JavaScript Classes and Objects

A Complete Guide

Master Object-Oriented Programming in JavaScript

With Detailed Explanations and Multiple Examples

Table of Contents

1. Introduction to JavaScript Data Types

JavaScript is a dynamically-typed language, which means you don't need to specify data types when declaring variables. Understanding the different types of data JavaScript works with is crucial for mastering classes and objects.

1.1 Primitive Types

Type	Description	Example
number	Numeric values	42, 3.14, -7
string	Text data	"Hello", 'World'
boolean	True or false	true, false
undefined	Variable not assigned	let x;
null	Absence of value	let y = null;

```
// Number examples
let age = 25;
let price = 19.99;

// String examples
let name = "Alice";
let greeting = 'Hello!';

// Boolean examples
let isActive = true;

console.log(typeof age);    // "number"
console.log(typeof name);   // "string"
```

1.2 Complex Types

```
// Object literal
let person = {
    name: "John",
    age: 30,
    city: "New York"
};

// Accessing properties
console.log(person.name);     // "John"
console.log(person["age"]);   // 30

// Arrays
let fruits = ["apple", "banana", "orange"];
console.log(fruits[0]);       // "apple"
console.log(fruits.length);   // 3
```

2. Understanding Objects in Depth

Objects are collections of key-value pairs. They are the foundation of JavaScript's object-oriented capabilities.

2.1 Creating Objects

```
// Object with properties and methods
let car = {
    brand: "Toyota",
    model: "Camry",
    year: 2022,

    startEngine: function() {
        console.log("Engine started!");
    },

    getInfo() {
        return this.year + " " + this.brand + " " + this.model;
    }
};

car.startEngine();            // "Engine started!"
console.log(car.getInfo());  // "2022 Toyota Camry"
```

2.2 Object Methods

```
let calculator = {
    value: 0,

    add(num) {
        this.value += num;
        return this;
    },

    subtract(num) {
        this.value -= num;
        return this;
    },

    getValue() {
        return this.value;
    }
};

// Method chaining
let result = calculator.add(10).subtract(3).getValue();
console.log(result);  // 7
```

3. Introduction to Classes

Classes were introduced in ES6 as a cleaner syntax for creating objects and implementing inheritance. A class is a blueprint for creating objects.

3.1 Basic Class Syntax

```
class Rectangle {
    constructor(width, height) {
        this.width = width;
        this.height = height;
    }

    area() {
        return this.width * this.height;
    }

    perimeter() {
        return 2 * (this.width + this.height);
    }
}

// Creating instances
let rect1 = new Rectangle(5, 10);
let rect2 = new Rectangle(3, 7);

console.log(rect1.area());       // 50
console.log(rect2.perimeter());  // 20
```

3.2 Real-World Example: Person Class

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

    getFullName() {
        return this.firstName + " " + this.lastName;
    }

    introduce() {
        return "Hi, I'm " + this.getFullName() +
               " and I'm " + this.age + " years old.";
    }

    isAdult() {
        return this.age >= 18;
    }
}

let person1 = new Person("Alice", "Johnson", 25);
console.log(person1.introduce());
// "Hi, I'm Alice Johnson and I'm 25 years old."

console.log(person1.isAdult());  // true
```

4. Class Constructors in Detail

The constructor is a special method that is automatically called when creating a new instance. It's used to initialize the object's properties.

4.1 Constructor with Default Parameters

```
class Product {
    constructor(name, price = 0, inStock = true) {
        this.name = name;
        this.price = price;
        this.inStock = inStock;
    }

    getInfo() {
        let status = this.inStock ? "In Stock" : "Out of Stock";
        return this.name + " - $" + this.price + " (" + status + ")";
    }
}

let product1 = new Product("Laptop", 999, true);
let product2 = new Product("Mystery Item");  // Uses defaults

console.log(product2.getInfo());
// "Mystery Item - $0 (In Stock)"
```

4.2 Constructor with Validation

```
class User {
    constructor(username, email, age) {
        if (!username || username.length < 3) {
            throw new Error("Username must be at least 3 characters");
        }

        if (!email.includes("@")) {
            throw new Error("Invalid email address");
        }

        if (age < 0 || age > 120) {
            throw new Error("Age must be between 0 and 120");
        }

        this.username = username;
        this.email = email;
        this.age = age;
    }
}

// Valid user
let user1 = new User("john_doe", "john@example.com", 25);

// Invalid - throws error
try {
    let user2 = new User("ab", "invalid", 25);
} catch (error) {
    console.log(error.message);
}
```

5. Class Methods in Detail

Methods are functions defined inside a class. They describe behaviors that objects can perform.

```
class BankAccount {
    constructor(accountNumber, initialBalance = 0) {
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
        this.transactions = [];
    }

    deposit(amount) {
        if (amount > 0) {
            this.balance += amount;
            this.transactions.push({
                type: 'deposit',
                amount: amount,
                date: new Date()
            });
            return "Deposited $" + amount +
                   ". New balance: $" + this.balance;
        }
        return "Invalid amount";
    }

    withdraw(amount) {
        if (amount > 0 && amount <= this.balance) {
            this.balance -= amount;
            this.transactions.push({
                type: 'withdrawal',
                amount: amount,
                date: new Date()
            });
            return "Withdrew $" + amount +
                   ". New balance: $" + this.balance;
        }
        return "Invalid amount or insufficient funds";
    }

    getBalance() {
        return "Current balance: $" + this.balance;
    }
}

let account = new BankAccount("12345", 1000);
console.log(account.deposit(500));
// "Deposited $500. New balance: $1500"
console.log(account.withdraw(200));
// "Withdrew $200. New balance: $1300"
```

6. Understanding the 'this' Keyword

The 'this' keyword refers to the current instance of the class. It's used to access properties and methods of that instance.

```
class Dog {
    constructor(name, breed) {
        this.name = name;
        this.breed = breed;
        this.tricks = [];
    }

    bark() {
        return this.name + " says Woof!";
    }

    learnTrick(trick) {
        this.tricks.push(trick);
        return this.name + " learned " + trick + "!";
    }

    showTricks() {
        if (this.tricks.length === 0) {
            return this.name + " doesn't know any tricks yet.";
        }
        return this.name + " knows: " + this.tricks.join(", ");
    }
}

let dog1 = new Dog("Max", "Golden Retriever");
let dog2 = new Dog("Bella", "Poodle");

console.log(dog1.bark());        // "Max says Woof!"
console.log(dog2.bark());        // "Bella says Woof!"

dog1.learnTrick("sit");
dog1.learnTrick("stay");
dog2.learnTrick("roll over");

console.log(dog1.showTricks());  // "Max knows: sit, stay"
console.log(dog2.showTricks());  // "Bella knows: roll over"
```

7. Inheritance in JavaScript

Inheritance allows a class to inherit properties and methods from another class. This promotes code reuse and establishes relationships between classes.

7.1 Basic Inheritance with 'extends'

```
// Parent class
class Animal {
    constructor(name, age) {
        this.name = name;
        this.age = age;
    }

    eat() {
        return this.name + " is eating";
    }

    sleep() {
        return this.name + " is sleeping";
    }
}

// Child class
class Dog extends Animal {
    constructor(name, age, breed) {
        super(name, age);  // Call parent constructor
        this.breed = breed;
    }

    // Override parent method
    makeSound() {
        return this.name + " barks: Woof!";
    }

    // New method
    fetch() {
        return this.name + " is fetching the ball";
    }
}

let dog = new Dog("Max", 3, "Golden Retriever");

console.log(dog.eat());        // "Max is eating" (inherited)
console.log(dog.makeSound());  // "Max barks: Woof!" (own)
console.log(dog.fetch());      // "Max is fetching the ball" (own)
```

7.2 The 'super' Keyword

```
class Vehicle {
    constructor(make, model, year) {
        this.make = make;
        this.model = model;
        this.year = year;
        this.mileage = 0;
    }

    drive(miles) {
        this.mileage += miles;
        return "Drove " + miles + " miles. Total: " + this.mileage;
    }
}

class ElectricCar extends Vehicle {
    constructor(make, model, year, batteryCapacity) {
        super(make, model, year);  // Call parent constructor
        this.batteryCapacity = batteryCapacity;
        this.batteryLevel = 100;
    }
    // Override and enhance
    drive(miles) {
        let result = super.drive(miles);  // Call parent method
        this.batteryLevel -= miles * 0.3;
        return result + ". Battery: " + this.batteryLevel + "%";
    }

    charge() {
        this.batteryLevel = 100;
        return "Battery fully charged!";
    }
}

let tesla = new ElectricCar("Tesla", "Model 3", 2023, 75);
console.log(tesla.drive(100));
// "Drove 100 miles. Total: 100. Battery: 70%"
```

8. Advanced Class Features

8.1 Private Fields

```
class BankAccount {
    #balance;  // Private field
    #pin;      // Private field

    constructor(initialBalance, pin) {
        this.#balance = initialBalance;
        this.#pin = pin;
    }

    #verifyPin(pin) {  // Private method
        return this.#pin === pin;
    }

    deposit(amount, pin) {
        if (!this.#verifyPin(pin)) {
            return "Invalid PIN";
        }
        this.#balance += amount;
        return "Deposited $" + amount;
    }

    getBalance(pin) {
        if (this.#verifyPin(pin)) {
            return this.#balance;
        }
        return "Invalid PIN";
    }
}

let account = new BankAccount(1000, "1234");
console.log(account.deposit(500, "1234"));
// "Deposited $500"

// Cannot access private fields
// account.#balance  // SyntaxError
```

9. Static Methods and Properties

Static methods and properties belong to the class itself, not to instances.

```
class MathUtils {
    static PI = 3.14159;  // Static property

    static add(a, b) {    // Static method
        return a + b;
    }

    static multiply(a, b) {
        return a * b;
    }
}

// Call on class, not instance
console.log(MathUtils.add(5, 3));       // 8
console.log(MathUtils.multiply(4, 7));  // 28
console.log(MathUtils.PI);              // 3.14159

// Factory method pattern
class User {
    constructor(username, role) {
        this.username = username;
        this.role = role;
    }

    static createAdmin(username) {
        return new User(username, 'admin');
    }

    static createRegular(username) {
        return new User(username, 'user');
    }
}

let admin = User.createAdmin("admin1");
let user = User.createRegular("john");
```

10. Getters and Setters

Getters and setters provide controlled access to properties with validation.

```
class Temperature {
    constructor(celsius = 0) {
        this._celsius = celsius;
    }

    // Getter - accessed like a property
    get celsius() {
        return this._celsius;
    }

    // Setter - set like a property
    set celsius(value) {
        this._celsius = value;
    }

    // Computed property
    get fahrenheit() {
        return (this._celsius * 9/5) + 32;
    }

    set fahrenheit(value) {
        this._celsius = (value - 32) * 5/9;
    }

    get kelvin() {
        return this._celsius + 273.15;
    }
}

let temp = new Temperature(25);
console.log(temp.celsius);     // 25
console.log(temp.fahrenheit);  // 77
console.log(temp.kelvin);      // 298.15

// Set fahrenheit, updates celsius
temp.fahrenheit = 32;
console.log(temp.celsius);     // 0
```

11. Built-in JavaScript Classes

11.1 Date Class

```
// Creating dates
let now = new Date();
let specificDate = new Date('2024-12-25');

console.log(now.toISOString());
console.log(now.getFullYear());  // 2024
console.log(now.getMonth());     // 0-11
console.log(now.getDate());      // Day of month

// Custom Date class
class CustomDate extends Date {
    addDays(days) {
        this.setDate(this.getDate() + days);
        return this;
    }

    format() {
        return this.toLocaleDateString();
    }
}

let myDate = new CustomDate('2024-01-15');
myDate.addDays(10);
console.log(myDate.format());  // "1/25/2024"
```

11.2 Map Class

```
// Creating a Map
let userMap = new Map();

userMap.set('john', { age: 30, city: 'NY' });
userMap.set('jane', { age: 25, city: 'LA' });

console.log(userMap.get('john'));  // { age: 30, city: 'NY' }
console.log(userMap.has('jane'));  // true
console.log(userMap.size);         // 2

// Iterate
for (let [key, value] of userMap) {
    console.log(key + ": " + value.age);
}
```

12. Best Practices

Principle	Description
Single Responsibility	One class, one purpose
Encapsulation	Hide internal details
Clear Naming	Use descriptive names
Validation	Check inputs early
Documentation	Comment complex logic

Summary

This guide covered JavaScript classes and objects comprehensively:

• Understanding primitive and complex data types • Creating and working with objects • Defining classes with constructors and methods • Using 'this' to reference instances • Implementing inheritance with extends and super • Advanced features: private fields, static methods, getters/setters • Working with built-in classes like Date, Map, Set • Following best practices for clean code

Classes enable organized, reusable code. Practice these concepts with real-world examples to build robust applications.