TypeScript Abstract Classes

Summary: in this tutorial, you will learn about TypeScript abstract classes and how to use them to define common behaviors for derived classes.

Introduction to TypeScript abstract classes

An abstract class is typically used to define common behaviors for derived classes to extend. Unlike a regular class, an abstract class cannot be instantiated directly.

To declare an abstract class, you use the `abstract` keyword:

```
abstract class Employee {
    //...
}
```

Typically, an abstract class contains one or more abstract methods.

An abstract method does not contain implementation. It only defines the signature of the method without including the method body. An abstract method must be implemented in the derived class.

The following shows the `Employee` abstract class that has the `getSalary()` abstract method:

```
abstract class Employee {
  constructor(private firstName: string, private lastName: string) {}
  abstract getSalary(): number;
  get fullName(): string {
    return `${this.firstName} ${this.lastName}`;
```

```
  }
  compensationStatement(): string {
    return `${this.fullName} makes ${this.getSalary()} a month.`;
  }
}
```

In the `Employee` class:

- The constructor declares the `firstName` and `lastName` properties.
- The `getSalary()` method is an abstract method. The derived class will implement the logic based on the type of employee.
- The `getFullName()` and `compensationStatement()` methods contain detailed implementation. Note that the `compensationStatement()` method calls the `getSalary()` method.

Because the `Employee` class is abstract, you cannot create a new object from it. The following statement causes an error:

```
let employee = new Employee('John','Doe');
```

Error:

```
error TS2511: Cannot create an instance of an abstract class.
```

The following `FullTimeEmployee` class inherits from the `Employee` class:

```
class FullTimeEmployee extends Employee {
    constructor(firstName: string, lastName: string, private salary: number) {
        super(firstName, lastName);
    }
    getSalary(): number {
        return this.salary;
    }
}
```

In this `FullTimeEmployee` class, the salary is set in the constructor. Because the `getSalary()` is an abstract method of the `Employee` class, the `FullTimeEmployee` class needs to implement this

method. In this example, it just returns the salary without any calculation.

The following shows the `Contractor` class that also inherits from the `Employee` class:

```
class Contractor extends Employee {
  constructor(
    firstName: string,
    lastName: string,
    private rate: number,
    private hours: number
  ) {
    super(firstName, lastName);
  }
  getSalary(): number {
    return this.rate * this.hours;
  }
}
```

In the `Contractor` class, the constructor initializes the rate and hours. The `getSalary()` method calculates the salary by multiplying the rate by the hours.

The following first creates a `FullTimeEmployee` object and a `Contractor` object and then shows the compensation statements to the console:

```
let john = new FullTimeEmployee('John', 'Doe', 12000);
let jane = new Contractor('Jane', 'Doe', 100, 160);

console.log(john.compensationStatement());
console.log(jane.compensationStatement());
```

Output:

```
John Doe makes 12000 a month.
Jane Doe makes 16000 a month.
```

It's a good practice to use abstract classes when you want to share code among some related classes.

Summary

- Abstract classes cannot be instantiated.
- An Abstract class has at least one abstract method.
- To use an abstract class, you need to inherit it and provide the implementation for the abstract methods.