Angular Components and Services

Introduction to Angular Components and Angular Services

Introduction

- Components are the **main building block** for Angular applications.
- Each component consists of:
  - An **HTML template** that declares what renders on the page.
  - A **Typescript class** that defines behavior.
  - A **CSS selector** that defines how the component is used in a template.
  - Optionally, **CSS styles** applied to the template.

Create a new component

```
ng generate component <component-name>
```

Specifying a component's CSS selector

- Every component requires a **CSS selector**.
- Specify a component's selector by adding a selector statement to the @Component decorator.

```
@Component({
  selector: 'app-my-component',
})
```

Defining a component's template

- A template is a block of HTML that tells Angular how to render the component in your application.
- To define a template as an external file, add a templateUrl property to the @Component decorator.

```
@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html',
})
```

- To define a template within the component, add a template property to the @Component decorator that contains the HTML you want to use.

```
@Component({
  selector: 'app-my-component',
  template: '<h1>Hello World!</h1>',
})
```

Declaring a component's styles

- To declare the styles for a component in a separate file, add a styleUrls property to the @Component decorator.

```
@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html',
  styleUrls: ['./my-component.component.css']
})
```

- To declare the styles within the component, add a styles property to the @Component decorator that contains the styles you want to use.

```
@Component({
  selector: 'my-component-overview',
  template: '<h1>Hello World!</h1>',
  styles: ['h1 { font-weight: normal; }']
})
```

Pass data from parent to child with input binding

- For details, see Sharing data between child and parent directives and components

`hero-parent.component.ts`

```
import { Component } from '@angular/core';

import { HEROES } from './hero';

@Component({
  selector: 'app-hero-parent',
  template: `
    <h2>{{master}} controls {{heroes.length}} heroes</h2>

    <app-hero-child
      *ngFor="let hero of heroes"
      [hero]="hero"
      [master]="master">
    </app-hero-child>
  `
})
export class HeroParentComponent {
  heroes = HEROES;
  master = 'Master';
}
```

Pass data from parent to child with input binding (cont.)

`hero-child.component.ts`

```
import { Component, Input } from '@angular/core';

import { Hero } from './hero';

@Component({
  selector: 'app-hero-child',
  template: `
    <h3>{{hero.name}} says:</h3>
    <p>I, {{hero.name}}, am at your service, {{masterName}}.</p>
  `
})
export class HeroChildComponent {
  @Input() hero!: Hero;
  @Input('master') masterName = ''; // tslint:disable-line: no-input-rename
}
```

Intercept input property changes with ngOnChanges()

- Detect and act upon changes to input property values with the ngOnChanges() method of the OnChanges lifecycle hook interface.

version-child.component.ts

```
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-version-child',
  template: `
    <h3>Version {{major}}.{{minor}}</h3>
    <h4>Change log:</h4>
    <ul>
      <li *ngFor="let change of changeLog">{{change}}</li>
    </ul>
  `
})
export class VersionChildComponent implements OnChanges {
  @Input() major = 0;
  @Input() minor = 0;
  changeLog: string[] = [];

  ngOnChanges(changes: SimpleChanges) {
    const log: string[] = [];
    for (const propName in changes) {
      const changedProp = changes[propName];
      const to = JSON.stringify(changedProp.currentValue);
      if (changedProp.isFirstChange()) {
        log.push(`Initial value of ${propName} set to ${to}`);
      } else {
        const from = JSON.stringify(changedProp.previousValue);
        log.push(`${propName} changed from ${from} to ${to}`);
      }
    }
    this.changeLog.push(log.join(', '));
  }
}
```

Intercept input property changes with ngOnChanges() (cont.)

- The VersionParentComponent supplies the minor and major values and binds buttons to methods that change them.

`version-parent.component.ts`

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-version-parent',
  template: `
    <h2>Source code version</h2>
    <button (click)="newMinor()">New minor version</button>
    <button (click)="newMajor()">New major version</button>
    <app-version-child [major]="major" [minor]="minor"></app-version-child>
  `
})
export class VersionParentComponent {
  major = 1;
  minor = 23;

  newMinor() {
    this.minor++;
  }

  newMajor() {
    this.major++;
    this.minor = 0;
  }
}
```

Parent listens for child event

- The child component exposes an **EventEmitter** property with which it emits events when something happens.
- The parent binds to that event property and reacts to those events.

`voter.component.ts`

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
  selector: 'app-voter',
  template: `
    <h4>{{name}}</h4>
    <button (click)="vote(true)"  [disabled]="didVote">Agree</button>
    <button (click)="vote(false)" [disabled]="didVote">Disagree</button>
  `
})
export class VoterComponent {
  @Input()  name = '';
  @Output() voted = new EventEmitter<boolean>();
  didVote = false;

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

Parent listens for child event (cont.)

- The parent VoteTakerComponent binds an event handler called onVoted() that responds to the child event payload $event and updates a counter.

`votetaker.component.ts`

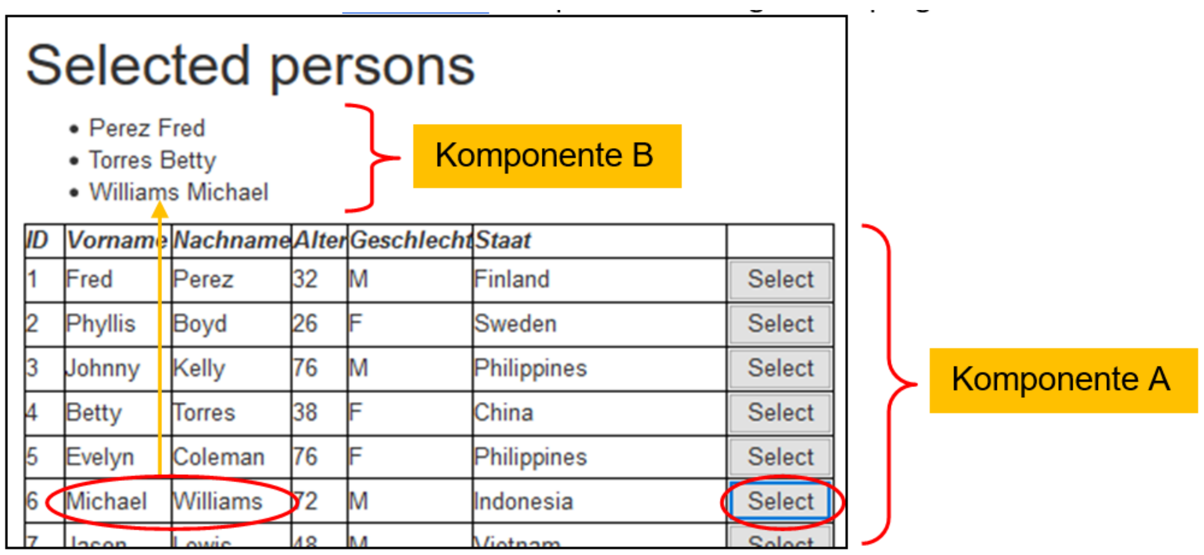
```
import { Component } from '@angular/core';

@Component({
  selector: 'app-vote-taker',
  template: `
    <h2>Should mankind colonize the Universe?</h2>
    <h3>Agree: {{agreed}}, Disagree: {{disagreed}}</h3>

    <app-voter
      *ngFor="let voter of voters"
      [name]="voter"
      (voted)="onVoted($event)">
    </app-voter>
  `
})
export class VoteTakerComponent {
  agreed = 0;
  disagreed = 0;
  voters = ['Narco', 'Celeritas', 'Bombasto'];

  onVoted(agreed: boolean) {
    agreed ? this.agreed++ : this.disagreed++;
  }
}
```

Parent and children communicate using a service



Selected persons

- Perez Fred
- Torres Betty
- Williams Michael

ID	Vorname	Nachname	Alter	Geschlecht	Staat	
1	Fred	Perez	32	M	Finland	Select
2	Phyllis	Boyd	26	F	Sweden	Select
3	Johnny	Kelly	76	M	Philippines	Select
4	Betty	Torres	38	F	China	Select
5	Evelyn	Coleman	76	F	Philippines	Select
6	Michael	Williams	72	M	Indonesia	Select
7	Jason	Lewis	48	M	Vietnam	Select

Create a new service

```
ng generate service <service-name>
```

- This service (which is a **singleton**) can be used in every component via **depency injection**.
- In the service, us a **Subject** to allow **observers** to register to an **observable**.

```
...
export class NotifierService {

  private stringRepository = new Subject<string>();

  public notify(msg: string): void {
    this.stringRepository.next(msg);
  }

  public listen(): Observable<string> {
    return this.stringRepository.asObservable();
  }
}
```

Sender

```
...
import { NotifierService } from '.notifier.service';

...
export class AppComponent implements OnInit {

  constructor(private notifier: NotifierService) {}

  selectPerson(person: Person): void {
    this.notifier.notify(`${person.lastname} ${person.firstname}`);
  }

  ...
}
```

Receiver

```
...
import { NotifierService } from '.notifier.service';

...
export class SelectPersonsComponent implements OnInit {
  names: string[] = [];

  constructor(private notifier: NotifierService) {}

  ngOnInit() {
    this.notifier
      .listen()
      .subscribe(x => this.names.push(x));
  }

  ...
}
```