## Domain Model

The goal of this lecture is to use a [domain model](https://en.wikipedia.org/wiki/Domain_model) to store our data instead of plain old objects.

**Learning Outcomes**

* Why domain models are better for storing application state.
* How to structure our application to use a domain model.

**What are Domain Models?**

A good practice when writing Angular code is to try to isolate the data structures you are using from the component code.

Right now our data structure is just an array of objects which we initialised on the component.

We also have this toggle function which doesn’t do anything other than modify a property of the object passed in, i.e. the function could exist outside of the component completely and still do it’s job just fine.

Imagine if this was a much larger application, if all the data was stored inside components as plain objects it would be hard for a new developer to find out *where* the data is stored and *which* function to edit.

To solve this, let’s create a class that represents a single joke and attach the toggle function to that class.

This class is what we call a *Domain Model*, it’s just a plain class which we will use to store data and functions.

**Our Domain Model**

We create a simple class with 3 properties, a constructor and a toggle function, like so:

class Joke {

setup: string;

punchline: string;

hide: boolean;

constructor(setup: string, punchline: string) {

this.setup = setup;

this.punchline = punchline;

this.hide = true;

}

toggle() {

this.hide = !this.hide;

}

}

As we’ve mentioned before we can instantiate a class by using the new keyword and when that happens javascript calls the constructor function where we can have code that initialises the properties.

However the constructors we’ve used so far have not had any arguments, the one for the joke class above does. It initialises the properties of the joke instance from the arguments passed into the constructor.

We can pass those arguments in when we instantiate a joke like so:

let joke = new Joke("What did the cheese say when it looked in the mirror?", "Hello-Me (Halloumi)");

We also added a toggle function which just flips the value of the hide property.

We use this in a class function to distinguish between properties that are *owned* by the class instance vs. arguments that are passed in or declared locally by the

function.

So this.joke in our toggle function is explicitly saying the joke property on the class instance.

Now lets change our JokeListComponent so it uses our domain model, like so:

class JokeListComponent {

jokes: Joke[];

①

constructor() {

this.jokes = [

②

new Joke("What did the cheese say when it looked in the mirror?", "Hello-me

(

Halloumi)"),

new Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-

pony (Mascarpone)"),

new Joke("A kid threw a lump of cheddar at me", "I thought

‘

That

’

s not very

mature

’

"),

];

}

toggle(joke) {

joke.hide = !joke.hide;

}

}

1. We changed the type of our jokes property to Joke[]. Now TypeScript will throw an error if we accidentally insert something other than an instance of a Joke class in this array.
2. We also converted our jokes array to now contain instance of the Joke class instead of plain objects.

The final thing to do is to change the markup in the template so we call the joke.toggle() function on the joke instance and not the toggle(joke) function on the component, like so:

<

a class="btn btn-warning" (click)="joke.toggle()">Tell Me</a

>

**Summary**

Although its possible to store all your data as objects it’s both recommend and simple to encapsulate your data into domain models in Angular.

Using a domain model clarifies for all developers where you should put functions that need to act on the data, like our toggle function.

**Listing**

*script.ts*

import {platformBrowserDynamic} from '@angular/platform-browser-dynamic';

import {NgModule} from '@angular/core'; import {BrowserModule} from '@angular/platform-browser'; import {Component} from '@angular/core'; class Joke {

public setup: string;

public punchline: string;

public hide: boolean;

constructor(setup: string, punchline: string) {

this.setup = setup;

this.punchline = punchline;

this.hide = true;

}

toggle() {

this.hide = !this.hide;

}

}

@Component({

selector: 'joke-list',

template: `

<

div class="card card-block"

\*ngFor="let joke of jokes">

<h4 class="card-title">{{joke.setup}}</h4>

<p class="card-text"

[hidden]="joke.hide">{{joke.punchline}}</p>

<a class="btn btn-warning" (click)="joke.toggle()">Tell Me</a>

<

/div

>

`

})

class JokeListComponent {

jokes: Joke[];

constructor() {

this.jokes = [

new Joke("What did the cheese say when it looked in the mirror?", "Hello-me

(

Halloumi)"),

new Joke("What kind of cheese do you use to disguise a small horse?", "Mask-a-

pony (Mascarpone)"),

new Joke("A kid threw a lump of cheddar at me", "I thought

‘

That

’

s not very

mature

’

"),

];

}

}

@NgModule({

imports: [BrowserModule],

declarations: [JokeListComponent],

bootstrap: [JokeListComponent]

})

export class AppModule {

}

platformBrowserDynamic().bootstrapModule(AppModule);

POINTS TO NOTE

1. **selector** is the property name of @Component annotation object
2. \***ngFor**=**"let** joke **of** jokes" syntax