TypeScript Inheritance

Summary: in this tutorial, you'll learn about the TypeScript inheritance concept and how to use it to reuse the functionality of another class.

Introduction to the TypeScript inheritance

A class can reuse the properties and methods of another class. This is called inheritance in TypeScript.

The class which inherits properties and methods is called the **child class**. The class whose properties and methods are inherited is known as the **parent class**. These names come from the nature that children inherit genes from their parents.

Inheritance allows you to reuse the functionality of an existing class without rewriting it.

JavaScript uses prototypal inheritance, not classical inheritance like Java or C#. ES6 introduces the class syntax that is simply the syntactic sugar of the prototypal inheritance. TypeScript supports inheritance like ES6.

Suppose you have the following `Person` class:

```
class Person {
  constructor(private firstName: string, private lastName: string) {}
  getFullName(): string {
    return `${this.firstName} ${this.lastName}`;
  }
  describe(): string {
    return `This is ${this.firstName} ${this.lastName}.`;
```

```
  }
}
```

To inherit a class, you use the `extends` keyword. For example the following `Employee` class inherits the `Person` class:

```
class Employee extends Person {
  //...
}
```

In this example, the `Employee` is a child class and the `Person` is the parent class.

Constructor

Because the `Person` class has a constructor that initializes the `firstName` and `lastName` properties, you need to initialize these properties in the constructor of the `Employee` class by calling its parent class' constructor.

To call the constructor of the parent class in the constructor of the child class, you use the `super()` syntax. For example:

```
class Employee extends Person {
    constructor(
        firstName: string,
        lastName: string,
        private jobTitle: string) {

        // call the constructor of the Person class:
        super(firstName, lastName);
    }
}
```

The following creates an instance of the `Employee` class:

```
let employee = new Employee('John','Doe','Front-end Developer');
```

Because the `Employee` class inherits properties and methods of the `Person` class, you can call the `getFullName()` and `describe()` methods on the `employee` object as follows:

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

console.log(employee.getFullName());
console.log(employee.describe());
```

Output:

```
John Doe
This is John Doe.
```

Method overriding

When you call the `employee.describe()` method on the `employee` object, the `describe()` method of the `Person` class is executed that shows the message: `This is John Doe`.

If you want the `Employee` class has its own version of the `describe()` method, you can define it in the `Employee` class like this:

```
class Employee extends Person {
    constructor(
        firstName: string,
        lastName: string,
        private jobTitle: string) {

        super(firstName, lastName);
    }

    describe(): string {
        return super.describe() + `I'm a ${this.jobTitle}.`;
    }
}
```

In the `describe()` method, we called the `describe()` method of the parent class using the syntax `super.methodInParentClass()`.

If you call the `describe()` method on the `employee` object, the `describe()` method in the `Employee` class is invoked:

```
let employee = new Employee('John', 'Doe', 'Web Developer');
console.log(employee.describe());
```

Output:

```
This is John Doe.I'm a Web Developer.
```

Summary

- Use the `extends` keyword to allow a class to inherit from another class.
- Use `super()` to call the constructor of the parent class in the constructor of the child class. Also, use the `super.methodInParentClass()` syntax to invoke the `methodInParentClass()` in the method of the child class.