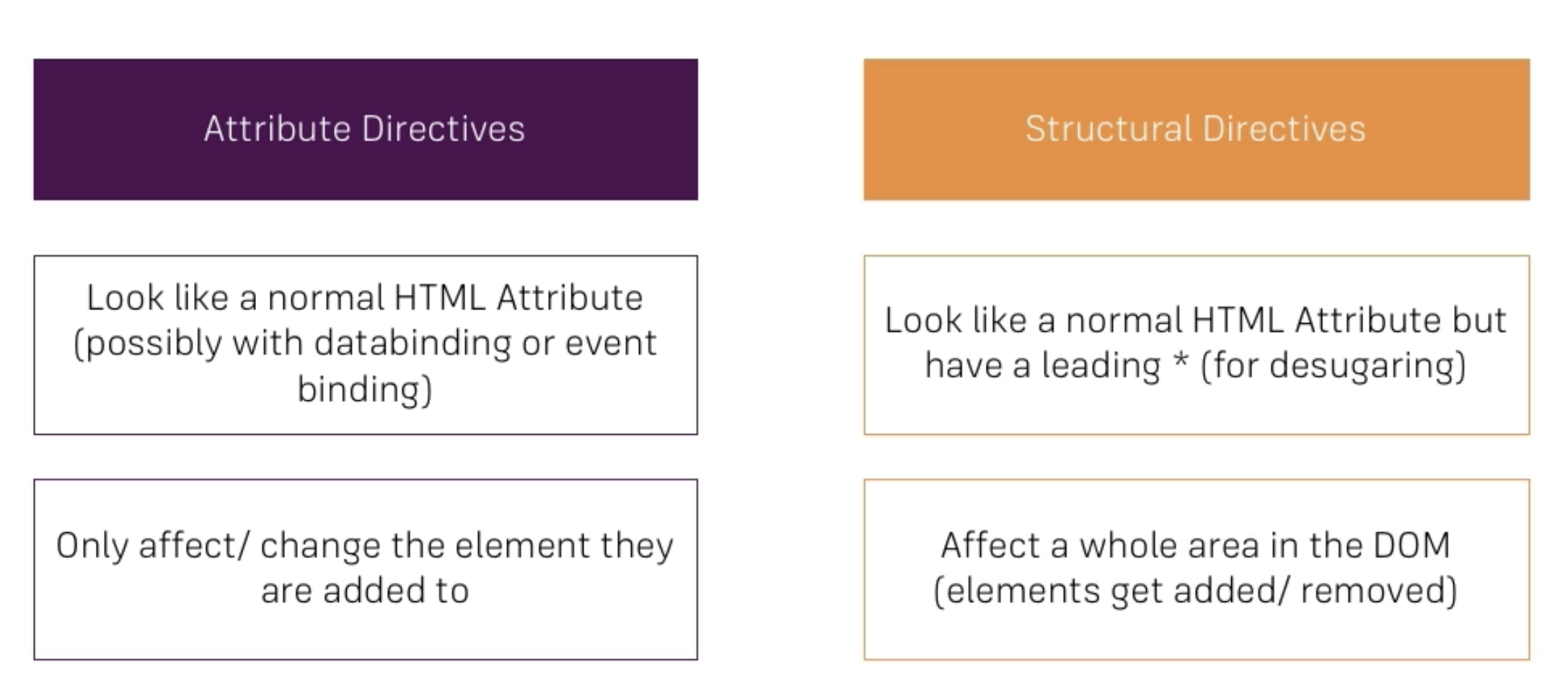
**Attribute and structural directives**

**Attribute:** are called like this because they sit on element just like attributes

* ngStyle: [ngStyle]="{backgroundColor: odd % 2 !== 0 ? 'yellow' : 'transparent'}"
* ngClass: [ngClass]="{odd: odd % 2 !== 0}"

**Structural**: they do the same but they also change structure of DOM around this element

* We can’t have more than one structural directive on the same element
* \*ngIf & ngFor



**YOUR OWN ATTRIBUTE DIRECTIVE:**

1. Create file basic-highlight.directive.ts
2. @Directive({

selector: '[appBasicHighlight]' –now you can use it without brackets

})

1. Define it

constructor(private elementRef: ElementRef) {}

ngOnInit() {

this.elementRef.nativeElement.style.backgroundColor = 'green';

}

1. Add it to app.module.ts > declarations: BasicHighlightDirective
2. Use it

<p appBasicHighlight>

Style me with basic directive!

</p>

1. Result:

Using the renderer to build a better attribute directive:

1. Crete file .directive.ts
2. @Directive({

selector: '[appBetterHighlight]'

})

export class BetterHighlightDirective implements OnInit {

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

ngOnInit() {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

}

**Host listener > zmeni pozadie pri mosueover**

@HostListener('mouseenter') mouseover(eventdata: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'blue');

}

Event

@HostListener('mouseleave') mouseleave(eventdata: Event) {

this.renderer.setStyle(this.elRef.nativeElement, 'background-color', 'transparent');

}

**To iste pomocou direktivy @Hostbinding**

@HostBinding(‘style.backgroundColor’)

Property of hosting element subproperty

@HostBinding('style.backgroundColor') backgroundColor = 'transparent';

constructor(private elRef: ElementRef, private renderer: Renderer2) { }

@HostListener('mouseenter') mouseover(eventdata: Event) {

this.backgroundColor = 'blue';

}

@HostListener('mouseleave') mouseleave(eventdata: Event) {

this.backgroundColor = 'transparent';

}