Directives in Angular

- In Angular, directives are classes that add functionality to elements in an application.
- They are markers in the Document Object Model (DOM) that tell the HTML compiler to attach a behavior to a DOM element.
- Directives can be used with any controller or HTML tag.

There are three main types of directives in Angular:

- **Component directives**

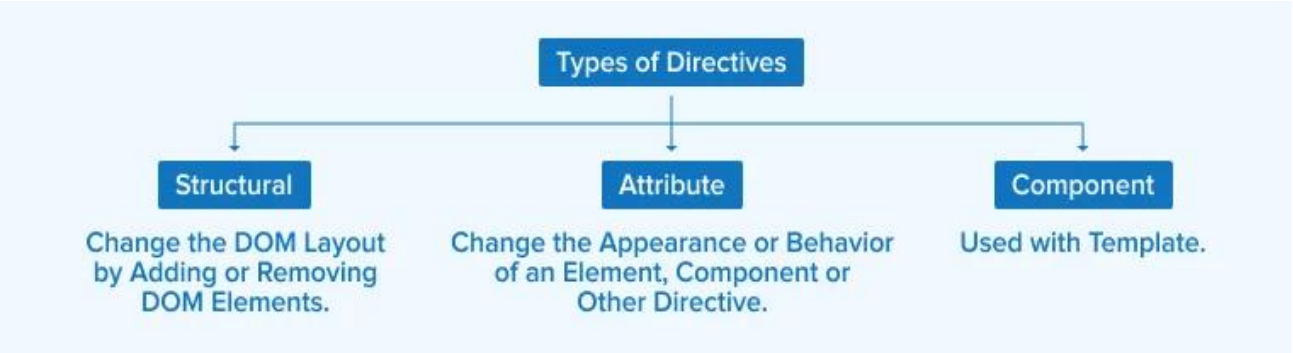
These directives have templates and are used to create reusable UI components.

- **Structural directives**

These directives modify the DOM structure by adding or removing elements. Examples include ngIf, ngFor, and ngSwitch.

- **Attribute directives**

These directives change the appearance or behavior of an element, component, or another directive. Examples include ngClass, ngStyle, and ngModel.



Structural Directives

Structural directive changes the structure of the DOM, adding or removing elements based on conditions.

1. ngIf
2. ngFor
3. ngSwitch

Here is the example for Nglf.

```
<div *ngif=""product"">{{product.name}}</div>

<div template=""ngIf" product"="">{{product.name}}</div>

<ng-template [ngif]=""product"">
```

```
<div>{{product.name}}</div>
</ng-template>
```

Example for NgFor.

```
<div *ngfor="let product of products">{{product.name}}</div>
<div template="ngFor let product of products">{{product.name}}</div>
<ng-template ngfor="" let-product="" [ngfor]="products">{{product.name}}
</ng-template>
```

And example for NgSwitch.

```
<div class="row" [ngswitch]="selectedValue">
   <div *ngswitchcase="'One'">One is Pressed</div>
   <div *ngswitchcase="'Two'">Two is Selected</div>
   <div *ngswitchdefault="">Default Option</div>
</div>
```

2. Attribute Directives

1) Attribute directives in Angular are a type of directive that can change the appearance or behaviour of an element, component, or another directive. They are applied to HTML elements as attributes and are used to enhance the functionality of these elements.

2) These directives are commonly used to dynamically modify the style, directive class, or behaviour of an element based on certain conditions. For example, the `ngClass` directive allows you to add or remove CSS classes based on the evaluation of an expression. Similarly, the `ngStyle` directive lets you set inline styles for a host element based on the evaluation of an expression.

ngClass

```
<div [ngClass]="{'text-success': isSuccess,'text-danger': !isSuccess}"> Text </div>
export class AppComponent { isSuccess: boolean = true; }
```

ngStyle

```
<div [ngStyle]="{ 'background-color': isSuccess ? 'yellow' : 'green', 'font-size': isSuccess ? '24px' : '20px' }"> This div's background color and font size change based on the value of isSuccess. </div>
```

4. Custom Directives

Custom Directives can also be created to add specific behaviors to elements, enhancing the functionality of an Angular application.

4. Custom Directives

i) Custom directives in Angular are directives that developers can create according to their requirements. These directives can either change the appearance or behaviour of elements in your web app.

ii) For example, you could create a custom directive called `myHighlight` that makes text change colour when you hover over it. This would be an attribute selector directive because it changes how an element looks.

iii) These custom directives help keep your code organised and easier to understand by encapsulating complex logic into reusable chunks.

ng generate directive my-highlight

This will create a new directive file my-highlight.directive.ts and add it to the declarations array in your module file (app.module.ts).

my-highlight.directive.ts

```
import { Directive, ElementRef, HostListener } from '@angular/core';

@Directive({
  selector: '[appMyHighlight]'
})
export class MyHighlightDirective {

  constructor(private el: ElementRef) { }

  @HostListener('mouseenter') onMouseEnter() {
    this.highlight('yellow');
  }

  @HostListener('mouseleave') onMouseLeave() {
    this.highlight(null);
  }

  private highlight(color: string | null) {
    this.el.nativeElement.style.backgroundColor = color;
```

```
  }
}
```

app.component.ts

```
@Component({
selector: 'app-root',
template: `
                    <p appMyHighlight>
                       Hover over this text to see it highlighted.
                    </p>
`,
Import:[ MyHighlightDirective]
})

export class AppComponent { }
```

note :do not forget to import directive class name