Modules and Namespaces

A module is a container to a group of related variables, functions, classes, and interfaces etc. Variables, functions, classes, and interfaces etc. declared in a module are not accessible outside the module unless they are explicitly exported using export keyword. Also, to consume the members of a module, you have to import it using import keyword.
Modules are declarative and the relationship between modules is specified in terms of imports and exports at the file level. Modules are also available in ES6/ECMAScript 2015.

Important Information

From TypeScript 1.5, "Internal modules" are "namespaces" and "External modules" are simply "modules" to match ES6 module terminology

Export

In TypeScript, you can export any declaration such as a variable, function, class, type alias, or interface by using export keyword.

myModule.ts

```
1. //exporting Employee type
2. export class Employee {
3.  constructor(private firstName: string, private lastName: string) { }
4.  showDetails() {
5.  return this.firstName + ", " + this.lastName;
6.  }
7. }
8.
9. //exporting Student type
10.export class Student {
11. constructor(private rollNo: number, private name: string) { }
12. showDetails() {
13. return this.rollNo + ", " + this.name;
14. }
15.}
16.
```

Import

In TypeScript, you can import an exported declaration by using import keyword. Let's import the myModule file types within app file as given below:

app.ts

```
1. //importing the exporting types Student and Employee from myModule file
2. import { Student, Employee } from "./myModule";
3.
4. let st = new Student(1, "Mohan");
5. let result1 = st.showDetails();
6. console.log("Student Details :" + result1);
7.
8. let emp = new Employee("Shailendra", "Chauhan");
9. let result2 = emp.showDetails();
10.console.log("Employee Details :" + result2);
11.
```

Re-Export

In TypeScript, sometimes modules extend other modules, and partially expose some of their features. In this case, you can re-export some of their features either using their original name or introducing a new name. Let's re-export the myModule file only Student type as show below:

re_export.ts

```
1. //re-exporting types Student as st from myModule file
2. export { Student as st } from "./myModule";
3.
4. export const numberRegexp = /^[0-9]+$/;
5.
```

Now, you can import the types form re_export file as given below:

app.ts

```
1. //importing the exporting types from reexport file
2. import { st as Student, numberRegexp } from "./reExport"; importing st as Student
3.
4. let st = new Student(1, "Mohan");
5. let result1 = st.showDetails();
6. console.log("Student Details :" + result1);
7.
```

Namespaces

Namespaces are simply named JavaScript objects in the global scope. Namespaces are used for grouping of variables, functions, objects, classes, and interfaces, so that you can avoid the naming collisions. Namespaces are declared using the namespace keyword.

namespace.ts

```
1. namespace Gurukulsight {
2.  //exporting outside the namespace body
3.  export class Student {
4.  constructor(private rollNo: number, private name: string) { }
5.  showDetails() {
6.  return this.rollNo + ", " + this.name;
7.  }
8.  }
9.  // Only available inside the namespace body
10. let maxCount: number = 100;
11. class Employee {
12. constructor(private firstName: string, private lastName: string) { }
13. showDetails() {
14. return this.firstName + ", " + this.lastName;
15. }
16. }
17.}
18.namespace DotNetTricks {
19. //accessing Gurukulsight namespace student class
20. import Student = Gurukulsight.Student;
21.
22. export class Person {
23. constructor(private firstName: string, private lastName: string) { }
24. fullName(): string {
25. return this.firstName + " " + this.lastName;
26. }
27. }
28. //creating object of student class
29. let st = new Student(1, "Mohan");
30. st.showDetails();
31.}
32.
```

Important Information

1. A namespace can be described in multiple files and allow to keep each file to a maintainable size.
2. The members of a module body, you can access only within that module body.
3. To make a member available outside the namespace body, you need to prefix the member with the export keyword.

Export and Import Namespaces

In TypeScript, you can export a namespace by prefixing export keyword and to use its members use import keyword. Also, to make a member available outside the namespace body, you need to prefix that member with the export keyword.
Let's understand the namespace import and export with the help of following example.

gurukulsight.ts

```
export namespace Gurukulsight {
 //exporting outside the namespace body
 export class Student {
 constructor(private rollNo: number, private name: string) { }
 showDetails() {
 return this.rollNo + ", " + this.name;
 }
 }
 // Only available inside the namespace body
 let maxCount: number = 100;
 class Employee {
 constructor(private firstName: string, private lastName: string) { }
 showDetails() {
 return this.firstName + ", " + this.lastName;
 }
 }
}
```

Now, let's import the Gurukulsight namespace as given below:

main.ts

```
//importing the namespace
import { Gurukulsight } from "./namespace";

//accessing Gurukulsight namespace student class
import Student = Gurukulsight.Student;

//creating object of student class
let st = new Student(1, "Mohan");
st.showDetails();
```