Inheritance

A class can inherit methods, properties, and other characteristics from another class

Swift terminology:

- **Base class**: A class that doesn't inherit from another class.
  - Also known as a superclass or parent class
- **Subclass**: A class that extends the behaviours of another class.

Overriding:

- In Swift, a child class (subclass) can *override* its parent's **methods** or **properties (computed and stored)**
- Use the `override` keyword to indicate when something is being overriden

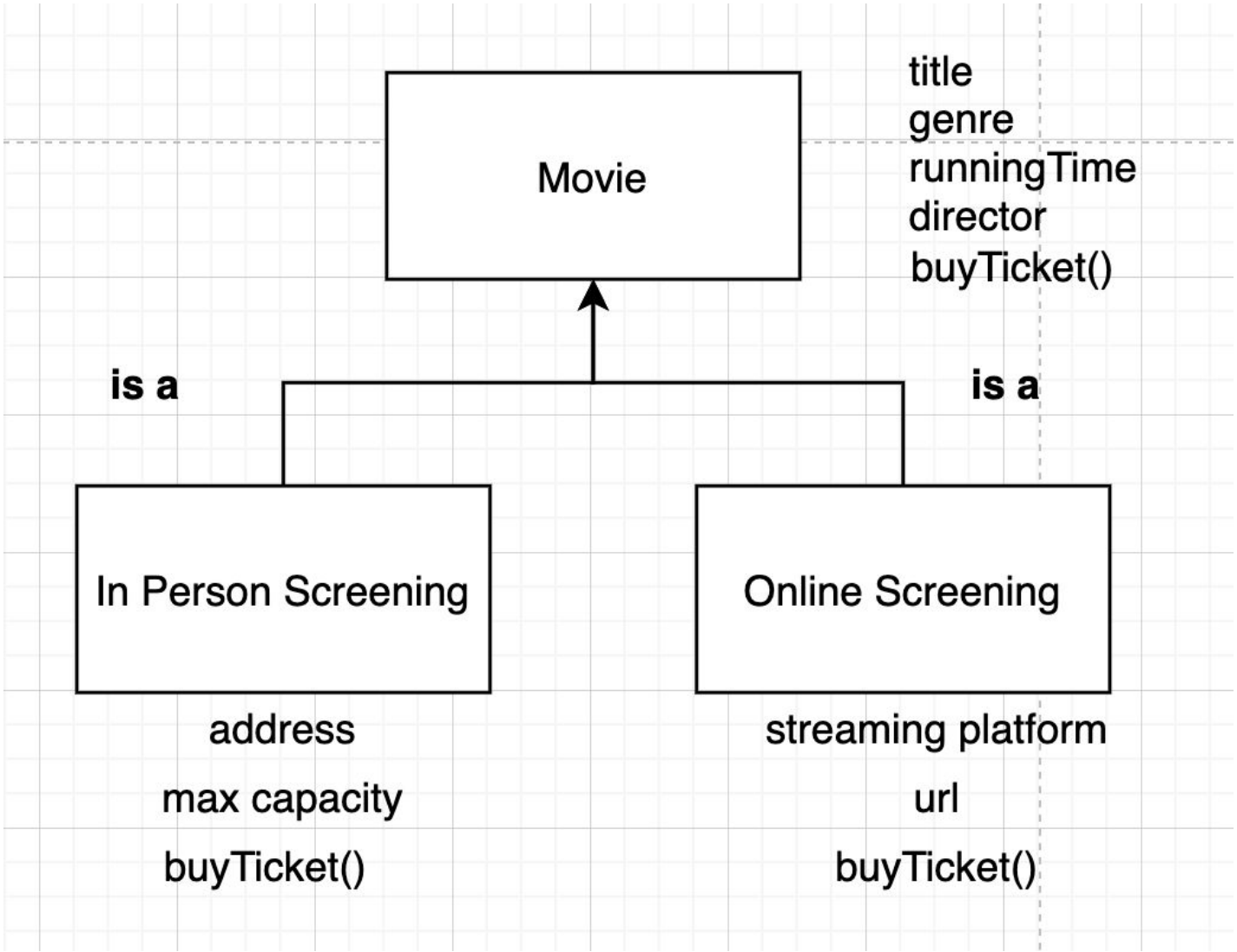
Inheritance

A class can inherit methods, properties, and other characteristics from another class

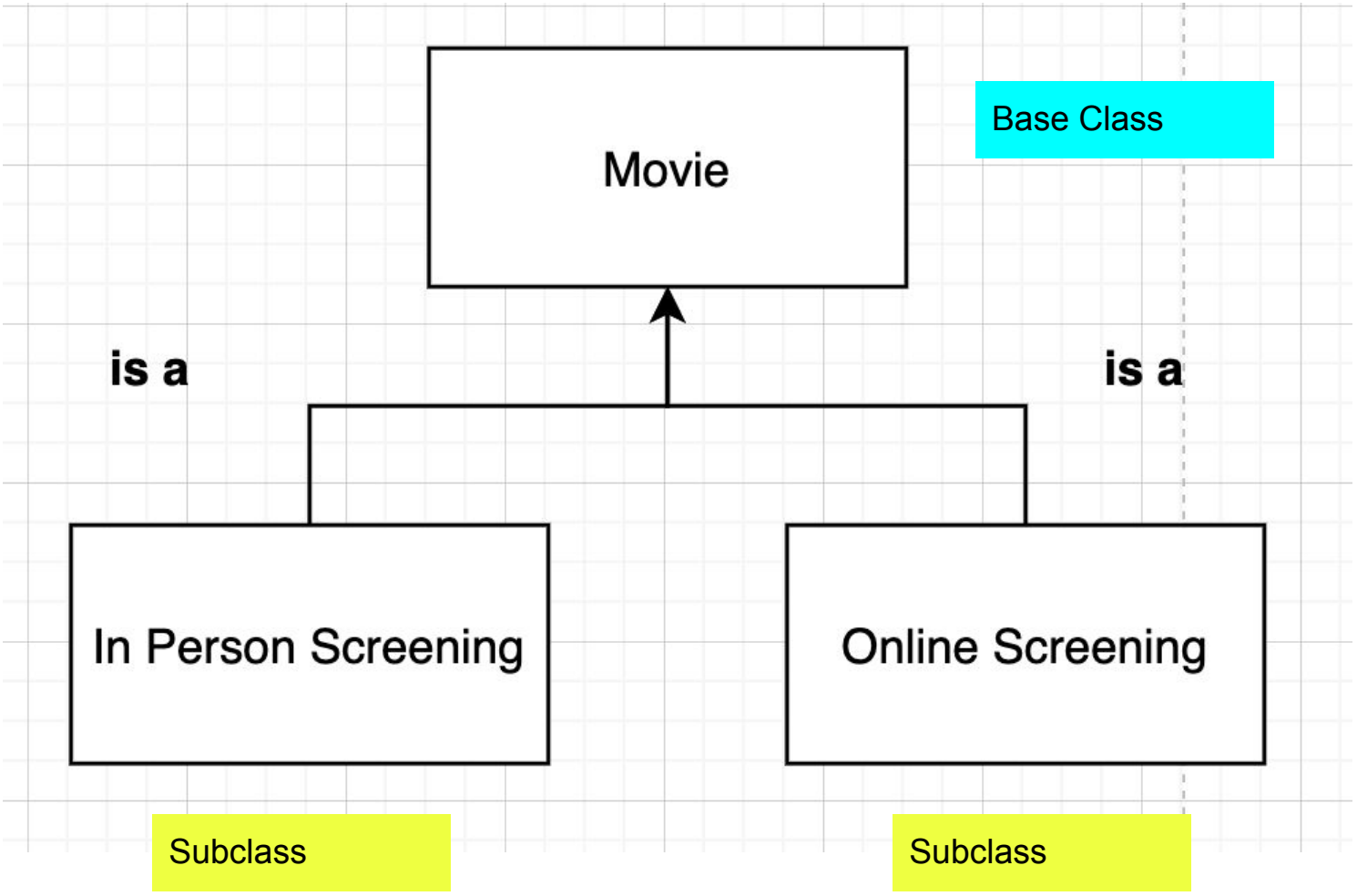
Overloading:

- Same method (function) name, different parameters

Example: Inheritance



Example: Inheritance



Protocols

Protocols define a *blueprint* of methods or properties, but do *not* provide any implementation details.

- Similar to an **interface** in other OOP languages

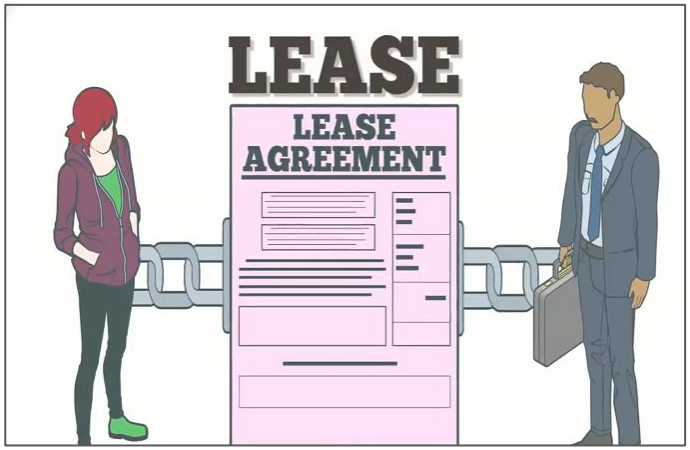
A class can then *conform* to the protocol by providing the implementation details of the protocol's method / property definitions.

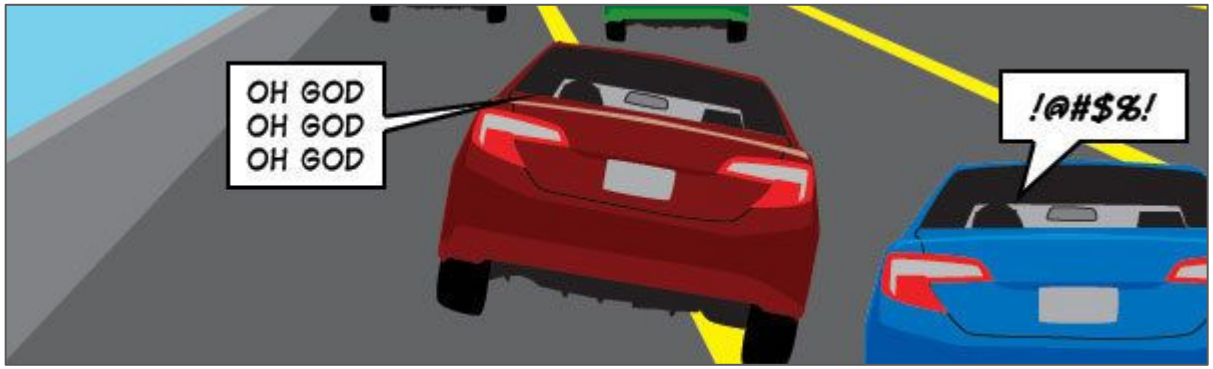
A class can *conform* to more than one protocol.

- This is a commonly used technique for allowing a class to inherit properties/methods from more than 1 parent.

What is an protocol

- A protocol captures specific attributes or behaviours of an object
- The protocol acts as a **contract** for how an object should behave



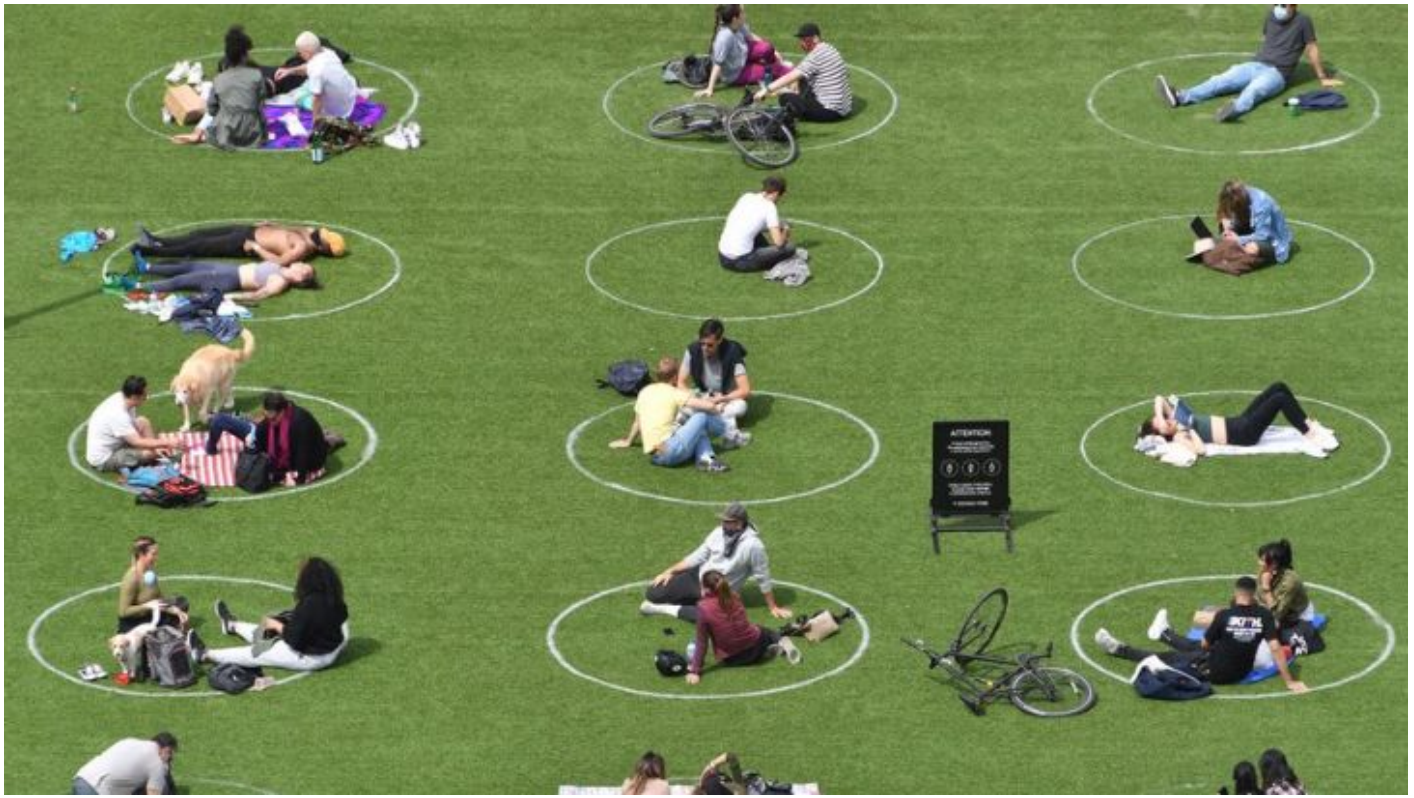


Protocols do not implement code

The contract specifies rules, but not implementation details.

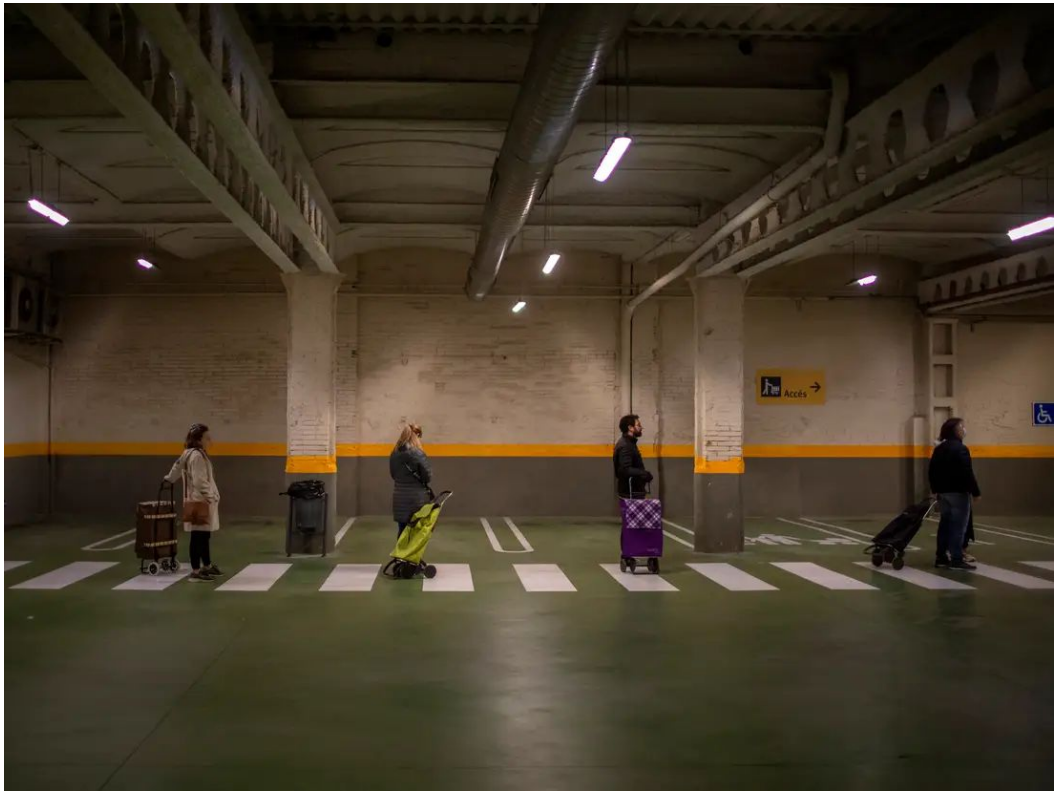
Implementation details can vary:

Practice Physical Distancing



Implementation details can vary:

Practice physical distancing



Implementation details can vary:

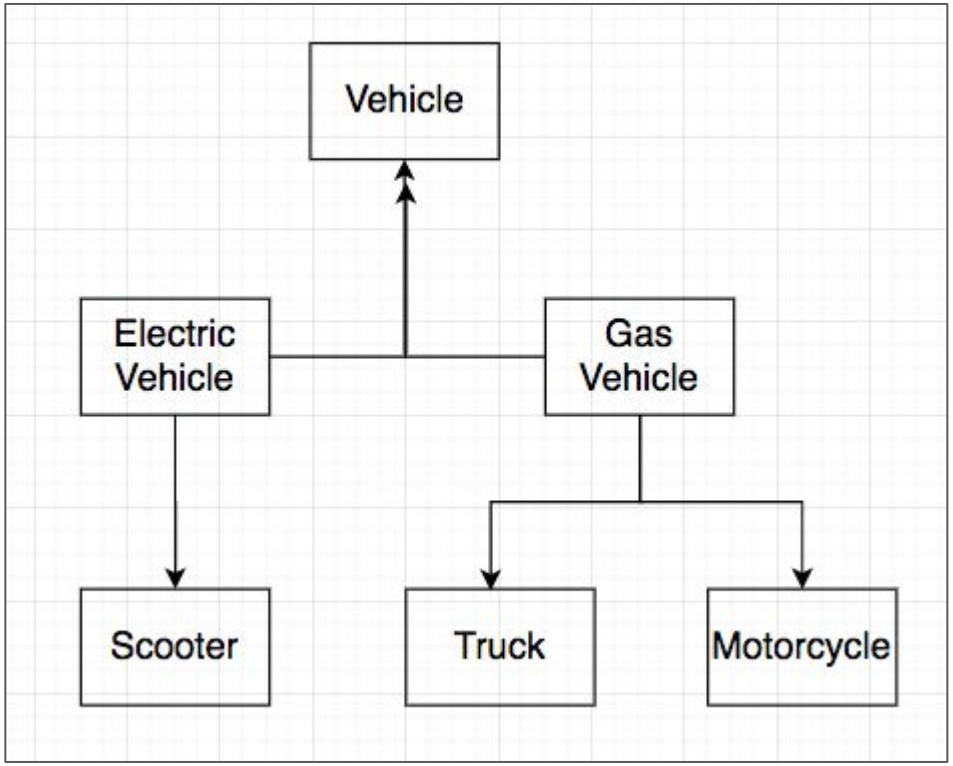
Practice physical distancing

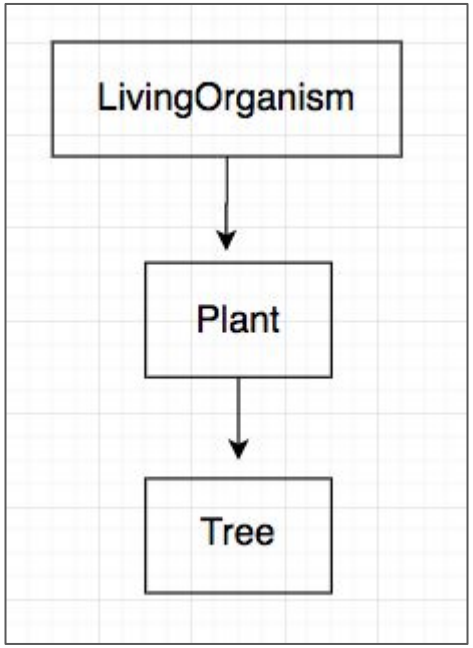
When to use a Protocol?

1. When traditional inheritance fails
2. When you want to model objects based on their characteristics, rather than their relationship to another object

Inheritance Works When Relationships are Obvious

Inheritance works when there are obvious relationships between entities:

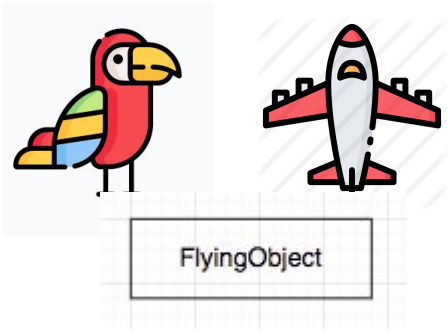




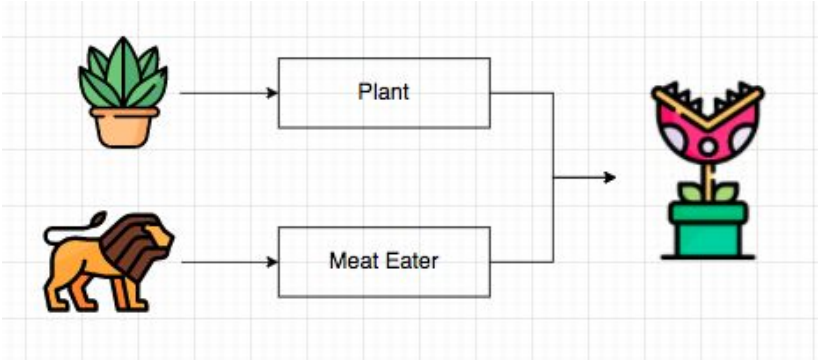
When Inheritance Fails

In real life, do not always fit neatly into a parent-child relationship. Here are two examples:

Example 1: Objects that share similar characteristics, but are not related:



Example 2: Objects that inherit from multiple parents



Protocol Based Design

Protocols enable us to develop classes based on *reusable behaviours*.

For example:

- Elephants and Cranes lift heavy objects!
- Elephants are Animals
- Cranes are Vehicles or Construction Equipment
    - There's no relationship between Animals and Vehicles!

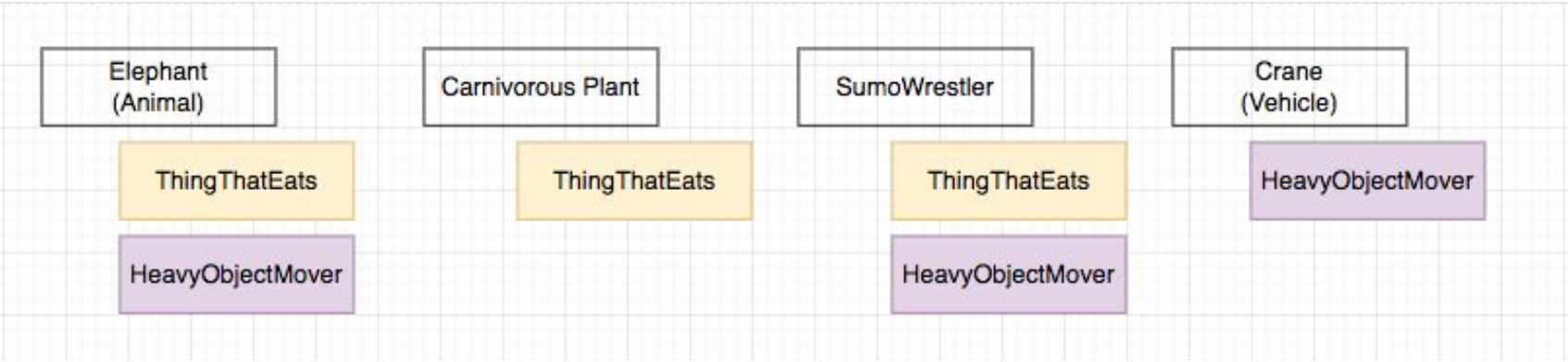
Elephants, Cranes, and Sumo Wrestlers → Lift heavy objects

Sumo Wrestlers, Venus Fly Traps, and Elephants → Eat a lot

Classifying Behaviours

Objects can be classified according to behaviour

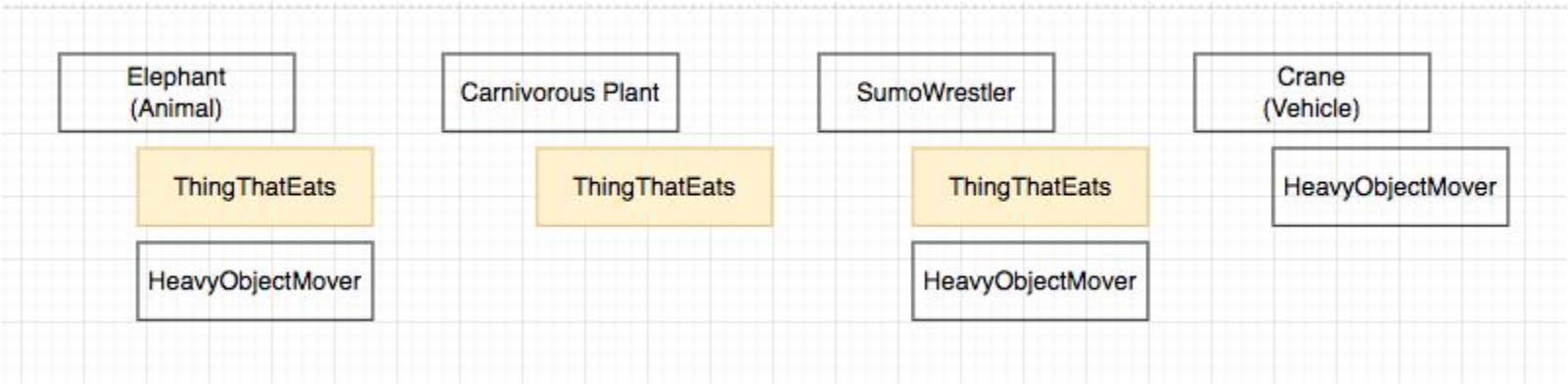
- Some objects may subscribe to more than 1 behaviour



Classifying Behaviours

Objects can be classified according to behaviour

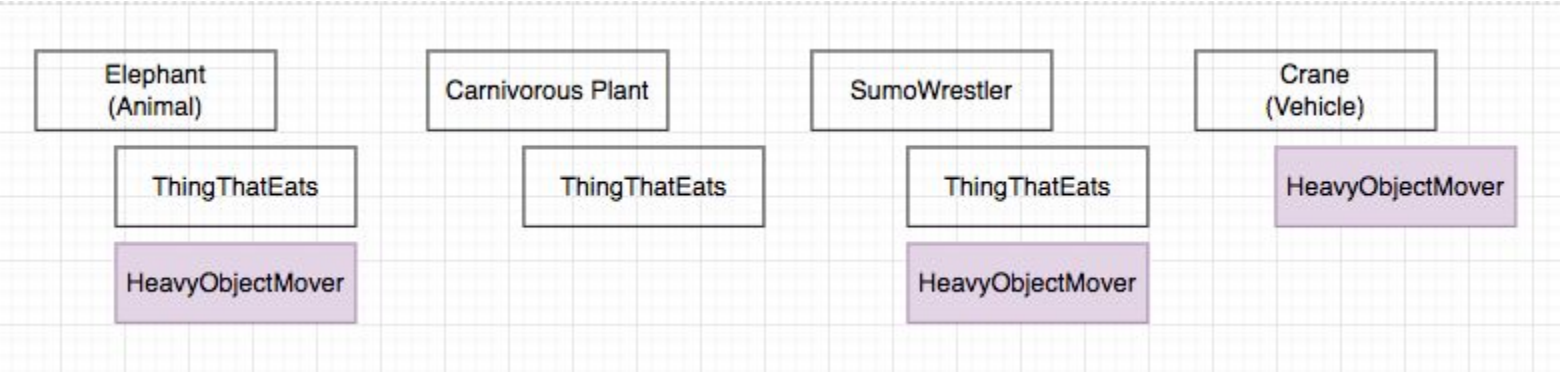
Things that Eat



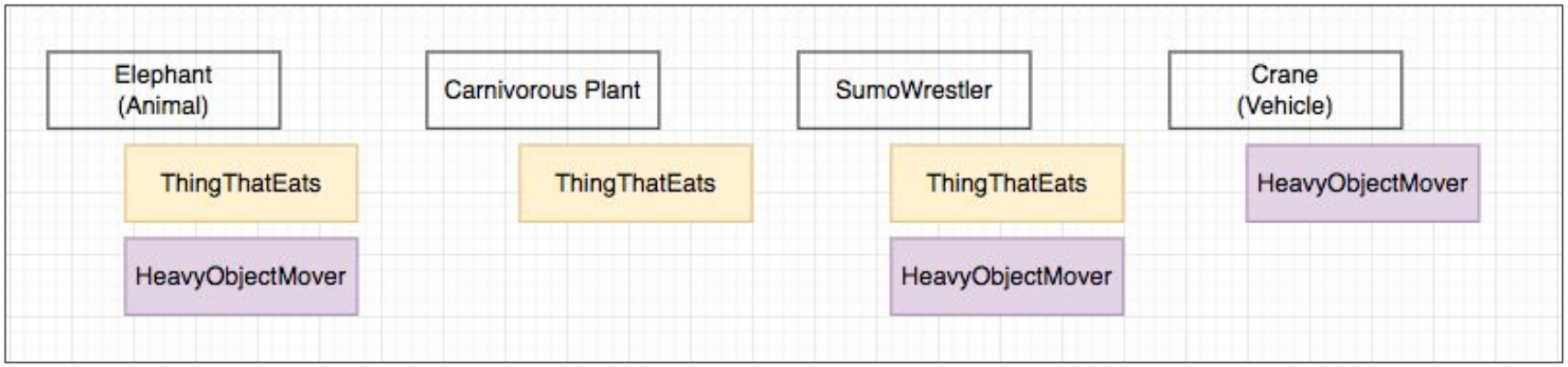
Classifying Behaviours

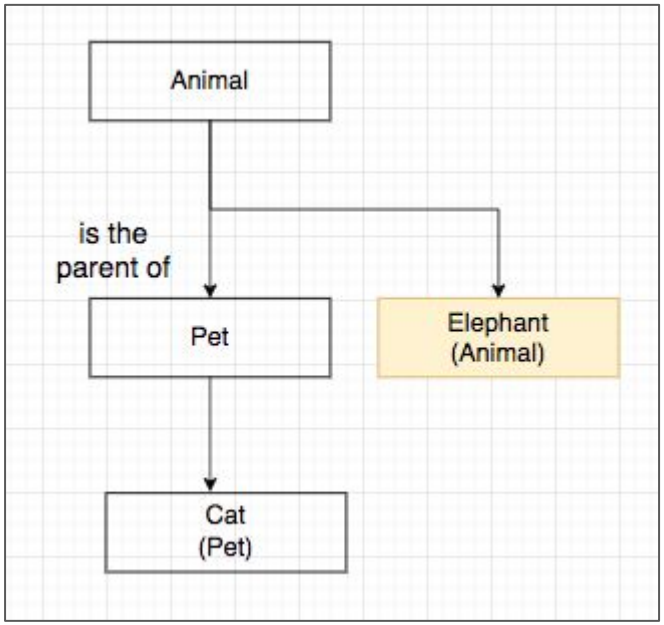
Objects can be classified according to behaviour

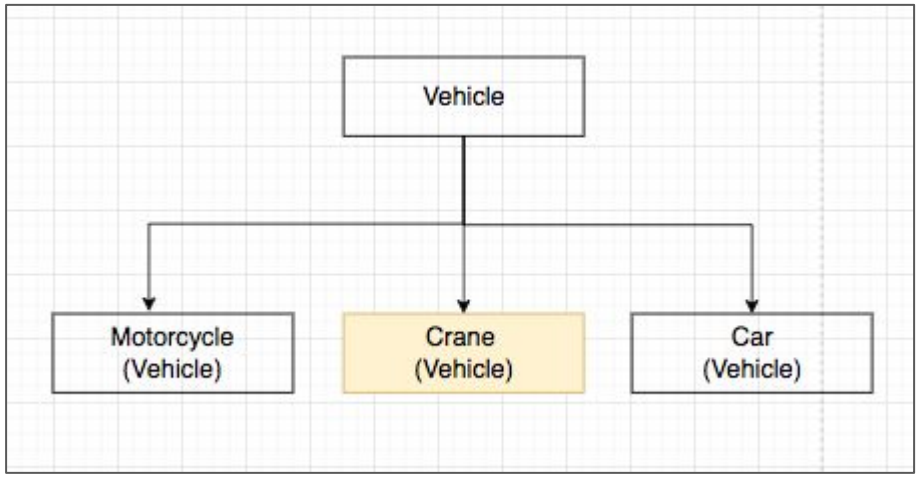
Things that Move Heavy Objects



Protocol Based Design vs. Inheritance Design

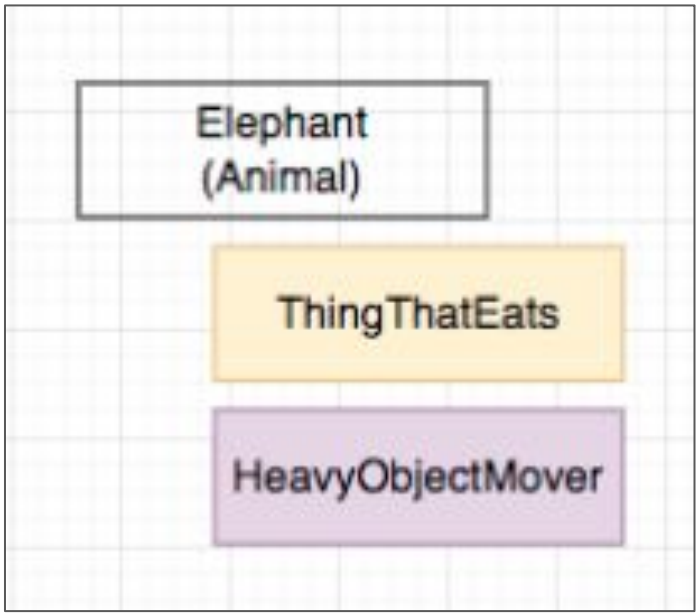






Capturing behaviour vs types

Behaviour



Type