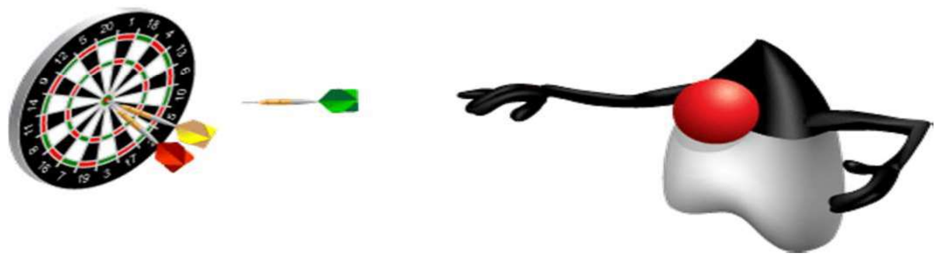
8

Modules and Namespaces

Objectives

After completing this lesson, you should be able to do the following:

- Modules
- Namespaces



Intro to Namespaces

Introduction

- A Namespace is TS/JS File, Which Contains Variables, Methods, Or Full Fledged Class.
- Making This Available to Who is in Need of it [But Within the File Itself].
- ES6 or TypeScript Provides a Solution to this Problem [Modules and Namespaces]. So that it Can Released as a Lib and Used Else Where .
- With a Simple Syntax
  - export

```
 namespace UtilityFunctions {

    export charAt(index: number): string {
       ...
    }

    export concat(stringVal1: string, StringVal2: string): string {
       ...
    }
}
```

```
  class Employee {

    Attributes

    Operations

      UtilityFunctions.charAt(2);
}
```

Introduction to Modules

Modules

- Another important concept when working on large apps is modularity. Having your code split into many small reusable components helps your project stay organized and understandable, compared to having a single 10000-line file for everything.
- TypeScript introduces a syntax for exporting and importing modules, but cannot handle the actual wiring between files. To enable external modules TS relies on third-party libraries: require.js for browser apps and CommonJS for Node.js.

- The first time we called the function we manually set the type to string.
- This isn't required as the compiler can see what argument has been passed and automatically decide what type suits it best, like in the second call.
- Although it's not mandatory, providing the type every time is considered good practice as the compiler might fail to guess the right type in more complex scenarios.

- Modules are executed within their own scope, not in the global scope; this means that variables, functions, classes, etc. declared in a module are not visible outside the module unless they are explicitly exported using one of the exportforms.
- Conversely, to consume a variable, function, class, interface, etc. exported from a different module, it has to be imported using one of the import forms.

Introduction

- A Module is TS/JS File, Which Contains Variables, Methods, Or Full Fledged Class.
  - Making This Available to Who is in Need of it.
  - ES6 or TypeScript Provides a Solution to this Problem [Modules and Namespaces]. So that it Can Released as a Lib and Used Else Where .

- With a Simple Syntax
  - import
  - export

Exporting and Importing

```
//exporter.ts
var sayHi = function(): void {
    console.log("Hello!");
}

export = sayHi;
```

```
//importer.ts
import sayHi = require('./exporter');
sayHi();
```

Export

Exporting a declaration

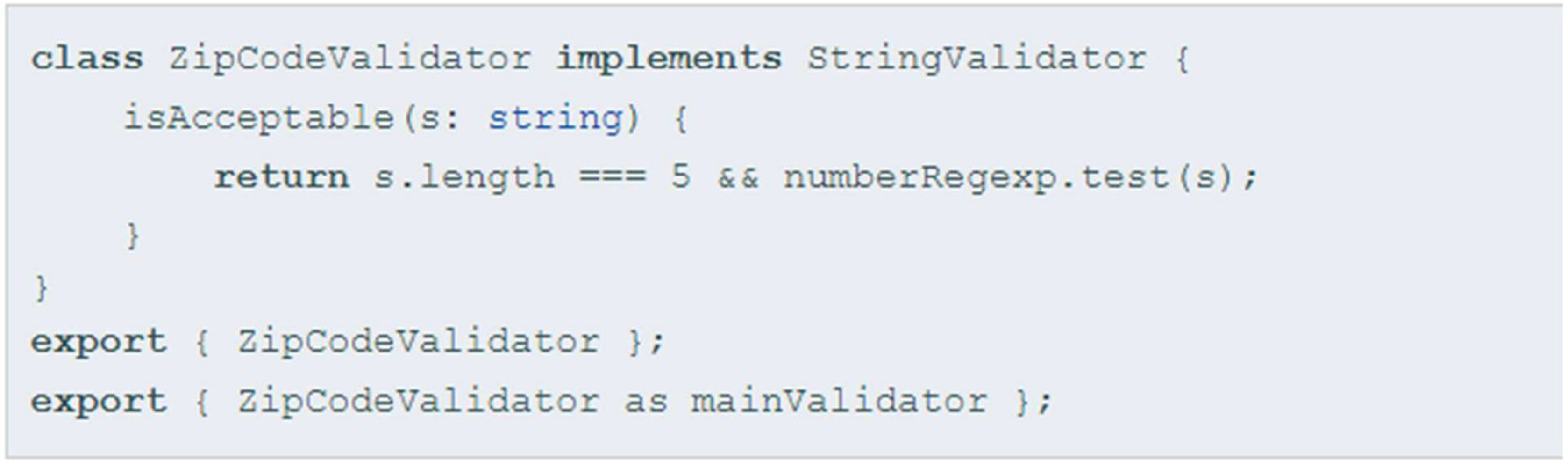
- Any declaration (such as a variable, function, class, type alias, or interface) can be exported by adding the export keyword.

```
//Validation.ts
export interface StringValidator
{
      isAcceptable(s: string): boolean;
}
```

Exporting Variable and A Class

```
//ZipCodeValidator.ts
export const numberRegexp = /^[0-9]+$/;
export class ZipCodeValidator implements StringValidator
    {
          isAcceptable(s: string) { return s.length === 5 && numberRegexp.test(s);
     }
}
```

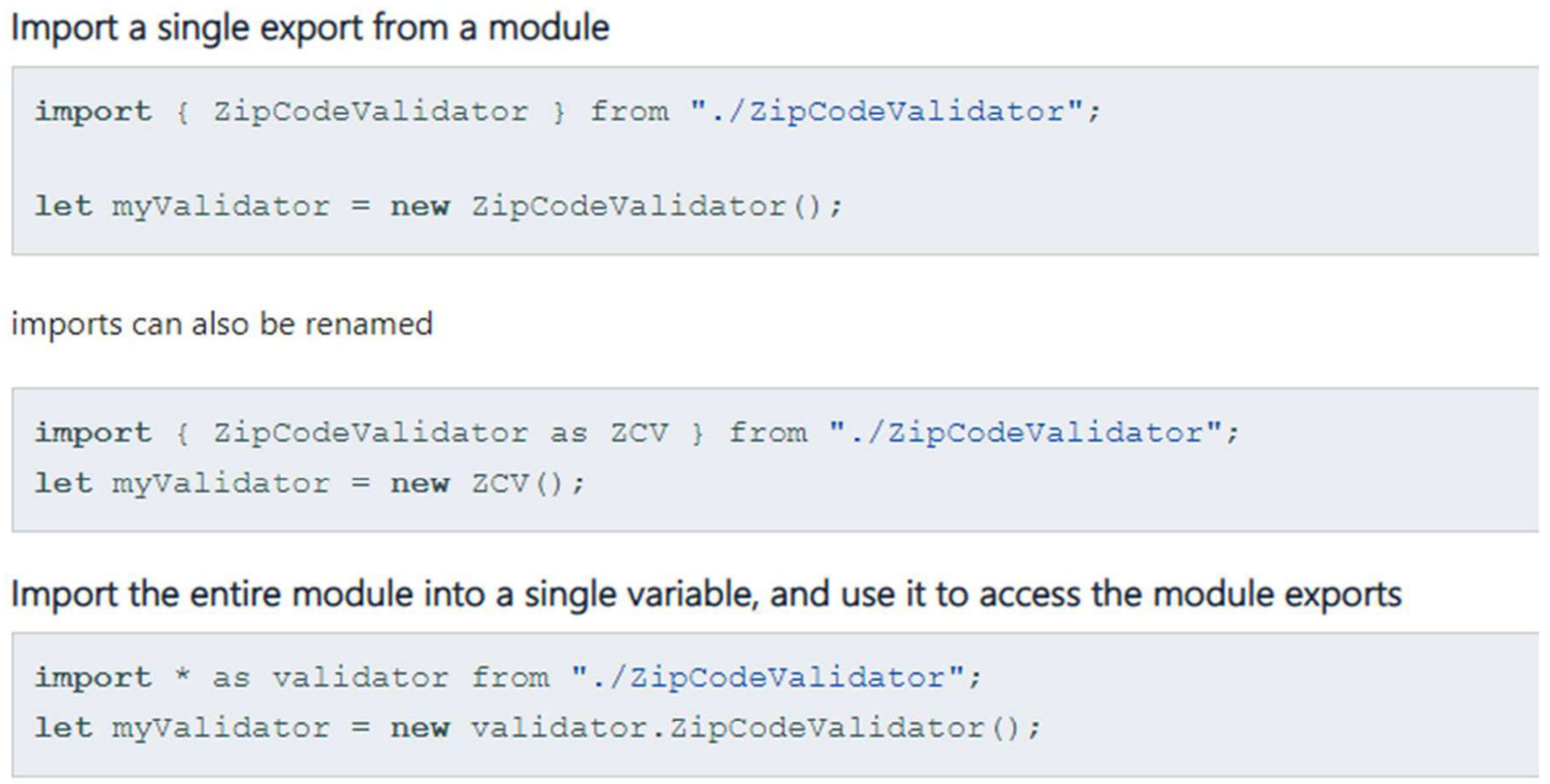
Export statements



```
class ZipCodeValidator implements StringValidator {
    isAcceptable(s: string) {
        return s.length === 5 && numberRegexp.test(s);
    }
}
export { ZipCodeValidator };
export { ZipCodeValidator as mainValidator };
```

Import

- Importing is just about as easy as exporting from a module. Importing an exported declaration is done through using one of the import forms below:



Import a single export from a module

```
import { ZipCodeValidator } from "./ZipCodeValidator";

let myValidator = new ZipCodeValidator();
```

imports can also be renamed

```
import { ZipCodeValidator as ZCV } from "./ZipCodeValidator";
let myValidator = new ZCV();
```

Import the entire module into a single variable, and use it to access the module exports

```
import * as validator from "./ZipCodeValidator";
let myValidator = new validator.ZipCodeValidator();
```

Summary

In this lesson, you should have learned how to:

- Modules
- Namespaces