Chapter 3 Javascript Multiple Values: Part 2: Objects

1 Lesson Objectives

- Understand the concept of objects in JavaScript.
- Create an object using the object literal and object constructor.
- Access and modify object properties.
- Create an array of objects.
- Nest objects and arrays.

2 Objects

Objects are a data structure that comprises a collection of **properties** and **methods**.

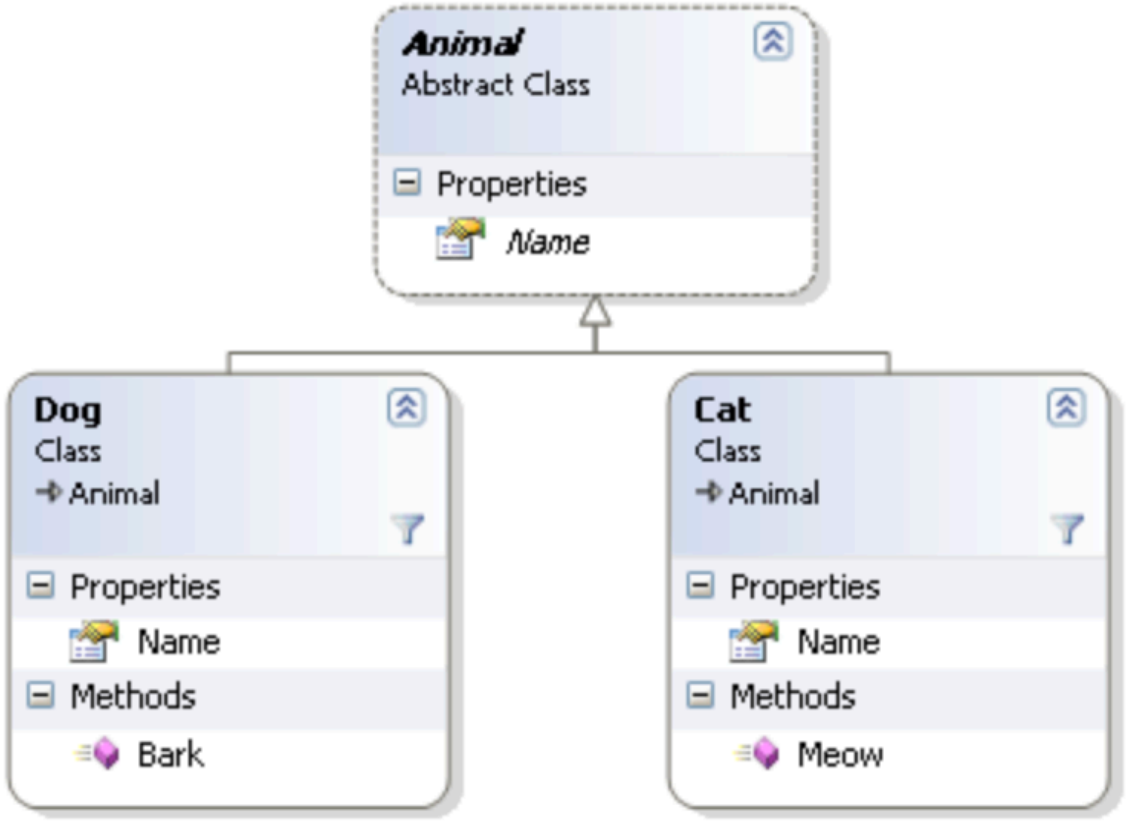
- A property is a key-value pair.
- A method is a function that can access the object's properties to perform a task.

Objects are the abstractions of real-world entities.

- Use them to model real-world entities in the program.

An object can inherit properties and methods from another object to extend its functionality.

- Be ease for developers to maintain and extend the code.



Example: The object hierarchy of Animal, Dog, and Cat.

- Dogs and Cats inherit the properties and methods from the Animal object.

See more about objects in the MDN web docs: JavaScript object basics - Learn web development | MDN

Creating an object

There are three ways to create an object in JavaScript:

- object literal,
- object constructor, and
- `Object.create()` method. (not covered in this chapter)

Object literal

Use the object literal `{}` to create an object declaratively.

- shortcoming: Can only create one object at a time.

Example: Create the car FIAT-500

The FIAT 500 in the physical world:

- Identify the properties and methods of the FIAT 500.

Fig source: [1]

Abstracting the FIAT 500 as an object:

Fig source: [1]

Create the FIAT 500 object in the JavaScript "world".

The object literal of the FIAT 500:

```
let fiat500 = {
  // properties
  maker: 'Fiat', // key: value
  model: '500',
  year: 1957,
  color: 'Blue',
  passengers: 2,
  mileage: 88000,

// methods
  drive_forward: function(distance_miles) {
    console.log('Driving forward');
    // use the "this" keyword to refer to the object itself
    this.mileage += distance_miles;
  }, // key: function() { method body } or function object

  drive_backward: function() {
    console.log('Driving backward');
  }
```

In the above object literal:

- The properties are key-value pairs separated by a colon `:`.
- The methods are key-function pairs separated by a colon `:`.
- The `this` keyword is used to refer to the object itself to access the object's properties and methods.

Object constructor: the template to create multiple objects

Object literal is suitable for creating a single object.

Limitations:

- Error-prone when creating multiple objects with the same properties and methods.
    - Repeatedly write the same properties and methods for each object.

Use the object constructor to create **multiple objects** from an object template (class).

The object constructor is a function that initializes the object's properties.

Use the `class` keyword to define an object constructor in ES6.

Example: Define the FIAT500 class in ES6.

```
class FIAT500 {
    // constructor function
    constructor(maker, model, year, color, passengers, mileage) {
        this.maker = maker; // add a property to the object
        // this['marker'] = maker;  // the same meaning as the above line
        this.model = model;
        this.year = year;
        this.color = color;
        this.passengers = passengers;
        this.mileage = mileage;
        }
    // methods: named functions
    drive_forward(distance_miles) {
        console.log('Driving forward');
        this.mileage += distance_miles;
    }
    drive_backward() {
        console.log('Driving backward');
    }
}
```

In the above code:

- The `constructor` function initializes the object's properties.
- Pass arguments to the constructor function to initialize the object's properties.
- In the constructor function, use the `this` keyword to refer to the object itself.
- Use the dot operator `.` to access the object's properties and methods.

Steps to create a class: Summary

```
class FIAT500 {
    // constructor function
    constructor(maker, model, year, color, passengers, mileage) {
        this.maker = maker; // add a property to the object
        // this['marker'] = maker;  // the same meaning as the above line
        this.model = model;
        this.year = year;
        this.color = color;
        this.passengers = passengers;
        this.mileage = mileage;
        }
    // methods: named functions
    drive_forward(distance_miles) {
        console.log('Driving forward');
        this.mileage += distance_miles;
    }
    drive_backward() {
        console.log('Driving backward');
    }
}
```

1. Define the class using the `class` keyword.
2. Define the `constructor` function to initialize the object's properties.
    - All required properties are defined in the constructor function.
3. Define the methods as named functions in the class definition.

Quick Practice

- Create a Cat class with the following properties and methods:
  - properties: name, age, color, and breed.
  - methods:
    - meow(): log "Meow!" to the console.
    - jump(): log "Jumping!" to the console.
    - info(): log the cat's name, age, color, and breed to the console.

▶ Click to see the answer

Create an object from the class

With the class definition, we can now create multiple objects of the same type using the `new` keyword.

Example 21: Create the myFiat500 and yourFIAT500 objects.

```
let myFiat = new FIAT500('Fiat', '500', 1957, 'Blue', 2, 6000);
let yourFiat = new FIAT500('Fiat', '500', 1957, 'Red', 2, 80000);
```

Note:

- There are other ways to create an object, such as the `Object.create()` method.
- We will cover these methods after discussing the `prototype` concepts in Chapter 7.
- See more about the class in Using classes - JavaScript | MDN

Quick Practice

Use the created Cat class to create a cat object named `myCat` with the following properties:

- name: "Fluffy"
- age: 3
- color: "white"
- breed: "Persian"

▶ Click to see the answer

Accessing object properties

Use the dot operator `.` or the square brackets `[]` with the property name (or key name) to access the object's properties.

Example: Log the `myFiat` object's mileage property.

```
console.log(myFiat.mileage);  // 6000
// or
console.log(myFiat['mileage']);
```

Add object's properties

JavaScript objects are dynamic.

- You can add, delete, and update properties (or even methods) of an object after the object is created.

When you specify a **new key-value pair** that does not exist in the object, JavaScript will add the new property to the object.

Example: Add the `fuel` property to the `myFiat` object.

```
myFiat.fuel = 'gasoline';   // Add a new property
console.log(myFiat.fuel);  // gasoline
```

Remove a property

To remove a property from an object, use the `delete` operator.

```
delete myFiat.fuel;  // remove the fuel property
console.log(myFiat);  // no fuel property in the object.
```

3 Working with objects and arrays

Array of objects

Dealing with an array of objects is a common task in JavaScript programming.

- Scenario
  - Query a list of **HTML element objects** with the same class name and store them in an array of objects.
  - Have a list of **File objects** when reading files from the file input element.

Scenario: Create your array of objects

Example 24: Create the `cars` array that contains 2 FIAT500 objects.

Using the object literals:

```
let cars = [{
    maker: 'Fiat',
    model: '500',
    year: 1957,
    color: 'Blue',
    passengers: 2,
    mileage: 6000
  },
  {
    maker: 'Fiat',
    model: '500',
    year: 1957,
    color: 'Red',
    passengers: 2,
    mileage: 80000
  }
```

Or, using the FIAT500 constructor:

```
let cars = [new FIAT500('Fiat', '500', 1957, 'Blue', 2, 6000),
            new FIAT500('Fiat', '500', 1957, 'Red', 2, 80000)];
```

Scenario: Handling an array of HTML element object

Example: Add a click event listener to each radio button in the HTML document below.

Show the radio button's value when the radio button is clicked.

- the value is displayed in the `<p>` element with the `display` id.

```
<fieldset>
        <legend>Select a maintenance drone:</legend>

        <div>
          <input type="radio" id="huey" name="drone" value="huey" checked />
          <label for="huey">Huey</label>
        </div>

        <div>
          <input type="radio" id="dewey" name="drone" value="dewey" />
          <label for="dewey">Dewey</label>
        </div>

        <div>
          <input type="radio" id="louie" name="drone" value="louie" />
          <label for="louie">Louie</label>
        </div>
      </fieldset>
    <div>
       Your selection: <p id="display"></p>
    </div>
```

Select a maintenance drone:

- Huey
- Dewey
- Louie

Your selection:

huey

To do that, we need to add a click event listener to each radio button.

- First, we get all the radio button elements in an array.
- Then, we iterate the array and add a click event listener to each radio button.
    - The listener function gets the radio button's value and shows it in the `<p>` element.

Here is the code:

```
let drones = document.getElementsByName('drone');

// NodeList(3) [input#huey, input#dewey, input#louie], an array of input elements
console.log(drones);

// iterate the array
drones.forEach( drone => {
    // add a click event listener to each radio button
    drone.addEventListener('click', function(e){
        // get the radio button's value
        let value = e.target.value;
        // show the value in the <p> element
        document.getElementById('display').textContent = value;
    });
})
```

See full code in the ex_03_array_of_objects.html file.

Note to this demo:

- This is not the best way to handle the radio button selection.
- Adding the click event listener to the parent element of the radio buttons is a more concise way.
    - Because the event can bubble up from the radio button to the parent element.

Object having an array property

You can use an array as a property of an object.

Example: Create the `myFiat` object with the `gear` property as an array of values: 1, 2, 3, 4, 5, and R

```
let myFiat = {
    maker: 'Fiat',
    model: '500',
    year: 1957,
    color: 'Blue',
    passengers: 2,
    mileage: 6000,
    gear: [1, 2, 3, 4, 5, 'R']
}
```

To log the first gear value of the `myFiat` object:

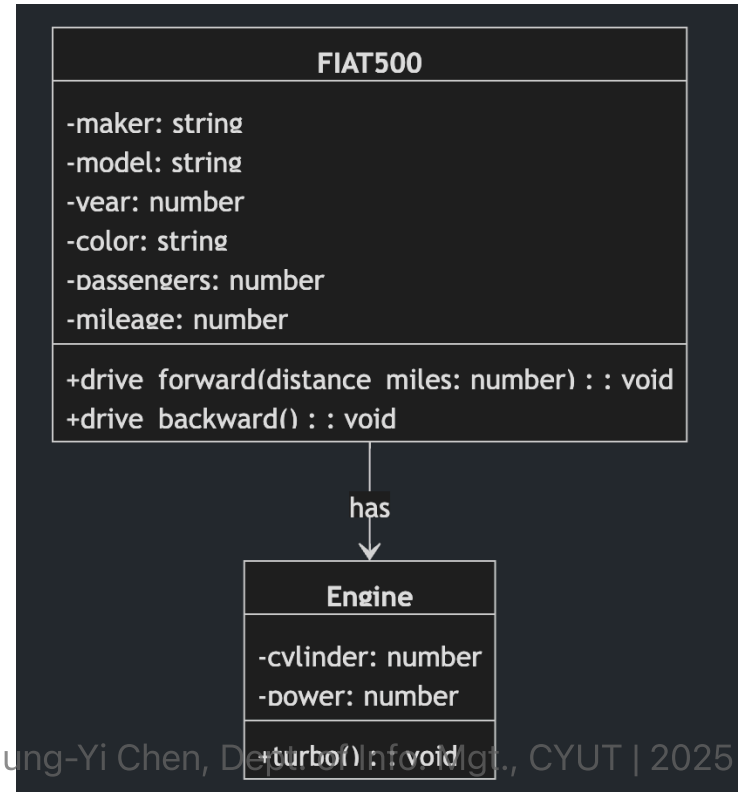
```
console.log(myFiat.gear[0]);  // 1
```

Nested objects

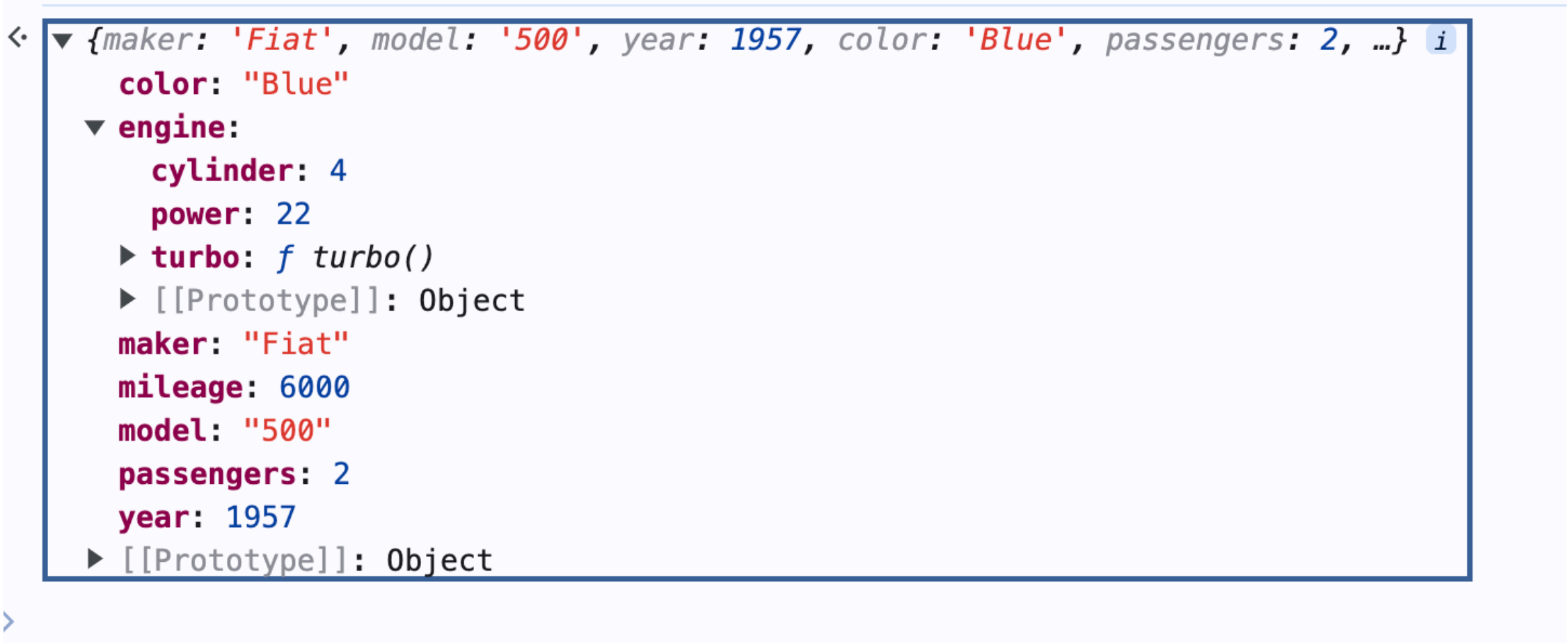
An object can have another object as a property.

Example 27: Create the `myFiat` object with the `engine` property as an object. The `engine` object has the `cylinder` and `power` properties.

```
let myFiat = {
    maker: 'Fiat',
    model: '500',
    year: 1957,
    color: 'Blue',
    passengers: 2,
    mileage: 6000,
    engine: {
        cylinder: 4,
        power: 22,
        // engine's method
        turbo(){
            console.log('Turbo is on' + this.power * 1.2);
        }
    }
}
```



Constructed object:



4 Summary

We have learned:

- Ways to create an object: object literal and object constructor.
- Accessing and modifying object properties.
- `this` keyword to refer to the object itself.
- Working with arrays and objects: array of objects, object having an array property, and nested objects.

5 References

[1] Eric T. Freeman and Elisabeth Robson, 2014. Head First JavaScript Programming: A Brain-Friendly Guide, O'Reilly Media