Symbols as Semi-Private Property Keys

advantages of making object properties private, its drawbacks and use cases

Creating truly private properties and operations is feasible, but it's not an obvious task in JavaScript. If it was as obvious as in Java, blog posts like this, this, this, this, and many more wouldn't have emerged.

Check out Exercise 2 to find out more about how to simulate private variables in JavaScript to decide whether it's worth for you.

Even though Symbols do not make attributes private, they can be used as a notation for private properties. You can use symbols to separate the enumeration of public and private properties, and the notation also makes it clear.

```
const _width = Symbol('width');
class Square {
    constructor( width0 ) {
        this[_width] = width0;
    }
    getWidth() {
        return this[_width];
    }
}
```

As long as you can hide the `_width` constant, you should be fine. One option to hide `_width` is to create a closure:

```
let Square = (function() {

    const _width = Symbol('width');

    return class {
        constructor( width0 ) {
            this[_width] = width0;
        }
        getWidth() {
            return this[_width];
        }
    }

} )();
```

The advantage of this approach is that it becomes intentionally harder to access the private `_width` value of our objects. It is also evident which of our properties are intended to be public, and which private. The solution is not bulletproof, but some developers do use this approach in favor of indicating privacy by starting a variable with an underscore.

The drawbacks are also obvious:

- By calling `Object.getOwnPropertySymbols`, we can get access to the symbol keys. Therefore, private fields are not truly private.
- Developer experience is also worse, as they have to write more code. Accessing private properties is not as convenient as in Java or TypeScript for example.

Some developers will express their opinion on using symbols for indicating privacy. In practice, your team has the freedom of deciding which practices to stick to, and which rules to follow. If you agree on using symbols as private keys, it is a working solution, as long as you don't start writing workarounds to access private field values publicly.

If you use symbols to denote private fields, you have done your best to indicate that a property is not to be accessed publicly. When someone writes code violating this common sense intention, they should bear the consequences.

There are various methods for structuring your code such that you indicate that some of your variables are private in JavaScript. None of them looks as elegant as a `private` access modifier.

If you want real privacy, you can achieve it even without using ES6. Exercise 2 deals with this topic. Try to solve it, or read the reference solution.

The question is not whether it is possible to simulate private fields in

JavaScript. The real question is whether you want to simulate them or not.

Once you figure out that you don’t need truly private fields for development, you can agree whether you use symbols, weak maps (covered later), closures, or a simple underscore prefix in front of your variables.

In the next lesson, let’s talk about enum types.