JavaScript Private Methods

Summary: in this tutorial, you'll learn about JavaScript private methods including private instance methods, private static methods, and private getter/setter.

Introduction to JavaScript private methods

By default, members of a class are public. ES2020 introduced the private members that include private fields and methods.

To make a public method private, you prefix its name with a hash `#`. JavaScript allows you to define private methods for instance methods, static methods, and getter/setters.

The following shows the syntax of defining a private instance method:

```
class MyClass {
  #privateMethod() {
    //...
  }
}
```

In this syntax, the `#privateMethod` is a private instance method. It can only be called inside the `MyClass`. In other words, it cannot be called from outside the class or in the subclasses of the `MyClass`.

To call the `#privateMethod` inside the `MyClass`, you use the `this` keyword as follows:

```
this.#privateMethod();
```

The following illustrates the syntax of defining a private static method:

```
class MyClass {
  static #privateStaticMethod() {
    //...
  }
}
```

To call the `#privateStaticMethod()` inside the `MyClass`, you use the class name instead of the `this` keyword:

```
MyClass.#privateStaticMethod();
```

The following shows the syntax of the private getters/setters:

```
class MyClass {
  #field;
  get #myField() {
      return #field;
  }
  set #myField(value){
      #field = value;
  }
}
```

In this example, the `#myField` is the private getter and setter that provides access to the private field `#field`.

In practice, you use private methods to minimize the number of methods that the object exposes.

As a rule of thumb, you should make all class methods private by default first. And then you make a method public whenever the object needs to use that method to interact with other objects.

JavaScript private method examples

Let's take some examples of using private methods

1) Private instance method example

The following illustrates how to define the `Person` class with private instance methods:

```
class Person {
  #firstName;
  #lastName;
  constructor(firstName, lastName) {
    this.#firstName = firstName;
    this.#lastName = lastName;
  }
  getFullName(format = true) {
    return format ? this.#firstLast() : this.#lastFirst();
  }

  #firstLast() {
    return `${this.#firstName} ${this.#lastName}`;
  }
  #lastFirst() {
    return `${this.#lastName}, ${this.#firstName}`;
  }
}

let person = new Person('John', 'Doe');
console.log(person.getFullName());
```

Output:

```
John Doe
```

In this example:

First, define two private fields `#firstName` and `#lastName` in the `Person` class body.

Second, define the private methods `#firstLast()` and `#lastFirst()`. These methods return the full name in different formats.

Third, define the public instance method `getFullName()` that returns a person's full name. The `getFullName()` method calls the private method `#firstLast()` and `#lastFirst()` to return the full name.

Finally, create a new `person` object and output the full name to the console.

2) Private static method example

The following adds the `#validate()` private static method to the `Person` class:

```
class Person {
  #firstName;
  #lastName;
  constructor(firstName, lastName) {
    this.#firstName = Person.#validate(firstName);
    this.#lastName = Person.#validate(lastName);
  }
  getFullName(format = true) {
    return format ? this.#firstLast() : this.#lastFirst();
  }
  static #validate(name) {
    if (typeof name === 'string') {
      let str = name.trim();
      if (str.length >= 3) {
        return str;
      }
    }
    throw 'The name must be a string with at least 3 characters';
  }

  #firstLast() {
    return `${this.#firstName} ${this.#lastName}`;
  }
  #lastFirst() {
    return `${this.#lastName}, ${this.#firstName}`;
  }
}

let person = new Person('John', 'Doe');
console.log(person.getFullName());
```

How it works.

First, define the static method `#validate()` that returns a value if it is a string with at least three characters. The method raises an exception otherwise.

Second, call the `#validate()` private static method in the constructor to validate the `firstName` and `lastName` arguments before assigning them to the corresponding private attributes.

Summary

- Prefix a method name the `#` to make it private.
- Private methods can be called inside the class, not from outside of the class or in the subclasses.