Experiment No.: 04

Title: Demonstrate Namespaces, Modules and Ambients in Typescript.

Objectives:

1. To study Namespaces, Modules and Ambients in Typescript.

Theory:

Namespace-

The namespace is used for logical grouping of functionalities. A namespace can include interfaces, classes, functions and variables to support a single or a group of related functionalities.

A namespace can be created using the namespace keyword followed by the namespace name. All the interfaces, classes etc. can be defined in the curly brackets { }.

```
namespace <name>
{   }
```

By default, namespace components cannot be used in other modules or namespaces. You must export each component to make it accessible outside, using the export keyword. To use it, it must be included using triple slash reference syntax.

e.g. ///<reference path="path to namespace file" />.

Modules-

A module is designed with the idea to organize code written in TypeScript. Modules are broadly divided into –

- Internal Modules
- External Modules

Internal Module

Internal modules was used to logically group classes, interfaces, functions into one unit and can be exported in another module. This logical grouping is named namespace in latest version of TypeScript. So internal modules are obsolete instead we can use namespace.

External Module

External modules in TypeScript exists to specify and load dependencies between multiple external js files. If there is only one js file used, then external modules are not relevant.

Traditionally dependency management between JavaScript files was done using browser script tags (<script></script>). But that's not extendable, as its very linear while loading modules. That means instead of loading files one after other there is no asynchronous option to load modules.

Ambients-

Ambient declarations are a way of telling the TypeScript compiler that the actual source code exists elsewhere. When you are consuming a bunch of third party js libraries like jquery/angularjs/nodejs you can't rewrite it in TypeScript. Ambient declarations help to seamlessly integrate other js libraries into TypeScript.

Defining Ambients

Ambient declarations are by convention kept in a type declaration file with following extension (d.ts)

Sample.d.ts

The above file will not be transcompiled to JavaScript. It will be used for type safety

The syntax for declaring ambient variables or modules will be as following −

declare module Module_Name {   }

The ambient files should be referenced in the client TypeScript file as shown −

/// <reference path = " Sample.d.ts" />

Key Concept: Namespaces, Modules, Ambients