Modules in TypeScript

The files created in TypeScript have global access, which means that variables declared in one file are easily accessed in another file. This global nature can cause code conflicts and can cause issues with execution at run-time. You have export and import module functionality which can be used to avoid global variable, function conflicts. This feature is available in JavaScript with ES6 release and also supported in TypeScript.

Why do you need Modules in TypeScript?

Following example shows the issue without modules:

Example test1.ts

```
let age : number = 25;
```

You have defined a variable age of type number in test1.ts.

Example test2.ts

In test2.ts file you are easily able to access the variable **age** defined in test1.ts and also modify it as shown below:

```
age = 30; // changed from 25 to 30.

let _new_age = age;
```

So, the above case can create a lot of problems as the variables are globally available and can be modified.

With **Modules**, the code written remains locale to the file and cannot be accessed outside it. To access anything from the file, it has to be exported using the export keyword. **It** is used when you want the variable, class, function, or interface to be used in another file. **Import** is used when you want to access the exported variable, class, or interface or function too. Doing so the code is written remains intact within the file, and even if you define same variable names, they are not mixed up and behave local to the file where they are declared.

Using Export and Import

There are many ways to export and import. So will discuss syntax here which is mostly used.

The syntax for import and export 1:

```
export  nameofthevariable or class name or interface name etc

//To import above variable or class name or interface you have to use import as shown below:

Import {nameof thevariable or class name or interfacename} from "file path here without.ts"
```

Here is a working example using export and import.

Example: test1.ts

```
export let age: number = 25;
```

Export keyword is used to share age variable in another file.

test2.ts

```
import { age } from "./test1"

let new_age :number = age;
```

Import keyword is used to access the **age** variable, and you need to specify the file location as shown above.

Syntax for import and export 2:

There is another way to export, and import and the syntax for the same is as shown below:

```
export = classname;

import classname = require("file path of modulename")
```

When you are using **export =** to export your module, the import has to use require("file path of modulename") to import it.

Here is a working example showing the above case:

Customer.ts

```
class Customer {
    name: string;
    age: number;
    constructor(name: string, age: number) {
        this.name = name;
        this.age = age;
    }
    getName(): string {
        return this.name;
    }
}
export = Customer;
```

testCustomer.ts

```
import Customer = require("./Customer");

let a = new Customer("Harry", 30);
alert(a.getName());
```

Example 2:

File: mathUtils.ts

```
export function add(a: number, b: number): number {
  return a + b;
}

export function subtract(a: number, b: number): number {
  return a - b;
}
```

File: app.ts

```
import { add, subtract } from './mathUtils';

const sum = add(5, 3);
const difference = subtract(5, 3);

console.log(`Sum: ${sum}`);
console.log(`Difference: ${difference}`);
```

- In mathUtils.ts, two functions, add and subtract, are defined and exported using the export keyword.
- In app.ts, these functions are imported using the import statement, allowing their usage within the file.
- This modular approach promotes code reusability and clarity by separating functionalities into distinct files.

Output:

```
Sum: 8
Difference: 2
```

Module Loader

Modules cannot work on its own, so you need module loader to locate the import dependencies as you have seen in TypeScript examples shown above. The module loader available is CommonJS for nodejs and Require.js to run in the browser.

To compile code using CommonJS module use following command:

```
tsc --module commonjs testCustomer.ts
```

To compile code using Requirejs module use following command:

```
tsc --module amd testCustomer.ts
```

The dependent files will get converted to js file with above command.