JAVASCRIPT

OOP #1

PARADIGM

- Paradigm is an example or model of something. That is, a certain pattern that must be followed. In this case, for creating computer programs.
- But besides the object-oriented paradigm, there are other varieties: functional programming, reactive programming, etc.

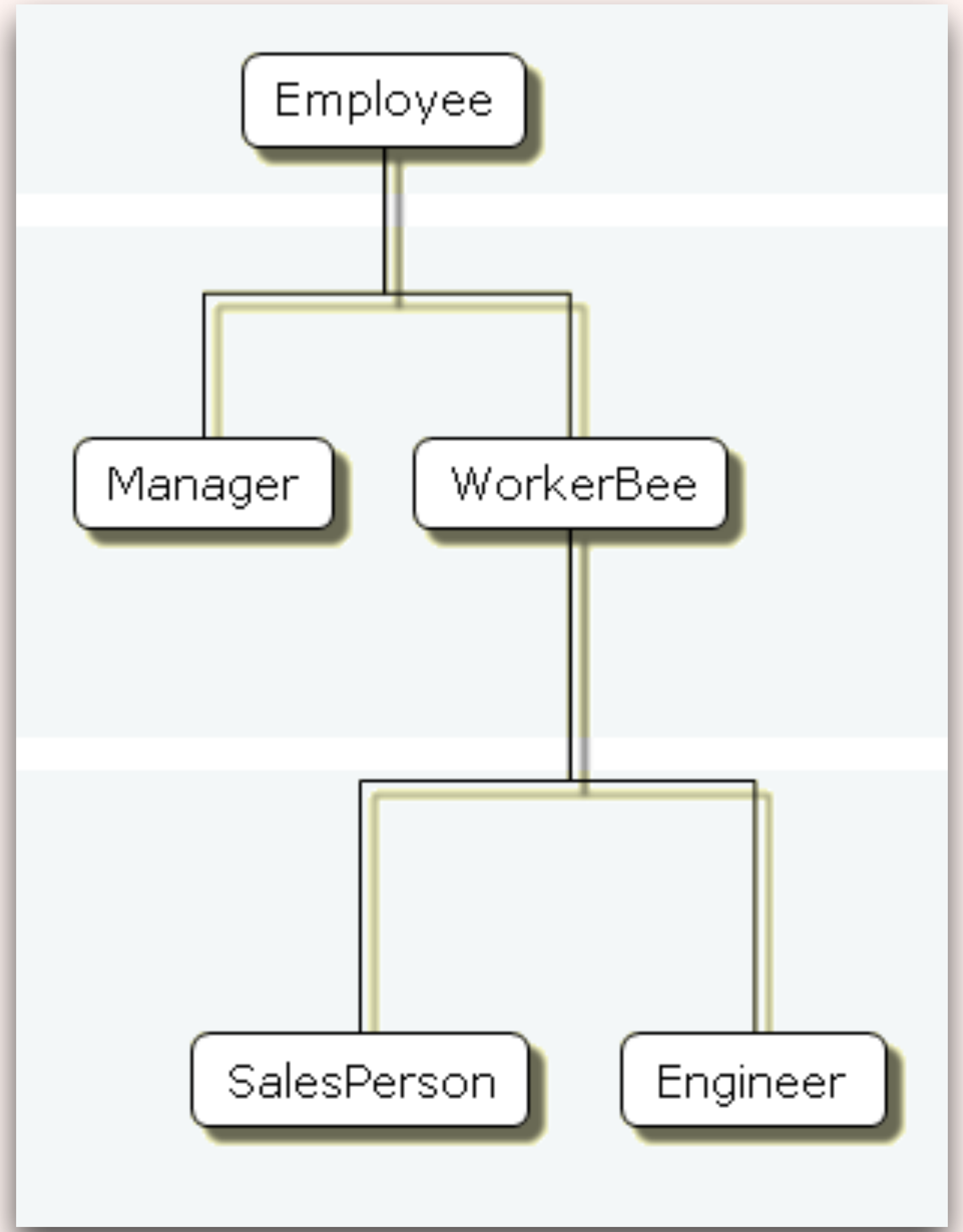
OOP

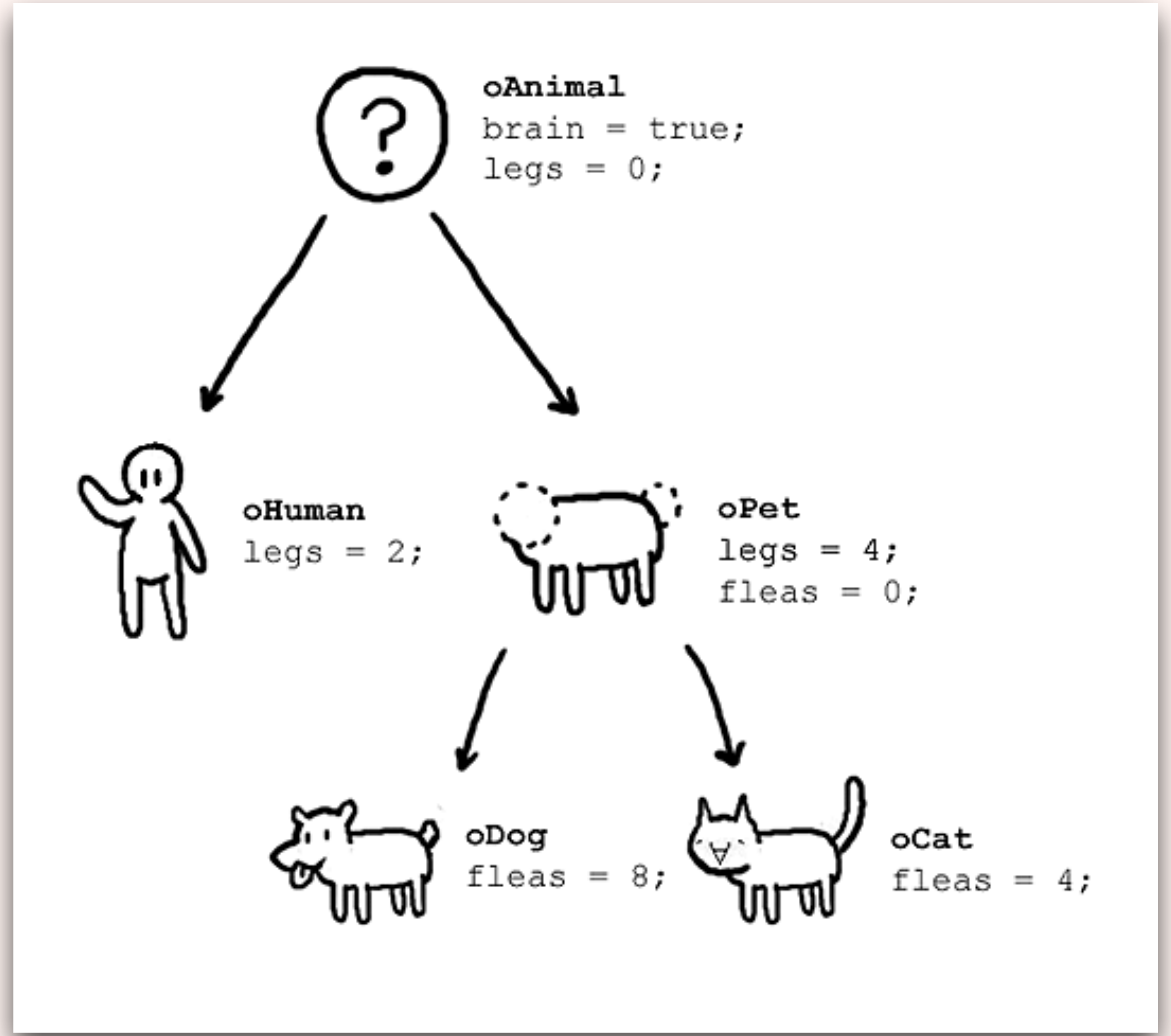
- The main problem with JavaScript is that it’s not the most object-oriented language. **Why?**
- **OOP -** programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods).
- **this** - JavaScript keyword. When used in a constructor, it refers to an instance created with the constructor. That is, the this keyword in the constructor points to its instance.

TERMINOLOGY

- **Class** - defines the characteristics of the object. A class is a description of a template for the properties and methods of an object.
- **Object** - instance of the class.
- **Property** - characteristic of an object, such as color.
- **Method** - capabilities of the object, such as walking. These are subroutines or functions associated with a class.

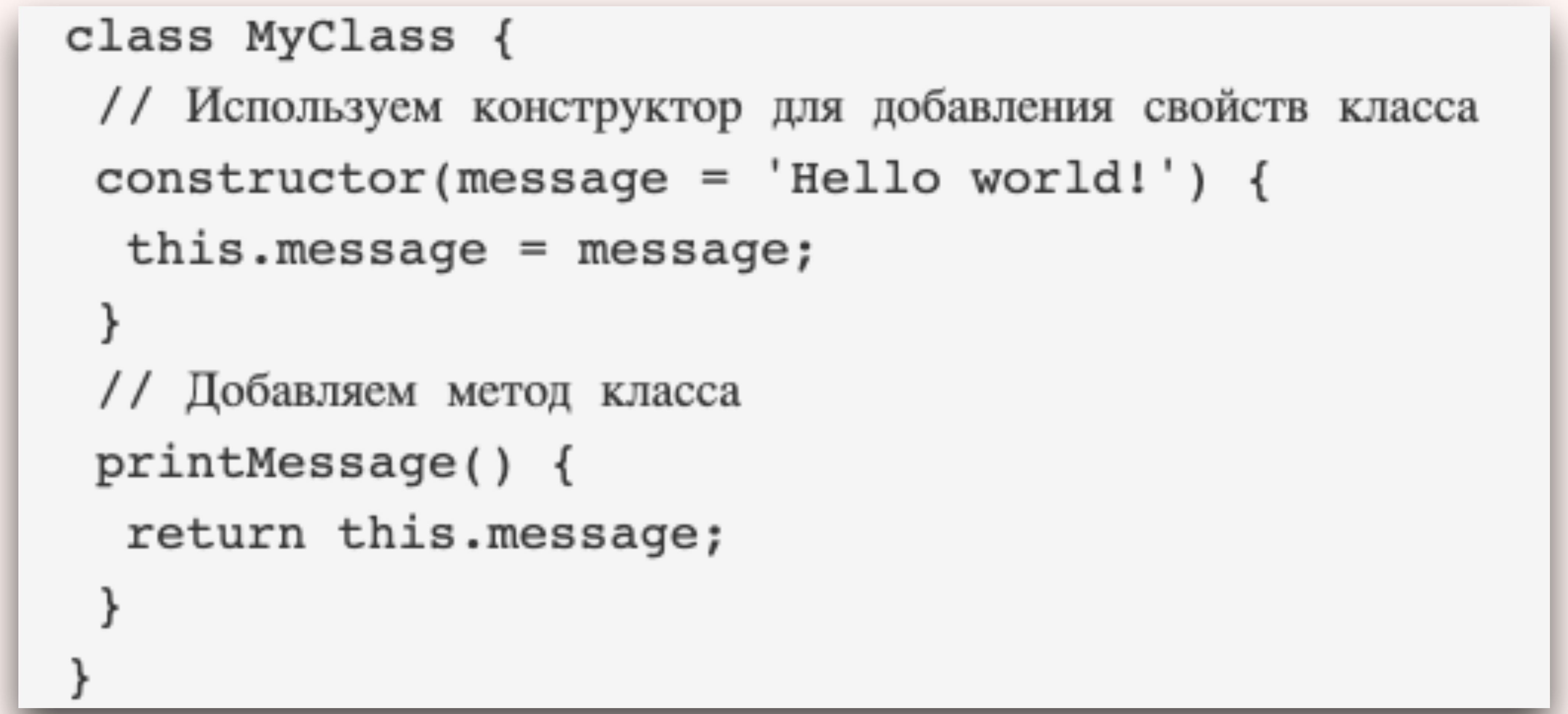
EXAMPLES


Employee
Manager
WorkerBee
SalesPerson
Engineer


oAnimal
brain = true;
legs = 0;
oHuman
legs = 2;
oPet
legs = 4;
fleas = 0;
oDog
fleas = 8;
oCat
fleas = 4;

CONSTRUCTOR

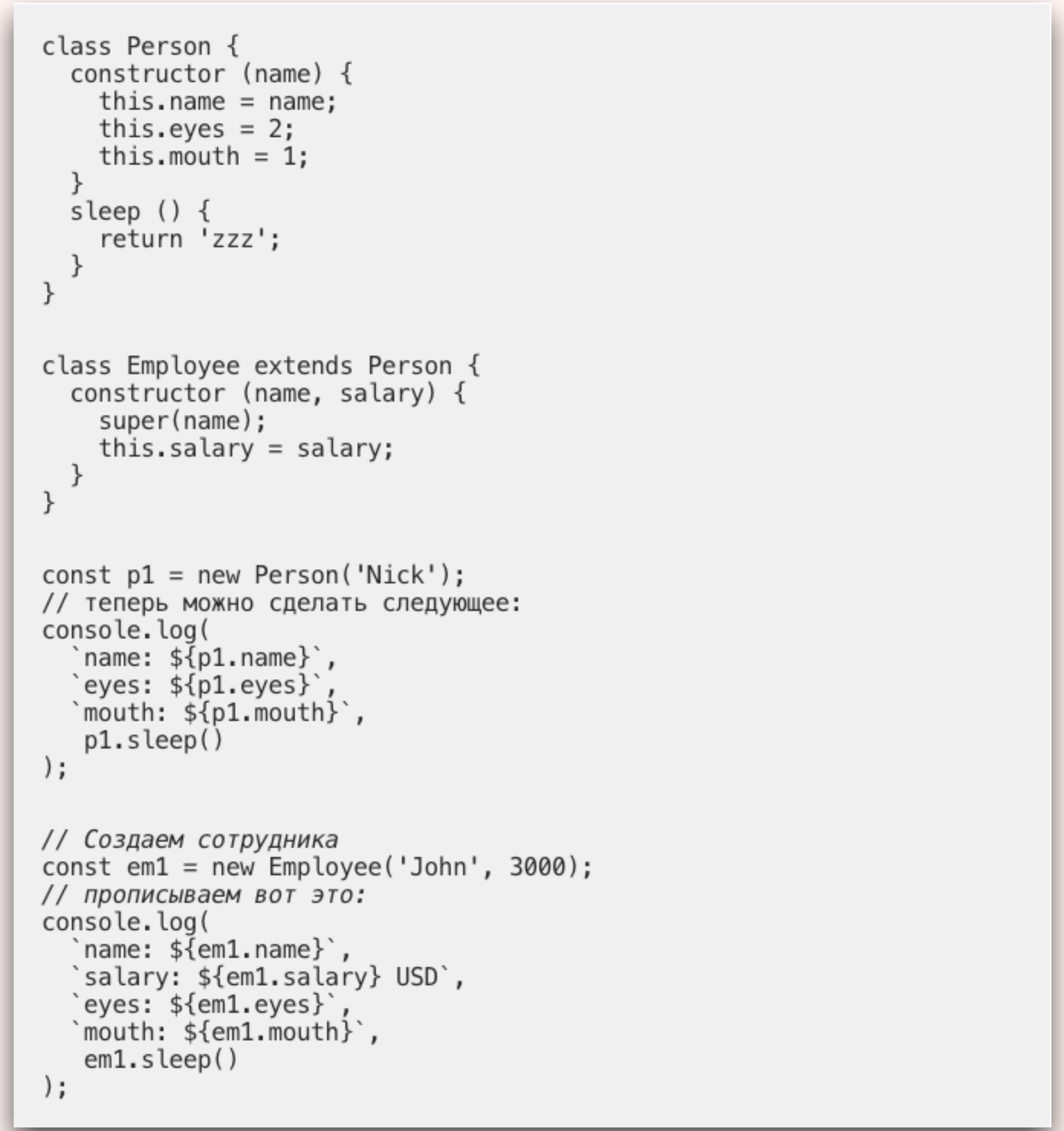
- Method of the class.
- Creates and initializes an object created with the class.
- Used to set properties of an object or to call methods that prepare an object for use.



```
class MyClass {
 // Используем конструктор для добавления свойств класса
 constructor(message = 'Hello world!') {
  this.message = message;
 }
 // Добавляем метод класса
 printMessage() {
  return this.message;
 }
}
```

INHERITANCE

- Ability to create a class as a specialized version of one or more classes.
- The main benefit of this is that you can add functionality without changing the original class
- Good for small and simple applications that rarely change and have no more than one inheritance level.



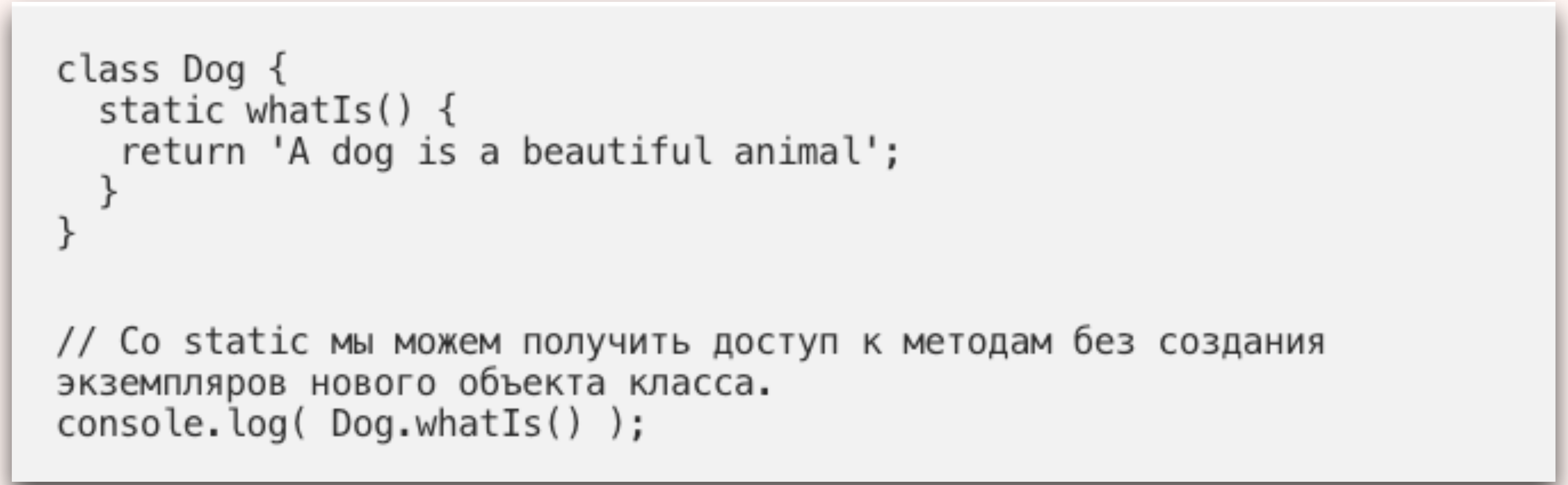
```
class Person {
  constructor (name) {
    this.name = name;
    this.eyes = 2;
    this.mouth = 1;
  }
  sleep () {
    return 'zzz';
  }
}

class Employee extends Person {
  constructor (name, salary) {
    super(name);
    this.salary = salary;
  }
}

const p1 = new Person('Nick');
// теперь можно сделать следующее:
console.log(
  `name: ${p1.name}`,
  `eyes: ${p1.eyes}`,
  `mouth: ${p1.mouth}`,
   p1.sleep()
);

// Создаем сотрудника
const em1 = new Employee('John', 3000);
// прописываем вот это:
console.log(
  `name: ${em1.name}`,
  `salary: ${em1.salary} USD`,
  `eyes: ${em1.eyes}`,
  `mouth: ${em1.mouth}`,
   em1.sleep()
);
```

ONCE YOU HAVE ENOUGH CLIENTS USING NEW, YOU WON'T EVEN BE ABLE TO CHANGE THE CONSTRUCTOR IMPLEMENTATION IF YOU WANT TO, AND IF YOU TRY, YOU'LL BREAK ALL THE CODE YOU DON'T OWN.

Eric Elliott

STATIC PROPERTIES AND METHODS

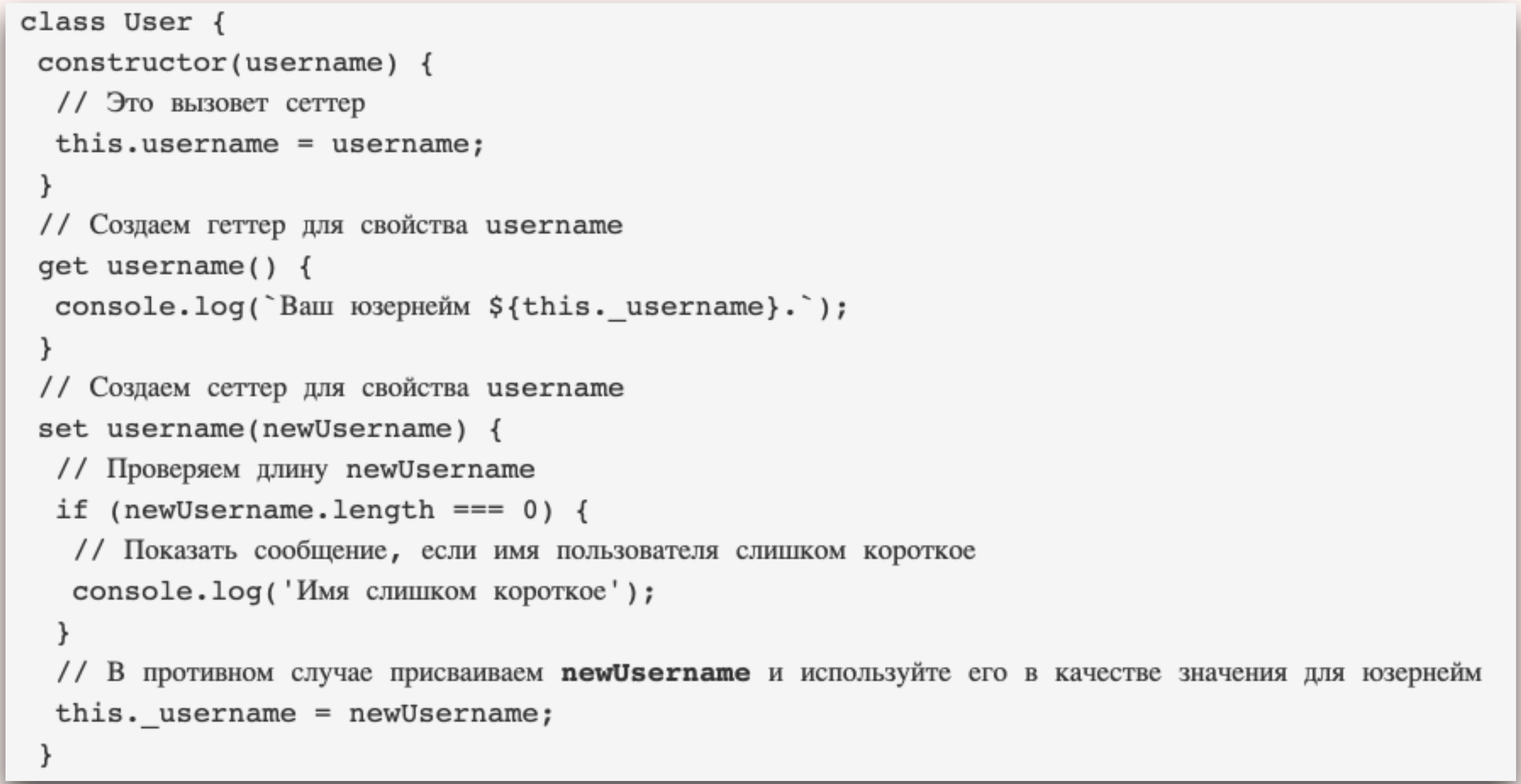
- Can be called on a class without creating a new instance.
- The syntax for declaring a property or method - **static** before the name of the property or method.



```
class Dog {
  static whatIs() {
   return 'A dog is a beautiful animal';
  }
}


// Co static мы можем получить доступ к методам без создания
экземпляров нового объекта класса.
console.log( Dog.whatIs() );
```

GETTER & SETTER

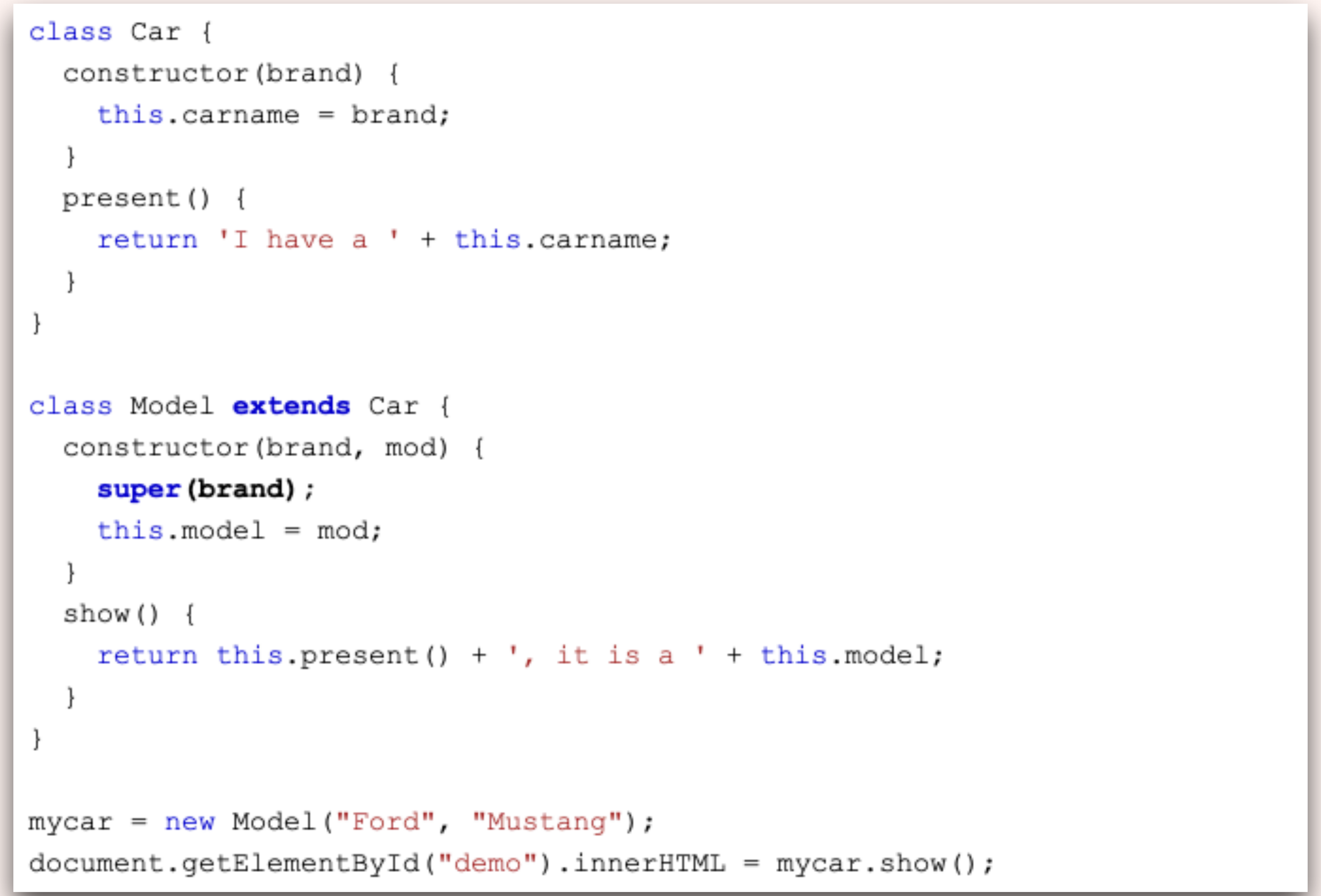
- To add **getters** and **setters** in the class, use the **get** and **set** keywords.
- Getter - methods that you use when you want to access and / or return the value of a property.
- You don't call them explicitly. All you have to do is just define them. JavaScript does the rest of the work for you.
- **setter** and **getter** method must have the same name as the property you want them to handle.



```
class User {
 constructor(username) {
  // Это вызовет сеттер
  this.username = username;
 }
 // Создаем геттер для свойства username
 get username() {
  console.log(`Ваш юзернейм ${this._username}.`);
 }
 // Создаем сеттер для свойства username
 set username(newUsername) {
  // Проверяем длину newUsername
  if (newUsername.length === 0) {
   // Показать сообщение, если имя пользователя слишком короткое
   console.log('Имя слишком короткое');
  }
  // В противном случае присваиваем newUsername и используйте его в качестве значения для юзернейм
  this._username = newUsername;
 }
```

SUPER

- By calling the **super()** method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods
- Syntax: super(…) or super.parentMethod(…) where … - arguments.



```
class Car {
  constructor(brand) {
    this.carname = brand;
  }
  present() {
    return 'I have a ' + this.carname;
  }
}

class Model extends Car {
  constructor(brand, mod) {
    super(brand);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model;
  }
}

mycar = new Model("Ford", "Mustang");
document.getElementById("demo").innerHTML = mycar.show();
```