Decorator

Decorators

A Decorator is a special kind of declaration that can be attached to a class declaration, method, accessor, property, or parameter.

Sometimes it is required to have additional features to support annotating or modifying classes and class members. Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members.

Decorators are a stage 2 proposal for JavaScript and are available as an experimental feature of TypeScript.

NOTE Decorators are an experimental feature that may change in future releases of Typescript.

Decorators use the form @expression, where expression must evaluate to a function that will be called at runtime with information about the decorated declaration.

Check the tutorial lab about Applying a Decorator in this module for more details about decorators.

Applying A Decorator

Decorators are a new concept in TypeScript. Because they are fairly new, they are only available for use if you use ES2015 or above as your target for compilation. Decorators provide a way to add both annotations and a meta-programming syntax for class declarations and members. Their purpose is to essentially be a modifying function for existing objects such as classes, parameters, and methods, allowing you to replace, modify, or otherwise alter these objects functionally.

In this exercise, we are going to use a Decorator to apply animation to our movingElement class, and finish applying the elements to the page.

1. Copy your advancedTypeScriptLab folder and name the copy advancedTypeScriptLab_animated. Open up your advancedTypeScriptLab.ts in your advancedTypeScriptLab_animated file in Visual Studio Code.

2. First we need to create a decorator function.

```
function animated(constructor: Function) {
  constructor.prototype.animated = true;
  return constructor;
}
```

Note that what is being passed into this function is the constructor function of the class. We are then modifying the prototype of the constructor function to include an animated variable value, then returning the constructor.

3. Next, we need to apply some behavior to our class to react to this value if it is present. We will add a animated property and set it to false and will also add a check in the constructor of the class.

```
class movingElement implements Rotater, Mover {
  rotate: (elem: HTMLElement) => any
  move: (elem: HTMLElement) => any
  moveBack: (elem: HTMLElement) => any
  rotateBack: (elem: HTMLElement) => any
  animated: false;

  element: HTMLElement
  constructor(elem: HTMLElement) {
    elem.onmousedown = () => {
      this.move(elem);
    }
    elem.onmouseup = () => {
      this.moveBack(elem);
    }
    elem.onmouseover = () => {
      this.rotate(elem);
    }
    elem.onmouseout = () => {
      this.rotateBack(elem);
    }
    if (this.animated) {
       elem.style.transition = "transform .5s ease"
    }
    this.element = elem;
  }
}
```

Note the block starting with if (this.animated). This is reacting to whether or not the animated variable is present on the constructor and applying the transition to the element.

4. Now that you have all of this set up, applying the decorator function is easy, simply type @animated on the line immediately preceding the class declaration.

```
@animated
class movingElement implements Rotater, Mover {
  rotate: (elem: HTMLElement) => any
  move: (elem: HTMLElement) => any
  moveBack: (elem: HTMLElement) => any
  rotateBack: (elem: HTMLElement) => any
  animated: false;

  element: HTMLElement
```

```
    constructor(elem: HTMLElement) {
        elem.onmousedown = () => {
            this.move(elem);
        }
        elem.onmouseup = () => {
            this.moveBack(elem);
        }
        elem.onmouseover = () => {
            this.rotate(elem);
        }
        elem.onmouseout = () => {
            this.rotateBack(elem);
        }
        if (this.animated) {
             elem.style.transition = "transform .5s ease"
        }
        this.element = elem;
    }
}
```

Caveat: You will likely get various different errors if you hover over your @animated decorator call in Visual Studio Code. VS Code does not quite have complete support for Decorators built into its TypeScript autocomplete and error checking functionality, so errors can exist that will not when you run the compiler.

Now you have a decorator applied to your function. This isn't going to work just yet though - we need to enable the experimentalDecorators option in our compiler configuration. To enable support for decorators, you must enable the experimentalDecorators compiler option either on the command line or in your tsconfig.json:

Command Line:

```
tsc advancedTypeScriptLab.ts --target ES5 --experimentalDecorators
tsconfig.json:

{
    "compilerOptions": {
        "target": "ES5",
        "experimentalDecorators": true
    }
}
```