(10.1.1) ES6 Intermediate - Class - Inheritance

Understanding Inheritance in JavaScript Classes
Inheritance is a fundamental concept in object-oriented programming (OOP), and JavaScript, despite its prototype-based nature, supports inheritance through the use of classes and the `extends` keyword introduced in ECMAScript 6 (ES6). In this article, we'll delve into the concept of inheritance in JavaScript classes, exploring how it works and why it's a crucial aspect of building modular and maintainable code.

Introduction to JavaScript Classes
Before we dive into inheritance, let's briefly revisit JavaScript classes. JavaScript classes provide a more structured and syntactic way to create objects and define their behavior compared to the traditional constructor functions and prototypes. A class encapsulates both data (properties) and behavior (methods) into a single entity.

Here's a simple example of a JavaScript class:

```
class Animal {
constructor(name) {
this.name = name;
}

speak() {
console.log(`${this.name} makes a sound.`);
}
}

const dog = new Animal("Fido");
dog.speak(); // Output: Fido makes a sound.
```

In this example, we have defined an `Animal` class with a constructor to set the `name` property and a `speak` method.

The `extends` Keyword
JavaScript classes support inheritance through the `extends` keyword. This keyword allows you to create a new class that inherits properties and methods

from an existing class, forming a parent-child relationship.

```
class Dog extends Animal {
constructor(name, breed) {
super(name);
this.breed = breed;
}

speak() {
console.log(`${this.name} barks.`);
}
}

const goldenRetriever = new Dog("Buddy", "Golden Retriever");
goldenRetriever.speak(); // Output: Buddy barks.
```

In this example, the `Dog` class extends the `Animal` class using `extends`. This means that `Dog` inherits the `name` property and the `speak` method from `Animal`. However, it also defines its own `speak` method, which overrides the method inherited from `Animal`.

The `super` Keyword

Inside the constructor of a child class, you can use the `super` keyword to call the constructor of the parent class. This is necessary because child classes often need to set up their own properties while also invoking the parent class's constructor to initialize inherited properties.

In the previous example, the `Dog` class uses `super(name)` to call the `Animal` constructor and set the `name` property inherited from `Animal`.

Method Overriding

Method overriding is a key aspect of inheritance. It allows child classes to provide their own implementation of a method that is already defined in the parent class. When a method is called on an instance of a child class, JavaScript will first look for the method in the child class. If it's found, it will execute the child class's implementation; otherwise, it will fall back to the parent class's method.

In our `Dog` class example, the `speak` method is overridden to provide a different behavior than the `speak` method in the `Animal` class.

```
const goldenRetriever = new Dog("Buddy", "Golden Retriever");
goldenRetriever.speak(); // Output: Buddy barks.
```

Multiple Inheritance in JavaScript

JavaScript supports single inheritance, which means a class can extend only one parent class. Unlike some other languages, JavaScript does not support multiple inheritance directly. However, you can achieve a form of multiple inheritance using a technique called "mixins" or by composing classes with various traits.

Mixins involve creating standalone classes with reusable functionality and then combining them into a single class using composition. This allows you to incorporate behaviors from multiple sources into a single class.

```
class Swimmer {
swim() {
console.log(`${this.name} is swimming.`);
}
}

class Flyer {
fly() {
console.log(`${this.name} is flying.`);
}
}

class Duck extends Animal {}

Object.assign(Duck.prototype, Swimmer.prototype, Flyer.prototype);

const mallard = new Duck("Mallard");
mallard.swim(); // Output: Mallard is swimming.
mallard.fly(); // Output: Mallard is flying.
```

In this example, we've created two mixin classes, `Swimmer` and `Flyer`, and applied them to the `Duck` class using `Object.assign`. This allows instances of `Duck` to use the `swim` and `fly` methods even though `Duck` directly extends the `Animal` class.

Conclusion

Inheritance is a powerful mechanism in JavaScript that enables you to create hierarchies of classes, share and extend functionality, and build modular and

maintainable code. With the `extends` keyword, `super` keyword, and method overriding, you can effectively implement inheritance in JavaScript classes. While JavaScript supports single inheritance, you can achieve similar results for multiple inheritance by using mixins or composition techniques. Understanding how inheritance works in JavaScript is essential for writing organized and reusable code in modern web development.