Namespaces and modules

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Content

Have a Question?

sli.do

#Typescript

Namespaces

Definition

- Namespaces are used to **logically grouped** functionalities
- Previously referred as **internal modules** in TypeScript
- Defined with **namespace** keyword
- Namespaces may include **functions**, **classes**, **interfaces** and **variables**

Access

- The elements of the namespace that must be accessed from the outside must be marked with **export** keyword
- In order to access namespaces from different files we must use the reference syntax **/// <reference path = “file.ts” />**

Example: Namespace

SoftUni Foundation

```
namespace printMessages {
    export function messenger(message: string | string[]) {
        return `${message}`;
    }
    export interface meetPerson {
        meetPerson(): string
    }
}
console.log(printMessages.messenger('Hello')); //Hello
```

namespace declaration

export to use the interface outside

Multiple Files Namespaces

- In order to access namespaces from different files we must use the reference syntax **/// <reference path = “file.ts” />**
- In order to compile the file we must
  - Compile the ts file - **tsc fileName.ts**
  - Use the outFile - **tsc --outFile fileName.js fileName.ts**
  - Compile the js file - **node fileName**

Example: Multiple File Namespace

```
/// <reference path = 'messages.ts'/>
class Person implements printMessages.meetPerson {
    public fullName: string;
    public greeting: string;

    constructor(fn: string, g: string) {
        this.fullName = fn;
        this.greeting = printMessages.messenger(g);
    }

    meetPerson(): string {
        return `Hello, ${this.fullName}, ${this.greeting}`;
    }
}
let p1 = new Person('Ivan Ivanov', 'pleasure to meet you!');
console.log(p1.meetPerson());
```

Aliases

- Used to simplify the work with namespaces
- Used with **import** keyword
- Often used as nested namespaces

```
namespace Shops {
    export namespace TechStores {
        export class PCStore {}
        export class AudioStore {}
        export class TVStore {}
    }
}
--Name of file – app.ts
import stores = Shops.TechStores;
let pcStore = new stores.PCStore();
```

Modules

Definition

- Modules are executed in their **own scope**, not the global
- A **set of functions** to be included in applications
- Resolve name collisions
- In order to be accessed from the outside they need to be marked with **export** keyword

Access

- To consume a function, class, interface or variable exported from another module we must use an **import** form
  - **import** { name } from "./location" - import specific element
  - **import** * as variable from "./location"; - imports the entire module in single variable

Export Statements

- There are three ways to use **export** statements:
  - A:
  - B:
  - C:
  - D:

```
export function numberValidation(num: number): number {…}
```

```
export { numberValidation };
```

```
export { numberValidation as isValidNum }; //isValidNum is alias
```

```
export default function stringValidations(string: string): string {…}
```

- In cases **A** and **B** there is **no difference** rather than syntax
- There might be only one **export default** in a file

Example: Export and Import Statements

```
--exports
export default function checkInput<T>(information: T): T {
    if (information) { return information; }
    else { throw new Error('The information passed is not valid') }
}
export function stringValidations(string: string): string {
    if (string.length > 0 && string.length <= 20) { return string;
}
    else { throw new Error('String is not valid'); }
}
export function numberValidation(num: number): number {
    if (num > 0 && num <= 999) { return num; }
    else { throw new Error('Number is not valid'); }
}
export { numberValidation as isValidNum };
```

Import Statements and File Compilation

```
--Imports
import * as validations from './validations'; //validations is
alias
import checkInput from "./validations";
import { isValidNum } from "./validations";

// Some code logic
```

- In order to compile the file we must
  - Compile the ts file - **tsc fileName.ts**
  - Use the outFile - **tsc --module commonjs fileName.ts**
  - Compile the js file - **node fileName**

Problem: Modules

- You are given a task to calculate the **area and perimeter** of a **Square**, **Circle** and **Rectangle**. You should split the respective classes in a separate files and use modules afterwards
  - Use **interfaces**, in a new file, to define the functions for calculating the area and the perimeter
  - In the main file make a **Calculator** class that makes the calculations

Solution: Modules

```
import * as make from "./interfacesCalc"
import * as shapes from "./shapes";
class Calculator //TODO: extends and implements {
    area(): number {
        return (this.height * this.base) / 2
    }
    perimeter(): number {
        return this.base * 3
    }
}
let calc = new Calculator(4, 4);
console.log(calc.area());
console.log(calc.perimeter());
```

Problem: Namespaces

- You are given a task to display the attendance to a business meeting. You should organize your program in different files using namespaces
  - Use **interface**, in a new file, to define the function for displaying the attendance
  - Use a **function** to invite the attendees. Note that the function takes two parameters -> full name of the employee and the position and returns them in an object
  - In the main file make a **Meeting** class that displays the attendees who went to the meeting

Solution: Namespaces

```
/// <reference path = "attendees.ts" />
const a1 = attendance.createAttendee('Ivan Ivanov', 'R&D');
//invite two more people
class Meeting implements layout.showAttendance {
    public att;
    constructor(a: any) {
        this.att = a;
    }
    showAttendance(): string {
        let output = '';
        //implement the logic that prints the attendees
        return output;
    }
}
```

Live Exercises

Summary

- Namespaces are logically grouped functionalities
- Modules are a **set of functions** to be included in applications
- Modules do not pollute the global scope

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

INDEAVR
Serving the high achievers

INFRAGISTICS®

STEMO®
Computer Systems & Software

SUPERHOSTING.BG

SoftUni Organizational Partners

SoftUni Foundation

IS ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

One SOFTW

Lukanet.com

WORLD OF MYTHS

Questions?

SoftUni

Software University

SoftUni Svetlina

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

License

Trainings @ Software University (SoftUni)

- Software University - High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg