Structuring Code

Roland Guijt
INDEPENDENT SOFTWARE DEVELOPER AND TRAINER

@rolandguijt www.rmgsolutions.nl

Module Overview

The global namespace

Namespaces

Modules

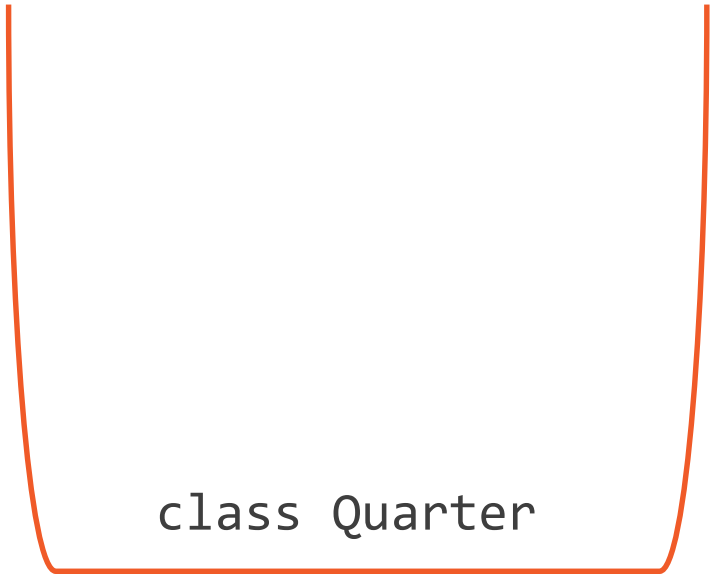
Generics

Advanced topic: Decorators

The Global Namespace

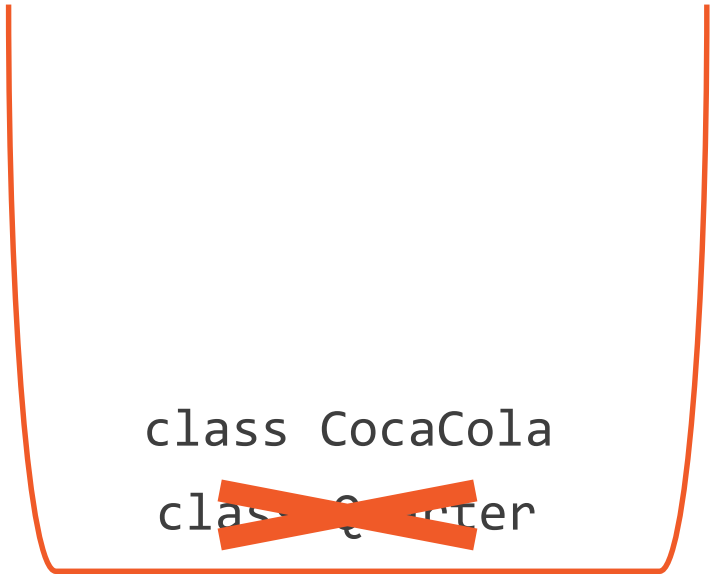
```
class Quarter
```

The Global Namespace



class CocaCola

The Global Namespace



class Quarter

Grouping Classes

- Less chance of naming conflicts
- Added semantics
- Extra level of encapsulation

Nesting and Aliases

```
namespace Money {
    export namespace Coins {
        export class Quarter {
        }
    }
}

var quarter = new Money.Coins.Quarter();
```

Nesting and Aliases

```
namespace Money {

    export namespace Coins {

        export class Quarter {

        }
    }
}

import coins = Money.Coins;

var quarter = new coins.Quarter();
```

Modules

- Way to group code
- The file is the container
- Other files have to import
- No reference paths needed
- All or nothing

Module Loader

External library

Loads module when imported

For bigger projects

Pick and choose

Selectable tsc output format

Dynamic loading

A Collection Class

```
class StringCollection {
    add(item: string) {
        //add item
    }
}
```

Another Collection Class

```
class NumberCollection {
    add(item: number) {
        //add item
    }
}
```

Another Collection Class

```
class QuarterCollection {
    add(item: Quarter) {
        //add item
    }
}
```

Generics

```
class Collection<T> {
    add(item: T) {
        //add item
    }
}
let stringCollection = new Collection<string>();
```

Generics

```
class Collection<string> {
    add(item: string) {
        //add item
    }
}
```

Instantiating a Generic Class

```
let numberCollection = new Collection<number>();

let quarterCollection = new Collection<Quarter>();
```

Contraints

```
class CoinCollection<T> {

    private allCoins = new Array<T>()

    add(item: T) {
        this.allCoins.push(item);
    }

    getTotalValue() {
        let total = 0;
        this.allCoins.forEach(c => total += c.value);
        return total;
    }
}
```

Contraints

```
class CoinCollection<T extends Coin> {

    private allCoins = new Array<T>()

    add(item: T) {
        this.allCoins.push(item);
    }

    getTotalValue() {
        let total = 0;
        this.allCoins.forEach(c => total += c.value);
        return total;
    }
}
```

Deriving From Generic Classes

```
class QuarterCollection extends CoinCollection<Quarter>
{
    //add functionality for quarters
}
```

Decorators

- Experimental
- Angular 2
- Functions
- Classes, methods, properties or parameters
- Reusable

Decorators

```
@Component({
        template: “<h1>Hello {{ name }}</h1>”
})
class HomePageComponent {
        name: “Angular2”
}
```

Summary

Expose to the global namespace as less as possible

Use namespace or modules for larger applications

Generics save you lines of code

Decorators add reusable functionality to methods, classes, properties and parameters