Decorators

PROGRAMMING WITH TYPESCRIPT

Objectives

- To understand what decorators are
- To understand how to implement a decorator and a decorator factory
- To understand how to decorate properties

2

Introduction

- Decorators provide a means through which existing classes and class members can be annotated and modified – used extensively in Angular 2+
  - Simply a way of wrapping one piece of code with another – literally decorating it
  - May have heard of functional composition or higher-order functions – same thing!
- They are an experimental feature for JavaScript that are available in TypeScript – so they may change in future releases!
- Decorators use a special @expression syntax where the expression evaluates to a function which takes information about the decorated declaration

```
//@myDecorator

function myDecorator(target: any) {
    //do some stuff with target
}
```

3

My First Decorator

- Where you can use a decorator is dependent on the parameters you supply the function with

```
//A simple class decorator

function simpleDecorator(target: any) {
    console.log(`My first decorator was called`)
}

@simpleDecorator
class DecoratedClass {

}
```

- Our (class) decorator takes the constructor as its only argument and simply logs the message to the console
- We don't need to instantiate the class for the decorator to run!

4

Decorator Factories

- We can use decorator factories to be able to provide our decerators with parameters
- Remember: a Decorator should evaluate to a function

```
function DecoratorFactory(name: string) {
    return function(target: Function) {
        console.log(`${name} decorator was called`)
    }
}

@DecoratorFactory("factory")
    class DecoratedClass {
}
```

5

CLASS DECORATOR PARAMETER

```
function merge(toMerge: Object) {
    return function (target: any) {
        for (let prop in toMerge) {
            target.prototype[prop] = toMerge[prop];
        }
    }
}

let user = {
    name: 'John Smith',
    age: 22,
    instructor: true
}

@merge(user)
class DecoratedClass {
    constructor() {};
    test = true;
}

let thing = new DecoratedClass();
console.log((<any>thing).name);
//cast to 'any' in order to use the name property
```

The runtime automatically passes the parameters to the evaluated function of our Decorators

In the case of a Class decorator this is the constructor function itself

6

Method Decorators

- Remember we can decorate any class or class member
- In the case of a method decorator the arguments required for the decorator are:
  - The target – the class prototype
  - The method name
  - The method descriptor

```
function readOnly(target: any, methodName: string, descriptor?: PropertyDescriptor) {
    descriptor.writable = false;
    descriptor.enumerable = false;
}

class DecoratedClass {
    @readOnly
    sayHello() { console.log("Hello") }
}

let thing = new DecoratedClass();
thing.sayHello()
thing.sayHello = false; //error (in strict mode - silent fail otherwise)
```

7

Objectives

- To understand what decorators are
- To understand how to implement a decorator and a decorator factory
- To understand how to decorate properties

8

QuickLab 7 - Decorators

- Examine when decorators are applied by creating a simple decorator

9

Hackathon Part 2 – Type-safe HTTP Requests

- In this part Hackathon, you will build on a partially developed solution (whether that be your previous iteration or the provided starting point) for QA Cinemas' website by allowing submission of the user data from the form to a remote backend. This should be simulated by using json-server. All the necessary tools, knowledge and techniques have been covered in the course so far.
- This part of the Hackathon is intended to help you develop your skills and knowledge to be able to use TypeScript to help submit type-safe data from a 'Sign-Up' form for users of the QA Cinemas website.

10