Typescript essentials. Classes

In this lesson we will look at the classes and interfaces in Typescript. Considering classes, we also will talk about constructors and access modifiers, accessors, extended class, abstract classes and so on.

Class – is a language construction that starts with keyword “class” and contains class name and class body. Class body can contain properties, methods and constructors. Properties define state of future object and methods define object behavior. Let’s have a look at the example of “User” class with two properties and one method (fig. 1).

```
1  class User {
2      firstName: string;
3      lastName: string;
4
5      print(): void {
6          console.log(`${this.firstName} ${this.lastName}`);
7      }
8  }
```

Figure 1 – Typescript class example

To try this class one object is created. Then we output it to the console, after this set values to it’s properties, output it to the console again and finally try its “print” method (fig. 2).

```
10  let user = new User();
11  console.log(user);
12  user.firstName = "John";
13  user.lastName = "Smith";
14  console.log(user);
15  user.print();
```

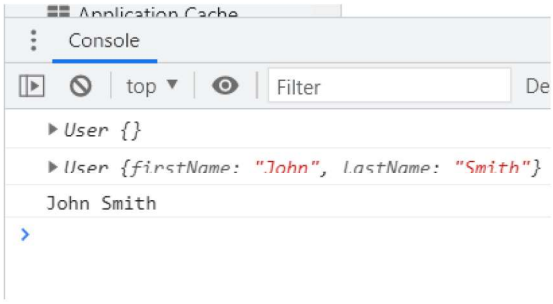

Figure 2 – Creating User object and working with it

At the first time we output empty object that was created but on property values have been set. Then we output the same object with properties that have been set previously. And, finally “print” method was called, which output concatenated properties again.

While working with classes in Typescript we can add to the class body special method called “constructor”. The task of this method is to initialize class instance. In previous example of “User” class, we had no constructor and after creating new User instance we set all properties of the object individually. If we want to initialize some or all User properties when we create it, it can be done by making User class constructor.

Let’s create constructor for “User” class (fig. 3).

```
1   class User {
2       firstName: string;
3       lastName: string;
4
5       constructor() {
6           this.firstName = 'User name';
7           this.lastName = 'User lastname';
8           console.log('User constructor');
9       }
10
11      print(): void {
12          console.log(`${this.firstName} ${this.lastName}`);
13      }
14  }
15
16  console.log('before User creating');
17  let user = new User();
18  console.log('after User creating');
```

Figure 3 – “User” class constructor

The result of executing code from figure 3 is given in fig. 4.

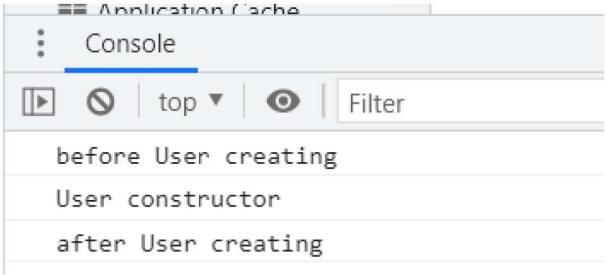

Figure 4 – The result of previously given code

On the next step, we add two input parameters to the constructor to pass values of “firstName” and “lastName” properties (fig. 5).

```
class User {
    firstName: string;
    lastName: string;

    constructor(firstName: string, lastName: string) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    print(): void {
        console.log(`${this.firstName} ${this.lastName}`);
    }
}

let user = new User('John', 'Smith');
user.print();
```

Figure 5 – Passing parameters to class constructor

Next, let's try to figure out with the concept of property access modifiers in Typescript. We have three property access modifier types: "private", "protected" and "public". This defines visibility area of class property. They work quite similar to other programming languages. Public property is visible for everyone, both inside the class and outside. Private property is accessible only inside the class. Protected property can be accessed from the class and also from the class inheritors. If we create class property and don't indicate any access modifier type, the access modifier will by "public" by default.

```
class AccessorsDemoClass {
    private privateProp: string;
    public publicProp: string;
    defaultProp: string;

    constructor(privateProp: string, publicProp: string, defaultProp: string) {
        this.privateProp = privateProp;
        this.publicProp = publicProp;
        this.defaultProp = defaultProp;
    }
}
```

```
12
13  let instance1 = new AccessorsDemoClass('private', 'public', 'default');
14  console.log(instance1.publicProp);
15  console.log(instance1.defaultProp);
16  console.log(instance1.privateProp);
```

```
Property 'privateProp' is private and only accessible within class
'AccessorsDemoClass'. ts(2341)
```

```
(property) AccessorsDemoClass.privateProp: string
```

Figure 6 – Class with public and private property

So if we want to have a property for class inner needs and don't want to give access to this properties from other parts of code, "private" access modifier must be used. But if users of the class need to have possibility to read/write this property outside of the class, we use "public" property access modifiers.

Also in Typescript there is a special keyword "readonly", with the help of which we can create properties that can be only read. It means that property can be used only to get value, but we won't be able to set value for it. The value of "readonly" property can be changed in two cases: when property is initialized and in class constructor (fig. 7).

```
1   class ReadOnlyClass {
2       public readonly readonlyProp: string = 'John Smith';
3
4       constructor(readonlyProp: string) {
5           this.readonlyProp = readonlyProp;
6       }
7
8       setReadOnlyProp(readonlyProp: string) {
9           this.readonlyProp = readonlyProp;
10      }
11  }
```

```
Cannot assign to 'readonlyProp' because it is a read-only
```

```
(property) ReadOnlyClass.readonlyProp: any
```

```
12
13  let readOnlyClassInstance = new ReadOnlyClass('John Bond');
14  console.log(readOnlyClassInstance.readonlyProp);
15  readOnlyClassInstance.readonlyProp = 'James Bond';
```

```
Cannot assign to 'readonlyProp' because it is a read-only property
```

```
(property) ReadOnlyClass.readonlyProp: any
```

Figure 7 – Readonly properties

As we can see, “readonly” property cannot be used to set value not only outside of the class but also inside the class, if it is not inside the constructor.

Typescript has some convenient approaches while working with class properties. For example, we don’t need to declare class property if we initialize it from constructor. Let’s rewrite class “User” from figure 5 with two properties: “firstName” and “lastName” (fig. 8).

```
1  class User {
2      constructor(public firstName: string, public lastName: string) {
3      }
4
5      print(): void {
6          console.log(`${this.firstName} ${this.lastName}`);
7      }
8  }
9
10 let user = new User('John', 'Smith');
11 user.print();
```

Figure 8 – Declaring class properties in the constructor

As can be seen, we simply added parameter access modifiers before it’s name in the constructor. Then we don’t need to declare class property, Typescript does it for us. Then we can simply get (or put) values from it using “this” link. It also need to be mentioned that “private” and “protected” access modifiers can be used.

Existence of “private” access modifiers makes it possible to encapsulate some properties inside the class. If we decide to hide something inside the class, we most likely will want to give some kind of limited access to it from outside the class. This can be done by creating special getter and setter methods with “public” access modifier. For example, let’s create “Student” class with the private “name” property and two methods for get and set values to it (fig. 9).

```
1   class Student {
2       private _name: string;
3
4       setName(name: string) {
5           this._name = name;
6       }
7
8       getName() {
9           return this._name;
10      }
11  }
12
13   let student = new Student();
14
15  student.setName('John Smith');
16  console.log(student.getName());
17
18  student._name = 'Piter Pen';
19  console.log(student._name);
```

Figure 9 – Getter and setter for private property

As we can see, private property "_name" can be simply accessed using "getName" and "setName" methods. And it also cannot be accessed directly.

Typescript also has special mechanism that is called accessor. With the help of accessor we can work with properties as usual, but when we address the property, special methods will be executed. Let's add private "age" property to the "Student" class and use accessors to make it possible to get and set values from/to this property.

```
2       private _name: string;
3       private _age: number;
4
5       get age(): number {
6           return this._age;
7       }
8
9       set age(age: number) {
10          this._age = age;
11      }
```

```
31
32  student.age = 20;
33  console.log(student.age);
```

Figure 10 – Using accessors to get and set values of private property

The “get” method might not has any parameters and need to return value with the datatype of getted parameter. The “set” method has one parameter with the same data type. The main difference from getter and setter methods is that with accessor we work as with common class property.

Typescript also has static properties and methods. They are shared for all instances of the class. For example, let’s create “Counter” class that has counter value. It can be increased and shown on the console. Each counter will have its own value. Besides, we will have global counter value, which will be increased when any of the counter increases its value (fig. 11).

```
1   class Counter {
2       private _counterValue = 0;
3       private static _globalCounterValue = 0;
4
5       increase() {
6           this._counterValue++;
7           Counter._globalCounterValue++;
8       }
9
10      show() {
11          console.log(this._counterValue);
12      }
13
14      static showGlobal() {
15          console.log(Counter._globalCounterValue);
16      }
17  }
```

```
18
19  let counter1 = new Counter();
20  let counter2 = new Counter();
21  counter1.increase();
22  counter1.increase();
23  counter2.increase();
24  counter1.show();
25  counter2.show();
26  Counter.showGlobal();
```

Figure 11 – Example of Typescript static properties and methods

To access such a property or method we use class name instead of “this” link. In the previous example we create two instances of “Counter” class, then increase first one two times and second one once. After that, first and second counter value is shown. Also global counter value is shown too (fig. 12).

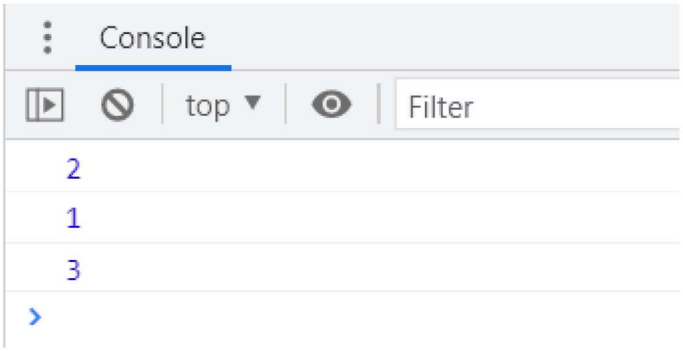

Figure 12 – The result of previously given code execution