

TypeScript and Object-Oriented Programming

Frameworks and Architectures of the Web

Spring 2018


Today's Program

TypeScript Walkthrough

A Small Exercise

Break

Feedback for Web Project: Implementation

 **What Arms are These For You!**
Vanessa Wagner
EP03

Course Outline

Frameworks

Week 13	Week 14	Week 15	Week 16	Week 17	Week 18	Week 19	Week 20
No Lecture (Easter Holiday)	Web Apps in React (Part 1)	Web Apps in React (Part 2)	TypeScript and Object-Oriented Programming	Web Apps in Angular (Part 1)	Web Apps in Angular (Part 2)	Augmented Reality in the Browser	No Lecture (End of Semester)
			Creative Implementation of a Web App				
Web Project - Implementation							

TypeScript

What is it?

A typed superset of JavaScript that compiles down to plain JavaScript.

Reasons to love it:

Static, compile-time type checking (no “silent fails”) - catch errors before you do.

Typed language means its much easier for IDEs to auto-complete code.

“Real” OO - proper classes, methods, encapsulation, interfaces etc.

Incorporates some of the nice features of ECMAScript 2015, such as arrow functions, optional parameters, let and string templates without the need to worry about compatibility (it can compile these features down to ES3).

Produces arguably cleaner, more robust code (encourages typing).

Is used in Angular 2.0!

NO TYPES

“I don’t know what I want”

Dynamic and flexible and easier to learn, but prone to runtime errors:

“object.x is undefined”

Difficulty understanding or interpreting another person’s code:

“Does this function accept a string or an object? I dunno.”

TYPES

“I know what I want”

You can specify types and have the compiler check them for you so you know about your bugs before your users do.

Produces arguably more robust, maintainable code.

Allows you to create well-defined interfaces to your classes and libraries, making them easier for other developers to work with.

Download the TypeScript Project

Use the terminal to cd to your project folder then run:

```
npm install
```

```
npm start
```



```
class Person {  
    private firstName: string;  
    private lastName: string;  
}
```

A Simple Class in TypeScript

We can define a class in TypeScript using the class keyword.

Within the class, we can define private members `firstName` and `lastName` and type them as strings.

```
class Person {  
  
    private firstName: string;  
    private lastName: string;  
  
    constructor(firstName: string, lastName: string) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
}
```

Adding the Constructor

Next, we add a constructor that allows us to create multiple instances of a Person object.

This allows us to create a Person object via their first and last name. The constructor assigns these values to the class's private fields.

In TypeScript, a class's fields and functions or public by default, unless you explicitly prefix them with the `private` keyword.


```
class Person {  
  
    private firstName: string;  
    private lastName: string;  
  
    constructor(firstName: string, lastName: string) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
  
    getFullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
}
```

Add a Public Method

Then, we add the public `getFullName()` method.

Do you notice the text encased in the back ticks? (``${this.firstName} ${this.lastName}``)? This is a template literal, a new ES2015 feature supported in TypeScript. It is an alternative to using: `this.firstName + ' ' + this.lastName`

```
class Person {  
  
    private firstName: string;  
    private lastName: string;  
  
    constructor(firstName: string, lastName: string) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
    }  
  
    getFullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
}  
  
let tim = new Person('Tim', 'Wray');  
let john = new Person('John', 'Holmes');  
  
document.writeln(tim.getFullName());  
document.writeln(john.getFullName());
```

Create Instances of Person

Now that we've defined the Person class, let's create some instances.

Notice the `let` keyword? It is similar to `var`, except that variables defined with `let` are block-scoped.

Try calling `tim.firstName` Your code won't compile because TypeScript won't let you access the private `firstName` field.

```

class Person {

    private firstName: string;
    private lastName: string;

    constructor(firstName: string, lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}

let tim = new Person('Tim', 'Wray');
let john = new Person('John', 'Holmes');

document.writeln(tim.getFullName());
document.writeln(john.getFullName());

```

TypeScript

```

var Person = (function () {

    function Person(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    Person.prototype.getFullName = function () {
        return this.firstName + " " + this.lastName;
    };

    return Person;

})();

var tim = new Person('Tim', 'Wray');
var john = new Person('John', 'Holmes');

document.writeln(tim.getFullName());
document.writeln(john.getFullName());

```

JavaScript

```
class Person {  
  
    private firstName: string;  
    private lastName: string;  
    private dateOfBirth: Date;  
  
    constructor(firstName: string, lastName: string, dateOfBirth:  
Date) {  
        this.firstName = firstName;  
        this.lastName = lastName;  
        this.dateOfBirth = dateOfBirth;  
    }  
  
    getFullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
  
}  
  
let tim = new Person('Tim', 'Wray', new Date('1 July 1984'));  
let john = new Person('John', 'Holmes', new Date('25 May 1982'));  
  
document.writeln(tim.getFullName());  
document.writeln(john.getFullName());
```

Add a dateOfBirth field

We can add another field, dateOfBirth of type Date so that it only accepts a JavaScript Date object.

```
class Person {  
    constructor(  
        public firstName: string,  
        public lastName: string,  
        public dateOfBirth: Date  
    ) { }  
  
    getFullName(): string {  
        return `${this.firstName} ${this.lastName}`;  
    }  
}  
  
let tim = new Person('Tim', 'Wray', new Date('1 July 1984'));  
let john = new Person('John', 'Holmes', new Date('25 May 1982'));  
  
document.writeln(tim.getFullName());  
document.writeln(john.getFullName());
```

Make our private fields public

Let's decide to make our private class members `firstName`, `lastName` and `dateOfBirth` public. We can do this by changing their `private` keyword to `public` (or remove it altogether).

Alternatively, we can refactor the code so that our private fields are explicitly declared public within the class's constructor, as shown on the left.

Additional fields / methods

We'll add an additional private field called `now` which is typed as a function that returns a new `Date()` object. (Calling `new Date()` retrieve's the current time).

Then, we'll add another method called `getAgeMessage()` which uses the private `getAgeThisYear()` method to determine how old that person will be turning this year.

Notice the arrow function syntax in the `now` declaration and the ternary expression in `getAgeMessage()`.


```
class Person {

    private now: () => Date = () => new Date();

    constructor(
        public firstName: string,
        public lastName: string,
        public dateOfBirth: Date
    ) { }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }

    getAgeMessage(): string {
        let ageThisYear = this.getAgeThisYear();
        let birthdayHasHappened: boolean = false;
        if (this.now().getMonth() > this.dateOfBirth.getMonth()) {
            birthdayHasHappened = true;
        } else if (this.now().getMonth() === this.dateOfBirth.getMonth()) {
            if (this.now().getDay() >= this.dateOfBirth.getDay()) {
                birthdayHasHappened = true;
            }
        }
        return birthdayHasHappened ? `${this.firstName} has turned ${ageThisYear} this year` : `${this.firstName} will turn ${ageThisYear} this year.`;
    }

    private getAgeThisYear(): number {
        return this.now().getFullYear() - this.dateOfBirth.getFullYear()
    }

}

let tim = new Person('Tim', 'Wray', new Date('1 July 1984'));
let john = new Person('John', 'Holmes', new Date('25 May 1982'));

document.writeln(tim.getAgeMessage());
document.writeln(john.getAgeMessage());
```

```
class Address {  
    constructor(  
        public streetName: string,  
        public number: string | number,  
        public floor?: 'st' | 'kld' | number  
    ) { }  
  
    toString(): string {  
        let addressString: string = `${this.streetName} ${this.number}`;  
        if (this.floor) {  
            if (this.floor === 'st' || this.floor === 'kld') {  
                addressString += ` ${this.floor}`;  
            } else {  
                addressString += ` fl. ${this.floor}`;  
            }  
        }  
        return addressString;  
    }  
}
```

The Address Class

We'll define another class, called Address. It contains an optional field called floor which is a compositional type.

```
interface Bilingual {  
    da: string,  
    en: string  
}  
  
class University {  
  
    constructor(  
        public name: Bilingual,  
        public address: Address  
    ) { }  
  
}
```

The University Class

We'll define another class, called University, which consists of a name and Address type.

The name is of type Bilingual. Bilingual is an interface that states at any members inheriting the interface must have an Danish string and an English string.

Inheritance

We'll modify the original Person class to add the address and an additional `getDescription()` method.

We'll create another class, called Student which inherits from Person.

```
class Person {

    private now: () => Date = () => new Date();

    constructor(
        public firstName: string,
        public lastName: string,
        public dateOfBirth: Date,
        public address: Address
    ) { }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }

    getAgeMessage(): string {
        let ageThisYear = this.getAgeThisYear();
        let birthdayHasHappened: boolean = false;
        if (this.now().getMonth() > this.dateOfBirth.getMonth()) {
            birthdayHasHappened = true;
        } else if (this.now().getMonth() === this.dateOfBirth.getMonth()) {
            if (this.now().getDay() >= this.dateOfBirth.getDay()) {
                birthdayHasHappened = true;
            }
        }
        return birthdayHasHappened ? `${this.firstName} has turned ${ageThisYear} this year` : `${this.firstName} will turn ${ageThisYear} this year.`;
    }

    getDescription(): string {
        return `${this.getFullName()} was born in ${this.dateOfBirth.getFullYear()} and lives at ${this.address.toString()}. ${this.getAgeMessage()}`
    }

    private getAgeThisYear(): number {
        return this.now().getFullYear() - this.dateOfBirth.getFullYear()
    }

}
```

```
class Student extends Person {

    constructor(
        public firstName: string,
        public lastName: string,
        public dateOfBirth: Date,
        public address: Address,
        public university: University
    ) {
        super(
            firstName,
            lastName,
            dateOfBirth,
            address
        )
    }

    getDescription(): string {
        return super.getDescription() + ` ${this.firstName} attends ${this.university.name.da} (${this.university.name.en})`;
    }

}

let itu = new University({ da: 'IT-Universitetet i København', en: 'IT University of Copenhagen' }, new Address('Rued Langgaards Vej', 7));
let ku = new University({ da: 'Københavns Universitet', en: 'University of Copenhagen' }, new Address('Nørregade', 10));

let tim = new Person('Tim', 'Wray', new Date('1 July 1984'), new Address('Amagerbrogade', 116));
let john = new Student('John', 'Holmes', new Date('25 May 1982'), new Address('Vesterbrogade', '200A', 2), itu);

document.writeln(tim.getDescription());
document.writeln(john.getDescription());
```


A Small Exercise

Create a class called Key which contains two private fields 'note' and 'color'.

The class should have a constructor that initialises the key.

The field 'note' should be of type string, and color should be a compositional type with values 'black' and 'white'.

You should write two methods:

- One that 'plays' the note sound file via the Audio() object.
- One that retrieves the note value (e.g. 'C', 'D', 'Bb') as a string.
- One that retrieves the octave (e.g. 1, 2, 3) as a number.