the Master Course

{CODENATION}

Intermediate JavaScript

Object Orientated Programming

{CODENATION}

Learning Objectives

To explore object orientated programming and use the class syntax

To be familiar with and use class inheritance

What is Object Orientated Programming?

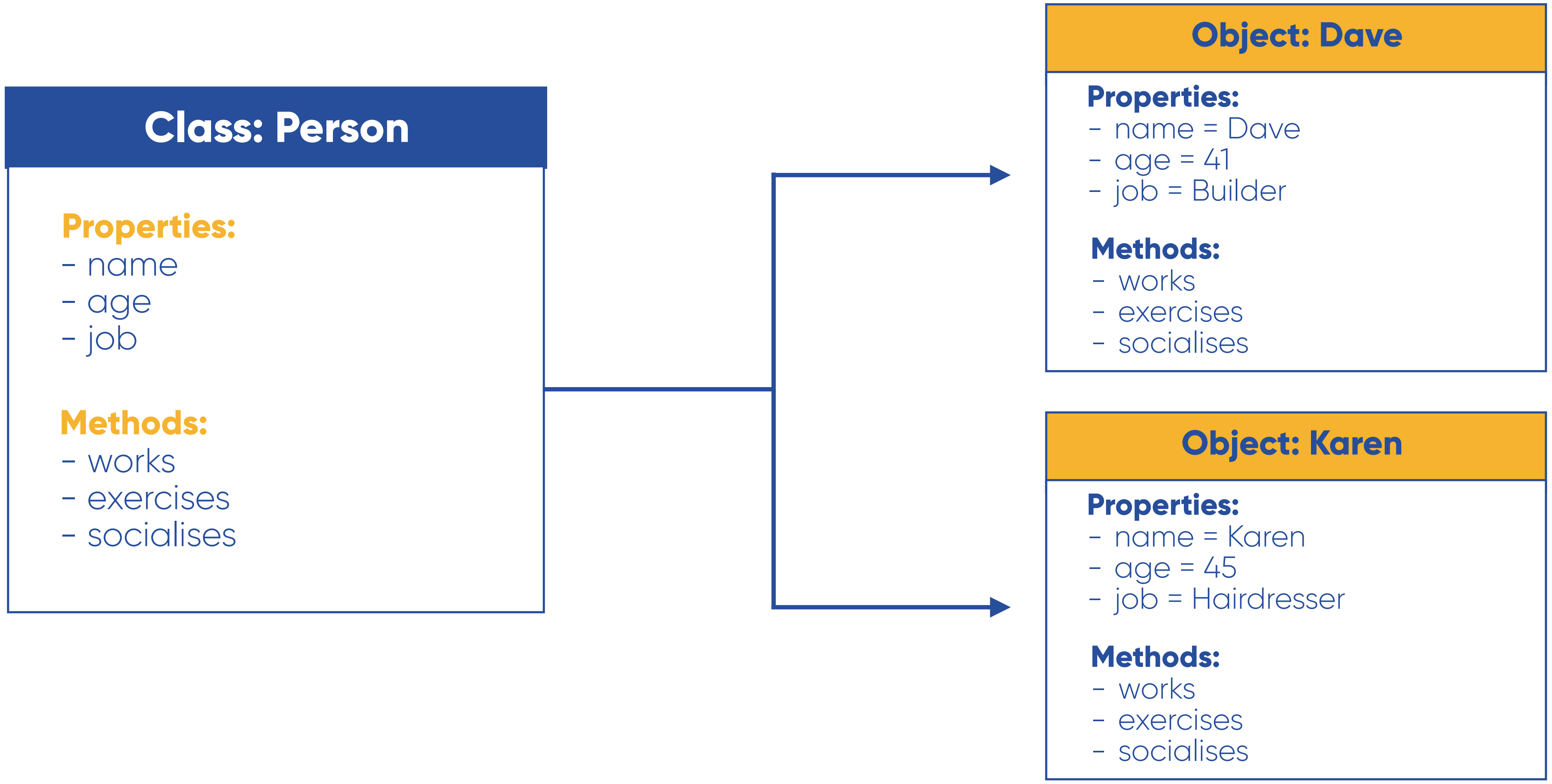
Object Oriented programming (OOP) is a programming pattern that relies on the concept of **classes, subclasses** and **objects**.

It is used to form a software program into simple, reusable pieces of code templates (classes), which are used to create individual instances of objects.

First let's revisit object literals

{CODENATION}

Intermediate JS



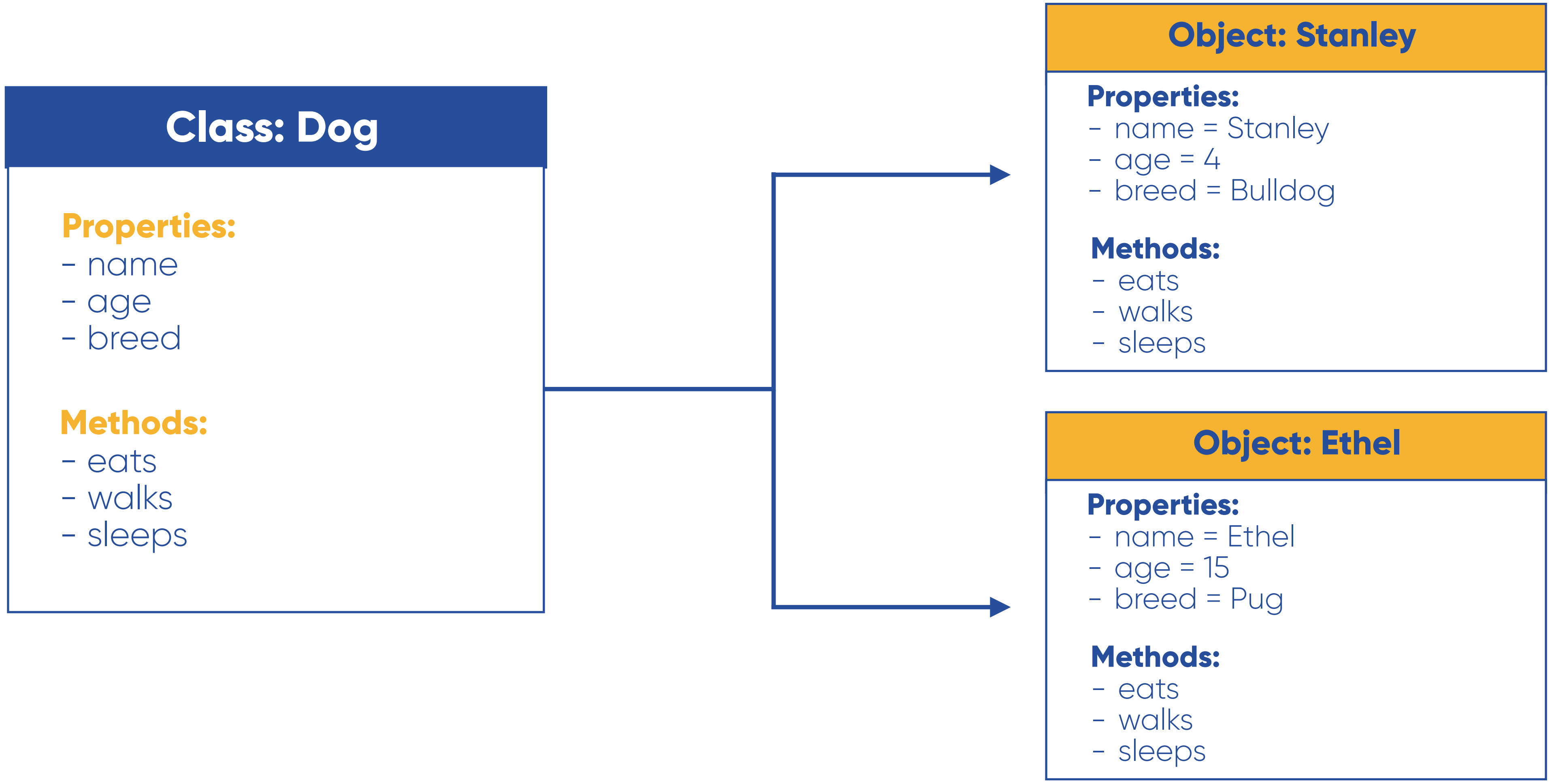
```
class Person {
    constructor(name, age, job) {
        //properties here
        this.name = name;
        this.age = age;
        this.job = job;
    }
    //methods here
    talks() {
        console.log(
            `Hi, my name is ${this.name}, I am ${this.age} and I work as a ${this.job}`,
        );
    }
    work() {
        console.log(`I am going to build a house, because I am a ${this.job}`);
    }
}
//create a new instance of the class
const dave = new Person('Dave', 41, 'Builder');

dave.talks();
dave.work();
```

Use the keyword class to create a template

Use the 'this' keyword inside of the class to refer to the current instance.

{CODENATION}

Intermediate JS



{CN}®

```
class Dog {
    constructor(name, breed) {
        this.name = name;
        this.breed = breed;
    }
    walks() {
        console.log(`Taking ${this.name} the ${this.breed} for a walk`);
    }
    eats() {
        console.log(`${this.name} has had some food`);
    }
}

const stanley = new Dog('Stanley', 'Bulldog');

stanley.walks();
stanley.eats();
```

Use the constructor method to create properties

We use the new keyword to create an instance of our dog class

```
class Dog {
    constructor(name, breed) {
        this.name = name;
        this.breed = breed;
    }
    walks() {
        console.log(`Taking ${this.name} the ${this.breed} for a walk`);
         return this;

    }
    eats() {
        console.log(`${this.name} has had some food` );
         return this;
    }
}
const stanley = new Dog('Stanley','Bull Dog');

stanley.walks().eats();
```

Explicitly return the instance at the end of methods.

To chain and use the methods together.

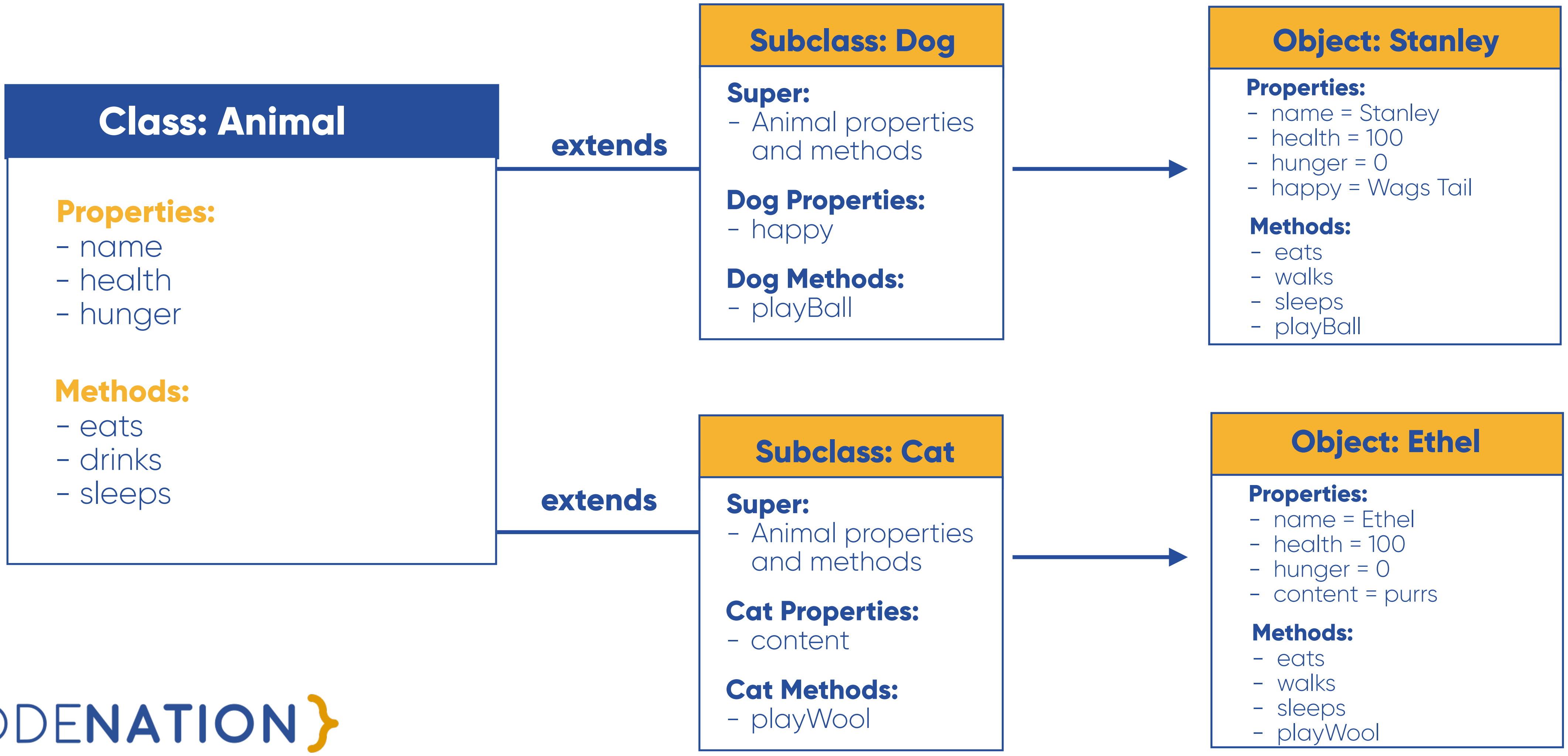
Class Inheritance

Subclasses

What is class inheritance?

Inheritance allows you to define a subclass that takes all the properties and methods from a **parent class** and will enable you to add more.

Class Inheritance



Parent Class

```
class Animal {
    constructor(name) {
        this.name = name;
        this.health = 100;
        this.hunger = 100;
    }
    drinks() {
        this.health += 5;
        return this;
    }

    eats() {
        this.health += 5;
        this.hunger += 10;
        console.log(`${this.name}'s health is ${this.health}`);
        return this;
    }
    stats() {
        return console.table({
            name: this.name,
            health: this.health,
        });
    }
}
```

Subclass

Using the **super keyword** inside a constructor runs the constructor from the parent class to set up the properties for the new subclass.

```
class Dog extends Animal {
    constructor(name, happy) {
        //Dog specific properties here
        super (name, happy);
        this.happy = happy;
    }
    //Dog specific methods
    playBall() {
        this.health += 10;
        this.hunger -= 10;
        console.log(`${this.name} is happy`);
        return this;
    }
    walks() {
        console.log(`Taking ${this.name}  for a walk, they are ${this.happy}`);
        this.health += 10;
        return this;
    }
}
```

{ CN }®

Intermediate JS

Subclass

Add the **parameters** of your **properties** that you want to use as **arguments** into both the subclass constructor and super.

```
class Cat extends Animal {
    constructor(name, content) {

        super(name, content);
        this.content = content;
    }

    playWool() {
        this.health += 10;
        this.hunger -= 10;
        console.log(`${this.name} is happy`);
        return this;
    }
    naps() {
        console.log(`${this.name} is taking a lovely nap, they are ${this.content}`);
        this.health += 10;
        return this;
    }
}
```

Getters and Setters

In a JavaScript class, **getters** and **setters** are used to get or set the properties values.

Get

Is the keyword used to define a getter method for retrieving the property value.

Set

Defines a setter method for changing the value of the specific property.

{CN}®

Intermediate JS

```
class Person {
    constructor(firstName, lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
    get fullName() {
        return `${this.firstName} ${this.lastName}`;
    }
    set fullName(value) {
        const names = value.split(' ');
        this.firstName = names[0];
        this.lastName = names[1];
    }
}

let person = new Person('Dave', 'Jones');
//set it
person.fullName = 'Will Smith';

//get it
console.log(person.fullName);
```

A **Setter** must have one parameter .

Further Information

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Classes_in_JavaScript

Learning Objectives

To explore object orientated programming and use the class syntax

To be familiar with and use class inheritance

For later...

Take a look at **Asynchronous** JavaScript....

https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing

https://www.youtube.com/watch?v=PoRJizFvM7s

What is **asynchronous JavaScript**?

Can you research **promises** and the **async** and **await** keywords?