Telerik Academy

TypeScript Overview

You wanted OOP – now you have it!

Telerik Software Academy
Learning & Development Team
http://academy.telerik.com

Telerik Academy

Table of Contents

Telerik Academy

What & Why

Getting ready for TypeScript

What & Why

- TypeScript
  - Typed superset of JavaScript
  - Any browser
  - Any OS
  - Open Source
- Starts with JS, ends with JS
- More information here:
  - http://www.typescriptlang.org/

What & Why

- Requirements
  - Basic JavaScript knowledge
    - Variables
    - Functions
  - Basic OOP knowledge
    - Classes
    - Interfaces
    - Inheritance

Telerik Academy

What & Why

Live Demo

Telerik Academy

Basic Types

Types? Where?

Basic Types

- Boolean - true or false
- String - text
- Number – integer or floating point number
- Array – collection of types
- Object – base object
- Enum – enumeration
- Any – dynamic types, can be everything
- Void – no value

Basic Types

```
var currentName: string;
var hasPassed: boolean;
var averageMark: number;
var currentCourses: string[]; // Array<string>
var additionalInfo: any;
var currentState: State;

enum State { Onsight, Online, NotEnrolled }

function setStudent(
         name: string, passed: boolean,
         mark: number, courses: string[], info: any,
         state: State) : void {
    …
}
```

Telerik Academy

Basic Types

Live Demo

Telerik Academy

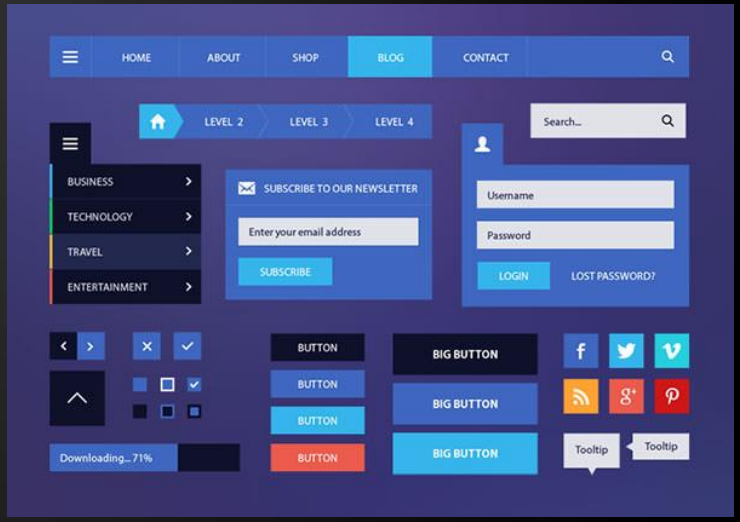
Interfaces

Interfaces? Interesting…

Telerik Academy

Interfaces

- Can define properties
- Can define optional properties
- Can define methods
- Can define functions
- Can define indexers
- Can extend other interfaces



Interfaces

```
interface Person {
    firstName: string;
    lastName: string;
    age?: number;
}
```

```
interface Driver extends Person {
    yearsExperience: number;
    vehicles: Vehicle[];
    addVehicle(vehicle: Vehicle): void;
    removeVehicle(vehicle: Vehicle): Vehicle;
}
```

Telerik Academy

Interfaces

Live Demo

Telerik Academy

Classes

Classes? Are you sure?

Classes

- Can implement properties
- Can have constructors
- Can have methods
- Have this referring to the current instance
- Can extend other classes (super is base)
- Can define private/public parts
- Can define getters or setters
- Can define static parts

Classes

```
class CarDriver extends BasePerson implements Driver {
    private static LicenseNumber: string = '1234-5678';
    private _health: number;
    vehicles: Vehicle[];

    constructor(name: string, public experience: number) {
        super(name);
    }

    get health() {…}
    set health(newHealth: number) {…}

    static CurrentLicenseNumber(): string {…}
    addVehicle(vehicle: Vehicle) {…}
    greet() : string {
        return super.greet() + …;
    }
}
```

Telerik Academy

Classes

Live Demo

Telerik Academy

Modules

Modules? You mean namespaces or what?

Modules

- Organize your code into subsystems
- Created by the module keyword
- Define the public parts by export keyword
- You can split one module into different files
- You can compile them to a single one
- Possibility of external modules (node/require)
- Can be used with external libraries

Modules

```
module Drivers {
    export class BasePerson implements Person {
        …
    }

    export class CarDriver extends BasePerson {
        …
    }
}
```

```
var someDriver = new Drivers.CarDriver(…);
```

Telerik Academy

Modules

Live Demo

Telerik Academy

Functions

Lambdas? You got to be kidding me!

Functions

- Can define the types of the parameters
- Can define their return value
- Can define typed pointers
- Can have optional or default parameters
- Can define collection parameters
- Can be used as lambda expressions
- Can have overloads based on their parameters

Functions

```
function calculateSum(x: number, y: number, z?: number,
...restNumbers: number[]): number {
    var sum = x + y;
    for (var i = 0; i < restNumbers.length; i++) {
        sum += restNumbers[i];
    }
    return sum;
}

var calc: (x: number, y: number)=> number = calculateSum;

var calcSum = (x, y) => x + y;
```

Telerik Academy

Functions

Live Demo

Telerik Academy

Generics

Generics? Yeah, right!

Generics

- Provides reusability
- Generic functions
- Generic classes

- Gives you types checking and constrains

Generics

```
class List<T> {
        private _collection: T[];

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

        remove(item: T) {
            …
        }

        get count() {
            return this._collection.length;
        }
    }
```

Telerik Academy

Generics

Live Demo

Telerik Academy

Additional Info

More is better!

Additional Info

- Documentation
  - http://www.typescriptlang.org/Content/TypeScript%20Language%20Specification.pdf
- Handbook
  - http://www.typescriptlang.org/Handbook
- Demos
  - http://www.typescriptlang.org/Samples
- Training room
  - http://www.typescriptlang.org/Playground

Telerik Academy

TypeScript

Questions?

http://academy.telerik.com

Telerik Academy

Homework

1. * Create a class hierarchy by your choice with TypeScript consisting of the following:
   - At least 2 modules
   - At least 3 interfaces
   - At least 6 classes
   - At least 2 uses of inheritance
   - At least 12 methods
   - At least one generic use
   - At least one static use
   - Everything should be strongly typed