Tip 38: Share Methods with Inheritance

In this tip, you'll learn how to extend classes and call parent methods.

We'll cover the following

In the previous tip, you learned how to create basic classes with properties and methods. You may recall that classes in JavaScript were highly anticipated and slightly dreaded. The core of the controversy is *inheritance*.

Inheritance in JavaScript classes

Inheriting methods on *prototypes* was a pretty complex process in early versions of JavaScript. First, you had to loop through the properties on an object; then, you had to check to see that each property existed specifically on an object as a property and not on the object prototype. Then you had to copy the prototype from the parent to a new object before adding further methods.

It was hard.

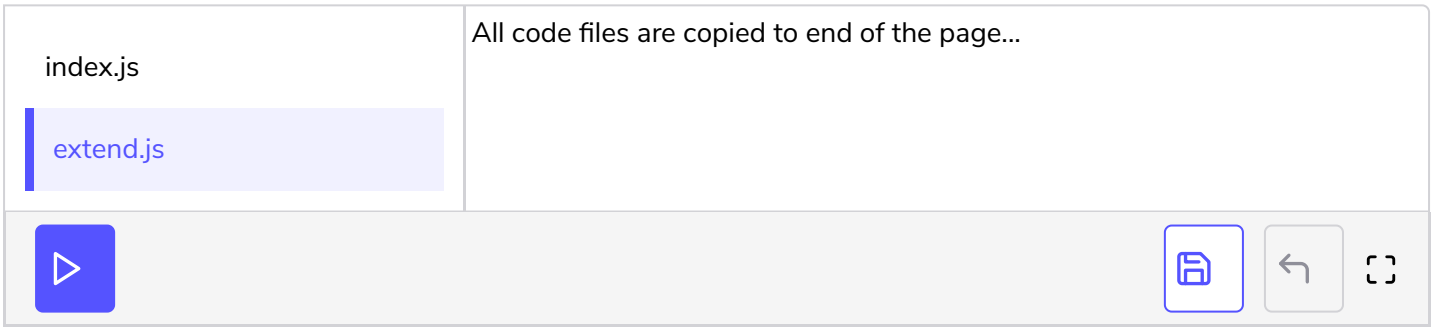
With classes, inheritance is easy. Still, in recent years, JavaScript developers have soured on inheritance. They argue that too much inheritance is a bad thing and that it leads to bloated code. There are other techniques for sharing methods that don't require inheritance (such as composition). In other words, use inheritance with caution.

Example: Inheritance

How does inheritance work? Return to your `Coupon` class. Suppose you want a `FlashCoupon` that has deeper discounts but a shorter time span. To create that class,

simply declare a new class called `FlashCoupon` that inherits from the `Coupon` class using the `extends` keyword.

Your new `FlashCoupon` class inherits all the *existing* properties and methods. For example, you can access the `price` and the `getPriceText()` method.

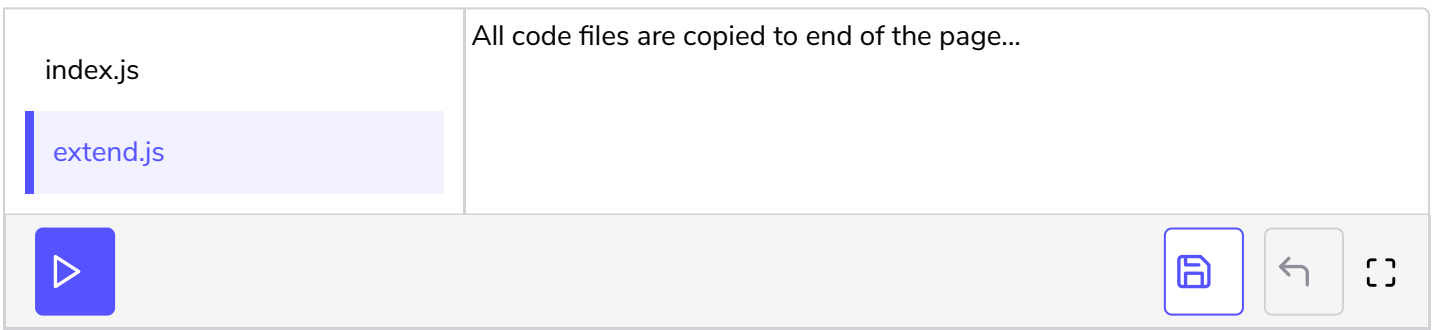


Of course, there's really no point in inheriting code if you aren't going to add new properties or methods. To make this coupon different, add a new default `expiration`. Make the new default expiration `"two hours"` instead of `"two weeks"`.

Calling `super` in a constructor

To make the change, set up a constructor function that takes `price` and `expiration`, as you did on the parent. In this `constructor`, you'll need to call `super()` to access the *parent* constructor. `super()` *calls* the parent constructor, so pass through any arguments the parent constructor might need. In this case, you'll need to pass the `price` to the parent constructor. After that, you can set any new properties or override any properties that the parent constructor might set.

For the `FlashCoupon`, you're setting the `expiration`, but you don't need to worry about setting the `price`. The parent `constructor` takes care of that.



You're using the parent `getExpirationMessage()` method, but you're using the child's `expiration` property. When you call `getExpirationMessage()`, you'll see the familiar message with the new default expiration.

Of course, you may not like that message. This is a flash coupon after all. You should alert your users that this coupon is special. Any time you call a method, the JavaScript engine first checks to see if the method exists on the current class. If not, the engine goes up the chain, checking each class or prototype along the way. This means you can *override* any method by creating a new method with the *same* name.

Overriding parent methods

Try adding a new method called `getExpirationMessage()` to the `FlashCoupon` class. This method will be the same as the parent method except that it returns a message of `This is a flash offer and expires in ${this.expiration}`.

At this point, you've created a class that inherits methods and properties. You called the parent `constructor` to set some properties and overrode other properties. You also wrote methods that *override* parent methods.

Invoking parent methods

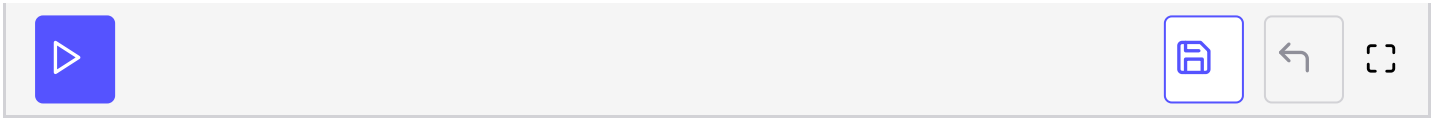
The last step is to write methods that invoke the parent methods. To start, add two new methods to your `Coupon` class. First, add the method `getRewards()`, which takes a user and then calls `isRewardsEligible()` to find out if the user is eligible for further discounts. If the user is eligible for further discounts, reduce the `price`.

> As a warning, any method you add to a parent class is inherited by a child class. This can be a huge benefit, but it's also easy to create bloat in child classes by adding methods to parents that aren't necessary in child classes.

```
class Coupon {
    constructor(price, expiration) {
        this.price = price;
        this.expiration = expiration || 'Two Weeks';
    }
    getPriceText() {
        return `${this.price}`;
    }
    getExpirationMessage() {
        return `This offer expires in ${this.expiration}`;
    }
    isRewardsEligible(user) {
        return user.rewardsEligible && user.active;
    }
    getRewards(user) {
        if (this.isRewardsEligible(user)) {
            this.price = this.price * 0.9;
```

```
        }
    }
}


const coupon = new Coupon(10);
const user = {
    rewardsEligible: true,
    active: true,
};
console.log(coupon.price);
coupon.getRewards(user);
console.log(coupon.price);
```



Giving users a discount is great, but because flash coupons are an even bigger savings, you'll probably want to give eligible users a larger discount on flash coupons. But you don't want to give away too much. Instead, you only want to give your eligible users a discount if the previous conditions are met, a user is active and is rewards-eligible, and the base price of the item is `$20` or more.

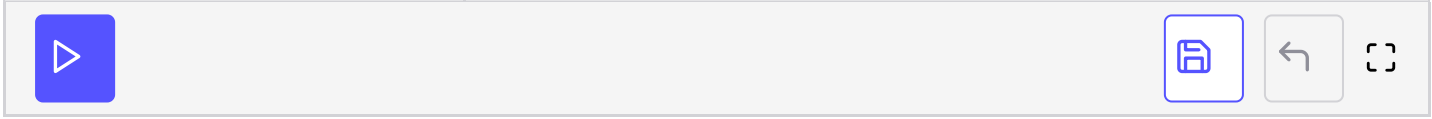
To add this in, first create a method of the same name in the `FlashCoupon` class. Then, in the `isRewardsEligible()` method, first call the parent method by calling the method name on `super()`. After that, add your additional code. Note that `super()` in the constructor doesn't need a specific method call, but if you want to call any other methods on the parent class, you'll have to specify them, even when they're in a method of the same name.

The result is a class that inherits some properties and functions from a parent while overriding others.

index.js

extend.js

All code files are copied to end of the page...



That's all there is to it. For those familiar with object-oriented programming, this should be very familiar. Still, you should remember that JavaScript isn't the same as Ruby, Java, or other languages that use classes. JavaScript is a prototype language. Classes as you're using them are simply a familiar syntax for a different paradigm. The benefit is that because they are using the same prototype actions under the hood, you can combine classes with legacy code.

Q

What will be the output of the following code?

```
class Vehicle{
    constructor(){
        this.color = "Red";
    }
}

class Car extends Vehicle{
    constructor(){
        super();
        this.color = "Blue";
    }
}

var car = new Car();
console.log(car.color);
```

Retake Quiz

In the next tip, you'll see how classes relate to pre-ES6 JavaScript and how you can combine the two approaches in the same codebase.

Code Files Content !!!

⤋ ⤋

```
--------------------------------------------------------------------------
|  index.js [1]
--------------------------------------------------------------------------


import Coupon from './extend';

class FlashCoupon extends Coupon {
}

const flash = new FlashCoupon(10);
console.log(flash.price);
console.log(flash.getPriceText());


--------------------------------------------------------------------------
|  extend.js [1]
--------------------------------------------------------------------------


class Coupon {
  constructor(price, expiration) {
    this.price = price;
    this.expiration = expiration || 'Two Weeks';
  }

  getPriceText() {
```

```
  getPriceText() {
    return `${this.price}`;
  }

  getExpirationMessage() {
    return `This offer expires in ${this.expiration}`;
  }

  isRewardsEligible(user) {
    return user.rewardsEligible && user.active;
  }

  getRewards(user) {
    if (this.isRewardsEligible(user)) {
      this.price = this.price * 0.9;
    }
  }
}

export default Coupon;
```

```
********************************************************************************
```

```
------------------------------------------------------------------------
|  index.js [2]
------------------------------------------------------------------------
```

```
import Coupon from './extend';

class FlashCoupon extends Coupon {
    constructor(price, expiration) {
        super(price);
        this.expiration = expiration || 'two hours';
    }
}

const flash = new FlashCoupon(10);
console.log(flash.price);
console.log(flash.getExpirationMessage());
```

```
------------------------------------------------------------------------
|  extend.js [2]
------------------------------------------------------------------------
```

```
class Coupon {
  constructor(price, expiration) {
    this.price = price;
    this.expiration = expiration || 'Two Weeks';
  }

  getPriceText() {
    return `${this.price}`;
  }

  getExpirationMessage() {
```

```
    return `This offer expires in ${this.expiration}`;
  }

  isRewardsEligible(user) {
    return user.rewardsEligible && user.active;
  }

  getRewards(user) {
    if (this.isRewardsEligible(user)) {
      this.price = this.price * 0.9;
    }
  }
}

export default Coupon;
```

****

```
------------------------------------------------------------------------
|  index.js [3]
------------------------------------------------------------------------


import Coupon from './extend';

class FlashCoupon extends Coupon {
    constructor(price, expiration) {
        super(price);
        this.expiration = expiration || 'two hours';
    }
    getExpirationMessage() {
        return `This is a flash offer and expires in ${this.expiration}.`;
    }
    isRewardsEligible(user) {
        return super.isRewardsEligible(user) && this.price > 20;
    }
    getRewards(user) {
        if (this.isRewardsEligible(user)) {
            this.price = this.price * 0.8;
        }
    }
}

const flash = new FlashCoupon(10);
const user = {
    rewardsEligible: true,
    active: true,
};
console.log(flash.price);
console.log(flash.getPriceText());
console.log(flash.getExpirationMessage());
flash.getRewards(user);
console.log(flash.price);


------------------------------------------------------------------------
|  extend.js [3]
```

```
--------------------------------------------------------------------------------


class Coupon {
  constructor(price, expiration) {
    this.price = price;
    this.expiration = expiration || 'Two Weeks';
  }

  getPriceText() {
    return `${this.price}`;
  }

  getExpirationMessage() {
    return `This offer expires in ${this.expiration}`;
  }

  isRewardsEligible(user) {
    return user.rewardsEligible && user.active;
  }

  getRewards(user) {
    if (this.isRewardsEligible(user)) {
      this.price = this.price * 0.9;
    }
  }
}

export default Coupon;




********************************************************************************
```