## Decorators

We’ve seen in the quickstart that in Angular we can *decorate* a class with extra info using the @ syntax, like so:

@Component({

selector: "thingy",

template: `foo`

})

class MyComponent {

}

This is a new feature that will *probably* make it into the *ES7* version of JavaScript, it’s not available right now however even in the ES6 version.

However the functionality is available in TypeScript, so we can already make use it.

It allows us to *decorate* classes and functions, similar to annotations in java and decorators in python.

Specific Angular implementations might be more complex and harder to read and understand but the concept is actually quite simple.

**Simple no-argument decorator**

I’m going to explain by creating a decorator called @course for our Person class

@course

class Person {

firstName;

lastName;

constructor(firstName, lastName) {

this.firstName = firstName;

this.lastName = lastName;

}

}

@course is just a *function*, like so:

function course(target) {

Object.defineProperty(target.prototype, 'course', {value: () => "Angular 2"})

}

The first argument to the course function is the target.

This is the *thing* the decorator is attached to, so for a class it’s going to be the *function constructor* for that class, the *under-the-hood* implementation of a class.

Knowing this we can actually dynamically add a function to our Person class by using the Object.defineProperty function,

The details of the Object.defineProperty function are beyond the scope of this chapter. We use it to add a function called course onto the class it decorates and for now this function just returns the string "Angular 2".

We can now call asim.course() and this prints out "Angular 2":

let asim = new Person("Asim", "Hussain");

console.log(asim.course()); // Angular 2

**Decorators with arguments**

But how do we pass arguments to our decorator, like the way the @Component decorator works?

We create a function that *returns* a decorator, like so:

function Course(config) { // 1

return function (target) {

Object.defineProperty(

target.prototype,

'course',

{value: () => config.course} // 2

)

}

}

1. We pass a config object to the *outer* Course function.
2. Then use that config in the returned *inner* decorator function.

Now we can use this decorator like so:

@Student({

course: "Angular 2"

})

class Person {

}

**Summary**

Decorators are a new feature of TypeScript and used throughout the Angular code, but they are nothing to be scared of.

With decorators we can configure and customise our classes at design time.

They are just functions that can be used to add meta-data, properties or functions to the thing they are attached to.

A collection of useful decorators, for use in your projects or just to read and learn, can be found here: <https://github.com/jayphelps/core-decorators.js>

**Listing**

[https://github.com/codecraftpro/angular2-sample-code/blob/master/2.es6typescript/11.decorators/script.ts](https://github.com/codecraftpro/angular2-sample-code/blob/master/2.es6-typescript/11.decorators/script.ts)

*script.js*

function Student(config) {

return function (target) {

Object.defineProperty(target.prototype, 'course', {value: () =>

config.course})

}

}

@Student({

course: "angular3"

})

class Person {

constructor(private firstName, private lastName) {

}

public name() {

return `${this.firstName} ${this.lastName}`;

}

protected whoAreYou() {

return `Hi i'm ${this.name()}`;

}

}

let asim = new Person("Asim", "Hussain");

//noinspection TypeScriptUnresolvedFunction

console.log(asim.course());