Spread Operator and Destructuring (Objects)

Destructuring

We've already discussed about destructuring before, but now we'll discuss it specifically for objects.

Destructuring an object is a bit different from destructuring an array. When destructuring an array, we can freely use any variable name to contain the destructured values of the array. But, when destructuring an object, we need to use the property's name as the variable. Of course another difference is we need to use curly braces (`{}`) instead of brackets (`[]`).

```
const { fullName, age, address } = {
  fullName: "John Doe",
  age: 34,
  address: "Foo Street"
}

console.log(fullName) // "John Doe"
console.log(age)      // 34
console.log(address)  // "Foo Street"
```

All other destructuring concepts in arrays also apply to objects. We can destructure objects stored within an object, we're also allowed to destructure only some of the properties in an object, and we can also access destructured properties using dot or bracket notations.

```
const user = {
  fullName: "John Doe",
  age: 34,
  address: "Foo Street"
}

const { fullName } = user

console.log(fullName)         // "John Doe"
console.log(user.fullName)    // "John Doe"
console.log(user.age)         // 34
```

Spread Operator

Again, the spread operator works the same for objects as it does for arrays. The spread operator also "remove" the curly braces in an object, so that means we can duplicate objects, add new properties, and even edit property values.

```
// Example 2.1
const oldUser = {
  fullName: "John Doe",
  age: 34,
  address: "Foo Street"
}

// Duplicating an object
const newUser1 = { ...oldUser }
console.log(newUser1)
// {
//  fullName: "John Doe",
//  age: 34,
//  address: "Foo Street"
// }

// Adding a property
const newUser2 = { ...oldUser, status: "married" }
console.log(newUser2)
// {
//  fullName: "John Doe",
//  age: 34,
//  address: "Foo Street",
//  status: "married"
// }

// Editing a property's value
const newUser3 = { ...oldUser, fullName: "Jane Foo" }
console.log(newUser3)
// {
//  fullName: "Jane Foo",
//  age: 34,
//  address: "Foo Street"
// }
```

Duplicating and adding a new property look quite simple since the syntax is nearly identical to arrays. Editing a property's value seems pretty easy too since all we have to do is "add" a new property but with the same property name as the property we want to edit. However there is one small detail that is very important. The new property's value should be written after the spread object.

```
// Example 2.2
const oldUser = {
  fullName: "John Doe",
  age: 34,
  address: "Foo Street"
}

// Correct
const newUser = { ...oldUser, fullName: "Jane Foo" }

// Wrong
const newUser2 = { fullName: "Jane Foo", ...oldUser }
```

Remember that the value of a property in an object is taken from the last written value. This means that if we have 2 properties which are identical in name, the final value that will be used is the value that is last written to that property.

```
// Example 2.3
const obj = {
  message: "Hello",
  message: "World",
}

console.log(obj) // { message: 'World' }
```

In the example above, `obj.message` has a value of `"World"` because even though it was initially given the value of `"Hello"`, it was later overwritten by the value `"World"`. Now if we take a look at example 2.2, we can see that `newUser.fullName` which was duplicated from `oldUser` and has a value of `"John Doe"` has been overwritten by a new value `"Jane Foo"`. However `newUser2` is the complete opposite of `newUser1`. `newUser2` initially has a property called `fullName` with the value of `"Jane Foo"`, but it was then overwritten by a spread `oldUser` that also has a `fullName` property, making the final value of `newUser2.fullName` , `"John Doe"`.