Lecture 5: OOP and JavaScript Classes

What is OOP?

- **Object-Oriented Programming (OOP) –** is a programming paradigm centered around the concept of objects.
- **Objects** can contain both data (properties) and functions (methods)
- **OOP** is based on four key concepts:

Encapsulation	**Inheritance**
Abstraction	**Polymorphism**

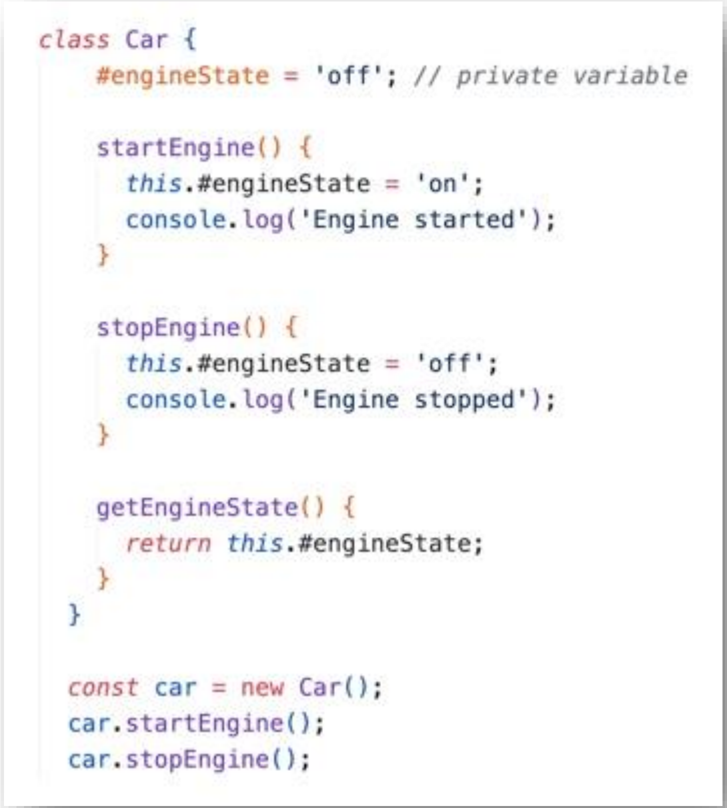
Encapsulation

Encapsulation is the process of wrapping data and methods that operate on that data within a single unit or class.

It hides the internal state of the object from the outside world and restricts direct access.

In JavaScript, we achieve this by defining private variables and providing public methods to interact with them.

Encapsulation



```
class Car {
  #engineState = 'off'; // private variable

  startEngine() {
    this.#engineState = 'on';
    console.log('Engine started');
  }

  stopEngine() {
    this.#engineState = 'off';
    console.log('Engine stopped');
  }

  getEngineState() {
    return this.#engineState;
  }
}

const car = new Car();
car.startEngine();
car.stopEngine();
```

Abstraction

Abstraction is the concept of hiding complex implementation details and showing only the necessary features of an object

It allows focusing on **what** an object does instead of **how** it does it

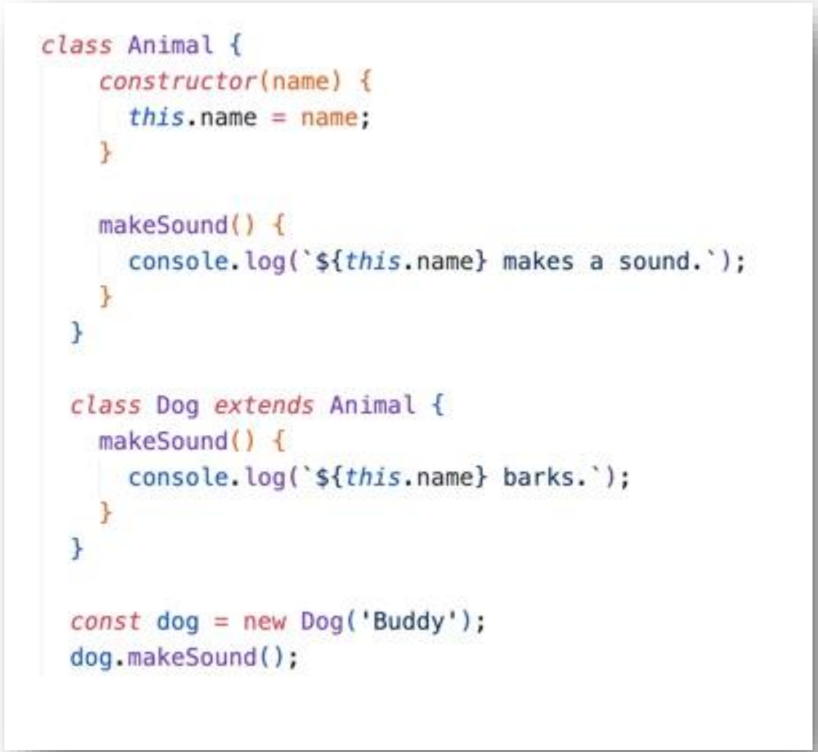
Ex: When using a car, we only need to know how to drive it (interface), without needing to understand how the engine works.

Inheritance

Inheritance allows one class to inherit properties and methods from another class, promoting code reusability

In JavaScript, a class can inherit form another class using the **extends** keyword.

Inheritance



```
class Animal {
    constructor(name) {
      this.name = name;
    }

    makeSound() {
      console.log(`${this.name} makes a sound.`);
    }
  }

  class Dog extends Animal {
    makeSound() {
      console.log(`${this.name} barks.`);
    }
  }

  const dog = new Dog('Buddy');
  dog.makeSound();
```

Polymorphism

Polymorphism allows objects of different classes to be treated as objects of a common base class.

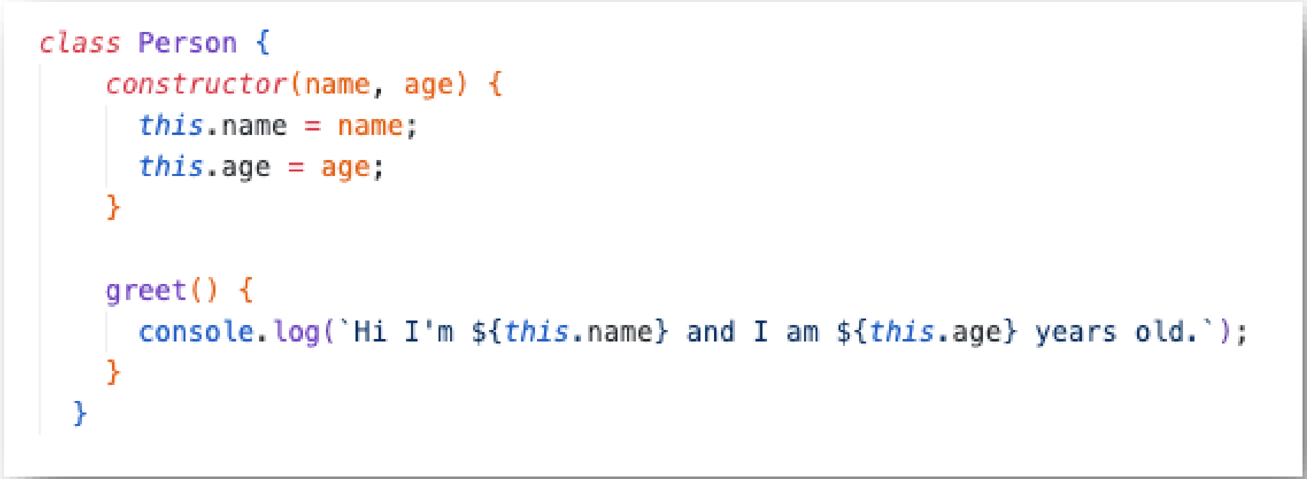
It also allows a method to behave differently based on which object it is invoked on.

Ex: Both **dog** and **cat** inherit from **Animal,** but they have their own implementations of **makeSound()**

Defining a Class

- **Constructor** is a special method used for initializing new objects
- **Methods** are functions inside a class
- **Properties** are data inside a class and they are specific for each instance (object) of the class

Defining a Class

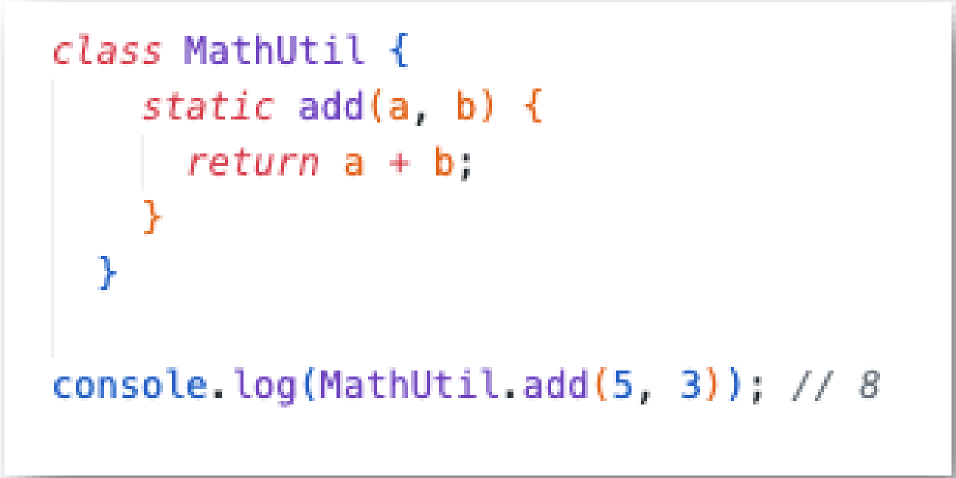


```javascript
class Person {
    constructor(name, age) {
      this.name = name;
      this.age = age;
    }

    greet() {
      console.log(`Hi I'm ${this.name} and I am ${this.age} years old.`);
    }
  }
```

Static Methods

Static methods are defined on the class itself, not on instances of the class. They can be accessed directly via the class.



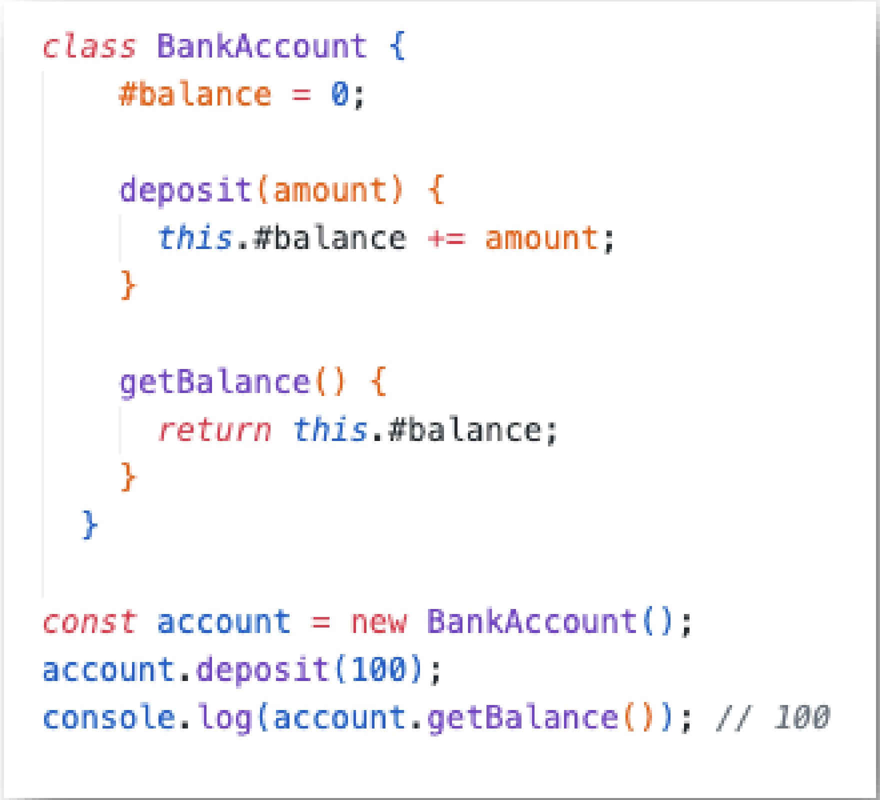
```
class MathUtil {
    static add(a, b) {
      return a + b;
    }
  }

console.log(MathUtil.add(5, 3)); // 8
```

Private Fields

- Introduced in ES2022, **private fields** are declared with **#** and are only accessible inside the class
- They enforce **encapsulation** by preventing external access.

Private Fields



```
class BankAccount {
    #balance = 0;

    deposit(amount) {
      this.#balance += amount;
    }

    getBalance() {
      return this.#balance;
    }
  }

const account = new BankAccount();
account.deposit(100);
console.log(account.getBalance()); // 100
```