TypeScript - Classes

In object-oriented programming languages like Java and C#, classes are the fundamental entities used to create reusable components. Functionalities are passed down to classes and objects are created from classes. However, until ECMAScript 6 (also known as ECMAScript 2015), this was not the case with JavaScript. JavaScript has been primarily a functional programming language where inheritance is prototype-based. Functions are used to build reusable components. In ECMAScript 6, object-oriented class based approach was introduced. TypeScript introduced classes to avail the benefit of object-oriented techniques like encapsulation and abstraction. The class in TypeScript is compiled to plain JavaScript functions by the TypeScript compiler to work across platforms and browsers.

A class can include the following:

* Constructor
* Properties
* Methods

The following is an example of a class in TypeScript:

Example: Class

 Copy

class Employee {

empCode: number;

empName: string;

constructor(code: number, name: string) {

this.empName = name;

this.empCode = code;

}

getSalary() : number {

return 10000;

}

}

The TypeScript compiler will convert the above class to the following JavaScript code using [closure](https://www.tutorialsteacher.com/javascript/closure-in-javascript):

var Employee = /\*\* @class \*/ (function () {

function Employee(name, code) {

this.empName = name;

this.empCode = code;

}

Employee.prototype.getSalary = function () {

return 10000;

};

return Employee;

}());

Constructor

The constructor is a special type of method which is called when creating an object. In TypeScript, the constructor method is always defined with the name "constructor".

Example: Constructor

 Copy

class Employee {

empCode: number;

empName: string;

constructor(empcode: number, name: string ) {

this.empCode = empcode;

this.name = name;

}

}

In the above example, the Employee class includes a constructor with the parameters empcode and name. In the constructor, members of the class can be accessed using this keyword e.g. this.empCode or this.name.

It is not necessary for a class to have a constructor.

Example: Class without Constructor

 Copy

class Employee {

empCode: number;

empName: string;

}

Creating an Object of Class

An object of the class can be created using the [new keyword](https://www.tutorialsteacher.com/javascript/new-keyword-in-javascript).

Example: Create an Object

 Copy

class Employee {

empCode: number;

empName: string;

}

let emp = new Employee();

Here, we create an object called emp of type Employee using let emp = new Employee();. The above class does not include any parameterized constructor so we cannot pass values while creating an object. If the class includes a parameterized constructor, then we can pass the values while creating the object.

class Employee {

empCode: number;

empName: string;

constructor(empcode: number, name: string ) {

this.empCode = empcode;

this.name = name;

}

}

let emp = new Employee(100,"Steve");

In the above example, we pass values to the object to initialize the member variables. When we instantiate a new object, the class constructor is called with the values passed and the member variables empCode and empName are initialized with these values.

Inheritance

Just like object-oriented languages such as Java and C#, TypeScript classes can be extended to create new classes with inheritance, using the keyword extends.

Example: Inheritance

 Copy

class Person {

name: string;

constructor(name: string) {

this.name = name;

}

}

class Employee extends Person {

empCode: number;

constructor(empcode: number, name:string) {

super(name);

this.empCode = empcode;

}

displayName():void {

console.log("Name = " + this.name + ", Employee Code = " + this.empCode);

}

}

let emp = new Employee(100, "Bill");

emp.displayName(); // Name = Bill, Employee Code = 100

In the above example, the Employee class extends the Person class using extends keyword. This means that the Employee class now includes all the members of the Person class.

The constructor of the Employee class initializes its own members as well as the parent class's properties using a special keyword 'super'. The super keyword is used to call the parent constructor and passes the property values.

 Note:

We must call super() method first before assigning values to properties in the constructor of the derived class.

Class Implements Interface

A class can implement single or multiple interfaces.

Example: Implement Interface

 Copy

interface IPerson {

name: string;

display():void;

}

interface IEmployee {

empCode: number;

}

class Employee implements IPerson, IEmployee {

empCode: number;

name: string;

constructor(empcode: number, name:string) {

this.empCode = empcode;

this.name = name;

}

display(): void {

console.log("Name = " + this.name + ", Employee Code = " + this.empCode);

}

}

let per:IPerson = new Employee(100, "Bill");

per.display(); // Name = Bill, Employee Code = 100

let emp:IEmployee = new Employee(100, "Bill");

emp.display(); //Compiler Error: Property 'display' does not exist on type 'IEmployee'

In the above example, the Employee class implements two interfaces - IPerson and IEmployee. So, an instance of the Employee class can be assigned to a variable of IPerson or IEmployee type. However, an object of type IEmployee cannot call the display() method because IEmployee does not include it. You can only use properties and methods specific to the object type.

Interface extends Class

An interface can also extend a class to represent a type.

Example: Interface Extends Class

 Copy

class Person {

name: string;

}

interface IEmployee extends Person {

empCode: number;

}

let emp: IEmployee = { empCode : 1, name:"James Bond" }

In the above example, IEmployee is an interface that extends the Person class. So, we can declare a variable of type IEmployee with two properties. So now, we must declare and initialize values at the same time.

Method Overriding

When a child class defines its own implementation of a method from the parent class, it is called method overriding.

Example: Method Overriding

 Copy

class Car {

name: string;

constructor(name: string) {

this.name = name;

}

run(speed:number = 0) {

console.log("A " + this.name + " is moving at " + speed + " mph!");

}

}

class Mercedes extends Car {

constructor(name: string) {

super(name);

}

run(speed = 150) {

console.log('A Mercedes started')

super.run(speed);

}

}

class Honda extends Car {

constructor(name: string) {

super(name);

}

run(speed = 100) {

console.log('A Honda started')

super.run(speed);

}

}

let mercObj = new Mercedes("Mercedes-Benz GLA");

let hondaObj = new Honda("Honda City")

mercObj.run(); // A Mercedes started A Mercedes-Benz GLA is moving at 150 mph!

hondaObj.run(); // A Honda started A Honda City is moving at 100 mph!

In the above example, we have a class Car with the name property. The constructor for this class initializes the member variables. The class also has a method display()with an argument speed initialized to 0.

We then create two classes, Mercedes and Honda, that extend from the parent class Car. Each child class extends the properties of the parent class. The constructor for each class calls the super constructor to initialize the parent class properties. Each class also defines a method run() that prints its own message in addition to calling the super class method for run().

Since each child class has its own implementation of the method run(), it is called method overriding, i.e. the children classes have a method of the same name as that of the parent class.