Protecting Properties

This lesson explains the steps to hide properties in the ES6 version of classes.

WE'LL COVER THE FOLLOWING

The method to protect the class properties in ES6 is similar to what we discussed in the previous chapter.

Hiding Properties in ES6

In order to hide the *class properties*, they can be declared inside the `constructor` using the `var` *keyword* so that they're only accessible through getters/setters, which can also be defined inside the `constructor`.

As discussed for the ES5 version, the convention of using `_` prefix before values that need to be protected is also followed in the ES6 version.

Example

Let's take a look at an example that implements data hiding in the ES6 version:

```
class Student {
  constructor(name,age,sex,grade) {
    //properties hidden
    var _name = name
    var _age = age
    var _sex = sex
    var _grade = grade
    this.getName = function() {
      return _name
    }
    this.getAge = function() {
        return _age
      }
      this.getSex = function() {
        return _sex
      }
      this.getGrade = function() {
        return _grade
      }
    }
}
var student1 = new Student('Kate',15,'F',8)
console.log("Name:",student1.getName())
console.log("Age:",student1.getAge())
console.log("Sex:",student1.getSex())
console.log("Grade:",student1.getGrade())
```

Explanation

As all class properties are declared locally using `var`, they can only be accessed through the public *get* functions. Since these functions are declared inside the constructor, they have access to these internal properties and are therefore used to *get* the values.

```
class Student {
  constructor(name,age,sex,grade) {
    //properties hidden
    var _name = name
    var _age = age
    var _sex = sex
    var _grade = grade
  }
}
var student1 = new Student('Kate',15,'F',8)
console.log("Name:",student1._name)
console.log("Age:",student1._age)
console.log("Sex:",student1._sex)
console.log("Grade:",student1._grade)
```

Accessing any of the protected properties directly outside the class would result in `undefined`, as discussed in the previous chapter.

In the next lesson, let's discuss what static methods are and how they are implemented.