# Composition vs Inheritance

TLDR:

By favoring composition over inheritance and thinking of what things do instead of what they are, you free yourself of fragile and tightly coupled inheritance structures.

Below is an example of how Inheritance is accomplished. We abstracted common features amongst every animal: name, energy,eat,sleep and play into a base-class called Animal, after which we created subclasses for each animal.

So the structure is seen below.

Animal

name

energy

eat()

sleep()

play()

Dog

breed

bark()

Cat

declawed

meow()

This works well as it allowed us to minimize code duplication and maximise code reuse.

Let’s say we are creating the software for the game, and we need to add users to the game.

The users enjoyed the game, and they want a more real-life experience. Namely, **Users want to be able to eat(), sleep(), and play() as well.**

User

email

username

pets

friends

adopt()

befriend()

Animal

name

energy

eat()

sleep()

play()

Dog

breed

bark()

Cat

declawed

meow()

But now the functions eat(), sleep() and play() are encompassed inside the Animal class.

We could abstract these common properties to another parent, and have one more step of inheritance.

But we can easily see that the structure is very **rigid**, and hard to **maintain**.

A way to work around this is to think about what it *does* instead of what it *is*.

A dog is a eater, sleeper, player, and barker.

A user will then be a sleeper, eater, player, adopter and friender.

Instead of defining methods and confining them within a particular class, if we abstract them into their own functions, we can compose them together with any type that needs them, using the **Object.assign** method.

const eater = (**state**) => ({

eat (amount) {

console.log(‘${**state**.name} is eating.’)

**state**.energy += amount

}

})

We can access the eater function for Dog as per below:

return **Object.assign**(

dog,

eater(**dog**),

sleeper(dog),

)

function User (email, username) {

let user = {

email,

username,

pets: [],

friends: []

}

return Object.assign(

user,

eater(user),

player(user),

adopter(user),

friender(user)

)

}