Private Members in JavaScript

Douglas Crockford
www.crockford.com

JavaScript is the world's most misunderstood programming language. Some believe that it lacks the property of *information hiding* because objects cannot have private instance variables and methods. But this is a misunderstanding. JavaScript objects can have private members. Here's how.

Objects

JavaScript is fundamentally about *objects*. Arrays are objects. Functions are objects. Objects are objects. So what are objects? Objects are collections of name-value pairs. The names are strings, and the values are strings, numbers, booleans, and objects (including arrays and functions). Objects are usually implemented as hashtables so values can be retrieved quickly.

If a value is a function, we can consider it a *method*. When a method of an object is invoked, the `this` variable is set to the object. The method can then access the instance variables through the `this` variable.

Objects can be produced by *constructors*, which are functions which initialize objects. Constructors provide the features that classes provide in other languages, including static variables and methods.

Public

The members of an object are all *public* members. Any function can access, modify, or delete those members, or add new members. There are two main ways of putting members in a new object:

In the constructor

This technique is usually used to initialize public instance variables. The constructor's `this` variable is used to add members to the object.

```
function Container(param) {
    this.member = param;
}
```

So, if we construct a new object

```
var myContainer = new Container('abc');
```

then `myContainer.member` contains `'abc'`.

In the prototype

This technique is usually used to add public methods. When a member is sought and it isn't found in the object itself, then it is taken from the object's constructor's `prototype` member. The prototype mechanism is used for inheritance. It also conserves memory. To add a method to all objects made by a constructor, add a function to the constructor's `prototype`:

```
Container.prototype.stamp = function (string) {
    return this.member + string;
}
```

So, we can invoke the method

```
myContainer.stamp('def')
```

which produces `'abcdef'`.

Private

Private members are made by the constructor. Ordinary `var`s and parameters of the constructor become the private members.

```
function Container(param) {
    this.member = param;
    var secret = 3;
    var that = this;
}
```

This constructor makes three private instance variables: `param`, `secret`, and `that`. They are attached to the object, but they are not accessible to the outside, nor are they accessible to the object's own public methods. They are accessible to private methods. Private methods are inner functions of the constructor.

```
function Container(param) {

    function dec() {
        if (secret > 0) {
            secret -= 1;
            return true;
        } else {
            return false;
        }
    }

    this.member = param;
    var secret = 3;
    var that = this;
}
```

The private method `dec` examines the `secret` instance variable. If it is greater than zero, it decrements `secret` and returns `true`. Otherwise it returns `false`. It can be used to make this object limited to three uses.

By convention, we make a private `that` variable. This is used to make the object available to the private methods. This is a workaround for an error in the ECMAScript Language Specification which causes `this` to be set incorrectly for inner functions.

Private methods cannot be called by public methods. To make private methods useful, we need to introduce a privileged method.

Privileged

A *privileged* method is able to access the private variables and methods, and is itself accessible to the public methods and the outside. It is possible to delete or replace a privileged method, but it is not possible to alter it, or to force it to give up its secrets.

Privileged methods are assigned with `this` within the constructor.

```
function Container(param) {

    function dec() {
        if (secret > 0) {
            secret -= 1;
            return true;
        } else {
            return false;
        }
    }

    this.member = param;
    var secret = 3;
    var that = this;

    this.service = function () {
        return dec() ? that.member : null;
    };
}
```

`service` is a privileged method. Calling `myContainer.service()` will return `'abc'` the first three times it is called. After that, it will return `null`. `service` calls the private `dec` method which accesses the private `secret` variable. `service` is available to other objects and methods, but it does not allow direct access to the private members.

Closures

This pattern of public, private, and privileged members is possible because JavaScript has *closures*. What this means is that an inner function always has access to the vars and parameters of its outer function, even after the outer function has returned. This is an extremely powerful property of the language. It is described in How JavaScript Works.

Private and privileged members can only be made when an object is constructed. Public members can be added at any time.

Patterns

Public

```
function Constructor(...) {

        this.membername = value;

}
Constructor.prototype.membername = value;
```

Private

```
function Constructor(...) {

    var that = this;
    var membername = value;

    function membername(...) {...}

}
```

Note: The function statement

```
    function membername(...) {...}
```

is shorthand for

```
    var membername = function membername(...) {...};
```

Privileged

```
function Constructor(...) {

    this.membername = function (…) {…};

}
```