Methods, Getters & Setters

Dan Geabunea

Passionate Software Developer

@romaniancoder | www.dangeabunea.com

Overview

Functions in the context of OOP

- Defining & invoking methods
- Getters
- Setters

Encapsulation

Object-oriented Concepts in JavaScript

Jim Cooper

OOP

Is a programming paradigm that organizes software around objects. In most cases, this is the most natural and pragmatic way of modeling the real world.

Objects

Data

Fields

Behavior

Functions

JavaScript Objects

Data

Fields

Behavior

~~Functions~~ Methods

Method

A function defined within an object or class

Methods Are Defined Inside Objects and Classes

Inside an object

```
const aircraft = {
  altitude: 2000,

  changeAltitude: function(value) {
    this.altitude += value;
    console.log(this.altitude);
  },
};

aircraft.changeAltitude(100);
```

Inside a class

```
class Aircraft {
  constructor(altitude) {
    this.altitude = altitude;
  }

  changeAltitude(value) {
    this.altitude += value;
    console.log(this.altitude);
  }
}

const a1 = new Aircraft(2000);
a1.changeAltitude(100);
```

```
const aircraftOne = new Aircraft(2000);

aircraftOne.changeAltitude(100);
```

Instance Methods

Functions that act on a particular object

Static Methods

Functions that operate on the class, not a particular instance of that class

Methods are functions inside objects or classes

Everything that applies to functions applies to methods

Creating methods

Getters & Setters

Getters & Setters

Special methods that protect the data of an object.

A getter is typically used to access the data of an object and setter is used to modify it.

Getter

```
class Passenger {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  // Getter
  get fullName() {
    return `${this.firstName} ${this.lastName}`;
  }
}

// No parenthesis after getter
const john = new Passenger('John', 'Doe');
console.log(john.fullName);
```

JavaScript Getter

Must be declared using the “get” keyword before their name

It needs an identifier / name

It must have exactly zero parameters

When you call a getter, you omit the parenthesis, use the identifier

Why Use Getters?

Computed properties

Encapsulation

Setter

```
class Passenger {
  internalName = '';

  set name(value) {
    if (!value) {
      throw new Error(‘name must have a value');
    }

    if (value.length < 1) {
      throw new Error('name has few characters');
    }

    this.internalName = value;
  }
}
```

Setter

```
class Passenger {
  internalName = ‘’;

  constructor(name) {
     // assign data to setter => modify private field
    this.name = name;
  }

  set name(value) {
    if (!value) {
      throw new Error('name must have a value');
    }

    if (value.length < 1) {
      throw new Error('name has few characters');
    }

    this.internalName = value;
  }
}
```

JavaScript Setter

Must be declared using “set” prefix

Must have exactly one parameter

Call it like you are assigning a variable, not as a function

Why Use Setters?

Validation of state

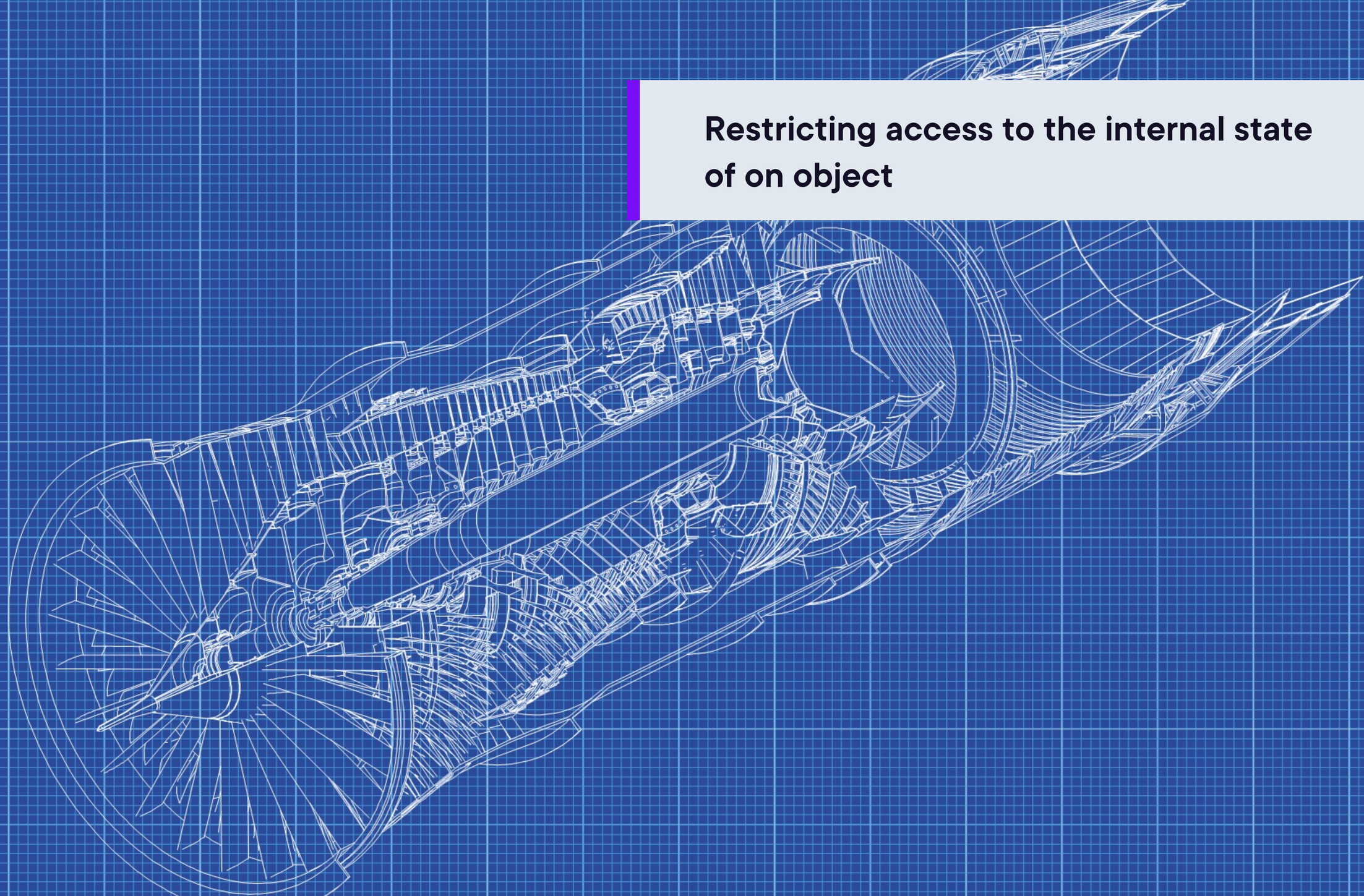
Encapsulation

Getter / Setter Pair

```
class Passenger {
  internalName = '';

  constructor(name) {
    this.name = name;
  }

  set name(value) {
    if (!value) {
      throw new Error('name must have a value');
    }
    if (value.length < 1) {
      throw new Error('name has few characters');
    }

    this.internalName = value;
  }

  get name() {
    return this.internalName;
  }
}

const john = new Passenger('John');
john.internalName = 'Dan';
```

Encapsulation

Restricting access to the internal state of on object



Weak Encapsulation

```
class Aircraft{
  speed = 0;
  altitude = 0;

  constructor(speed, altitude) {
    this.speed = speed;
    this.altitude = altitude;
  }
}

const aircraft = new Aircraft(900, 20000);

// Not safe
aircraft.speed = 0;
```

We must design classes in a way that empowers others to use them safely and correctly

Step 1: Hide Internal Data

You

“I thought JavaScript does not support access modifiers that well.”

Private Members Using

Adding a # to the name of a field or method will make it private. This feature is available starting with ECMAScript 2020

Stronger Encapsulation

```
class Aircraft{
  #speed = 0;     // private
  #altitude = 0;  // private

  constructor(speed, altitude) {
    this.#speed = speed;
    this.#altitude = altitude;
  }

}

const aircraft = new Aircraft(900, 20000);

aircraft.#speed = 0;
//Private field '#speed' must be declared in an
//enclosing class
```

Step 2: Expose Meaningful Actions

Use methods, getters and setters

Strong Encapsulation

```
class Aircraft{
  #speed = 0;     // private
  #altitude = 0;  // private

  land() {
    // Logic to change data in a safe way
  }

  get altitudeInFL() {
    return this.#altitude / 100;
  }
}

const aircraft = new Aircraft(900, 20000);

aircraft.land();
console.log(aircraft.altitudeInFL);
```

Private Methods

```
class Aircraft{
  #speed = 0;     // private
  #altitude = 0;  // private

  land() {
    this.#setLandingSpeed();
  }

  #setLandingSpeed() {
    if (this.#altitude > 10000) {
      this.#speed = 500;
    } else {
      this.#speed = 380;
    }
  }

}

const aircraft = new Aircraft(900, 20000);
aircraft.land();
```

Encapsulating JS Code

Summary

When we are in the context of a class or object, functions are called methods

Getters are a special kind of method with no parameters, which are used to access an object's data

Setters are another particular kind of method with a single parameter, which are used to mutate the state of objects safely

JavaScript has evolved and supports encapsulation using the # modifier. This makes members private to their enclosing class

Up Next:

Understanding Function Scope and Closure