Object oriented Programming (OOP)

1. Code Reuse

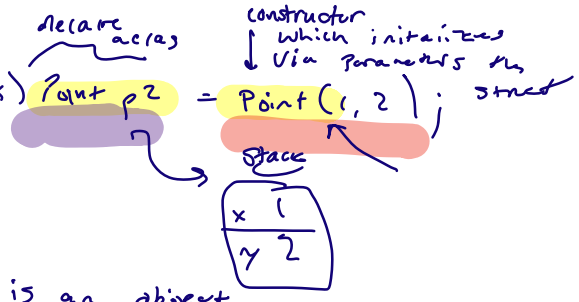
Classes code reuse → because they encapsulate both data and behavior

Class is a template for behavior of a collection of data,

— i.e.

```
struct Point {
    int x, y;
    int magnitude();
};
```

Object an instance of a class, ex) Point p2 = Point(1, 2); 3



p2 is an object

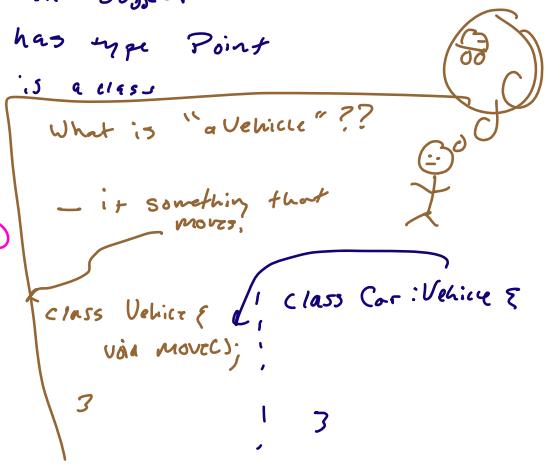
has type Point

Point is a class

Two types of relationships:

- → is-a → inheritance
- → has-a (composition)
  - data (private public variables)

a car is-a vehicle.



A car is-a vehicle

A car is a derived class from Vehicle

Vehicle is the base class of a car

A car is a subclass of vehicle.

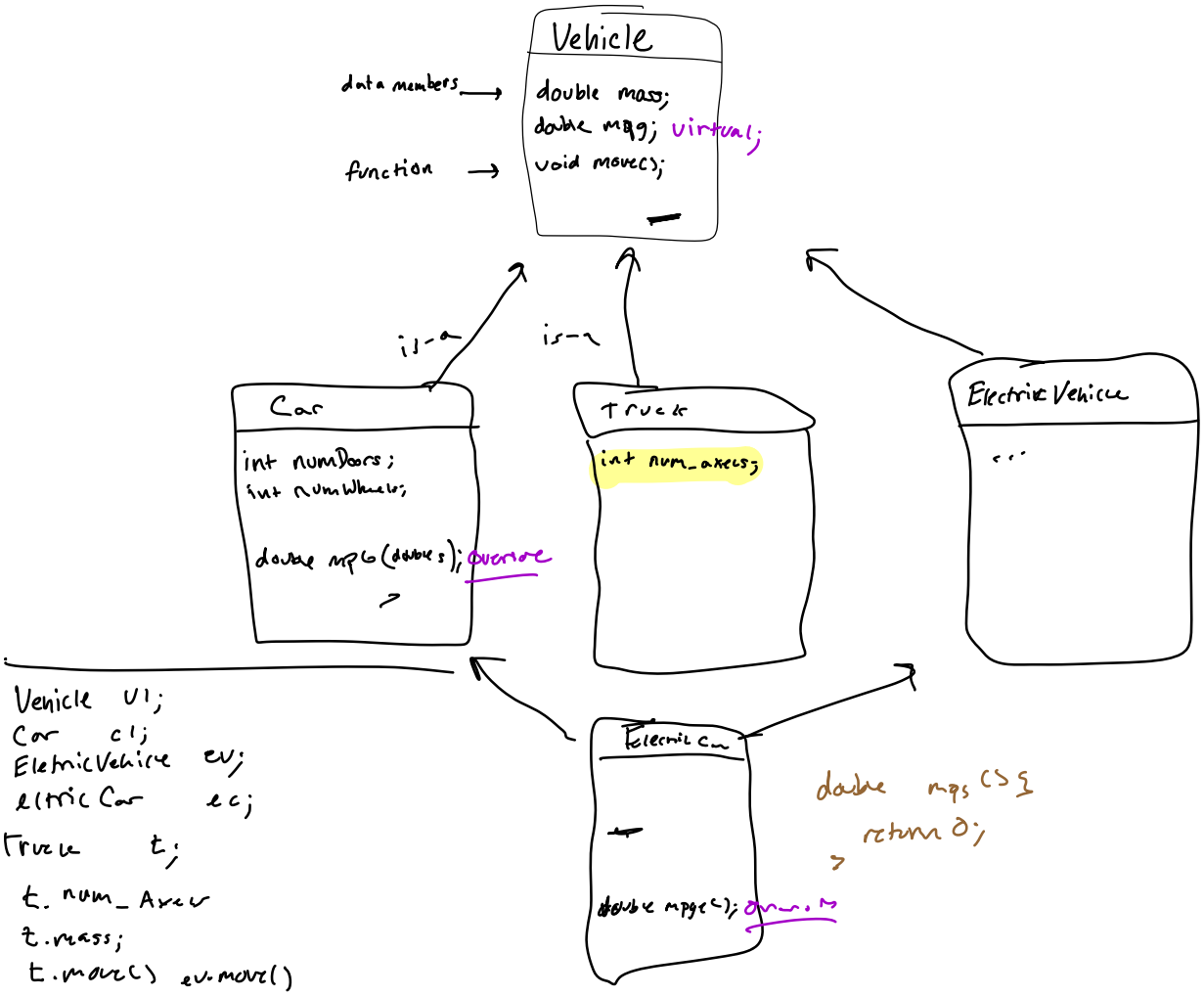
You can have multiple inheritance in C++.

```
Class ParkingLot {

    std::vector<Vehicle> vehiclesInLot;
};
```

Composition

Diagram




```
Vehicle v1;
Car c1;
ElectricVehicle ev;
ElectricCar ec;
Truck t;
t.num_Axles
t.mass;
t.move() ev.move()
```

Override → to change your derived class's behavior

→ our way altering our base class behavior.

ElectricCar ec;

ec.mpg(); → treats ec as-a car, and calls car's mpg function

now override car's mpg function for electric car.

→ if we overide

Polymorphism

```
void ParkingLot::AddVehicle(Vehicle const& v) {
  // does stuff.....
  double cost = v.mpg();
  // does stuff
}
```

if mpg virtual and overriden

this might be a car pretending to be a car vehicle.

Whose!?

So this calls Vehicles NOT whatever derived class

```
int main() {
  ParkingLot p;
  ElectricCar c;
  p.AddVehicle(c);
}
```

hasType Car but v should be a Vehicle!!

well... a car - is-a vehicle so let's cast it. implicitly static_cast<Vehicle&>(c);

Dynamic Dispatch → a runtime system which keeps track of the real/underlying class of an object even when casted.

how do we use it!

override and virtual.

SFINE → templates to correctly dispatch the correct

Public data/functions ⟶ avaible for anyone to use or to call

protected data/function ⟶ avaible only to that class or its derived classes

private ⇒ only that class