Agenda: Namespaces

- Introduction
- Syntax with Example
- Compiling multiple TS files as one JS file

Introduction to Namespaces

- Namespaces are previously termed as **"internal modules"** in typescript, now they are referred as **"namespaces".**
- Namespaces are simply named JavaScript objects in the **global namespace**.
- Namespaces are important to avoid naming collisions in the global scope.
- It can span over multiple files.
- Namespaces can be a good way to structure your code in a Web Application, with all dependencies included as <script> tags in your HTML page.
- Only Exported types are visible outside the namespace.
- Types used outside the namespace must be qualified by its namespace.
- If required namespace can be nested. Nested namespaces also should be exported.

File: IPoint.ts

```
namespace MyExamples {
  export namespace TypeScript {
    export interface IPoint {
      x: number;
      y: number;
      distanceFromOrigin(): number;
    }
  }
}
```

File: Point.ts

```
///<reference path="IPoint.ts"/>
 namespace MyExamples {
   export class Point implements TypeScript.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.ts

```
///<reference path="IPoint.ts"/>
///<reference path="Point.ts"/>
namespace MyDemos {
    let pt: MyExamples.TypeScript.IPoint
    pt = new MyExamples.Point(10, 20);
    alert(pt.distanceFromOrigin())
}
```

Once there are multiple files involved, we'll need to make sure all of the compiled code gets loaded.

There are two ways of doing this..

1. First, we can use concatenated output using the --outFile flag to compile all of the input files into a single JavaScript output file:

   The compiler will automatically order the output file based on the **reference** tags present in the files.

   You can also specify each file individually:

   ```
   tsc --outFile sample.js IPoint.ts Point.ts Main.ts
   ```

OR

Visual Studio → Project Properties → TypeScript Build → Check Combined JavaScript output to File = "sample.js"

OR

tsconfig.json

```
{
  "compilerOptions": {
    "outFile": "./sample.js",
    "module": "amd",
   . . .
  },
}
```

2. Alternatively, we can use per-file compilation (the default) to emit one JavaScript file for each input file. If multiple JS files get produced, we'll need to use <script> tags on our webpage to load each emitted file in the appropriate order, for example:

Demo.html:

```
<script src="IPoint.js"></script>
<script src="Point.js"></script>
<script src="Main.js"></script>
```

Aliases: It's used to create shorter names.

import Eg = MyExamples.TypeScript

Now "Eg" can be used through the file instead of MyExamples.TypeScript.

Note: Starting with ECMAScript 2015, modules are native part of the language, and should be supported by all compliant engine implementations. Thus, for new projects modules would be the recommended code organization mechanism.