ARS

Motivation

- Komplexe Funktionalität innerhalb einer Komponente gruppieren
- Wiederkehrende Logik und HTML-Markup externalisieren
- Zusammengehörige Logik und HTML-Markup zu einer Einheit kapseln
- Abhängigkeiten / Hierarchien von Komponenten abbilden

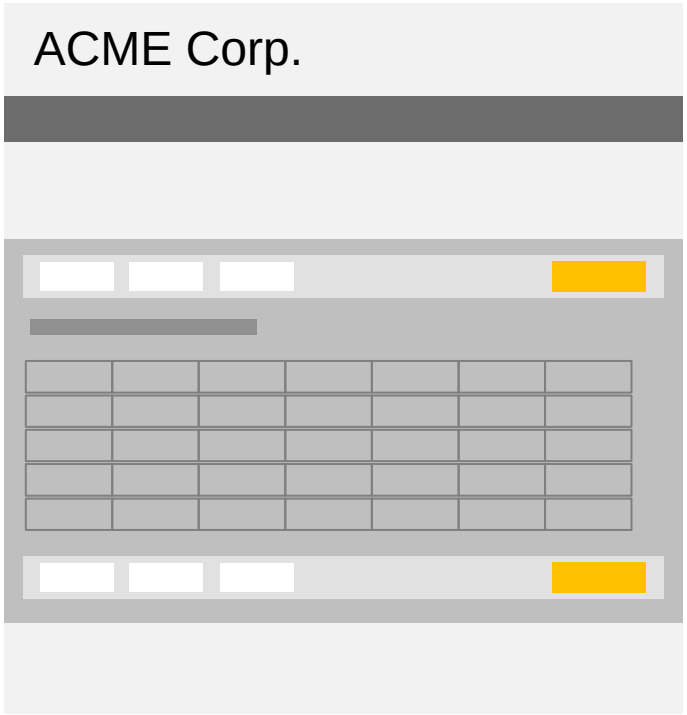
ARS

Lernziele

- Sie beschreiben Möglichkeiten zur Kapselung von Funktionalität.
- Sie erklären die Zusammenhänge zwischen Eltern- und Kind-Komponenten.
- Sie erläutern den wesentlichen Unterschied zwischen Input und Output Property.

ARS

Das Problem

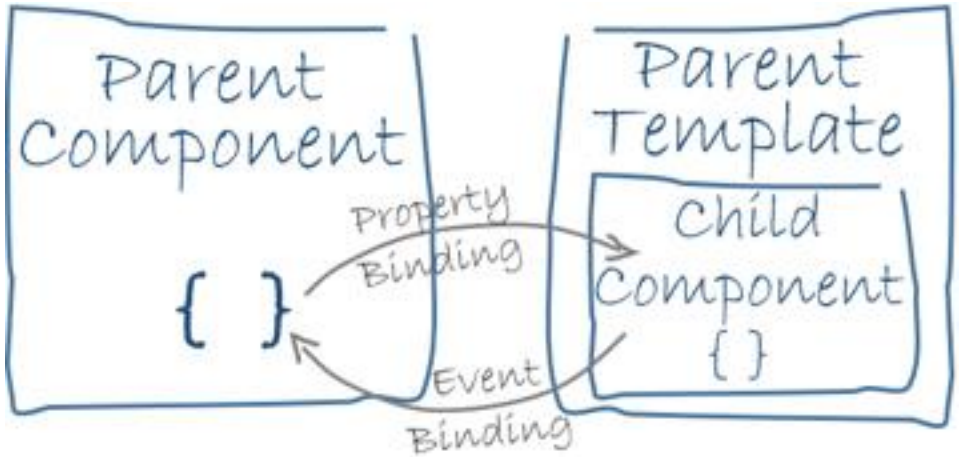


- Komplexes Feature
- Gleiche Logik und gleiches HTML-Markup an mehreren separaten Stellen!
- Mögliche Probleme:
  - Schlechte Lesbarkeit
  - Geringe Wiederverwendbarkeit (Copy-Paste Gefahr)
  - Erhöhter Test- und Wartungsaufwand
- **Lösung**
  Aufteilung auf Kind-Komponenten
- **Neues Problem**
  Datenweitergabe an Kind-Komponente?

ARS

Arten der Kommunikation

- Zwei Kommunikationskanäle existieren:
  - Daten werden von Eltern an Kinder übergeben (Property-Binding)
    - Input Property
    - @Input-Decorator
  - Eltern warten auf Events von Kindern (Event-Binding)
    - Output Property
    - EventEmitter
    - @Output-Decorator
- Vergleichbar mit Kommunikation zwischen Template und Komponentenklasse



ARS

Daten von Eltern an Kind (Property-Binding)

- Übergabe vergleichbar mit der Bindung eines HTML-Attributes
  → Entspricht Property-Binding
- Übergebene Daten werden mit @Input-Decorator markiert
- Notation: Eckige-Klammer-Notation „[]“ im HTML-Markup

ARS

Daten von Eltern an Kind (Property-Binding)

Eltern-Komponente

```
@Component({
   selector: 'hero-parent',
   template: `
     <h2>
       {{master}} controls
       {{availableHeroes.length}} heroes
     </h2>
     <hero-child
       *ngFor="let someHero of allHeroes"
       [hero] ="someHero">
     </hero-child>`
})
export class HeroParentComponent {
   allHeroes = [...];
   master = 'Master';
}
```

Kind-Komponente

```
@Component({
  selector: 'hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>`
})
export class HeroChildComponent {
   @Input() hero: Hero;
}
```

ARS

Event von Kind an Eltern (Event-Binding)

- EventEmitter sendet Events an Empfänger
- Empfangen von Daten vergleichbar mit einem HTML-Event
  → Entspricht Event-Binding
- Ausgehende Objektvariable wird mit @Output-Decorator markiert
- Notation: Runde-Klammer-Notation „()“ im HTML-Markup
- Übergebene Funktion fungiert als Empfänger von Events

ARS

Event von Kind an Eltern (Event-Binding)

Eltern-Komponente

```
<h2>Do you smoke?</h2>
<h3>Yes: {{yes}}, No: {{no}}</h3>

<vote (voted)="onVoted($event)">
</vote>
```

```
@Component({selector: 'poll',
  templateUrl: 'poll.component.html'
})
export class VoteTakerComponent {
  yes = 0;
  no = 0;

  onVoted(agreed: boolean) {
    agreed ? this.yes++ : this.no++;
  }
}
```

Kind-Komponente

```
<button (click)="vote(true)">Yes</button>
<button (click)="vote(false)">No</button>
```

```
@Component({
 selector: 'vote',
 template: 'vote.component.html'

})
export class VoterComponent {
  @Output() voted = new EventEmitter<boolean>();

  vote(agreed: boolean) {
    this.voted.emit(agreed);
  }
}
```

ARS

Zusammenfassung

Um was ging es in diesem Modul?

- Aufteilung von komplexen Komponenten
- Auflösen von Redundanz in Logik und HTML-Markup
- Kommunikation zwischen Eltern- und Kind-Komponenten

Wozu brauche ich das? Was will ich damit machen?

- Funktionalität und Zuständigkeiten gruppieren und kapseln
- Redundanz vermeiden
- Wartbarkeit und Wiederverwendbarkeit erhöhen

ARS

Kontrollfragen

- Welche Ansätze gibt es, um innerhalb einer Komponente Redundanzen in der Logik und im HTML-Markup zu vermeiden?
- Worin liegt der wesentliche Unterschied zwischen Input und Output Properties?
- Welche Schritte sind notwendig, um Daten von der Eltern-Komponente an die Kind-Komponente zu übergeben?
- Über welche Schritte lassen sich Daten über Events von der Kind-Komponente an die Eltern-Komponente übertragen?