



Angular



Session Objectives for Angular

- Session 1
 - Introduction to Angular
 - Setting up Angular Project
 - Understanding the Angular Application Structure
 - Creating Components
- Session 2
 - Using Components
 - Managing properties
 - Managing events and callback
 - Managing UI Templates

Session Objectives for Angular

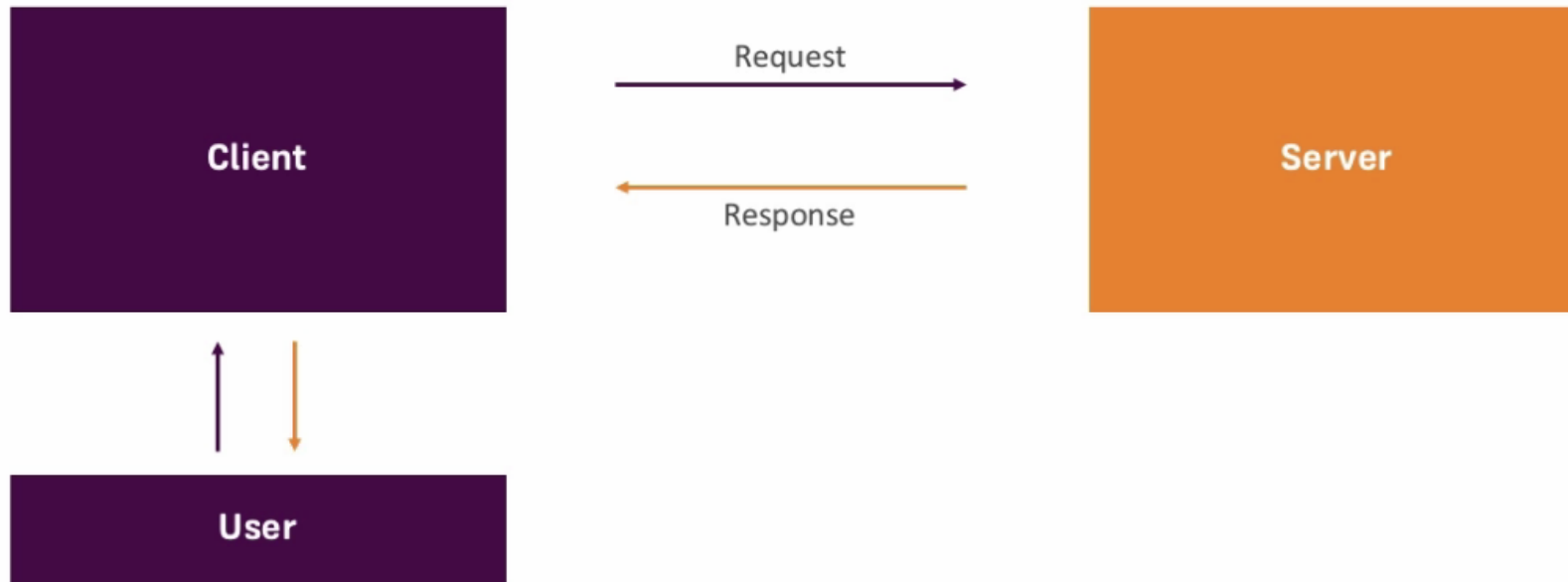
- Session 3
 - Using Angular Services
 - Using HTML Storage
 - Managing Forms and Validations
- Session 4: Angular Advanced
 - Introduction to Angular Pipe
 - Introduction to Angular Directives
 - Routing in Angular

Software/Plugins/Tools

- AngularJS CLI
- NodeJS & NodeJS modules
- Node Package Manager
- Express Application

Why Angular?

„Traditional“ Web Application



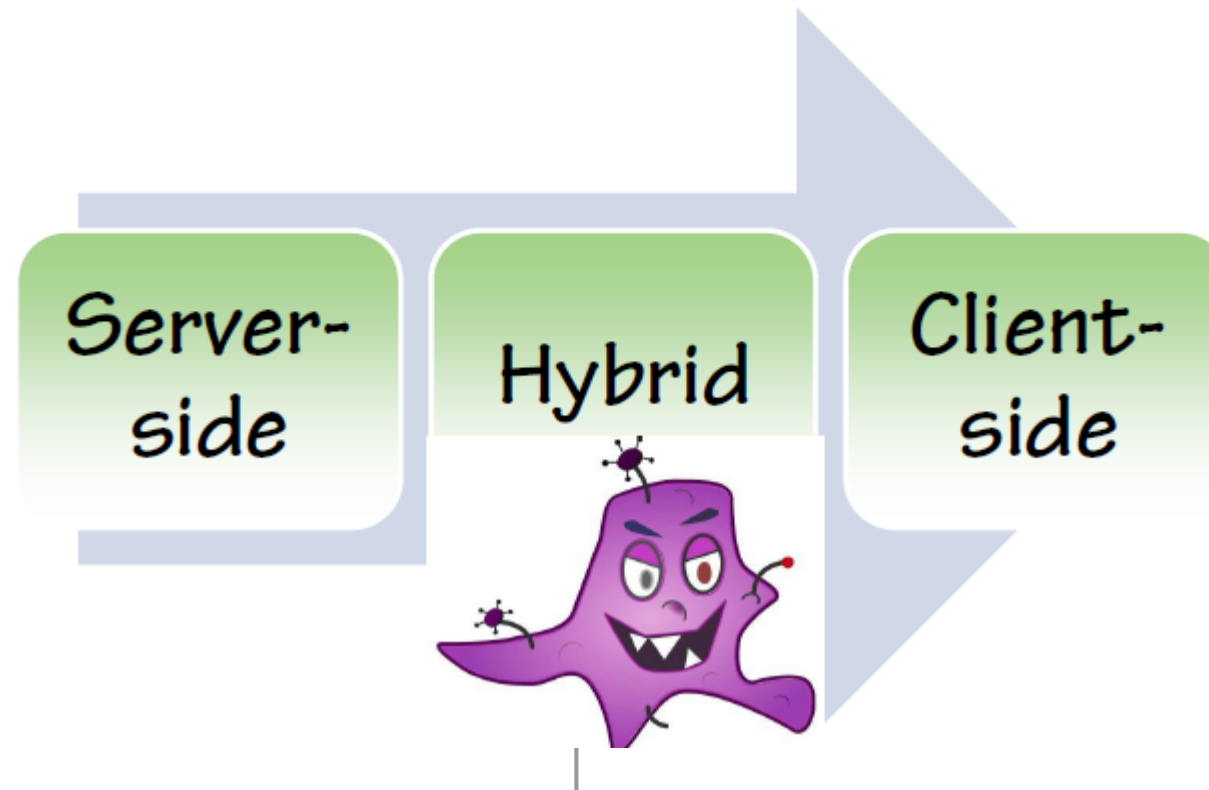
Why Angular?

„Traditional“ Web Application



SPAs

- **Single page applications**
 - Improved user experience



Why Angular?

Single Page Application



Why AngularJS?

- AngularJS was started to answer these questions...
 - Could the concepts like testable code, SRP, MVC, MVVM, be applied to client web apps.
 - Script based client side applications?
 - Could we have the best of both worlds—the capability of JavaScript and the pleasure of rapid, maintainable development

What is AngularJS?

- Created by Miško Hevery and Adam Abrons in 2009
- Open source, client-side JavaScript framework that promotes a high-productivity web development experience.

Why AngularJS?

- Doesn't require an explicit DOM refresh, is capable of tracking user actions, browser events, and model changes to figure out when and which templates (HTML) to refresh

What Angular is not

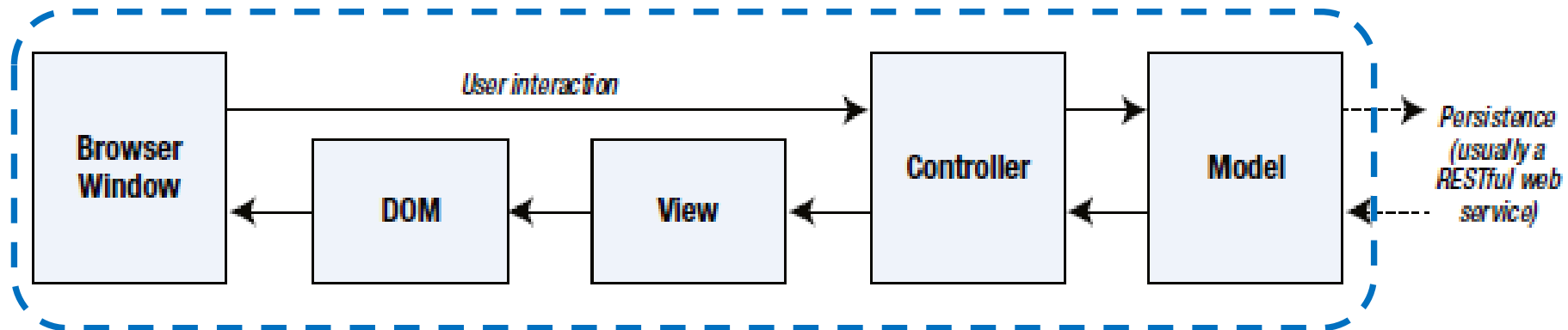
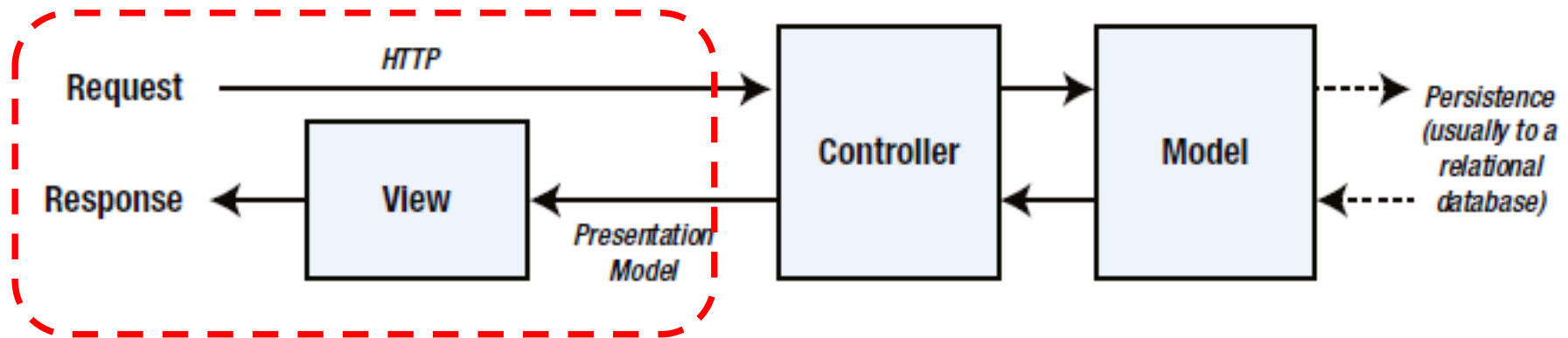
- A server-side framework/technology
- Javascript Library (Jquery,react, etc)
- Design Pattern
- Platform Lanugage (Java/.net)
- Plugin or extension

Angular Offerings

- Dynamic HTML
- Powerful Templates
- Fast Rendering
- Http Services
- Component Encapsulation
- Form & Input Handling
- Event Handling
- Routing
- Latest JS Standards
- And Many more...

What is AngularJS ?

- Server Side MVC



What is Typescript??

- A strict superset of Javascript with added features
- Maintained by Microsoft
- Optional static typing
- Class Based Object oriented programming
- Like C++/Java

Compatibility with JavaScript

- TypeScript is a strict superset of ECMAScript 2015, which is a superset of ECMAScript 5, commonly referred to as JavaScript.
- Class-based object-oriented programming.
- TypeScript may be used to develop JavaScript applications for client-side or server-side (Node.js) execution
- **European Computer Manufacturer's Association

TypeScript

TypeScript Walkthrough: Classes

```
1
2 class Greeter {
3     greeting: string;
4     constructor (message: string) {
5         this.greeting = message;
6     }
7     greet() {
8         return "Hello, " + this.greeting;
9     }
10 }
11
12 var greeter = new Greeter("world");
13
14 var button = document.createElement('button')
```

Run JavaScript

```
1 var Greeter = (function () {
2     function Greeter(message) {
3         this.greeting = message;
4     }
5     Greeter.prototype.greet = function () {
6         return "Hello, " + this.greeting;
7     };
8     return Greeter;

13 button.onclick = function () {
14     alert(greeter.greet());
```

TypeScript

Why should I use TypeScript?

General

Strong Typings

Allows compile-time Errors, IDE support
(autocomplete, errors, ...)

Next Gen JS Features

Classes, Imports / Exports, ...

Missing JS Features

Interfaces, Generics, ...

With Angular 2

Documentation & Support

Has by far the most Documentation and
Example-base

Main Language

Angular 2 chose TypeScript as main
Language

Angular App Execution Cycle

- How does it work??
- main.ts
- app module
 - index.ts
 - app.component.ts
 - selector
 - templateUrl
 - styleURLs
 - AppComponent class
 - title property

Components : Decorator

- Main way to build and specify elements and logic on the page
- @Component decorator gets imported from Angular Core Lib.
- Decorators allow us to transpose normal TypeScript classes into a Component .
- Components are custom HTML elements

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

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

Components : Templates

- Each components need some html support to displays its content which is called a template.
- Template can be linked to an external .html file or can be contained in this same .ts file. (only one .html per .ts)

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

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

Components: Styles

- Each components could optionally have a style linked to a .css or defined in the .ts file.
- This is an array [] , so multiple .css files can be provided.

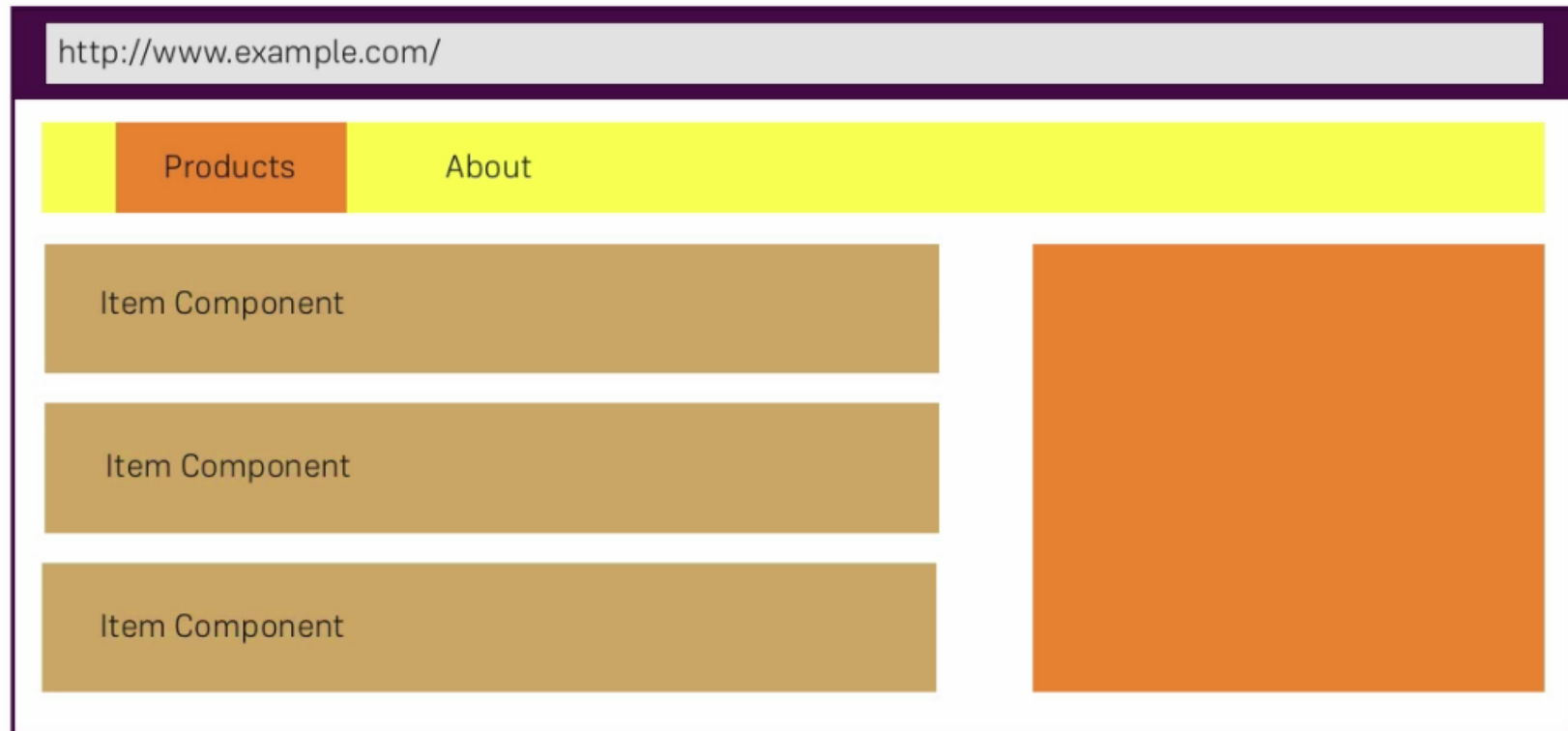
```
import { Component } from '@angular/core';

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

Views

- Lets say this our page

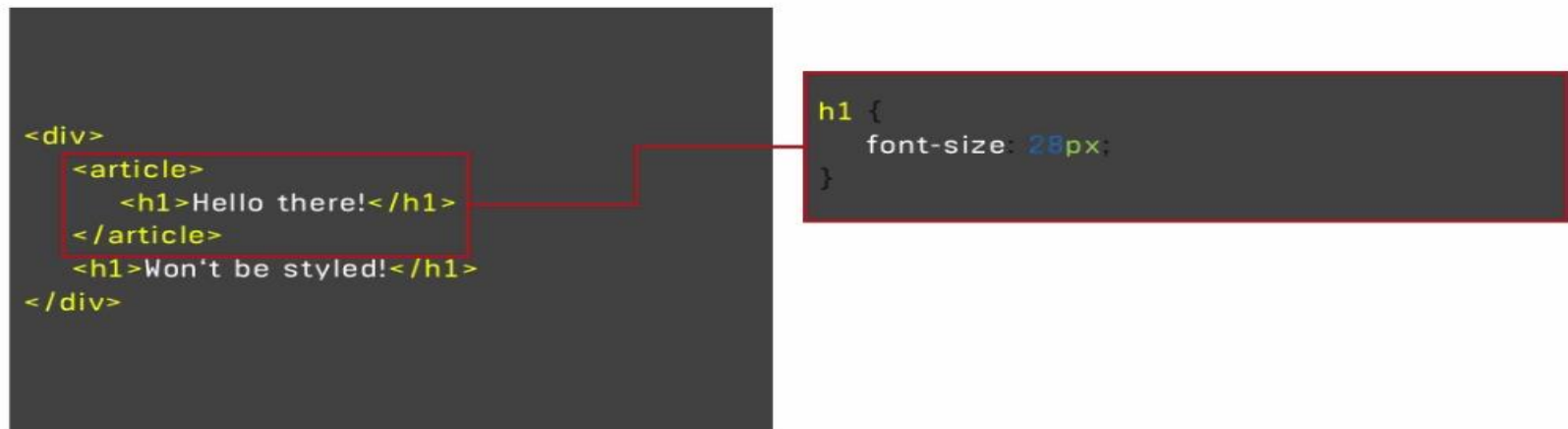
What is View Encapsulation?



Views

- HTML elements in Angular have a separate shadow DOM behind the scenes
- The h1 elements on the page don't get the font-size applied as the browser is displaying the shadow DOM

The Shadow DOM



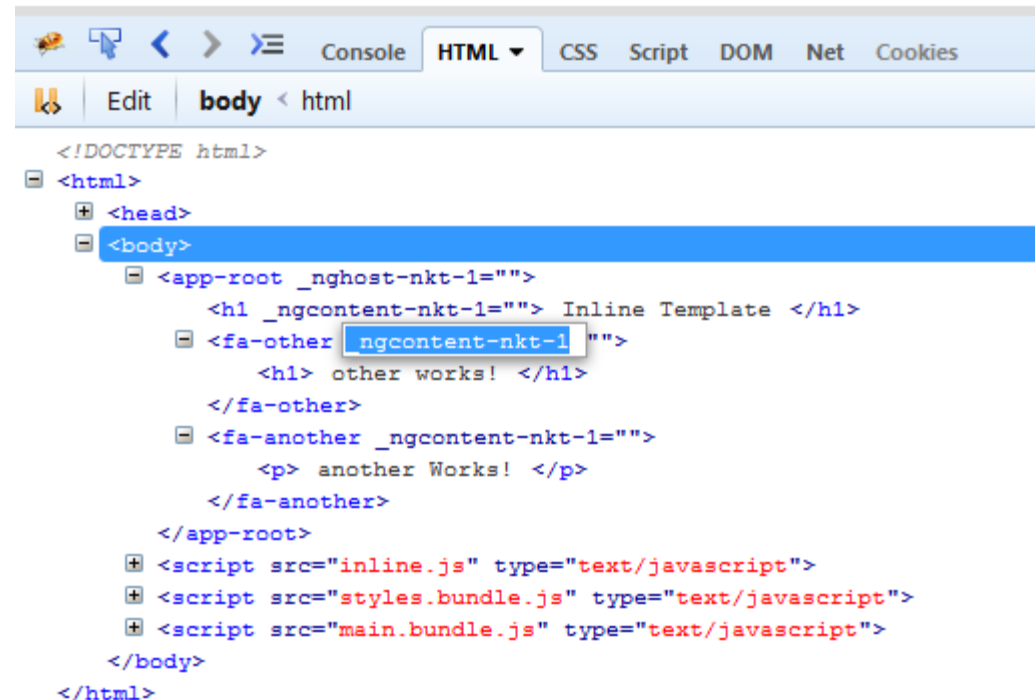
Views

- View the HTML in FireFox Dev Tools & watch the Angular attribute that got added
- Its only there on the first `<h1>` tag and not the one inside `<fa-other>`

Inline Template

other works!

another Works!



```
<!DOCTYPE html>
<html>
  <head>
  <body>
    <app-root _ngghost-nkt-1="">
      <h1 _ngcontent-nkt-1=""> Inline Template </h1>
      <fa-other ngcontent-nkt-1="">
        <h1> other works! </h1>
      </fa-other>
      <fa-another _ngcontent-nkt-1="">
        <p> another Works! </p>
      </fa-another>
    </app-root>
    <script src="inline.js" type="text/javascript">
    <script src="styles.bundle.js" type="text/javascript">
    <script src="main.bundle.js" type="text/javascript">
  </body>
</html>
```

DataBinding

Databinding = Communication



Data Binding

- In angular by default all data flow is unidirectional unless specified as two way

Databinding Methods

String Interpolation

```
{{ Expression resolving to a string }}
```

Property Binding

```
<button [disabled]="expression resolving to required value type">
```

Event Binding

```
<button (click)="expression handling the event">
```

Two-Way Binding

```
<input [(ngModel)]="bound model (e.g. object)">
```

Data Binding

- Components are also directives so we can bind to component properties

Property & Event Binding

Availability		
DOM Properties	Directive Properties	Component Properties
<code></code>	<code><div [ngClass]="..."></code>	<code><cmp [initObj]="..."></code>
<code></code>	<code><div (ngSubmit)="..."></code>	<code><cmp (rndEvent)="..."></code>

Lifecycle events

Lifecycle Hooks

#	Lifecycle Hook	Timing
1	ngOnChanges	Before #2 and when data-bound Property Value Change
2	ngOnInit	On Component Initialization, after first ngOnChanges
3	ngDoCheck	During every Angular 2 Change Detection Cycle
4	ngAfterContentInit	After inserting Content (<ng-content>)
5	ngAfterContentChecked	After every Check of inserted Content

Services

- Services are used to share data between components throughout the application
- Managing Data Access to a separate service keeps the component lean and focus on supporting view
- Services are invariably asynchronous and can provide data as a promise



THANK YOU