آموزش TypeScript – معرفی و کار با Namespaces در تایپ اسکریپت(TypeScript)

Namespaceروشی برای گروه بندی منطقی کدهای مرتبط است. این ویژگی به صورت inbuilt یا توکار در تایپ اسکریپت وجود دارد. این درحالی است که برخلاف تایپ اسکریپت، اعلان متغیرها در جاوا اسکریپت در حیطه ی جهانی (global scope) انجام می شود، و در صورتی که از فایل های جاوا اسکریپتی متعددی داخل پروژه ی یکسانی استفاده شود، احتمال overwrite شدن یا سوء تعبیر (misconstruing) متغیرهای یکسان وجود دارد، که این اتفاق در جاوا اسکریپت باعث "global namespace pollution problem" خواهد شد.

تعریف Namespace

برای این کار عبارت کلیدی namespace به دنبال اسم namespace می آید. مانند زیر:

```
namespace SomeNameSpaceName {

  export interface ISomeInterfaceName {    }

  export class SomeClassName {    }

}
```

کلاس ها یا رابط هایی که قرار است در خارج از namespace در دسترس باشند، باید با عبارت کلیدی export مشخص شوند.

برای اینکه بتوان در namespace دیگری به کلاس یا interface دسترسی پیدا کرد، باید از سینتکس زیر استفاده کرد: namespaceName.className

```
SomeNameSpaceName.SomeClassName;
```

در صورتی که اولین namespace در فایل تایپ اسکریپت جداگانه ای باشد، باید با استفاده از سینتکس رفرنس دارای سه اسلش به آن اشاره کرد.

```
/// <reference path = "SomeFileName.ts" />
```

در برنامه ی زیر استفاده از namespace ها نشان داده شده است.

```
FileName :IShape.ts

----------

namespace Drawing {

  export interface IShape {
```

```
    draw();
  }
}

FileName :Circle.ts
----------
/// <reference path = "IShape.ts" />
namespace Drawing {
  export class Circle implements IShape {
    public draw() {
      console.log("Circle is drawn");
    }

    FileName :Triangle.ts
    ----------
    /// <reference path = "IShape.ts" />
    namespace Drawing {
      export class Triangle implements IShape {
        public draw() {
          console.log("Triangle is drawn");
        }
      }

      FileName : TestShape.ts
      /// <reference path = "IShape.ts" />
```

```
    /// <reference path = "Circle.ts" />
    /// <reference path = "Triangle.ts" />
    function drawAllShapes(shape:Drawing.IShape) {
      shape.draw();
    }
    drawAllShapes(new Drawing.Circle());
    drawAllShapes(new Drawing.Triangle());
  }
 }
}
```

کد بالا را می توان با استفاده از دستور زیر کامپایل و اجرا کرد.

```
tsc --out app.js TestShape.ts

node app.js
```

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود (app.js).

```
//Generated by typescript 1.8.10
/// <reference path = "IShape.ts" />
var Drawing;
(function (Drawing) {
  var Circle = (function () {
    function Circle() {
    }
    Circle.prototype.draw = function () {
      console.log("Cirlce is drawn");
    };
    return Circle;
```

```
  }());

  Drawing.Circle = Circle;
})(Drawing || (Drawing = {}));

/// <reference path = "IShape.ts" />
var Drawing;
(function (Drawing) {
  var Triangle = (function () {
    function Triangle() {
    }
    Triangle.prototype.draw = function () {
      console.log("Triangle is drawn");
    };
    return Triangle;
  }());
  Drawing.Triangle = Triangle;
})(Drawing || (Drawing = {}));

/// <reference path = "IShape.ts" />
/// <reference path = "Circle.ts" />
/// <reference path = "Triangle.ts" />
function drawAllShapes(shape) {
  shape.draw();
}
```

```
drawAllShapes(new Drawing.Circle());

drawAllShapes(new Drawing.Triangle());
```

و خروجی به صورت زیر نمایش داده می شود.

```
Circle is drawn
Triangle is drawn
```

Namespace های تو در تو

می توان Namespace ای را داخل Namespace دیگر تعریف کرد. مانند زیر:

```
namespace namespace_name1 {

  export namespace namespace_name2 {

    export class class_name {   }

  }

}
```

جهت دسترسی به اعضای namespace های تو در تو می توان از عملگر (.) استفاده کرد. مانند زیر:

```
FileName : Invoice.ts

namespace tutorialPoint {

  export namespace invoiceApp {

    export class Invoice {

      public calculateDiscount(price: number) {

        return price * .40;

      }

    }

  }

}
```

FileName: InvoiceTest.ts

```
/// <reference path = "Invoice.ts" />

var invoice = new tutorialPoint.invoiceApp.Invoice();

console.log(invoice.calculateDiscount(500));
```

کد بالا را می توان با استفاده از دستور زیر کامپایل و اجرا کرد.

```
tsc --out app.js InvoiceTest.ts
node app.js
```

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود (app.js).

```
//Generated by typescript 1.8.10

var tutorialPoint;

(function (tutorialPoint) {

  var invoiceApp;

  (function (invoiceApp) {

    var Invoice = (function () {

      function Invoice() {

      }

      Invoice.prototype.calculateDiscount = function (price) {

        return price * .40;

      };

      return Invoice;

    }());

    invoiceApp.Invoice = Invoice;

  })(invoiceApp = tutorialPoint.invoiceApp || (tutorialPoint.invoiceApp = {}));
```

```
})(tutorialPoint || (tutorialPoint = {}));

/// <reference path = "Invoice.ts" />



var invoice = new tutorialPoint.invoiceApp.Invoice();

console.log(invoice.calculateDiscount(500));
```

و خروجی به صورت زیر نمایش داده می شود.

```
200
```