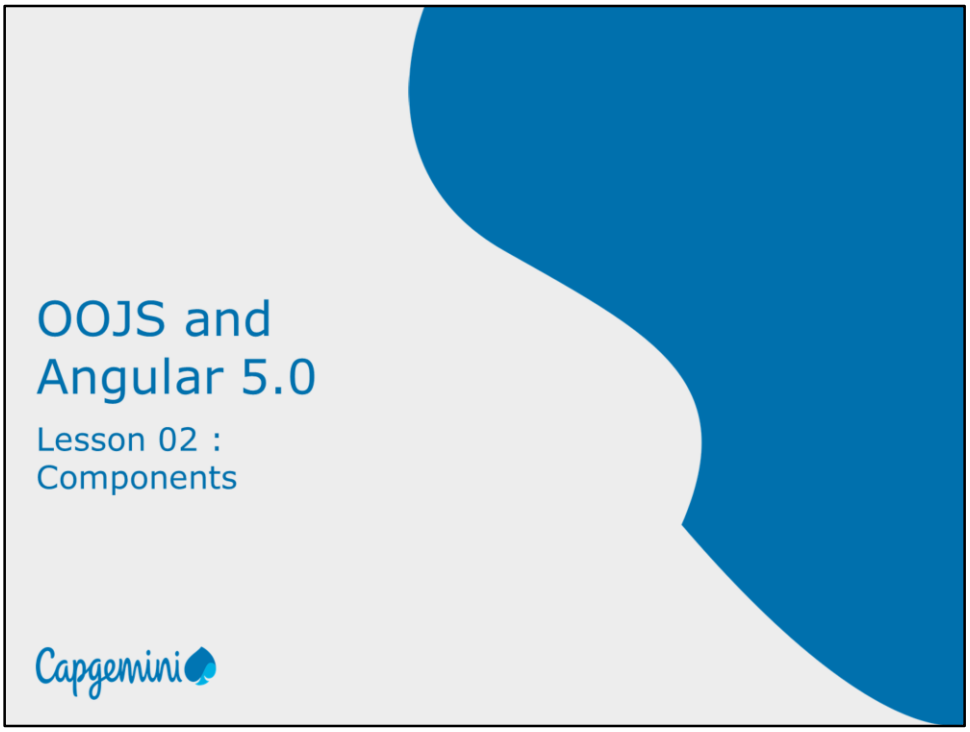
Instructor Notes:

Add instructor notes here.



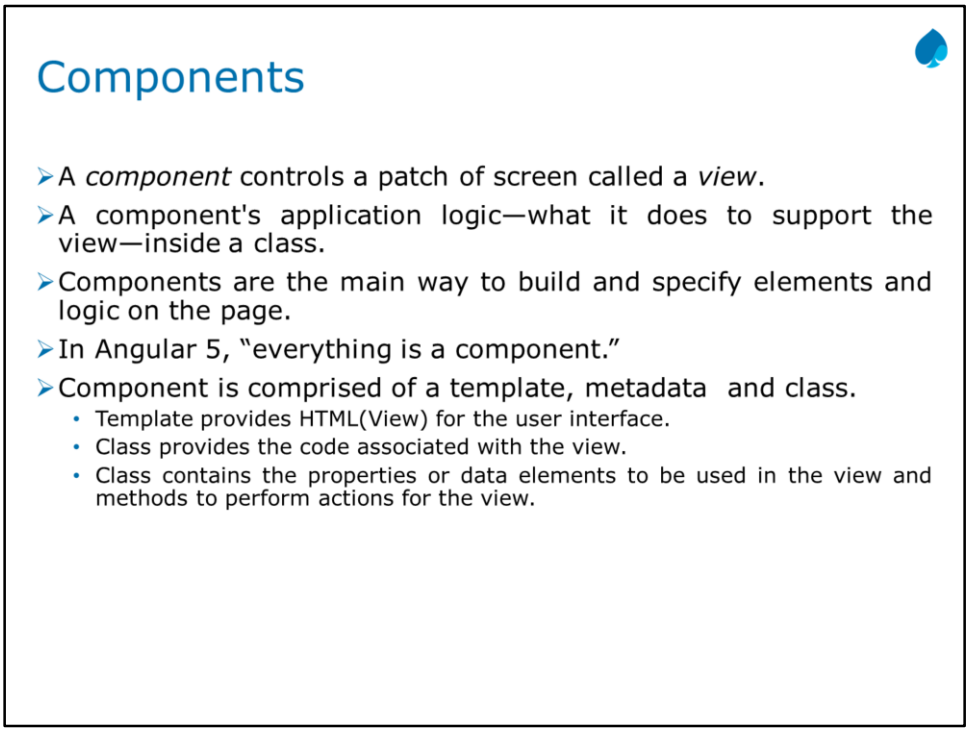
Instructor Notes:

Add instructor notes here.

Lesson Objectives

- Introduction of component
- Developing a simple component
- Templates for a component
- Component style
- Component lifecycle

Instructor Notes:



Description

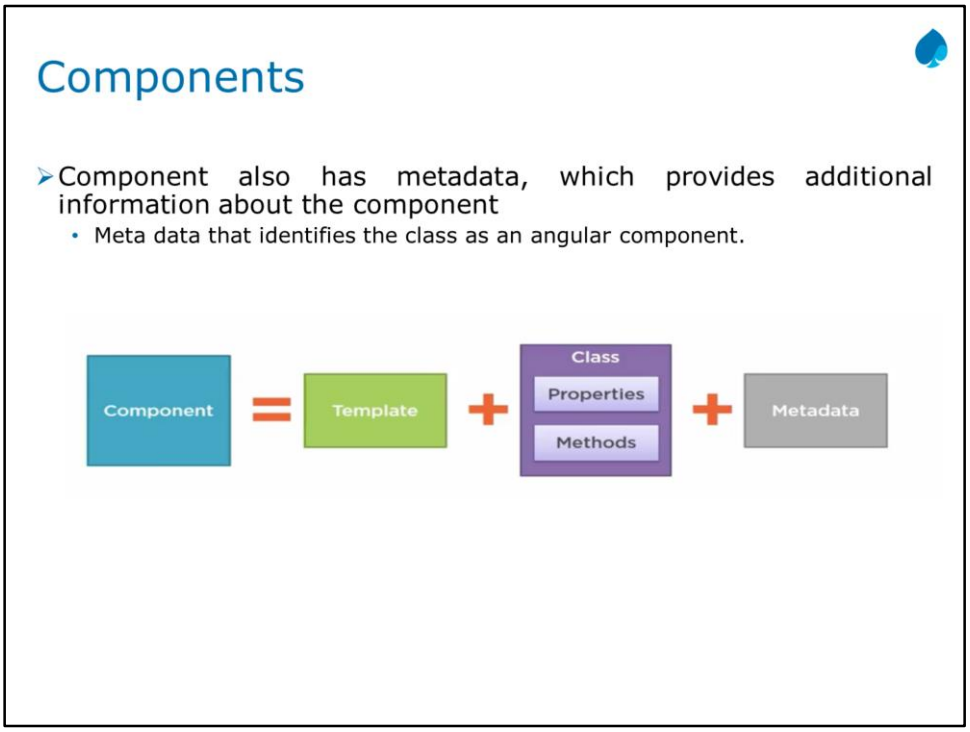
Component decorator allows you to mark a class as an Angular component and provide additional metadata that determines how the component should be processed, instantiated and used at runtime.

Components are the most basic building block of an UI in an Angular application. An Angular application is a tree of Angular components. Angular components are a subset of directives. Unlike directives, components always have a template and only one component can be instantiated per an element in a template.

A component must belong to an NgModule in order for it to be usable by another component or application. To specify that a component is a member of an NgModule, you should list it in the declarations field of that NgModule.

In addition to the metadata configuration specified via the Component decorator, components can control their runtime behavior by implementing various Life-Cycle hooks.

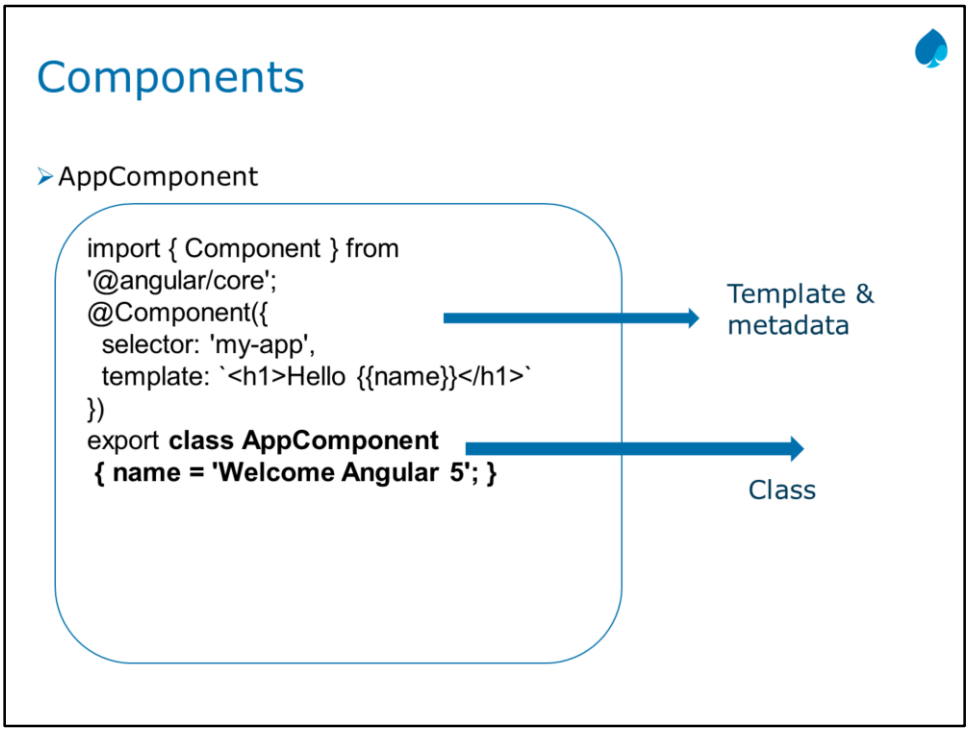
Instructor Notes:



Metadata Properties:
animations - list of animations of this component
changeDetection - change detection strategy used by this component
encapsulation - style encapsulation strategy used by this component
entryComponents - list of components that are dynamically inserted into the view of this component
exportAs - name under which the component instance is exported in a template
host - map of class property to host element bindings for events, properties and attributes
inputs - list of class property names to data-bind as component inputs
interpolation - custom interpolation markers used in this component's template
moduleId - ES/CommonJS module id of the file in which this component is defined
outputs - list of class property names that expose output events that others can subscribe to
providers - list of providers available to this component and its children
queries - configure queries that can be injected into the component
selector - css selector that identifies this component in a template
styleUrls - list of urls to stylesheets to be applied to this component's view
styles - inline-defined styles to be applied to this component's view
template - inline-defined template for the view
templateUrl - url to an external file containing a template for the view
viewProviders - list of providers available to this component and its view children
Example
content_copy@Component({selector: 'greet', template: 'Hello {{name}}!'}) class Greet { name: string = 'World'; }

Instructor Notes:

Add instructor notes here.



Components are a logical piece of code for Angular JS application. A Component consists of the following −
Template − This is used to render the view for the application. This contains the HTML that needs to be rendered in the application. This part also includes the binding and directives.
Class − This is like a class defined in any language such as C. This contains properties and methods. This has the code which is used to support the view. It is defined in TypeScript.
Metadata − This has the extra data defined for the Angular class. It is defined with a decorator.

We are using the import keyword to import the 'Component' decorator from the angular/core module.
We are then using the decorator to define a component.
The component has a selector called 'my-app'. This is nothing but our custom html tag which can be used in our main html page.

<body> <my-app></my-app> </body>

Instructor Notes:

Components-Metadata

- Metadata tells Angular how to process a class.

  export **class AppComponent**

  { name = 'Welcome Angular 5'; }

- To tell Angular that AppComponent is a component, attach metadata to the class. In TypeScript, we can attach metadata by using a decorator,
- @Component decorator, which identifies the class.
- The metadata in the @Component tells Angular where to get the major building blocks .
- export keyword exports the class; thereby making it available for use by other components of the application.

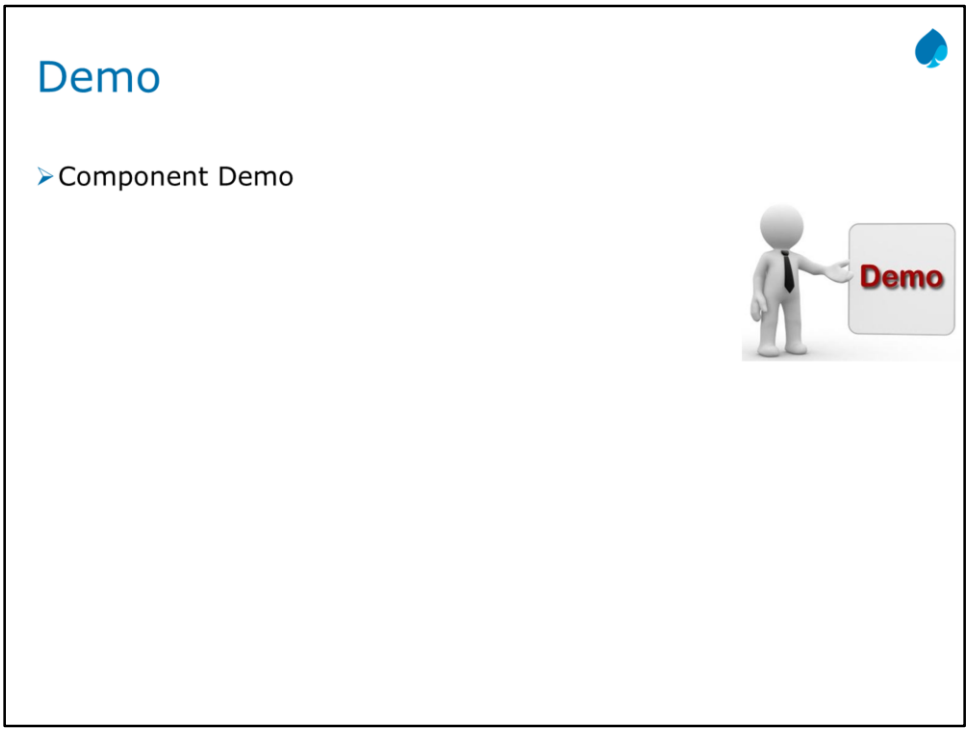
Instructor Notes:

Components-Metadata

- @Component configuration options:
  - 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.
  - template : This is the portion of our component that holds template. It is an integral part of the component as it allows to tie logic from component directly to a view. Its call inline
  - templateUrl: module-relative address of this component's HTML template, its call external
  - providers: array of dependency injection providers for services that the component requires.

Instructor Notes:

Add instructor notes here.



Add the notes here.

Instructor Notes:

Template

- HTML is the language of the Angular template
- Template are mostly HTML which is used to tell Angular how to render the component.
- Template for a component can be created using
  - Inline template (Embedded template string)
  - Linked template (Template provided in external html file)
- Interpolation ({{...}})-use interpolation to weave calculated strings into the text between HTML element tags and within attribute assignments. Example
  - `<h1>Hello {{name}}</h1>
  - <h1>Hello world {{10 + 20 + 30}}</h1>
  - <h3> {{title}} <img src="{{heroImageUrl}}" style="height:30px"></h3>

The expression can invoke methods of the host component such as getVal()
<!-- "The sum of 1 + 1 is not 4" --> The sum of 1 + 1 is not {{1 + 1 + getVal()}}
A template **expression** produces a value. Angular executes the expression and assigns it to a property of a binding target; the target might be an HTML element, a component, or a directive.
The interpolation braces in {{1 + 1}} surround the template expression 1 + 1. a template expression appears in quotes to the right of the = symbol as in [property]="expression".
We can write these template expressions in a language that looks like JavaScript. Many JavaScript expressions are legal template expressions, but not all.
JavaScript expressions that have or promote side effects are prohibited, including:
assignments (=, +=, -=, ...)
new
chaining expressions with ; or ,
increment and decrement operators (++ and --)

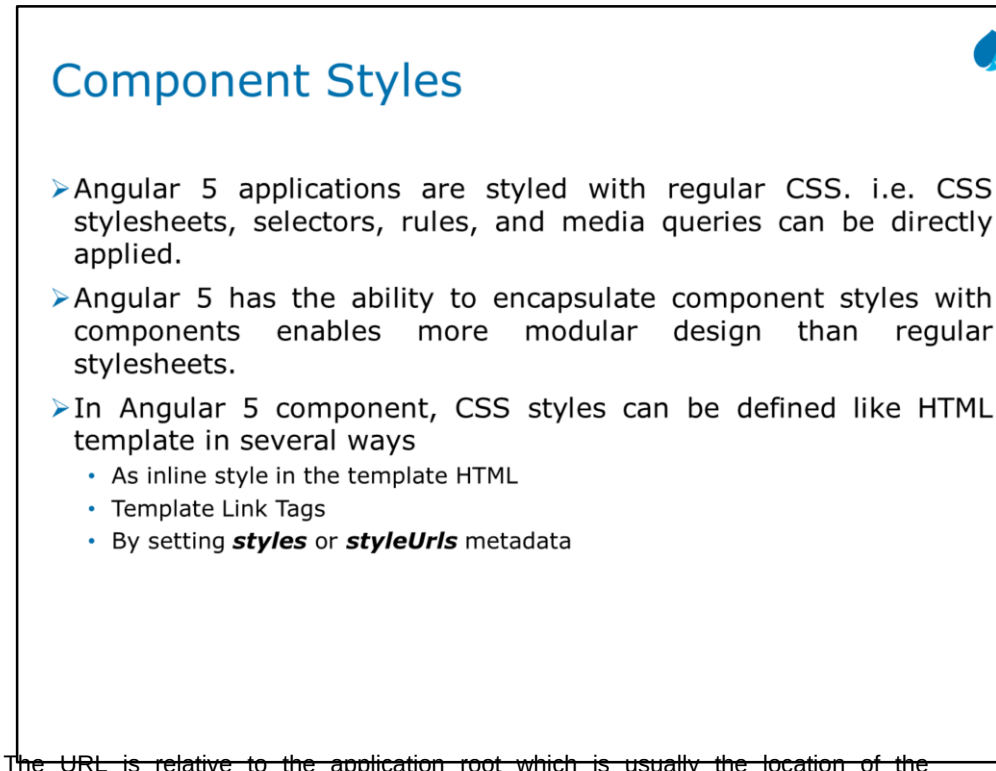
Instructor Notes:

Demo

- Component Demo Inline Template
- Component Demo External Template

Instructor Notes:

Add instructor notes here.



The URL is relative to the application root which is usually the location of the index.html web page that hosts the application. The style file URL is not relative to the component file.

Special selectors
Component styles have a few special selectors from the world of shadow DOM style scoping:

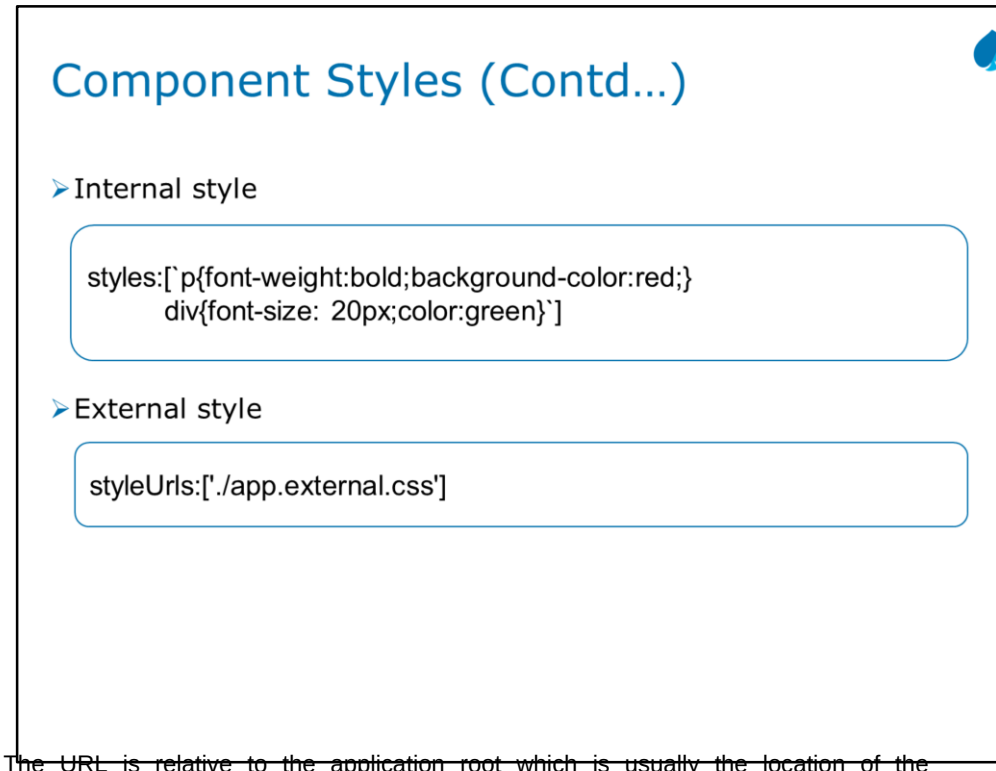
:host is a pseudo-class selector that applies styles in the element that hosts the component. It means if a component has a child component using component binding then child component will use :host selector that will target host element in parent component. :host selector can be used in component with styles metadata as well as with styleUrls metadata of @Component decorator.
@Component({ --- styles: [':host { position: absolute; top: 10%; }'] })

:host-context() : Looks for a CSS class in any ancestor of the component host element, all the way up to the document root. It's useful when combined with another selector. :host-context selector is used in the same way as :host selector but :host-context is used when we want to apply a style on host element on some condition outside of the component view. For the example a style could be applied on host element only if a given CSS class is found anywhere in parent tree up to the document root. In our example we have following components in parent-child relationship.
:host-context(.my-theme) h3 { background-color: green; font-style: normal; }

/deep/ : selector to force a style down through the child component tree into all the child component views. The /deep/ selector works to any depth of nested components, and it applies both to the view children and the content children of the component.
deep/ selector has alias as >>> . Component style normally applies only to the component's own template. Using /deep/ selector we can force a style down through the child component tree into all child component views. /deep/selector forces its style to its own component, child component, nested component, view children and content children.
:host /deep/ h3 { color: yellow; font-style: italic; } :host >>> p { color: white; font-style: Monospace; font-size: 20px; }

Instructor Notes:

Add instructor notes here.



The URL is relative to the application root which is usually the location of the index.html web page that hosts the application. The style file URL is not relative to the component file.

Special selectors
Component styles have a few special selectors from the world of shadow DOM style scoping:

:host is a pseudo-class selector that applies styles in the element that hosts the component. It means if a component has a child component using component binding then child component will use :host selector that will target host element in parent component. :host selector can be used in component with styles metadata as well as with styleUrls metadata of @Component decorator.
@Component({ --- styles: [':host { position: absolute; top: 10%; }'] })

:host-context() : Looks for a CSS class in any ancestor of the component host element, all the way up to the document root. It's useful when combined with another selector. :host-context selector is used in the same way as :host selector but :host-context is used when we want to apply a style on host element on some condition outside of the component view. For the example a style could be applied on host element only if a given CSS class is found anywhere in parent tree up to the document root. In our example we have following components in parent-child relationship.
:host-context(.my-theme) h3 { background-color: green; font-style: normal; }

/deep/ : selector to force a style down through the child component tree into all the child component views. The /deep/ selector works to any depth of nested components, and it applies both to the view children and the content children of the component.
deep/ selector has alias as >>> . Component style normally applies only to the component's own template. Using /deep/ selector we can force a style down through the child component tree into all child component views. /deep/selector forces its style to its own component, child component, nested component, view children and content children.
:host /deep/ h3 { color: yellow; font-style: italic; } :host >>> p { color: white; font-style: Monospace; font-size: 20px; }

Instructor Notes:

Component Styles (Contd…)

- :host:host is a pseudo-class selector that applies styles in the element that hosts the component.
- It means if a component has a child component using component binding then child component will use :host selector that will target host element in parent component.
- :host selector can be used in component with styles metadata as well as with styleUrls metadata of @Component decorator.
- Example

```
@Component({
  ---
  styles: [ ':host { position: absolute; top: 10%; }' ]
})
```

Instructor Notes:

Component Styles (Contd…)

- :host-context selector is used in the same way as :host selector but :host-context is used when we want to apply a style on host element on some condition outside of the component view.
- Example

```
:host-context(.my-theme) h3 {
    background-color: green;
    font-style: normal;
}
```

Instructor Notes:

Demo

- Component Demo Style

Instructor Notes:

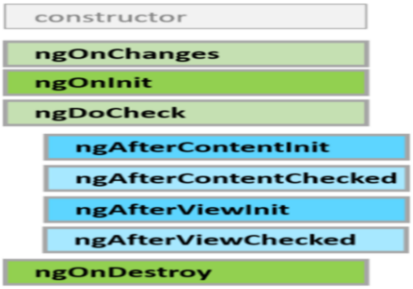
Component Lifecycle

- Each Angular application goes through a lifecycle.
- If we want to access the value of an input - to load additional data from the server for example - you have to use a lifecycle phase.
- The constructor of the component class is called before any other component lifecycle hook.
- For best practice inputs of a component should not be accessed via constructor.
- To access the value of an input for instance to load data from server component's life cycle phase should be used.

Instructor Notes:

Component Lifecycle (Contd…)

- A component has a lifecycle managed by Angular.
- Angular creates it, renders it, creates and renders its children, checks it when its data-bound properties change, and destroys it before removing it from the DOM.
- Angular offers **lifecycle hooks** that provide visibility into these key life moments and the ability to act when they occur.



Instructor Notes:

Component Lifecycle (Contd…)

- *After* creating a component by calling its constructor, Angular calls the lifecycle hook methods in the following sequence at specific moments:

Hooks	Purpose and Timing
ngOnChanges()	Respond when Angular (re)sets data-bound input properties. The method receives a SimpleChanges object of current and previous property values. Called before ngOnInit() and whenever one or more data-bound input properties change.
ngOnInit()	Initialize the directive/component after Angular first displays the data-bound properties and sets the directive/component's input properties. Called once, after the first ngOnChanges().
ngDoCheck()	Detect and act upon changes that Angular can't or won't detect on its own. Called during every change detection run, immediately after ngOnChanges() and ngOnInit().

Instructor Notes:

Component Lifecycle (Contd…)

Hooks	Purpose and Timing
ngAfterContentInit()	Respond after Angular projects external content into the component's view. Called once after the first ngDoCheck(). A component-only hook.
ngAfterViewInit()	Respond after Angular initializes the component's views and child views. Called once after the first ngAfterContentChecked(). A component-only hook.
ngAfterViewChecked()	Respond after Angular checks the component's views and child views. Called after the ngAfterViewInit and every subsequent ngAfterContentChecked(). A component-only hook.
ngOnDestroy()	Cleanup just before Angular destroys the directive/component. Unsubscribe Observables and detach event handlers to avoid memory leaks. Called *just before* Angular destroys the directive/component.

Instructor Notes:

Demo

- Component Life Cycle

Instructor Notes:

Add instructor notes here.

Summary

- Every component must be declared in some NgModule and a component can belong to one and only one NgModule
- exports key is nothing but the list of public components for NgModule.
- Angular4 Application is a tree of components and the top level component is nothing but the application.
- Components are Composable.
- Template for a component can be created using InlineTemplate and LinkedTemplate using template and templateUrl respectively.
- **styles** and **styleUrls** keys are used in components to work with styles in Angular 5

Add the notes here.