آموزش TypeScript – آموزش کلاس (Classes) در TypeScript

تایپ اسکریپت جاوا اسکریپت شیء گرا است. تایپ اسکریت از برنامه نویسی شیء گرا مانند کلاس ها، رابط ها و ... پشتیبانی می کند. یک کلاس بر حسب OOP طرح اولیه ای برای ساخت اشیاء محسوب می شود. نقش کلاس ها نگهداری داده ها برای اشیاء است. در خود تایپ اسکریپت از این مفهوم تحت عنوان کلاس پشتیبانی شده است. در جاوا اسکریپت ES5 و قبل تر از کلاس ها پشتیبانی نمی شد. تایپ اسکریپت این ویژگی را از ES6 گرفته است.

آموزش ایجاد کلاس ها در TypeScript

برای اعلان کلاس ها در تایپ اسکریپت از عبارت کلیدی class استفاده می شود. سینتکس این کار در زیر آمده است.

سینتکس

```
class class_name {
  //class scope
}
```

عبارت کلیدی class به دنبال اسم کلاس می آید. در زمانی که بخواهیم اسم کلاسی را انتخاب کنیم، باید قوانین مربوط به شناسه ها را در نظر بگیریم.

در تعریف کلاس موارد زیر باید لحاظ شود.

- Fields : به متغیری گفته می شود که در یک کلاس اعلان می شود. فیلدها بیانگر داده های مربوط به اشیاء هستند.
- Constructors : مسئول تخصیص حافظه به اشیاء کلاس هستند.
- Functions : بیانگر کارهایی هستند که یک شیء انجام می دهد. در بعضی مواقع به Function ها متد نیز گفته می شود.

به ترکیب این اجزا در کنار هم، اعضای داده ای کلاس گفته می شود. کلاس Person را در تایپ اسکریپت در نظر بگیرید.

```
class Person {

}
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می شود.

```
//Generated by typescript 1.8.10
```

```
var Person = (function () {
  function Person() {
  }
  return Person;
}());
```

مثال: اعلان کلاس

```
class Car {
  //field
  engine:string;

  //constructor
  constructor(engine:string) {
    this.engine = engine
  }

  //function
  disp():void {
    console.log("Engine is :  "+this.engine)
  }
}
```

در این مثال کلاس Car اعلان شده است. این کلاس فیلدی به نام engine دارد. عبارت کلیدی var طی اعلان یک فیلد استفاده نمی شود. در مثال بالا constructor ای برای کلاس اعلان شده است.

constructor ها توابع ویژه ای از کلاس هستند که مسئول مقداردهی اولیه ی متغیرهای کلاس می باشند. تایپ اسکریپت با استفاده از عبارت کلیدی constructor، کنستراکتوری را تعریف می کند. با توجه به اینکه constructor ها تابع هستند، می توان به آن ها پارامتر داد.

عبارت کلیدی this به نمونه فعلی کلاس اشاره دارد. در اینجا اسم پارامتر و اسم فیلد کلاس یکی است. به همین دلیل برای جلوگیری از ابهام، فیلد کلاس پیش از عبارت کلیدی this آمده است.

disp() نمونه ای ساده از تعریف تابع است. توجه داشته باشید که در اینجا از عبارت کلیدی function استفاده نشده است.

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10
var Car = (function () {
  //constructor
  function Car(engine) {
    this.engine = engine;
  }

  //function
  Car.prototype.disp = function () {
    console.log("Engine is :  " + this.engine);
  };
  return Car;
}());
```

آموزش ایجاد نمونه اشیاء در TypeScript

برای اینکه بتوان نمونه ای از کلاس را ایجاد کرد، می توان در کنار اسم کلاس از عبارت کلیدی new استفاده کرد. سینتکس انجام این کار در زیر آمده است.

سینتکس

```
var object_name = new class_name([ arguments ])
```

- عبارت کلیدی new مسئول نمونه سازی است.
- سمت راست این عبارت مسئول احضار کردن constructor است. در صورتی که constructor دارای پارامتر باشد، باید به آن مقدار داد.

مثال: نمونه سازی یک کلاس

```
var obj = new Car("Engine 1")
```

آموزش دسترسی به Attribute ها و توابع در TypeScript

به توابع و attribute های یک کلاس می توان از طریق شیء دسترسی پیدا کرد. برای دسترسی به اعضای داده ای یک کلاس، بایستی از نقطه ' . ' که به آن period گفته می شود، استفاده کرد.

```
//accessing an attribute
obj.field_name

//accessing a function
obj.function_name()
```

مثال : ترکیب موارد بالا در کنار هم

```
class Car {

  //field

  engine:string;


  //constructor

  constructor(engine:string) {

    this.engine = engine

  }


  //function
```

```
  disp():void {
    console.log("Function displays Engine is :  "+this.engine)
  }
}

//create an object
var obj = new Car("XXSY1")

//access the field
console.log("Reading attribute value Engine as :  "+obj.engine)

//access the function
obj.disp()
```

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10
var Car = (function () {
  //constructor
  function Car(engine) {
    this.engine = engine;
  }

  //function
  Car.prototype.disp = function () {
    console.log("Function displays Engine is :  " + this.engine);
  };
```

```
  return Car;

}());


//create an object

var obj = new Car("XXSY1");


//access the field

console.log("Reading attribute value Engine as :  " + obj.engine);


//access the function

obj.disp();
```

و خروجی به صورت زیر نمایش داده می شود.

```
Reading attribute value Engine as :  XXSY1
Function displays Engine is  :  XXSY1
```

آموزش وراثت در کلاس در TypeScript

تایپ اسکریپت از مفهوم وراثت پشتیبانی می کند. وراثت به توانایی یک برنامه در ایجاد کلاس های جدید از یک کلاس موجود گفته می شود. به کلاسی که جهت ایجاد کلاس های جدیدتر توسعه داده می شود، parent class یا super class گفته می شود. به کلاسی که به تازگی ایجاد شده است، sub class یا child class گفته می شود.

کلاس ها با استفاده از عبارت کلیدی 'extends' وارث کلاس دیگری می شوند. Child class ها تمامی مشخصه ها و متدها به غیر از اعضای خصوصی و constructor های parent class به ارث می برند.

سینتکس

```
class child_class_name extends parent_class_name
```

هرچند که تایپ اسکریپت از وراثت های متعدد پشتیبانی نمی کند.

مثال : وراثت درکلاس

```
class Shape {
  Area:number

  constructor(a:number) {
    this.Area = a
  }
}

class Circle extends Shape {
  disp():void {
    console.log("Area of the circle:  "+this.Area)
  }
}

var obj = new Circle(223);
obj.disp()
```

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10
var __extends = (this && this.__extends) || function (d, b) {
  for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
  function __() { this.constructor = d; }
  d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
};
var Shape = (function () {
  function Shape(a) {
```

```
    this.Area = a;
  }
  return Shape;
}());

var Circle = (function (_super) {
  __extends(Circle, _super);
  function Circle() {
    _super.apply(this, arguments);
  }

  Circle.prototype.disp = function () {
    console.log("Area of the circle: " + this.Area);
  };
  return Circle;
}(Shape));

var obj = new Circle(223);
obj.disp();
```

و خروجی به صورت زیر نمایش داده می شود.

```
Area of the Circle: 223
```

در مثال بالا، کلاس Shape اعلان شده است. این کلاس توسط کلاس Circle توسعه داده شده است. با توجه به اینکه بین کلاس ها رابطه ی وراثت وجود دارد، child class یا همان کلاس Car به صورت غیرمستقیم به attribute مربوط به parent class خود یا همان area دسترسی دارد.

وراثت را می توان به صورت زیر دسته بندی کرد.

- Single : هر کلاس می تواند حداکثر از یک parent class توسعه یابد.
- Multiple : کلاس ها می توانند وارث کلاس های متعددی باشند. تایپ اسکریپت از این حالت پشتیبانی نمی کند.
- Multi-level : در مثال زیر چگونگی کارکرد وراثت چند سطحی نشان داده شده است.

مثال

```
class Root {

  str:string;

}


class Child extends Root {}

class Leaf extends Child {} //indirectly inherits from Root by virtue of inheritance


var obj = new Leaf();

obj.str ="hello"

console.log(obj.str)
```

کلاس Leaf با استفاده از وراثت چند سطحی attribute های Root class و Child class را به دست می آورد.

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10

var __extends = (this && this.__extends) || function (d, b) {

  for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];

  function __() { this.constructor = d; }

  d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());

};


var Root = (function () {
```

```
  function Root() {

  }

  return Root;

}());


var Child = (function (_super) {

  __extends(Child, _super);

  function Child() {

    _super.apply(this, arguments);

  }

  return Child;

}(Root));


var Leaf = (function (_super) {

  __extends(Leaf, _super);

  function Leaf() {

    _super.apply(this, arguments);

  }

  return Leaf;

}(Child));


var obj = new Leaf();

obj.str = "hello";

console.log(obj.str);
```

و خروجی به صورت زیر نمایش داده می شود.

خروجی

```
hello
```

آموزش وراثت کلاس و Override کردن متد در تایپ اسکریپت

Method Overriding به مکانیزمی گفته می شود که در آن child class متد superclass را مجددا تعریف می کند. در مثال زیر این مکانیزم نشان داده شده است.

```
class PrinterClass {

  doPrint():void {

    console.log("doPrint() from Parent called...")

  }

}


class StringPrinter extends PrinterClass {

  doPrint():void {

    super.doPrint()

    console.log("doPrint() is printing a string...")

  }

}


var obj = new StringPrinter()

obj.doPrint()
```

عبارت کلیدی super در اشاره به parent بی واسطه ی (immediate parent) یک کلاس کاربرد دارد. این عبارت کلیدی در اشاره به نسخه ی super class یک متغیر، مشخصه یا متد نیز به کار می رود. در خط سیزدهم، نسخه ی super class تابع ()doWork احضار شده است.

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10
var __extends = (this && this.__extends) || function (d, b) {
  for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p];
  function __() { this.constructor = d; }
  d.prototype = b === null ? Object.create(b) : (__.prototype = b.prototype, new __());
};

var PrinterClass = (function () {
  function PrinterClass() {
  }
  PrinterClass.prototype.doPrint = function () {
    console.log("doPrint() from Parent called...");
  };
  return PrinterClass;
}());

var StringPrinter = (function (_super) {
  __extends(StringPrinter, _super);

  function StringPrinter() {
    _super.apply(this, arguments);
  }

  StringPrinter.prototype.doPrint = function () {
    _super.prototype.doPrint.call(this);
```

```
    console.log("doPrint() is printing a string...");
  };

  return StringPrinter;
}(PrinterClass));

var obj = new StringPrinter();
obj.doPrint();
```

و خروجی به صورت زیر نمایش داده می شود.

```
doPrint() from Parent called...
doPrint() is printing a string...
```

آموزش عبارت کلیدی استاتیک در TypeScript

این عبارت های کلیدی را می توان در اعضای داده ای کلاس ها استفاده کرد. متغیرهای استاتیک تا زمانی که برنامه کار اجرای خود را به اتمام برساند، مقدار خود را حفظ می کند. به اعضای استاتیک توسط نام کلاس اشاره می شود.

مثال

```
class StaticMem {
  static num:number;

  static disp():void {
    console.log("The value of num is"+ StaticMem.num)
  }
}

StaticMem.num = 12     // initialize the static variable
```

```
StaticMem.disp()    // invoke the static method
```

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می شود.

```
//Generated by typescript 1.8.10
var StaticMem = (function () {
  function StaticMem() {
  }

  StaticMem.disp = function () {
    console.log("The value of num is" + StaticMem.num);
  };

  return StaticMem;
}());

StaticMem.num = 12;    // initialize the static variable
StaticMem.disp();    // invoke the static method
```

و خروجی به صورت زیر نمایش داده می شود.

```
The value of num is 12
```

آموزش عملگر instanceof در TypeScript

در صورتی که شیء متعلق به نوع خاصی باشد، true را برگشت می دهد.

مثال

```
class Person{ }
```

```
var obj = new Person()

var isPerson = obj instanceof Person;

console.log(" obj is an instance of Person " + isPerson);
```

بعد از کامپایل کردن کد بالا، کد زیر در جاوا اسکریپت ایجاد می شود.

```
//Generated by typescript 1.8.10

var Person = (function () {

  function Person() {

  }

  return Person;

}());


var obj = new Person();

var isPerson = obj instanceof Person;

console.log(" obj is an instance of Person " + isPerson);
```

و خروجی به صورت زیر نمایش داده می شود.

```
obj is an instance of Person True
```

آموزش مخفی کردن داده در TypeScript

کلاس ها می توانند تعیین کنند که اعضای کلاس های دیگر، اعضای داده ای خود را ببینند یا نبینند. به این قابلیت اصطلاحا "مخفی کردن داده" یا "Encapsulation" گفته می شود.

شیء گرایی برای پیاده سازی این مفهوم از اصلاح کننده های دسترسی "access modifiers" یا مشخص کننده های دسترسی "access specifiers" بهره می برند. این دو وظیفه ی تعیین قابلیت دیدن اعضای داده ای یک کلاس را در خارج از کلاس تعریف خود برعهده دارند.

access modifier هایی که توسط تایپ اسکریپت پشتیبانی می شوند، در جدول زیر آمده اند.

ردیف	Access Specifier و توضیحات
.1	Public عضو داده ای عمومی به صورت جهانی قابل دسترسی است. اعضای داده ای موجود در کلاس ها به صورت پیش فرض public یا عمومی هستند.
.2	Private اعضای داده ای خصوصی تنها از داخل کلاسی که این اعضا را تعریف می کند، قابل دسترسی هستند. در صورتی که عضو کلاسی از خارج بخواهد به عضوی خصوصی دسترسی پیدا کند، کامپایلر خطا می دهد.
.3	Protected اعضای داده ای محافظت شده مانند اعضای داده ای خصوصی توسط اعضای داخل کلاس خودشان قابل دسترس هستند. همچنین توسط اعضای child class ها نیز در دسترس هستند.

مثال

برای درک چگونگی کارکرد مخفی کردن داده ها به مثال زیر نگاه کنید.

```
class Encapsulate {

  str:string = "hello"

  private str2:string = "world"

}



var obj = new Encapsulate()

console.log(obj.str)    //accessible

console.log(obj.str2)  //compilation Error as str2 is private
```

این کلاس دو attribute به نام های str1 و str2 دارد. که این attribute ها به ترتیب اعضای عمومی و خصوصی هستند. بعد از اینکه کلاس نمونه سازی شود، کامپایلر خطا می دهد. زیرا str2 که attribute خصوصی است، از خارج از کلاسی که آن را اعلان می کند در دسترس قرار می گیرد.

آموزش کلاس ها و رابط ها در TypeScript

در کلاس ها می توان رابط ها را نیز پیاده سازی کرد.

```
interface ILoan {

  interest:number

}


class AgriLoan implements ILoan {

  interest:number

  rebate:number


  constructor(interest:number,rebate:number) {

    this.interest = interest

    this.rebate = rebate

  }

}


var obj = new AgriLoan(10,1)

console.log("Interest is : "+obj.interest+" Rebate is : "+obj.rebate )
```

کلاس AgriLoan رابط Loan را پیاده سازی کرده است. به همین دلیل این رابط برای لحاظ کردن مشخصه ی interest به عنوان عضو خود به کلاس مقید شده است.

بعد از کامپایل کردن کد بالا، کد جاوا اسکریپت زیر ایجاد می شود.

```
//Generated by typescript 1.8.10

var AgriLoan = (function () {

  function AgriLoan(interest, rebate) {

    this.interest = interest;

    this.rebate = rebate;

  }
```

```
  return AgriLoan;

}());



var obj = new AgriLoan(10, 1);

console.log("Interest is : " + obj.interest + " Rebate is : " + obj.rebate);
```

و خروجی به صورت زیر نمایش داده می شود.

```
Interest is : 10 Rebate is : 1
```