In JavaScript, **multiple inheritance** (inheriting from more than one class) is not supported *directly* like in some languages (e.g., C++). JavaScript uses **prototypal inheritance** and supports **single inheritance**, but you can simulate multiple inheritance using **mixins**, **Object.assign()**, or **composition patterns**.

Ways to Implement Multiple Inheritance in JavaScript

1. Using Mixins

Mixins are a way to add properties and methods from multiple objects into a class.

```
let sayHi = {
  greet() {
    console.log("Hello!");
  }
};

let sayBye = {
  farewell() {
    console.log("Goodbye!");
  }
};

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

Object.assign(Person.prototype, sayHi, sayBye);

const user = new Person("Kunj");
user.greet();   // Hello!
user.farewell(); // Goodbye!
```

2. Using Function Composition

```
const canEat = (state) => ({
  eat: () => console.log(`${state.name} is eating`)
});

const canWalk = (state) => ({
  walk: () => console.log(`${state.name} is walking`)
});

const Person = (name) => {
  let state = { name };
  return Object.assign(state, canEat(state), canWalk(state));
};

const user = Person("Kunj");
user.eat();  // Kunj is eating
user.walk(); // Kunj is walking
```

3. Using Classes with Composition and Mixins

You can create reusable behavior via classes and then merge them.

```
class Animal {
  sleep() {
    console.log("Sleeping...");
  }
}

let flyer = {
  fly() {
    console.log("Flying...");
  }
};

let swimmer = {
  swim() {
    console.log("Swimming...");
  }
};

class Bird extends Animal {
  constructor() {
    super();
    Object.assign(this, flyer, swimmer); // Merging other behaviors
  }
}

const penguin = new Bird();
penguin.sleep(); // Sleeping...
penguin.fly();   // Flying...
penguin.swim();  // Swimming...
```

4. Using Utility Function for Mixin Inheritance

```
function mix(...mixins) {
  class Mix {}

  for (let mixin of mixins) {
    Object.assign(Mix.prototype, mixin.prototype || mixin);
  }

  return Mix;
}

class A {
  methodA() {
    console.log("From A");
  }
}

class B {
  methodB() {
    console.log("From B");
  }
}

class C extends mix(A, B) {}

const obj = new C();
obj.methodA(); // From A
obj.methodB(); // From B
```

Important Notes

- JavaScript **does not support** multiple inheritance through classes (i.e., `class C extends A, B` is invalid).
- Use **composition over inheritance** where possible — it's more flexible and clean.
- **Mixins and Object.assign()** are most commonly used for simulating multiple inheritance.