ANGULAR 6

Lesson 02

OBJECTIVES

Essentials of Angular

- Component Basics
- Setting up the templates
- Creating Components using CLI
- Nesting Components
- Data Binding - Property & Event Binding, String Interpolation, Style binding
- Two-way data binding
- Input Properties, Output Properties, Passing Event Data
- Case Study

ESSENTIALS OF ANGULAR

Component Basics

- A component controls a patch of screen called a view
- You define a component's application logic—what it does to support the view—inside a class.
- The class interacts with the view through an API of properties and methods.
- For example, this HeroListComponent has a heroes property that returns an array of heroes that it acquires from a service. HeroListComponent also has a selectHero() method that sets a selectedHero property when the user clicks to choose a hero from that list.

Component Basics

```
export class HeroListComponent implements OnInit {

        heroes: Hero[];

        selectedHero: Hero;

        constructor(private service: HeroService) { }

        ngOnInit() {

                this.heroes = this.service.getHeroes();

        }

        selectHero(hero: Hero) { this.selectedHero = hero; }

}
```

TEMPLATES

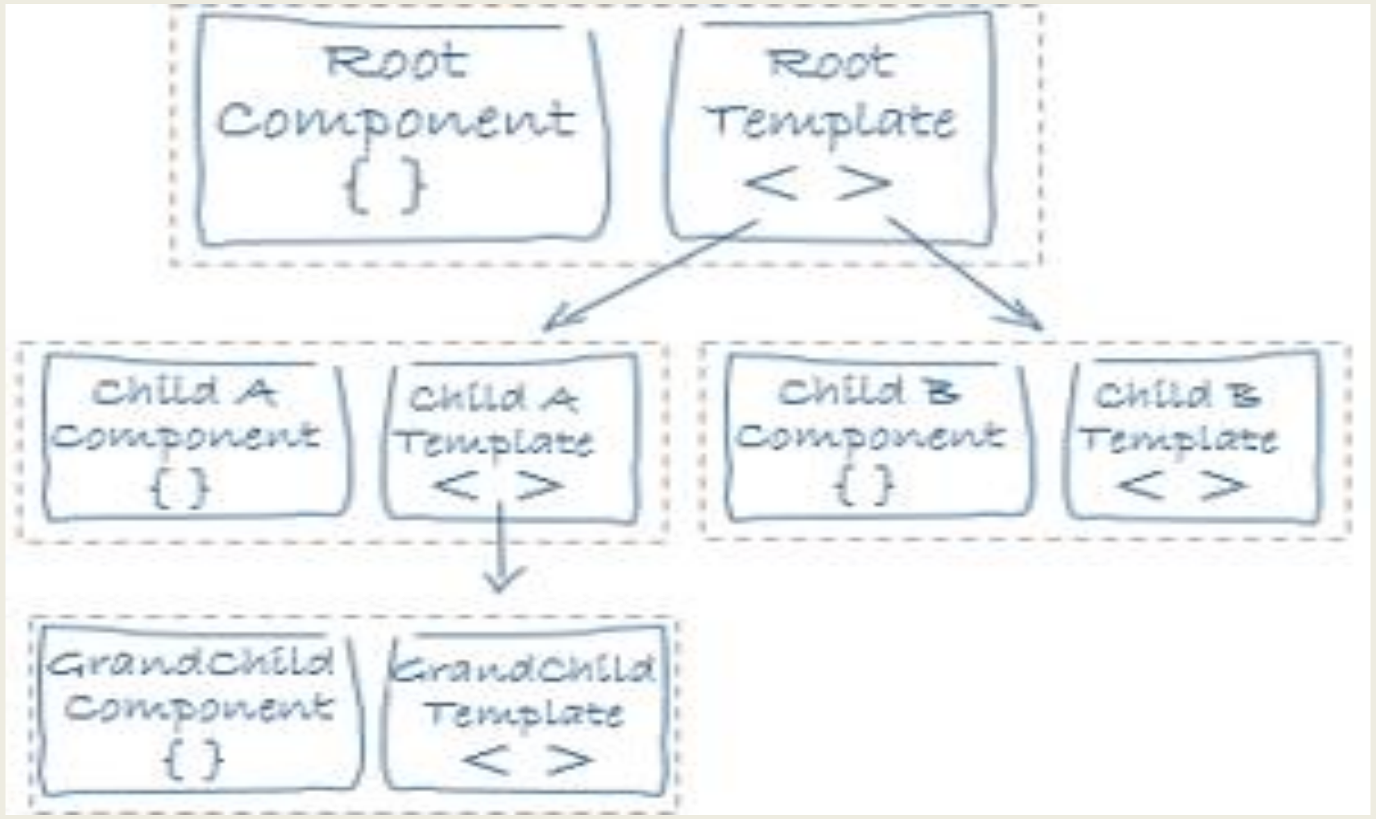
- You define a component's view with its companion **template**. A template is a form of HTML that tells Angular how to render the component
- A template looks like regular HTML, except for a few differences. Here is a template for our HeroListComponent

```
<h2>Hero List</h2>

<p><i>Pick a hero from the list</i></p>
<ul>
  <li *ngFor="let hero of heroes" (click)="selectHero(hero)">
    {{hero.name}}
  </li>
</ul>

<app-hero-detail *ngIf="selectedHero" [hero]="selectedHero"></app-hero-detail>
```

TEMPLATES


Root Component { }
Root Template < >
Child A Component { }
Child A Template < >
Child B Component { }
Child B Template < >
GrandChild Component { }
GrandChild Template < >

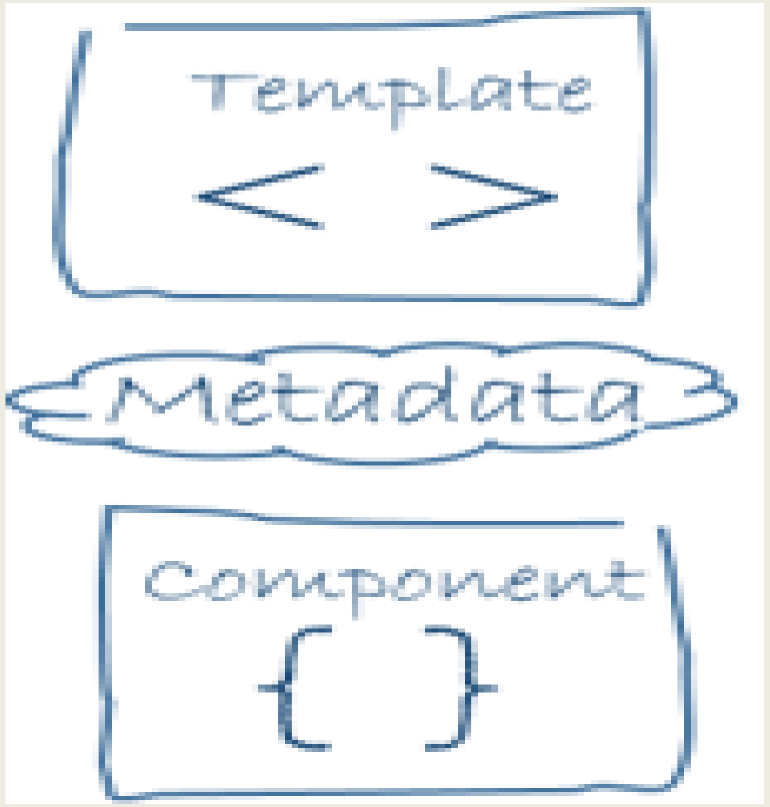
METADATA

- Metadata tells Angular how to process a class.
- HeroListComponent really is just *A CLASS*. It's not a component until you tell Angular about it.

```
@Component({
        selector:    'app-hero-list',
        templateUrl: './hero-list.component.html',
        providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
/* . . . */
}
```

METADATA

- selector: CSS selector that tells Angular to create and insert an instance of this component where it finds a <hero-list> tag in parent HTML. For example, if an app's HTML contains <hero-list></hero-list>, then Angular inserts an instance of the HeroListComponent view between those tags.
- templateUrl: module-relative address of this component's HTML template
- providers: array of **dependency injection providers** for services that the component requires. This is one way to tell Angular that the component's constructor requires a HeroService so it can get the list of heroes to display



```
@Component({
        selector:    'app-hero-list',
        templateUrl: './hero-list.component.html',
        providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
      heroes: Hero[];
      selectedHero: Hero;
      constructor(private service: HeroService) { }
      ngOnInit() {
            this.heroes = this.service.getHeroes();
      }
      selectHero(hero: Hero) { this.selectedHero = hero; }
}
```

Creating Components using CLI

- Angular components are the basic building blocks of your app. Each component defines:
- Any necessary imports needed by the component
- A component decorator, which includes properties that allow you to define the template, CSS styling, animations, etc..
- A class, which is where your component logic is stored.
- Angular components reside within the /src/app folder:

```
> src
  > app
    app.component.ts     // A component file
    app.component.html  // A component template file
    app.component.css   // A component CSS file
    > about             // Additional component folder
    > contact           // Additional component folder
```

Creating Components using CLI

- Let's open up the /src/app/app.component.ts componen file that the CLI generated for us:

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

@Component({
          selector: 'app-root',
          templateUrl: './app.component.html',
          styleUrls: ['./app.component.scss']
})
export class AppComponent {
         title = 'app';
}
```

Creating Components using CLI

- The Angular CLI is used for more than just starting new projects. You can also use it to generate new components. In the console (you can open a second console so that your ng serve command is not halted), type::

```
> ng generate component home        or     > ng g c home

 // Output:
 create src/app/home/home.component.html (23 bytes)
 create src/app/home/home.component.spec.ts (614 bytes)
 create src/app/home/home.component.ts (262 bytes)
 create src/app/home/home.component.scss (0 bytes)
 update src/app/app.module.ts (467 bytes)
```

Creating Components using CLI

- Let’s look at the component code and then take these one at a time. Open our first

  TypeScript file: src/app/home/home.component.ts

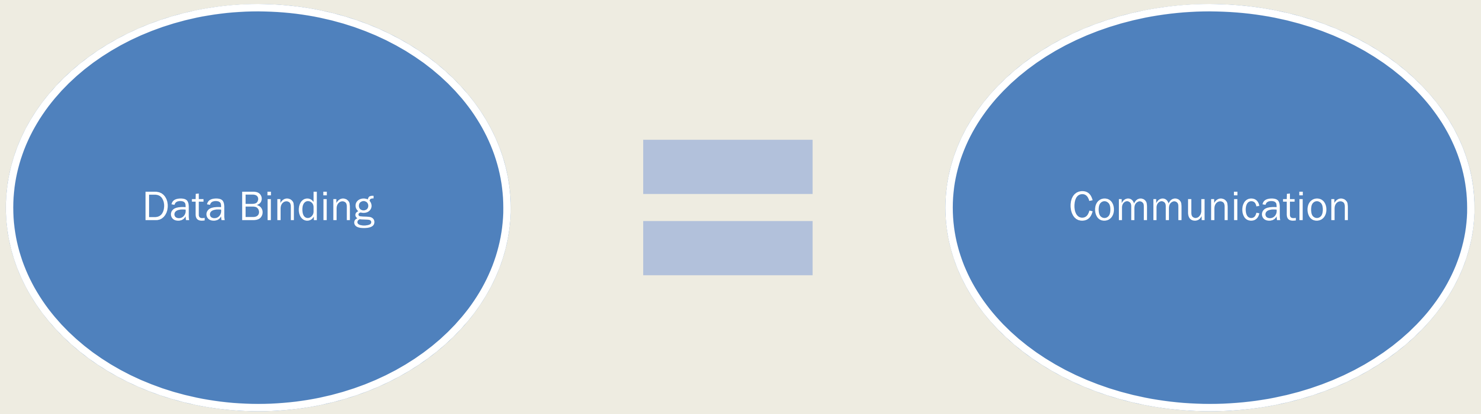
```
import { Component, OnInit } from '@angular/core';

@Component({
	selector: 'app-home',
	templateUrl: './home.component.html',
	styleUrls: ['./home.component.css']
})
export class HomeComponent implements OnInit {
	constructor() { }
	ngOnInit() {
	}
}
```

Creating Components using CLI

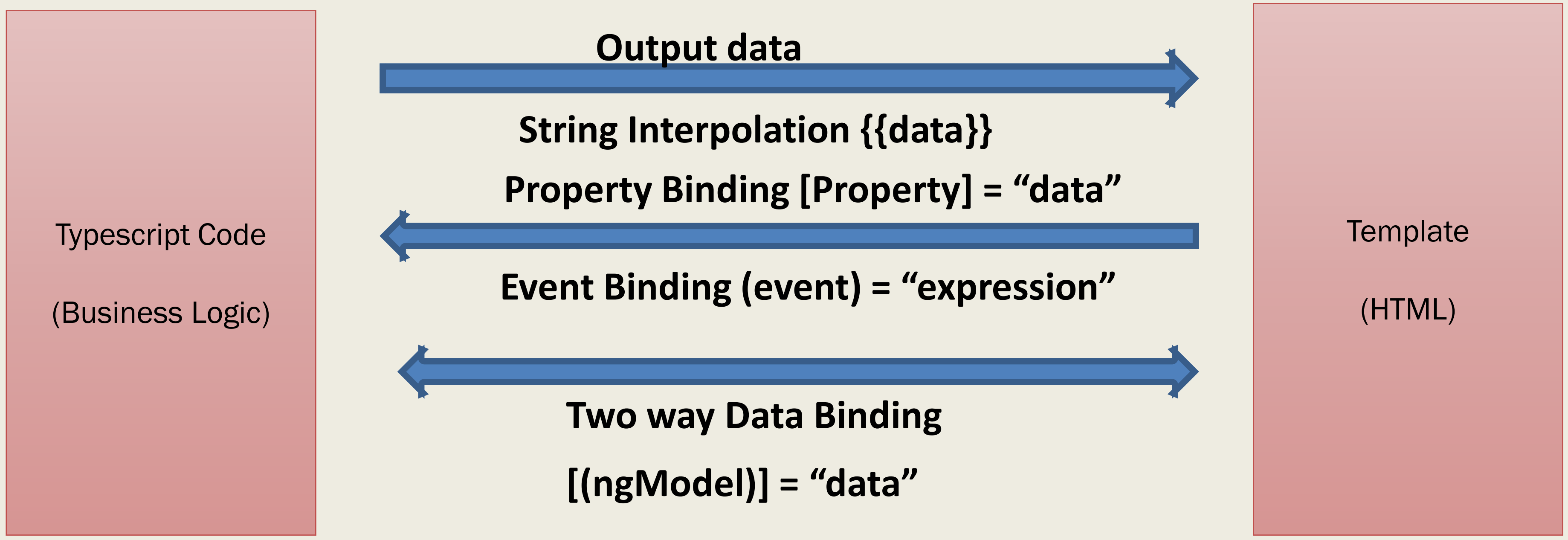
- IMPORTING DEPENDENCIES
  - *The import statement defines the modules we want to use to write our code. Here we're importing two things: Component, and OnInit.*
- COMPONENT DECORATORS
  - *Selector*
  - *ADDING A TEMPLATE OR TEMPLATEURL*
  - *ADDING CSS STYLES WITH STYLEURLS*

Data Binding



- Communication between your typescript code of your business logic and the template what the user sees is called Data Binding

Data Binding


Typescript Code
(Business Logic)
Output data
String Interpolation {{data}}
Property Binding [Property] = “data”
Event Binding (event) = “expression”
Two way Data Binding
[(ngModel)] = “data”
Template
(HTML)

- Communication between your typescript code of your business logic and the template what the user sees is called Data Binding

Interpolation(One-Way binding)

We met the double curly braces of interpolation, {{ and }}

The syntax between the interpolation curly braces is called as template expression.

Angular evaluates that expression using the component as the context.

- *Angular looks to the component to obtain property values or to call methods.*
- *Angular then converts the result of the template expression to a string and assigned that string to an element or directive property*

Interpolation is used to insert the interpolated strings into the text between HTML elements.

- *<li>{{hero.name}}</li>*
- *<p>The sum of 1 + 1 is {{1 + 1}}</p>*
- *<p>The sum of 1 + 1 is not {{1 + 1 + getVal()}}</p>*

Demo

Demo One Way Binding

Property Binding

The template expressions in quotes on the right of the equals are used to set the DOM properties in square brackets on the left.

target]="expression“

- *Example*
- *<img [src]="user.img">*
- *<span [hidden]="isUnchanged">changed</span>*

Like interpolation property binding is one way from the source class property to the target element property

Property binding effectively allows to control the template DOM from a component class.

The general guideline is to prefer property binding all for interpolation. However to include the template expression as part of a larger expression then use interpolation

Demo

Demo Property Binding

Event Binding

When an event in parentheses on the left of the equals is detected, the template statement in quotes on the right of the equals is executed.

- *<button (click)="onSave()">Save</button>*

The string in quotes is a *template statement*.

Template statements respond to an event by executing some JavaScript-like code.

The name of the bound event is enclosed in parentheses identifying it as the target event.

Template statement is often the name of a component class method enclosed in quotes.

Demo

Demo Event Binding

Two-way Binding

To display a component class property in the template and update that property when the user makes a change with user entry HTML elements like input element two-way binding is required.

In Angular ***ngModel*** directive is used to specify the two way binding.

- *[(target)]="expression“*
- *input type="text" [(ngModel)]="name">*

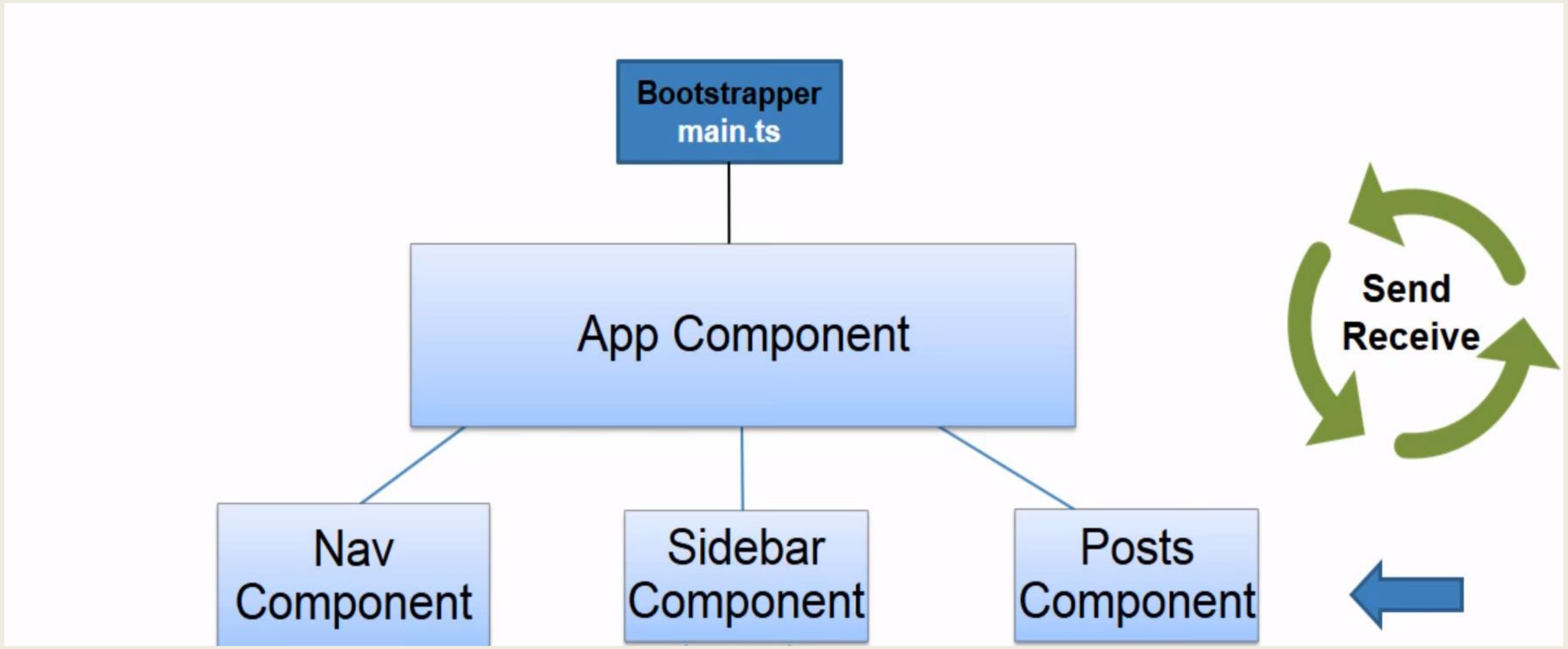
ngModel in square brackets is used to indicate property binding from the class property to the input element

Parentheses to indicate event binding to send the notification of the user entered data back to the class property

Demo

Demo Two Way Binding

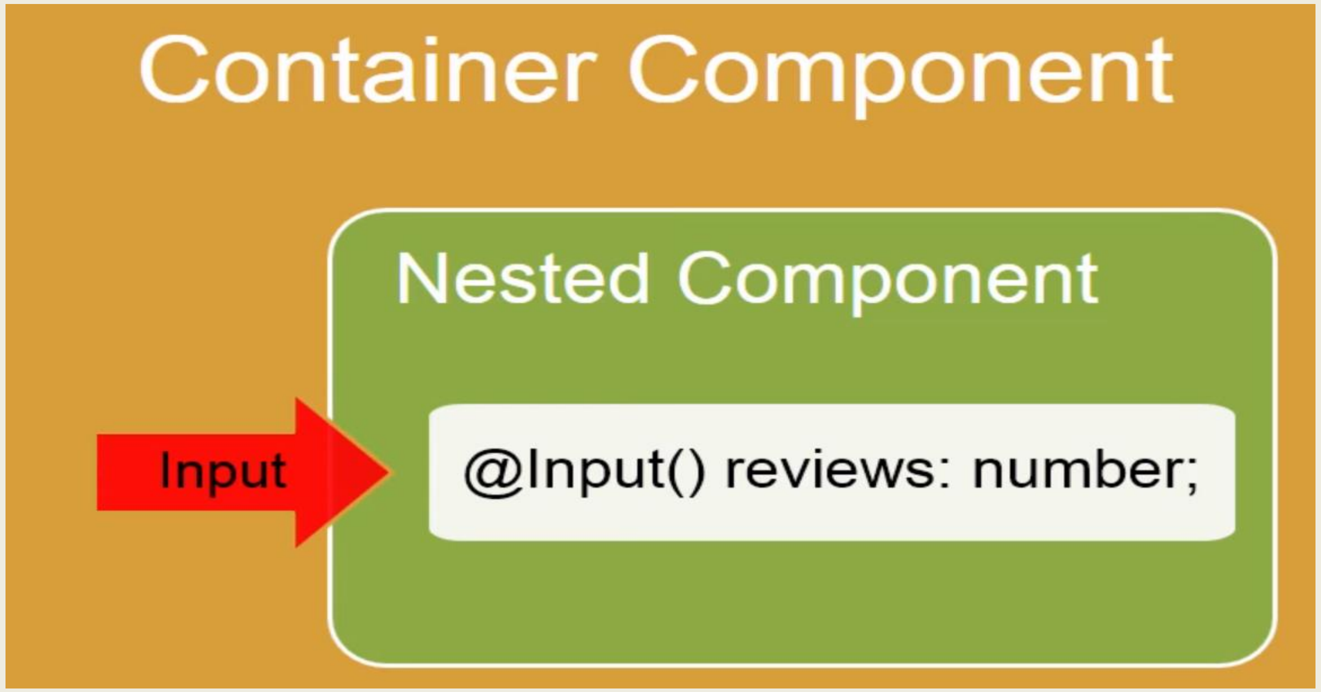
Nested Compoments



Using @Input and @Output

@Input –Allows data to flow from parents component to child component

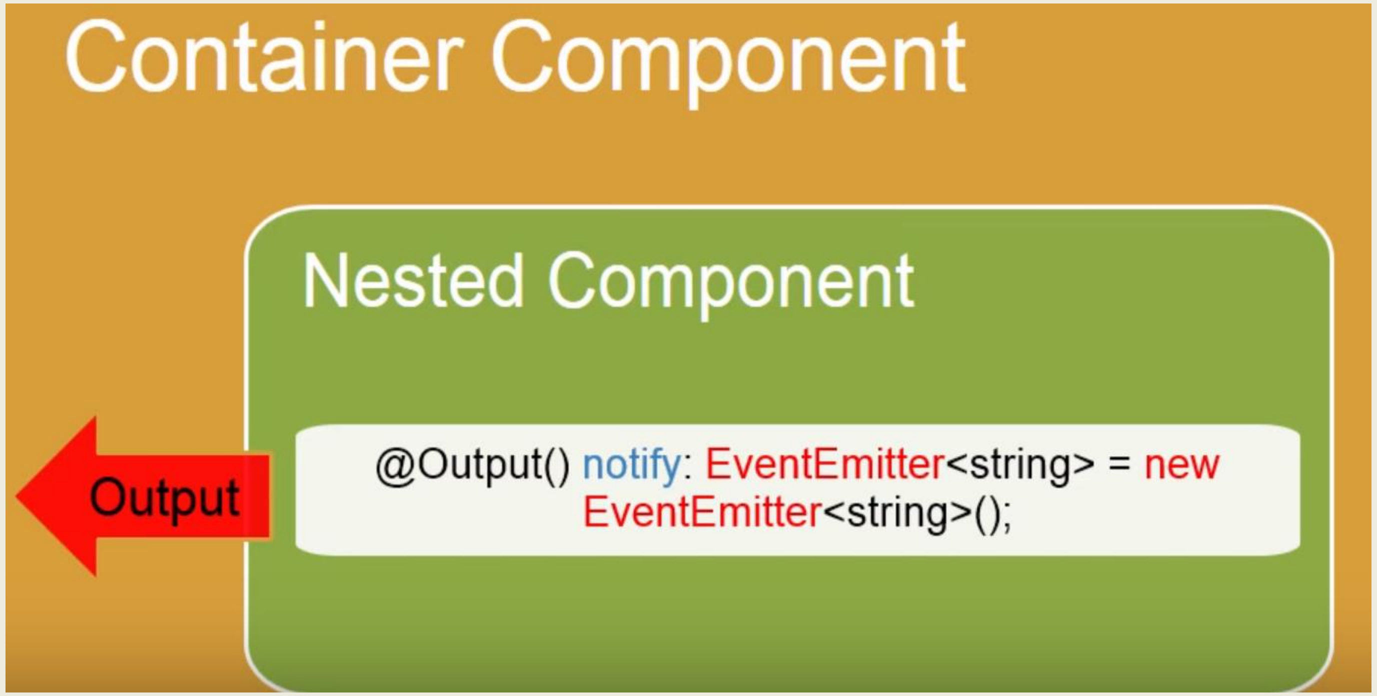
@Input allows you to pass data into your controller and templates through html and defining custom properties.



Using @Input and @Output (Contd...)

@Output that pass data from child component to ParentComponent

Components push out events using a combination of an @Output and an EventEmitter. This allows a clean separation between reusable Components and application logic.



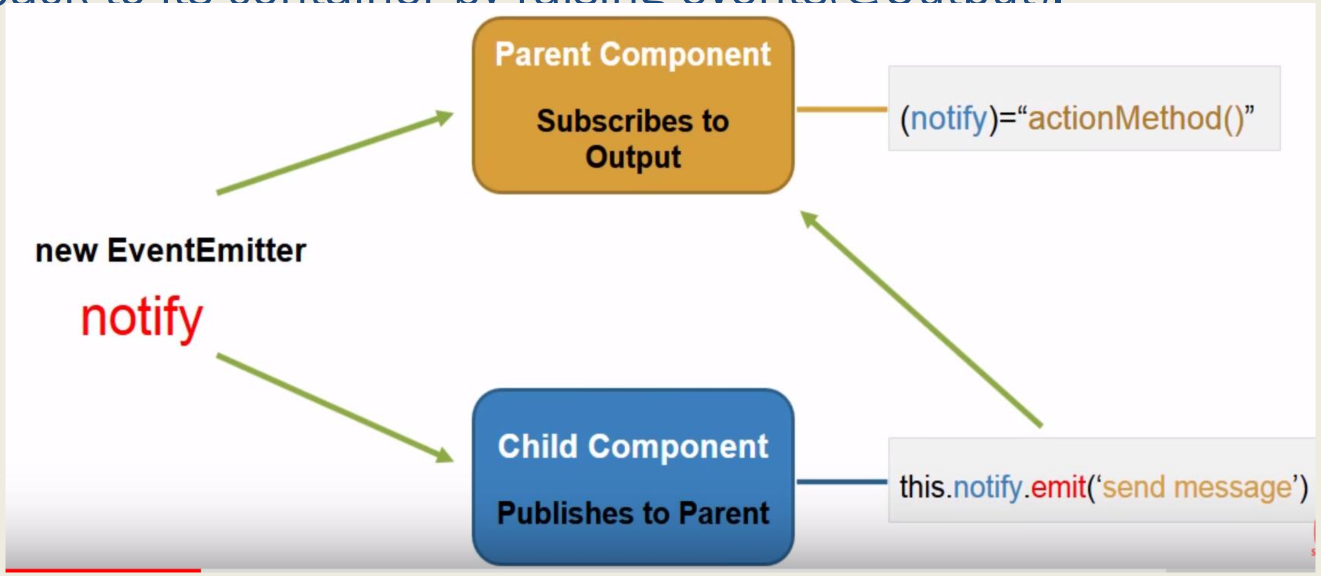
```
@Output() notify: EventEmitter<string> = new
EventEmitter<string>();
```

Using @Input and @Output (Contd…)

EventEmitter-Listen for something to happen & emit a event when triggered

emit() method is used to trigger the event by emitting data from inner component to outer component which can be accessed via $event

Nested component receives information from its container using input properties(@Input) and outputs information back to its container by raising events(@Output).



Demo

Demo Nested Components input output

Lab

Case Study