TypeScript

Tahaluf Training Center 2021

Chapter 04

1. Class
2. Composition
3. Interface
4. Enum
5. Access Modifier

Class

- TypeScript is object oriented JavaScript which supports object-oriented programming features like classes, interfaces, etc. .

- A class encapsulates data for the object.

Class

- JavaScript ES5 or earlier didn't support classes. Typescript gets this feature from ES6.

- The Syntax to declare a class :

```
class class_name {
    //properties
    //constructor
}
```

❖ **Example:**

```
class Person {
name: string = '';
age: number = 0;
constructor() {
console.log('New Person is under creation !!!');
            }
        }
```

❖ To create new instance of a class :

Syntax :

```
const object_Name = new
class_Name();
```

❖ Create new instance for our example:

```
const personOne = new Person();
```

❖ Get this object new values :

```
const personOne = new Person();
      personOne.name = 'John’;
      personOne.age = 20;
      console.log(personOne);
```

- You can pass a parameter of the constructor and give for each parameter data type .

- Example :

```
constructor(fName: string, lName: string){
            this.fName = fName;
            this.lName = lName;
        }
```

Exercise :

Create a class called User , give it the information for the any user . Then create an instance of this class .

Solution

```
class User {
lName: string = ‘’;
dob: Date = new Date();
age: number = 0;
 address: string = ‘’;
   constructor() {
                console.log('New
User is under creation !!!');
            }
```

Solution

```
const userA = new User();
userA.fName = 'John';
userA.lName = 'doh’;
userA.age = 20;
userA.address = 'Jordan/Irbid';
console.log(userA);
```

- Default parameter in the constructor:

1. You can pass default parameter for the constructor. This means if the user does not pass the value for the constructor , default value will be execute .

2. Default parameter must be after the required parameter .

Example:

```
constructor(fName: string = 'unknown’,
   lName: string = 'unknown'){ }
```

- TypeScript provides a **Optional** parameters feature

- By using Optional parameters features, we can declare some parameters in the function optional, so that client need not required to pass value to optional parameters.

Example:

```
constructor(fName ?: string, lName: string =
'unknown')

const userA = new User('deo');//Ok
const userA = new User('John', 'deo');//Ok
```

Day 04

1. Class
2. **Composition**
3. Interface
4. Enum
5. Access Modifier

Composition

- Composition is one of the fundamental concepts in object-oriented programming

- Its a class that references one or more objects of other classes in instance variables.

Composition

```
class User {
 fName?: string;
 lName?: string;
 dob?: Date;
 age?: number;
 type: string = '';
 address: Address = new   Address();
 print() {
                console.log(this.fName);
           }
```

Composition

```
class Address {
 streetName?: string;
 city?: string;
 buliding?: number;
       }
```

Composition

```
const userA = new
   User(20,'John','doh');
userA.age=20;
userA.address.buliding=5;
userA.address.city='Irbid';
userA.address.streetName='';
console.log(userA);
```

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Day 04

1. Class
2. Composition
3. **Interface**
4. Enum
5. Access Modifier

Interface

- Interface is a structure that defines the contract in your application.
- It defines the syntax for classes to follow.
- Classes that are derived from an interface must follow the structure provided by their interface.

Interface

- The TypeScript compiler does not convert interface to JavaScript. It uses interface for type checking. This is also known as "duck typing" or "structural subtyping".

- An interface is defined with the keyword interface and it can include properties and method declarations using a function or an arrow function

❖ Example

```
interface IUser{
    fname:string;
    lname:string;
    age:number
}

const userB:IUser={
    fname:'john',
    lname:'Doh',
    age:30
}
```

Day 04

1. Class
2. Composition
3. Interface
4. **Enum**
5. Access Modifier

Enum

- Enums are one of the few features in TypeScript
- Enums allow a developer to define a set of named constants.
- TypeScript provides both numeric and string-based enums.

❖ Example

```
enum typeUser {
Manager = 'manager',
Employee = 'Employee',
Guest = 'Guest'
```

Enum

```
const creatUser = () => {
 //Create new user
 const employee = new User(20, 'Ahmad',
‘Mohammad');
 //set Type as employee
employee.type = typeUser.Employee;
const employeeMeg = setGreets(employee.type);
            console.log(employeeMeg)
```

Enum

```
const guest = new User(20, 'John', 'Doh');
guest.type = typeUser.Guest;
const guestMsg = setGreets(guest.type);
console.log(guestMsg);
```

Enum

```
const manger = new User(10, 'Jack',
'Mark');
manger.type = typeUser.Manager;
const mangerMsg = setGreets(manger.type);
console.log(mangerMsg)
        }
```

Enum

```
const setGreets=(userType:string)=>{
//send employee mess='Welcome, pleace
complete your task'
if(userType==typeUser.Employee)
return ' Welcome, pleace complete your
task';
```

```
 //send guest mess='welcome here, we hope you
finde what you want '
else if(userType===typeUser.Guest)
return 'welcome here, we hope you finde what you
want';
//send manger mssg='welcome boss !!'
else if(userType===typeUser.Manager)
return 'welcome boss !!';
```

شركة تحالف الإمارات للحلول التقنية ذ.م.م.
TAHALUF AL EMARAT TECHNICAL SOLUTIONS L.L.C.

Day 04

1. Class
2. Composition
3. Interface
4. Enum
5. **Access Modifier**

Access Modifier

- TypeScript supports access modifiers at the class level.

- TypeScript supports three access modifiers - public, private, and protected.

Access Modifier

- **Public** - By default, members (properties and methods) of TypeScript class are public.

- **Private** - A private member cannot be accessed outside of its containing class. Private members can be accessed only within the class

Access Modifier

- **Protected** - A protected member cannot be accessed outside of its containing class. Protected members can be accessed only within the class and by the instance of its sub/child class.

Example

```
class Person {
private name: string;
public weight: number;
protected age: number;
constructor(name: string, weight: number, age:
number) {
this.name = name;
this.weight = weight;
this.age = age;
                }
```

Access Modifier

```
// protected display()
 protected display(): void {
    "I'm Person ";
     }

 // get & set function for name filed
     get Name() : string {
         return this.name;
     }
     set Name(name: string){
         this.name = name;
     }
     }
```

Access Modifier

```
var obj = new Person("Ahmed", 50, 40);
console.log(obj.Name);
console.log(obj.weight);
//public filed
obj.weight = 60;
//private Field
obj.Name = "Sora";
//protected Field
//obj.age=24;//count access outside the class
console.log(obj.Name);
console.log(obj.weight);
```