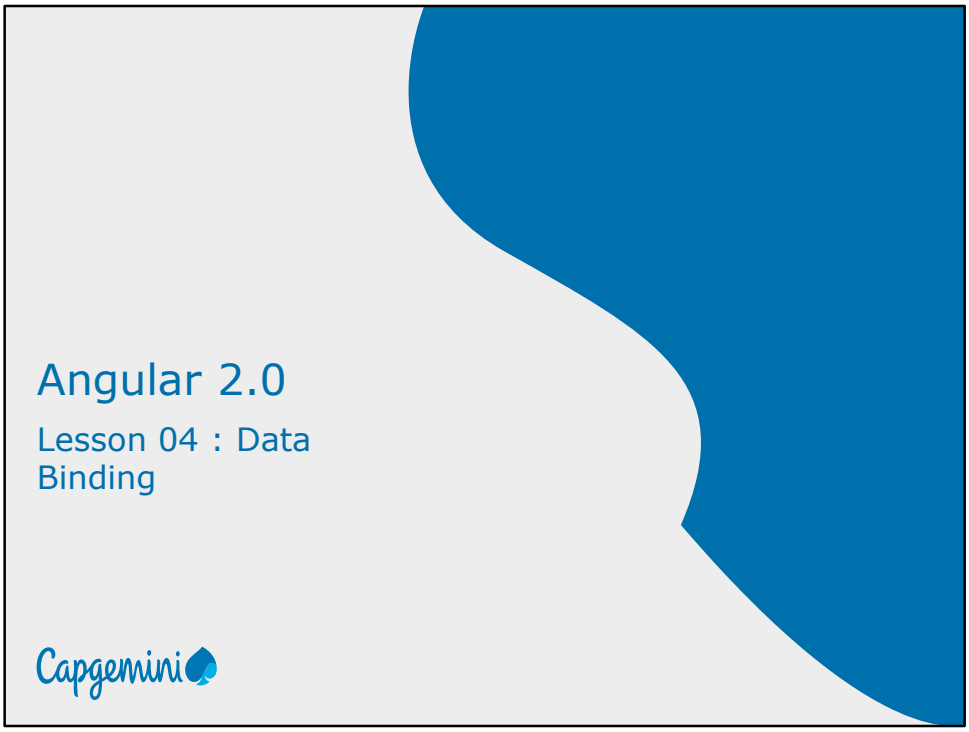
Instructor Notes:

Add instructor notes here.



Instructor Notes:

Add instructor notes here.

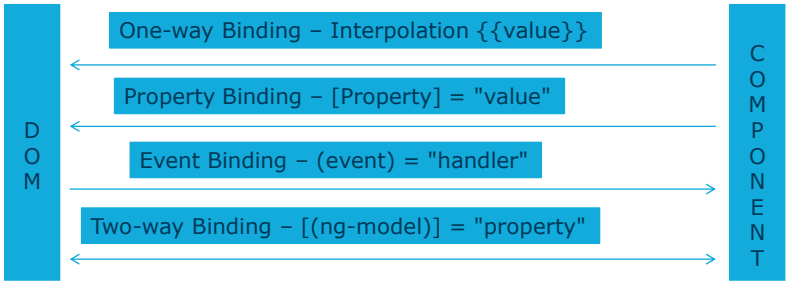
Lesson Objectives

- Introduction of Data binding
- One way data binding
- Two way data binding
- Nested components
- @Input, @output

Instructor Notes:

Data binding

- Angular supports **data binding**, a mechanism for coordinating parts of a template with parts of a component.
- Add binding markup to the template HTML to tell Angular how to connect both sides.
- Each form has a direction — to the DOM, from the DOM, or in both directions.



{{hero.name}}
<hero-detail [hero]="selectedHero"></hero-detail>
<li (click)="selectHero(hero)">

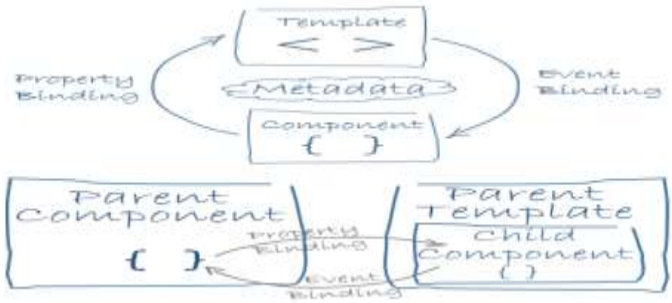
{{hero.name}} *interpolation* displays the component's hero.name property value within the <li> element.
The [hero] *property binding* passes the value of selectedHero from the parent HeroListComponent to the hero property of the child HeroDetailComponent.
The (click) *event binding* calls the component's selectHero method when the user clicks a hero's name.

<input [(ngModel)]="hero.name">**Two-way data binding** is an important fourth form that combines property and event binding in a single notation, using the ngModel directive

Instructor Notes:

Data binding

- Data binding plays an important role in communication between a template and its component.
- Data binding is also important for communication between parent and child components.



Instructor Notes:

Interpolation(One-Way binding)

- Interpolation performs one way binding from the class property to the template given in a double curly braces.
- Using Interpolation operations like concatenation, simple calculations, calling method on a class can be performed.
- 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.
  - {{hero.name}}
  - The sum of 1 + 1 is {{1 + 1}}
  - The sum of 1 + 1 is not {{1 + 1 + getVal()}}

Interpolation with Curly Braces
Double curly braces contain *template expressions* which allow us to read primitive or object values from component properties. Template expressions are similar to JavaScript and include features such as the ternary operator, concatenation and arithmetic.
interpolation automatically escapes any HTML, so here {{html}} displays this is a div , or more precisely it displays <div>this is a div</div>.

Instructor Notes:

Demo

- Demo One Way Binding

Instructor Notes:

Property Binding

- Property binding allows us to set a property of an element to the value of a template expression.
- 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
- Binding target is always enclosed in square brackets and the Binding source is always enclosed in quotes and specifies the template expression
- 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.

Property Binding with Square Brackets
We specify one-way bindings to DOM properties using square brackets and template expressions.
The template expressions in quotes on the right of the equals are used to set the DOM properties in square brackets on the left.
For example, textContent is set to a person's name, buttons are disabled based on the sex property of person, and the src, alt and title properties of the img element are bound to properties of the person object.
innerHTML sets the HTML content of the span to a number of stars (✰ is a star as you can see in the application output below). We use [innerHTML] here rather than interpolation here because interpolation would escape the HTML
Template expressions are similar to JavaScript and include features such as the ternary operator, concatenation and boolean conditions. We can call built-in methods such as repeat as well.
First, setting an attribute on an element will not evaluate the template expression in quotes. This is just a standard attribute assignment using a string, and we can confirm this in the application output below where we see the 'Hello ' + name string and not Hello World.
However, when we use interpolation in the quotes (identified by the double curly braces), we are now binding to the DOM *property* instead, and the template expression is evaluated. This is a subtle but important difference.
Property bindings using square brackets or the bind- prefix (known as the canonical form) always interpret the string in quotes as a template expression, so there is no need to include the curly braces in this case.

Instructor Notes:

Demo

- Demo Property Binding

Instructor Notes:

Event Binding

- Event binding is used to send information from the element property to the component class property to respond for user events.
- 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.
- If the defined event occurs; the template statement is executed calling the specified method in the component

The only way to know about a user action is to listen for certain events such as keystrokes, mouse movements, clicks, and touches. We declare our interest in user actions through Angular event binding.

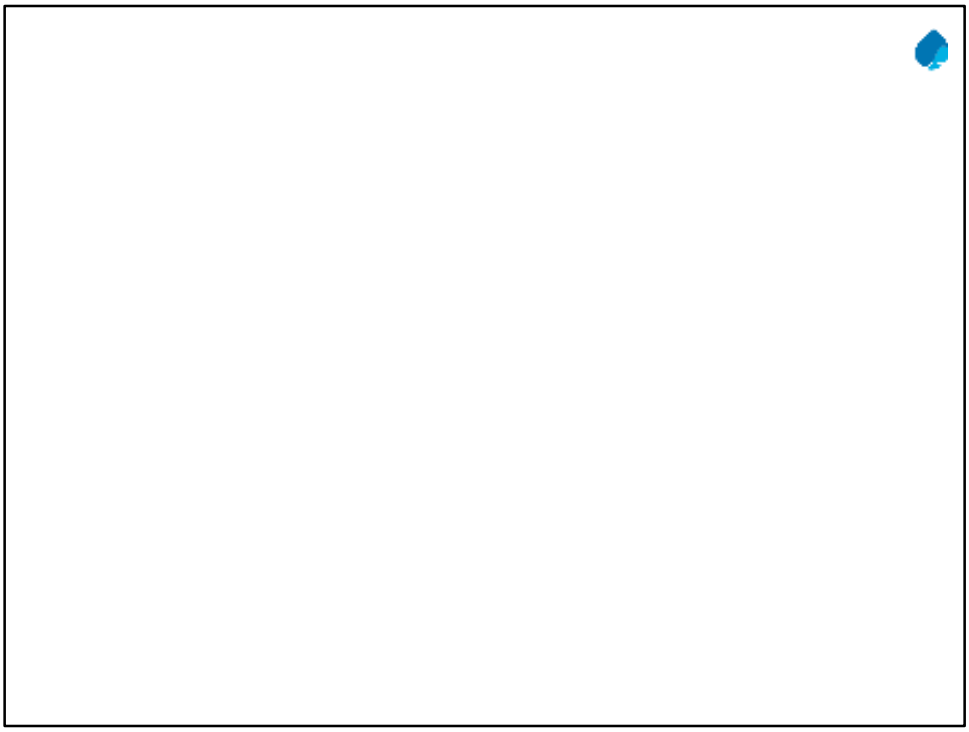
Event binding syntax consists of a target event within parentheses on the left of an equal sign, and a quoted template statement on the right.

$event object is used to listen to an event and grab's the user event

Event Binding with Parentheses
We handle DOM events using parentheses and template statements.
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.
Alternatively, we can use the *canonical* form by prefixing the event name with on-. These three bindings are equivalent
to (click), (dblclick) and (contextmenu).
The string in quotes is a *template statement*. Template statements respond to an event by executing some JavaScript-like code. Here we simply call methods on the component class and pass in the $event object.

Keyboard Events:Template statements are expected to change the state of the application based on the user's input. Technically, they can contain statements that directly alter data such as "key = $event.key" but this is usually best handled by calling a method.

Instructor Notes:



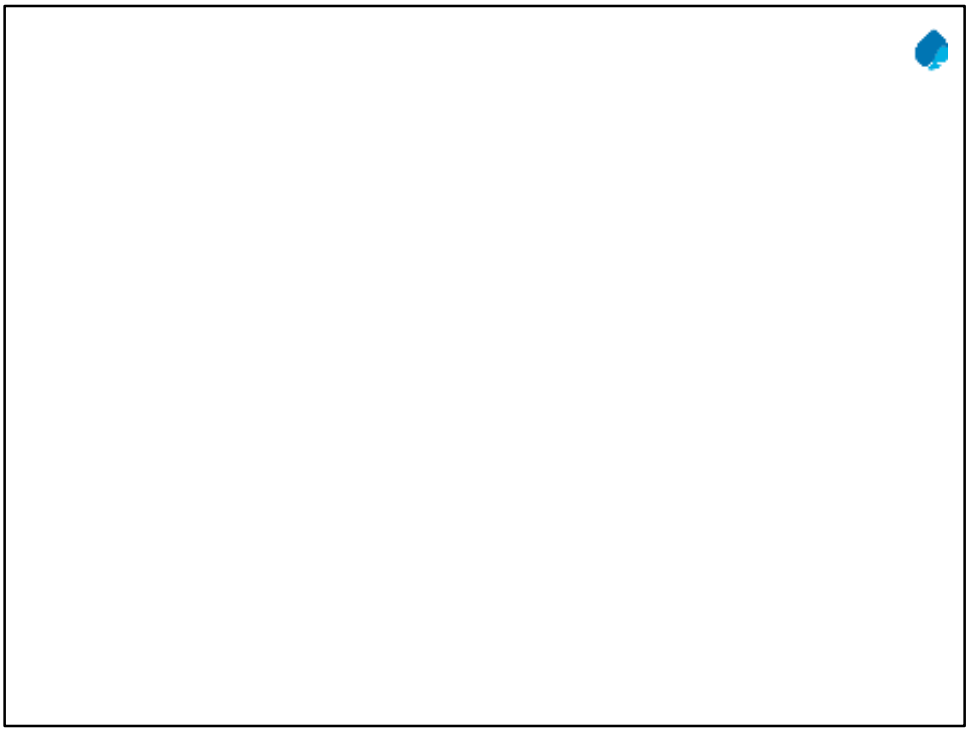
Checkbox---The change event is triggered when the user clicks on a checkbox

```
import { Component } from '@angular/core';
@Component({
selector: 'app-checkbox', template: ` <h1>Checkbox</h1> <p> <label
[class.selected]="cb1.checked"> <input #cb1 type="checkbox" value="one"
(change)="logCheckbox(cb1)"> One </label> <label
[class.selected]="cb2.checked"> <input #cb2 type="checkbox" value="two"
(change)="logCheckbox(cb2)"> Two </label> <label
[class.selected]="cb3.checked"> <input #cb3 type="checkbox" value="three"
(change)="logCheckbox(cb3)"> Three </label> </p> <h2>Log <button
(click)="log=''">Clear</button></h2> <pre>{{log}}</pre>`, styles: ['.selected
{color: OrangeRed;}'] })
export class CheckboxComponent
{ log = ''; logCheckbox(element: HTMLInputElement): void { this.log +=
`Checkbox ${element.value} was ${element.checked ? '' : 'un'}checked\n`; } }
```

Text Areatextarea behaves in a similar way to the textbox

```
import { Component } from '@angular/core';
@Component({
selector: 'app-text-area', template: ` <h1>Text Area</h1> <textarea ref-
textarea [(ngModel)]="textValue" rows="4"></textarea><br/> <button
(click)="logText(textarea.value)">Update Log</button> <button
(click)="textValue=''">Clear</button> <h2>Log <button
(click)="log=''">Clear</button></h2> <pre>{{log}}</pre>` })
export class TextAreaComponent { textValue = 'initial value'; log = '';
logText(value: string): void { this.log += `Text changed to '${value}'\n`; } }
```

Instructor Notes:



Radio
The radio field behaves in a similar way to the checkbox.

```
import { Component } from '@angular/core';
@Component({ selector: 'app-radio', template: ` <h1>Radio</h1> <p> <label
[class.selected]="r1.checked"> <input #r1 type="radio" name="r" value="one"
(change)="logRadio(r1)"> One </label> <label [class.selected]="r2.checked">
<input #r2 type="radio" name="r" value="two" (change)="logRadio(r2)"> Two
</label> <label [class.selected]="r3.checked"> <input #r3 type="radio"
name="r" value="three" (change)="logRadio(r3)"> Three </label> </p>
<h2>Log <button (click)="log=''">Clear</button></h2> <pre>{{log}}</pre>`,
styles: ['.selected {color: OrangeRed;}'] })
 export class RadioComponent { log = ''; logRadio(element:
HTMLInputElement): void { this.log += `Radio ${element.value} was
selected\n`; } }
```

Instructor Notes:

Demo

- Demo Event Binding

Instructor Notes:

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

Two-way data binding combines the square brackets of property binding with the parentheses of event binding in a single notation using
the ngModel directive. Tip, to remember that the parentheses go inside the brackets, visualize a banana in a box. [()]
We bind to the ngModel property of the ngModel directive to update the contents of the input field. Conversely, the ngModelChange event takes care of setting of the value of the component property when the user changes the value in the text box.

Instructor Notes:

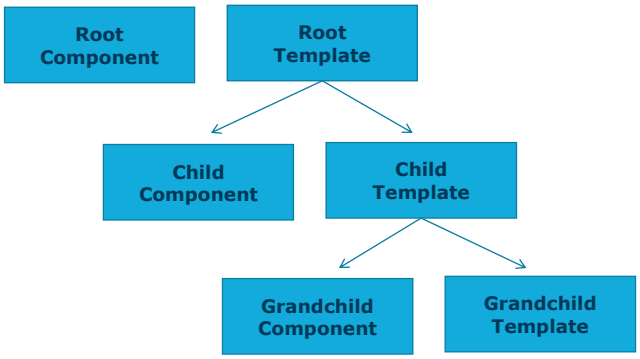
Demo

- Demo Two Way Binding

Instructor Notes:

Nested Components

- Components have templates which may contain other components.
- Outer component is referred as the container or parent component
- Inner component is referred as the nested or child component.

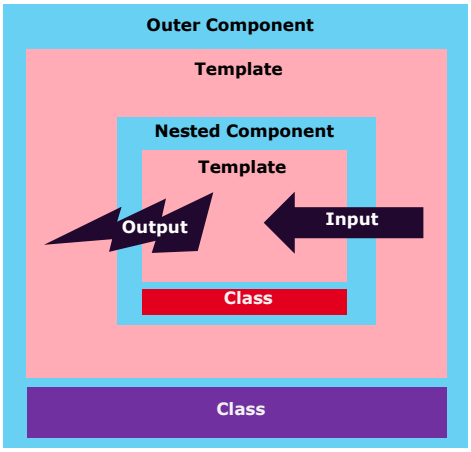


Instructor Notes:

Add instructor notes here.

Using @Input and @Output

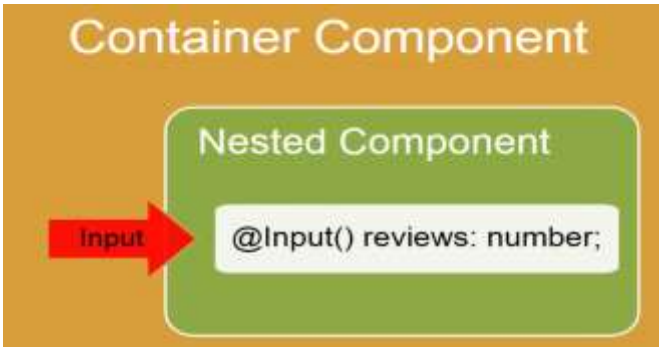
- Nested component receives information from its container using input properties(@Input) and outputs information back to its container by raising events(@Output).
- Input, Output & EventEmitter need to imported in Nested component from @angular/core module.
- emit() method is used to trigger the event by emitting data from inner component to outer component which can be accessed via $event.



Instructor Notes:

Using @Input and @Output (Contd…)

- @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.

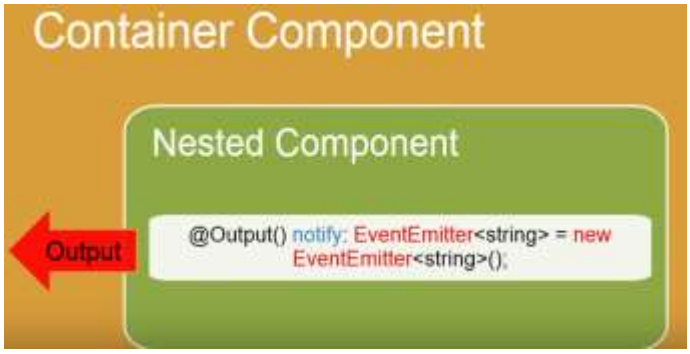


Use property binding and the @Input decorator to pass data into a component, and @Output and EventEmitter to pass data out and bind to an event.

Instructor Notes:

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.

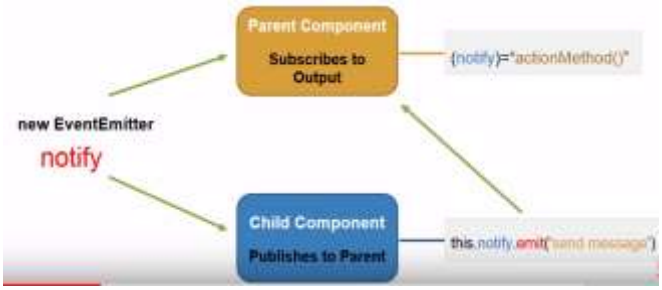


Event Binding with @Output
We can also pass values *out* of a child component back to the parent using an event binding. Let's see how.
In the child component we include an @Output decorator on a variable of type EventEmitter. This object will be used to fire our custom events.
@Output() colorValue: EventEmitter<string> = new EventEmitter();The variable name decorated by @Output is the name of the event, which in this case is colorValue. The <string>type in angle brackets is the data type of the payload to be sent with the event.
We trigger the event by calling the emit method on the EventEmitter object. This method takes one argument which is the payload of the event.

Instructor Notes:

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).



Instructor Notes:

Demo

- Demo Nested Components input output

Instructor Notes:

Add instructor notes here.

Summary

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

Add the notes here.

Instructor Notes:

Add instructor notes here.

Lab

- Lab 1.2

Add the notes here.