Agenda: Modules

- Introduction
- Exporting and Importing of Modules
- Re-Export of Modules
- Default Exports

Modules

- 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 export forms. 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.
- Modules are declarative; the relationships between modules are specified in terms of imports and exports **at the file level.**
- Modules import one another using a module loader. At runtime the **module loade**r is responsible for locating and executing all dependencies of a module before executing it.
- In TypeScript, just as in ECMAScript 2015, any file containing a top-level import or export is considered a module.

Syntax for Export:

1. Either precede declaration with "**export**"
   OR
   Use **export { TypeName }** at the end of the file.
2. Make sure that your project contains **tsconfig.json**

If **tsconfig.json** is not installed. Add a json file and name it as **tsconfig.json** in the root folder, and now add the following code.

```
{
  "compilerOptions": {
    "module": "amd",
    "noImplicitAny": false,
    "noEmitOnError": true,
    "removeComments": false,
    "sourceMap": true,
    "target": "es5"
  }
}
```

File: IPoint.ts

```
export interface IPoint {
    x: number;
```

```
    y: number;
    distanceFromOrigin(): number;
}
```

File: Point.ts

```
import {IPoint} from "./IPoint"
export class Point implements IPoint {
    x: number;
    y: number;
    constructor(x: number, y: number) {
        this.x = x; this.y = y;
    }
    distanceFromOrigin(): number {
        return Math.sqrt((this.x * this.x) + (this.y * this.y));
    }
}
```

File: Main.cs

```
import { Point } from "./Point"
import { IPoint } from "./IPoint"

let pt: IPoint
pt = new Point(1, 2);
```

File: Sample.html

```
<script src="~/Scripts/require.js" data-main="main.js"></script>
```

Note: Add require.js to scripts folder.

Alternate syntax for Export / Import

Renaming:

```
export { Point as PointClass } // Export can be renamed if used outside the file.
import { Point as PointClass } //Imports can also be renamed.
```

Optionally, a module can wrap one or more modules and combine all their exports using `export * from "module"` syntax.

File: AllExports.ts

```
export * from "./IPoint";
```

```
export * from "./Point";
```

File: Main.ts

```
import * as MyDemos from "./AllExports"

let pt: MyDemos.IPoint
pt = new MyDemos.Point(1, 2);
```

Re-export

Previously we have learnt how to export and how we can expose (or) make the features of the module visible. But we can re-export any component of that module by giving alias name.

File: **Point.ts**

```
export class Point {
   sum = function (num1: number, num2: number) {
      return num1 + num2;
   }
}
export {IPoint} from "./Point"
```

Here we are exporting the same I**Point** interface.

Default exports

Each module can optionally export a default export. Default exports are marked with the keyword default; and there can only be one default export per module. default exports are imported using a different import form.

File: Point.ts

```
export default class Point {
   x: number;
   y: number;
   constructor(x: number, y: number) {
      this.x = x; this.y = y;
   }
   distanceFromOrigin(): number {
      return Math.sqrt((this.x * this.x) + (this.y * this.y));
   }
}
```

File: Main.ts

```
import DefPoint from "Point"
```

```
let pt: DefPoint
pt = new DefPoint(1, 2);
```

export = and import = require()

- Both **CommonJS and AMD** generally have the concept of an exports object which contains all exports from a module.
- The **export =** syntax specifies a **single object that is exported** from the module. This can be a class, interface, namespace, function, or enum.
- When importing a module using **export =**, TypeScript-specific **import module = require("module")** must be used to import the module.

File: Point.ts

```
class Point {
   x: number;
   y: number;
   constructor(x: number, y: number) {
      this.x = x; this.y = y;
   }
   distanceFromOrigin(): number {
      return Math.sqrt((this.x * this.x) + (this.y * this.y));
   }
}
export = Point
```

File: Main.ts

```
import Point = require("Point")

let pt: Point
pt = new Point(1, 2);
```

Note: For the above code to compile, go to Project → Properties → TypeScript Build Tab → Module System = AMD (Asynchronous Module Definition)

Do not use namespaces in modules:

When first moving to a module-based organization, a common tendency is to wrap exports in an additional layer of namespaces. Modules have their own scope, and only exported declarations are visible from outside the module. With this in mind, namespace provide very little, if any, value when working with modules.