

# ANGULAR MEETUP #3

Angular Framework verstehen


Alexander Gräfenstein

# ANGULAR

## 19 Dekoratoren

 Attribute

 ContentChildren

 HostBinding

 Injectable

 Optional

 Self


 ViewChildren

 Component

 Directive

 HostListener

 Input

 Output

 SkipSelf

 ContentChild

 Host

 Inject

 NgModule

 Pipe

 ViewChild

# WAS IST EIN DECORATOR?

- ein Typescript Feature (noch)
- eine Funktion
- reichert Zielobjekt an

```
function myDecorator(target) {  
    // do something with target  
}
```

# Das können wir doch schon in JS

```
let components = []
```

```
function registerComponent(target) {  
  components.push(target)  
}
```

```
class MyClass {  
  // 1.000.000 Zeilen Code  
}
```

```
registerComponent(MyClass)
```

# Schreibweise in Typescript

```
let components = []
```

```
function registerComponent(target) {  
    components.push(target)  
}
```

```
@registerComponent
```

```
class MyClass {  
    // 1.000.000 Zeilen Code  
}
```

Dekoratoren können angewendet werden auf

## Klassen

```
@Component  
class MyClass { /* ... */ }
```

## Methoden, getter/setter u. Eigenschaften

```
class MyClass {  
    @foo  
    public myMember  
}
```

## Parameter

```
class MyClass {  
    constructor(@foo myArg: number) { /* ... */ }  
}
```

DEMO

# ANGULAR DEKORATOREN

## Bausteine

@Pipe  
@Directive  
@Component  
@Injectable

## Legokiste

@NgModule

## Data binding

@Input  
@Output

## Host binding

@HostBinding  
@HostListener

## Content binding

@ContentChild  
@ContentChildren  
@ViewChild  
@ViewChildren

## DI

@Inject  
@Host  
@Attribute  
@Optional  
@Self  
@SkipSelf

# Bausteine / Legokiste

- @Pipe
- @Directive
- @Component
- @Injectable
- @NgModule



## @Pipe

Transformiert einen Wert im Template,  
bevor dieser ausgegeben wird

```
<div> Geschrieben am {{ myDate | date:'d.MM.YYYY' }} </div>
```

```
<ul>
  <li *ngFor="let item of items | sortBy:'name'">
    {{ item.name }}
  </li>
</ul>
```

```
@Pipe({ name: 'sortBy' })
class SortByPipe {
```

```
  public transform(input: any[], byField: string) {
    return input.sort(/* ... */)
  }
```

```
}
```

## DEMO

# @Directive

gibt einem HTML Element ein 'Verhalten'

```
<div trackClicks>Click mich</div>
```

```
@Directive({ selector: '[trackClicks]' })  
class ClickTracker {
```

```
    public counter = 0  
  
    @HostListener('click')  
    public track() {  
        this.counter++  
    }  
}
```

```
}
```

## DEMO

# @Component

ist im Grunde eine Direktive

- beschränkt auf 1 Komponente pro HTML Element
- mit Template und Style

```
@Component({  
  selector: 'my-component',  
  template: "Hello Component"  
})  
class MyComponent {  
  
}
```

```
<my-component></my-component>
```

```
<my-component>Hello Component</my-component>
```

## DEMO

# @Injectable

- Serviceklassen
- ohne View
- ohne HTML Element

```
@Injectable()  
class TrackService {
```

```
    public clicks = 0  
  
    public track() {  
        this.clicks++  
    }  
}
```

```
}
```

```
class MyComponent {  
    constructor(service: TrackService) {  
  
    }  
}
```

## DEMO



# @NgModule

bündelt die Bausteine zu einem Modul zusammen

```
@NgModule({  
  declarations: [ /* eigene @Pipe, @Directive, @Component */ ],  
  imports: [ /* fremde @NgModule */ ],  
  exports: [ /* fremde @NgModule, eigene declarations */ ],  
  providers: [ /* eigene @Injectable */ ],  
})  
class MyModule {}
```

- Declarables müssen deklariert werden
- Injectables können weggelassen werden

DEMO

# Data binding

- @Input
- @Output

# @Input

ist der Kanal von Außen nach Innen

```
@Component({
  selector: "my-component",
  template: "",
})
class MyComponent {
  @Input()
  public isActive: boolean
}
```

```
<my-component [isActive]="true"></my-component>
```

!!! [ ] !!!

```
<my-component isActive="true"></my-component>
```



# @Output

ist der Kanal von Innen nach Außen

```
@Component({  
  selector: "my-component",  
  template: "",  
})  
class MyComponent {
```

```
  @Output()  
  public changed = new EventEmitter()
```

```
  public notify() {  
    this.changed.emit("some value")  
  }
```

```
}
```

```
<my-component (changed)="onChanged($event)"></my-component>
```

# @Input / @Output und die Bananenbox

```
<my-component [(isActive)]="value"></my-component>
```

```
@Component({ /* ... */ })  
class MyComponent {  
    private active: boolean
```

```
    @Output()  
    public isActiveChange = new EventEmitter<boolean>()
```

```
    @Input()  
    public get isActive() { return this.active }  
    public set isActive(v: boolean) {
```

```
        if (v !== this.active) {  
            this.isActiveChange.emit(v)  
        }
```

```
        this.active = v  
    }
```

```
}
```

# Host binding

- `@HostBinding`
- `@HostListener`

# @HostBinding

setzt HTML Attribute am Host Element

```
@Directive({ selector: "img[size]" })  
class MyImgDirective {
```

```
    @Input()  
    public size: string
```

```
    @HostBinding("attr.width")  
    @HostBinding("attr.height")  
    public get sizeInPx() {  
        return this.size === "small" ? 64 : 128  
    }
```

```
}
```

```

```

```

```

# @HostBinding

setzt auch CSS Klassen

```
@Directive({ selector: "[myLayout]" })  
class MyLayoutDirective {
```

```
    @Input()  
    public myLayout: string
```

```
    @HostBinding("class.layout-vertical")  
    public get isVertical() {  
        return this.myLayout === "vertical"  
    }
```

```
    @HostBinding("class.layout-horizontal")  
    public get isHorizontal() {  
        return this.myLayout === "horizontal"  
    }
```

```
}
```

# @HostListener

reagiert auf DOM Events

```
@Directive({ selector: "[myDirective]" })  
class MyDirective {
```

```
    @HostListener("click")  
    public onClick() {  
        // ...  
    }
```

```
    @HostListener("mousemove", ["$event"])  
    public onMouseMove(e: MouseEvent) {  
        // ...  
    }
```

```
    @HostListener("document:click")  
    public onDocumentClick() {  
        // ...  
    }
```

```
}
```

# Content binding

- `@ContentChild`
- `@ContentChildren`
- `@ViewChild`
- `@ViewChildren`

@ViewChild / -ren

selektiert Elemente

- im Content der Direktive
- im ng-content der Komponente

@ViewChild / -ren

selektiert Elemente

- im Template der Komponente

DEMO



# DI

- @Optional
- @SkipSelf
- @Self
- @Host
- @Attribute
- @Inject

## @Optional

wie der Name schon sagt, ist optional

```
constructor(@Optional() service: MyService) {}
```

## @Self

holt einen Service nur aus dem lokalen Injector

```
@Component({  
    // ...  
    providers: [MyService]  
})  
class MyComponent {  
    constructor(@Self() service: MyService) {}  
}
```

## @SkipSelf

überspringt den lokalen Injector

```
@Component({  
    // ...  
    providers: [MyService]  
})  
class MyComponent {  
    constructor(@SkipSelf() service: MyService) {}  
}
```

@Host

sucht nur bis zum Injector vom Host des Templates

a visual guide

## @Attribute

übergibt den Wert eines Attributs vom HTML Element

```
<input type="email" />
```

```
@Directive({  
  selector: 'input'  
})  
class MyInputDirective {  
  constructor(@Attribute() type: string) {  
  
  }  
}
```

ja, das ginge auch mit @Input

# @Inject

bestimmt den Lookup Key

```
constructor(service: MyService) {}
```

ist gleichbedeutend mit

```
constructor(@Inject(MyService) service) {}
```

```
import { LOCALE_ID } from "@angular/core"  
// ...  
constructor(@Inject(LOCALE_ID) language) {}
```

```
@NgModule({  
  providers: [{ provide: LOCALE_ID, useValue: "de" }],  
})
```

ohne Demo, ist eh schon spät

Danke  
viel Erfolg



