Decorators in TypeScript

SoftUni Team

Technical Trainers

Software University

SoftUni

Software
University

sli.do

#TypeScript

Table of Contents

Introduction to Decorators

Definition

- Used in frameworks like **Angular**, **MobX** and others

- They are used to extend a functionality or add meta-data

- Use the form **@example** where **example** must evaluate to function that will be called at runtime

```javascript
function example(target) {
    //some code logic
}
```

Decorate

- We can decorate **five** different **things**:
    - Class definitions, properties, methods, accessors, parameters
- The function that we implement is **dependent** on the **thing** we are **decorating**
- The **arguments** required to **decorate a class** are different to the **arguments** required **to decorate a method**

Enable Decorators

- In the **tsconfig.json** file:

```json
{

    "compilerOptions": {
      "experimentalDecorators": true,
      "emitDecoratorMetadata": true
    }

}
```

```
tsc --experimentalDecorators --emitDecoratorMetadata
```

Decorator Evaluation

- There is well **defined order** to how decorators are applied:

  - **Parameter Decorators**, followed by **Method**, **Accessor**, or **Property Decorators** are applied for each **instance member**

  - **Parameter Decorators**, followed by **Method**, **Accessor**, or **Property Decorators** are applied for each **static member**

  - **Parameter Decorators** are applied for the **constructor**

  - **Class Decorators** are applied for the **class**

Syntax and Basic Usage

Syntax and Basic Usage

```typescript
function classDecorator(constructor: Function) {
    console.log("Class decorator called.");
}


@classDecorator
class ExampleClass {}
```

```typescript
function methodDecorator() {
    console.log("Method decorated.");
}
class ExampleClass {
        @methodDecorator
        exampleMethod(){}
}
```

Types of Decorators

Types

- **Class Decorators**
- **Method Decorators**
- **Accessor Decorators**
- **Property Decorators**
- **Parameter Decorators**
- **Decorators Factories**

Class Decorators

Definition

- **Class Decorator** is **added just before** the class declaration

- The Class Decorator is applied to the **constructor** of the class

- Used to **observe**, **modify** or **replace** a class definition

- If the Class Decorator **returns a value**, it will **replace** the class declaration with the **provided constructor function**

```
function Frozen(construtor: Function) {
    Object.freeze(construtor);
    Object.freeze(construtor.prototype);
}

@Frozen class Person {
    constructor(private name: string) { }
}
```

@Frozen is a class decorator

Decorators and Inheritance

- Subclasses do not inherit the decorations of the super class

- Every subclass needs to be decorated on its own

```typescript
@ClassDecorator class Person {
    constructor(public name: string) { }
    @enumerable(false) greet() {
        return `Hello ${this.name}`
    }
}
class Teacher extends Person {
    constructor(private subject: string, name: string) {
        super(name);
    }
    introduce() {
        // some code logic
    }
}
```

> The Teacher class does not inherit the decorator

Method Decorators

Definition

- The decorator function takes **three** arguments:
  - **target** - the parent **class**
  - **key** - the **name** of the function
  - **descriptor** - the actual **function** itself

```
function enumerable(value: boolean) {
    return function (target: Object,
                     key: string,
                     descriptor: PropertyDescriptor) {
        descriptor.enumerable = value;
    };
}
```

Example: Method Decorator

```typescript
function Confirmable(message: string) {
    return function (target: Object,
      key: string,
      descriptor: PropertyDescriptor) {
        const original = descriptor.value;
        descriptor.value = function (...args: any[]) {
            const allow = confirm(message);
            if (allow) {
                const result = original.apply(this,
args);
                return result;
            } else { return null; }
        };
        return descriptor;
    };
}
```

Accessor Decorators

Definition

- TypeScript **does not allow** to decorate **both** the **getter** and the **setter**

- The **Property Descriptor** combines **both get** and **set** not each declaration separately

- Takes the following three arguments:
  - Either the **constructor function of the class** for a static member, or the **prototype of the class** for an instance member
  - The **name** of the member
  - The **Property Descriptor** for the member

Example: Accessor Decorator

```typescript
class Point {
    private _x: number;
    private _y: number;
    constructor(x: number, y: number) {
        this._x = x;
        this._y = y;
    }

    @configurable(false)
    get x() { return this._x; }

    @configurable(false)
    get y() { return this._y; }
}
```

Property Decorators

Definition

- Property decorator can only be used to **observe** that a property of a specific name has been declared for a class

- Takes the following **two** arguments:

  - Either the **constructor function** of the class for a static member, or the **prototype** of the class for an instance member

  - The **name** of the member

Example: Accessor Decorator

```
class Greeter {
    @format("Hello, %s")
    greeting: string;

    constructor(message: string) {
        this.greeting = message;
    }

    greet() {
        let formatString = getFormat(this, "greeting");
        return formatString
            .replace("%s",   this.greeting);
    }
}
```
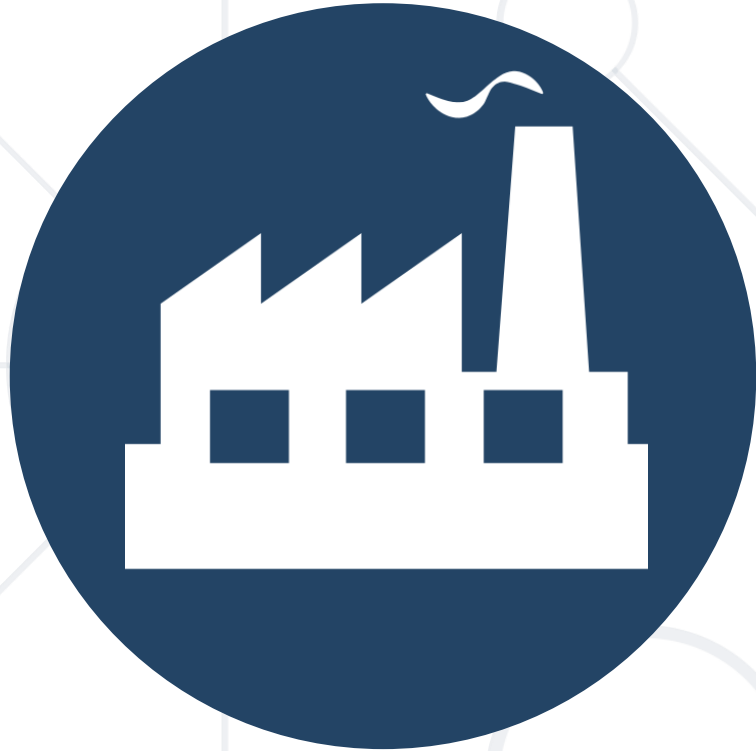
Parameter Decorators

- Parameter decorator can be used in parameter of **function** or **method**

```
class MyClass {
    myMethod(@MyParameterDecorator param: string) {
        // method implementation
    }
}
```

Decorator Factories

Definition

- **Function** that **returns** the **decorator** function **itself**

- Gives the flexibility to pass **custom data** when needed

- Mainly used in **method** and **property decoration**

```typescript
function enumerable(value: boolean) {
    return function (target: Object, propertyKey:
string, descriptor: PropertyDescriptor) {
        descriptor.enumerable = value;
    };
}
```

Multiple Decorators

- Decorators are **composable**
- We can chain **multiple decorators** for each class declaration, method, property, access or parameter
- In those cases the decorators are applied from **top to bottom**

```
@Frozen
@Configurable
@OtherDecorator
class Person {
    constructor(private name: string) { }
}
```

Advanced Usage

Advanced Usages

- Dependency Injection with Decorators

- Logging with Decorators

- Authorization with Decorators

- Validation with Decorators

Summary

- **Decorators are basically functions**

- **Add additional functionalities to a class or class members**

- **We can decorate class declaration, methods, accessors, properties and parameters**

- **To decorate different classes or class members the decorator functions takes different arguments**

Questions?
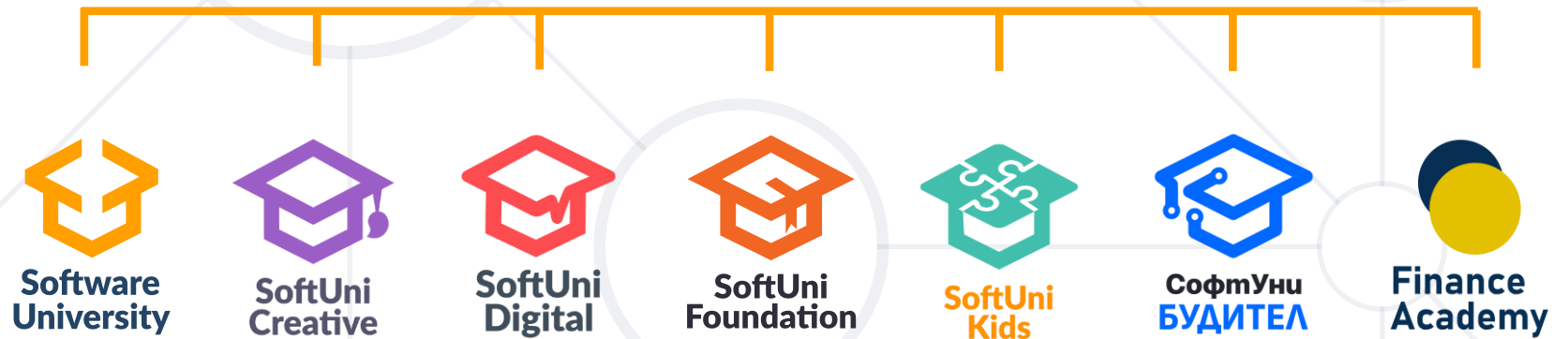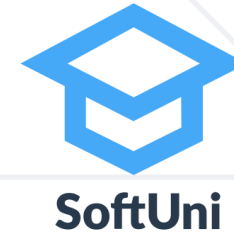
SoftUni Diamond Partners

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
    - softuni.bg, about.softuni.bg
- Software University Foundation
    - softuni.foundation
- Software University @ Facebook
    - facebook.com/SoftwareUniversity

License

- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**

- Unauthorized copy, reproduction or use is illegal

- © SoftUni – https://about.softuni.bg/

- © Software University – https://softuni.bg