Angular Components and Data Binding

Components

Components are the fundamental building blocks of Angular applications. Each component consists of:

- A template: Defines the view.

- A class: Handles the business logic and data.

- Metadata: Provides additional information about the component to Angular.

Example:

```typescript
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  title = 'Angular Components';
}
```
```html
<!-- app.component.html -->
<h1>{{ title }}</h1>
```

Manually Creating Components

To create a component manually, follow these steps:

1. Create the Component Class: Create a TypeScript file.

2. Create the Template: Create an HTML file.

3. Create the Styles: Create a CSS file.

4. Register the Component: Add it to a module.

Example:

```typescript
// my-component.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-my-component',
  templateUrl: './my-component.component.html',
  styleUrls: ['./my-component.component.css']
})
export class MyComponent {
  message = 'Hello from MyComponent!';
}
```
```html
<!-- my-component.component.html -->
<p>{{ message }}</p>
```
```css
/* my-component.component.css */
p {
  color: blue;
}
```
Add MyComponent to the module:
```typescript
// app.module.ts
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
import { MyComponent } from './my-component/my-component.component';

@NgModule({
  declarations: [
    AppComponent,
    MyComponent
  ],
  imports: [
    BrowserModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Working with Component Templates and Component Styles

Inline Template and Styles

Templates and styles can be defined inline using backticks and the styles array.

```typescript
@Component({
  selector: 'app-inline-component',
  template: `<p>Inline Template</p>`,
  styles: [`p { color: green; }`]
})
export class InlineComponent { }
```

External Template and Styles

Templates and styles can be referenced from external files.

```typescript
@Component({
  selector: 'app-external-component',
  templateUrl: './external-component.component.html',
  styleUrls: ['./external-component.component.css']
})
export class ExternalComponent { }
```

Data Binding

Data binding in Angular binds data between the component and the view. There are four forms:

1. String Interpolation

2. Property Binding

3. Event Binding

4. Two-Way Binding

String Interpolation

String interpolation uses the {{ }} syntax to bind data from the component to the template.

```typescript
// app.component.ts
export class AppComponent {
  title = 'String Interpolation Example';
}
```
```html
<!-- app.component.html -->
<h1>{{ title }}</h1>
```

Property Binding

Property binding binds data from the component to an HTML element property.

```typescript
// app.component.ts
export class AppComponent {
  isDisabled = true;
}
```
```html
<!-- app.component.html -->
<button [disabled]="isDisabled">Click Me</button>
```

Event Binding

Event binding binds an event from the template to a method in the component.

```typescript
// app.component.ts
export class AppComponent {
  handleClick() {
    alert('Button clicked!');
  }
}
```
```html
```

```
<!-- app.component.html -->
<button (click)="handleClick()">Click Me</button>
```

Two-Way Binding

Two-way binding allows for data to be synchronized between the component and the template. It uses [(ngModel)].

```typescript
// app.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  name = '';
}
```
```html
<!-- app.component.html -->
<input [(ngModel)]="name" placeholder="Enter your name">
<p>Hello, {{ name }}!</p>
```

Passing Data between Components

Parent to Child

Use @Input() to pass data from a parent component to a child component.

```typescript
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `<app-child [childMessage]="parentMessage"></app-child>`
})
export class ParentComponent {
  parentMessage = "Message from Parent";
}
```

~~~
```typescript
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `<p>{{ childMessage }}</p>`
})
export class ChildComponent {
  @Input() childMessage: string;
}
~~~

Child to Parent

Use @Output() and EventEmitter to pass data from a child component to a parent component.

```typescript
// parent.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-parent',
  template: `<app-child (messageEvent)="receiveMessage($event)"></app-child>
             <p>Message: {{ message }}</p>`
})
export class ParentComponent {
  message: string;

  receiveMessage($event) {
    this.message = $event;
  }
}
```
```typescript
// child.component.ts
import { Component, Output, EventEmitter } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `<button (click)="sendMessage()">Send Message</button>`
```

```
})
export class ChildComponent {
  message = 'Message from Child';

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

  sendMessage() {
    this.messageEvent.emit(this.message);
  }
}
```