FAQ: What is the purpose of “declare” for module components?

- Declare is not a keyword.
- Declare is a marker that is used to make any variable or component so that it allows only declaration.
- Initialization of values in not allowed on ambient declaration.

  Syntax:
  declare let x:number = 10; //invalid
  declare let x:number; //valid
  x=10;

FAQ: What is the purpose of “default”?

- You can export many components from a file.
- Default is used to restrict any one component for exporting. When you try import without using “{ }”.
- Default component can’t be imported by using “{}”
- Every file can have multiple default components. Developer can import any one there is no priority for components.

Note: Every component marked with export can be imported. But not as default import.

Default import is defined without “{ }”.

Ex:

ProjectContracts.ts

```
export default interface ICategory
{
   CategoryId:number;
}
export default interface IProduct
{
   Name:string;
   Price:number;
}
```

App.ts

```
import ICategory from './ProjectContracts';   // valid
import IProduct from './ProjectContracts';  // valid
import { ICategory} from './ProjectContracts';  // invalid
import {ICategory,IProduct} from './ProjectContracts'; // invalid
```

Namespace

- Namespace is a collection of related type of sub namespaces and classes.
- You can configure a library under namespace, which contains set of contracts, templates and components that you can import into any application.

- The Namespace is defined as a container in which the components must be marked with “export”.

Ex:

1. Add a new folder
   Project

1. Add following folders into project folder

- Contracts
- Templates
- Services
- App

1. Add following file into contracts

“ProductContract.ts”

```
namespace Project
{
   export namespace Contracts
   {
      export interface IProduct
      {
         Name:string;
         Price:number;
         Qty:number;
         Total():number;
         Print():void;
      }
   }
}
```

1. Add file into Templates folder

“ProductTemplate.ts”

```
///<reference path="../Contracts/ProductContract.ts" />
```

```
import contracts = Project.Contracts;
namespace Project
{
   export namespace Templates
   {
      export abstract class ProductTemplate implements
contracts.IProduct
       {
       public Name:string;
       public Price:number;
       public Qty:number;
       public Total():number {
          return this.Qty * this.Price;
       }
       abstract Print():void;
      }
   }
}
```

1. Add file into Services folder
“ProductService.ts”

```
///<reference path="../Templates/ProductTemplate.ts" />

import templates = Project.Templates;

namespace Project
{
   export namespace Services
   {
      export class Product extends templates.ProductTemplate
      {
         public Name:string = '';
         public Price:number = 0;
         public Qty:number = 1;
         public Total():number {
            return this.Qty * this.Price;
         }
```

```
        public Print():void {
            console.log(`Name=${this.Name}\nPrice=${this.Price}
\nQty=${this.Qty}\nTotal=${this.Total()}`);
        }
    }
  }
}
```

1. Add File into APP folder

   MyApp.ts

```
///<reference path="../Services/ProductService.ts" />

import services = Project.Services;

let tv = new services.Product();
tv.Name = "Samsung TV";
tv.Price = 34000.44;
tv.Qty = 2;
tv.Print();
```

Compile

```
> tsc --outFile  myapp.js  myapp.ts
```