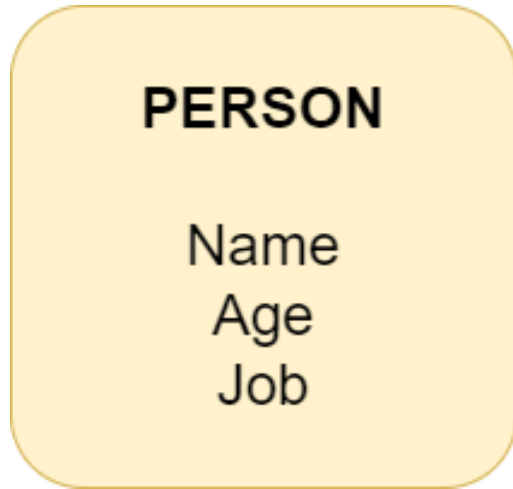# Programming for Social Scientists

Johan A. Dornschneider-Elkink
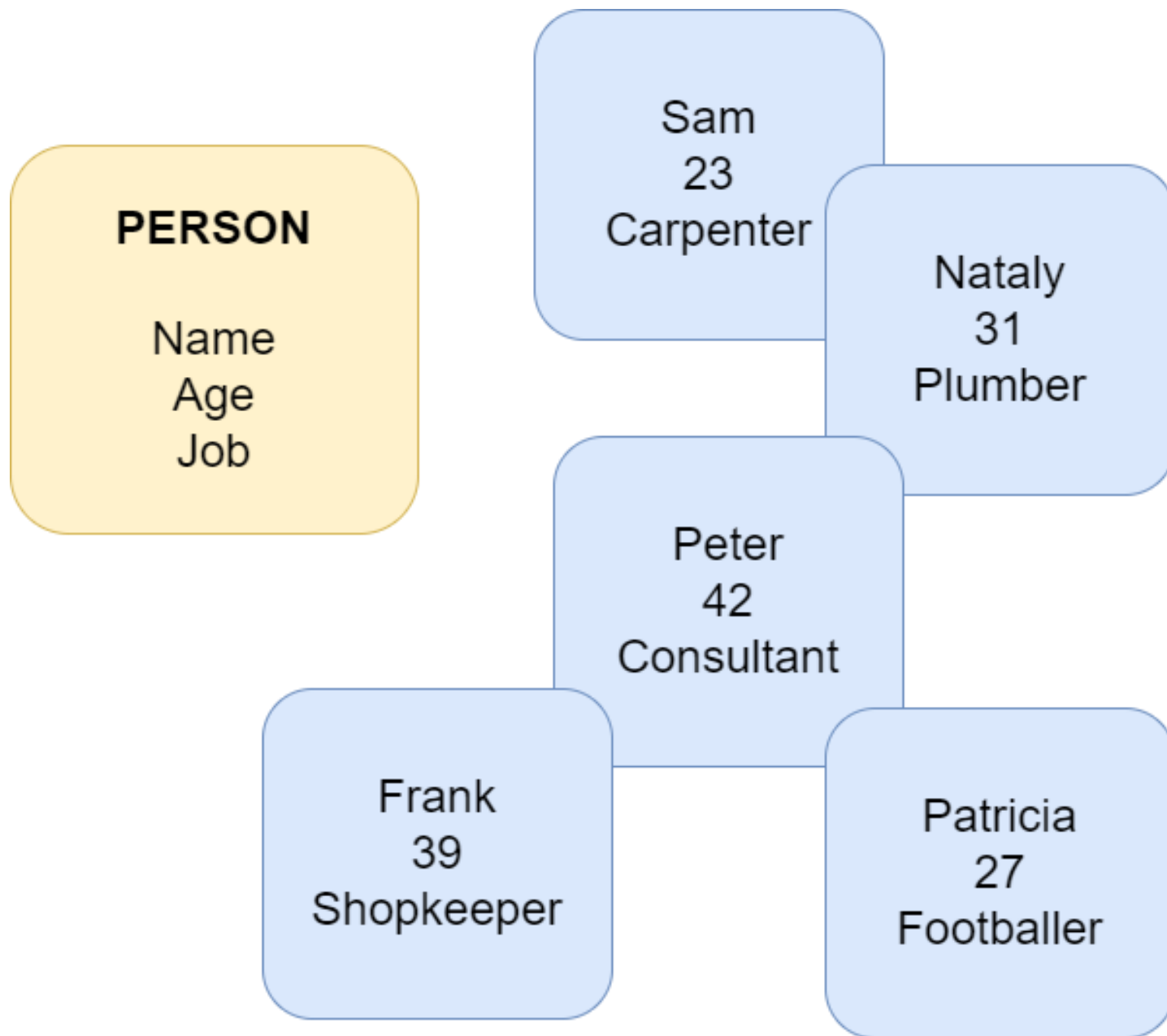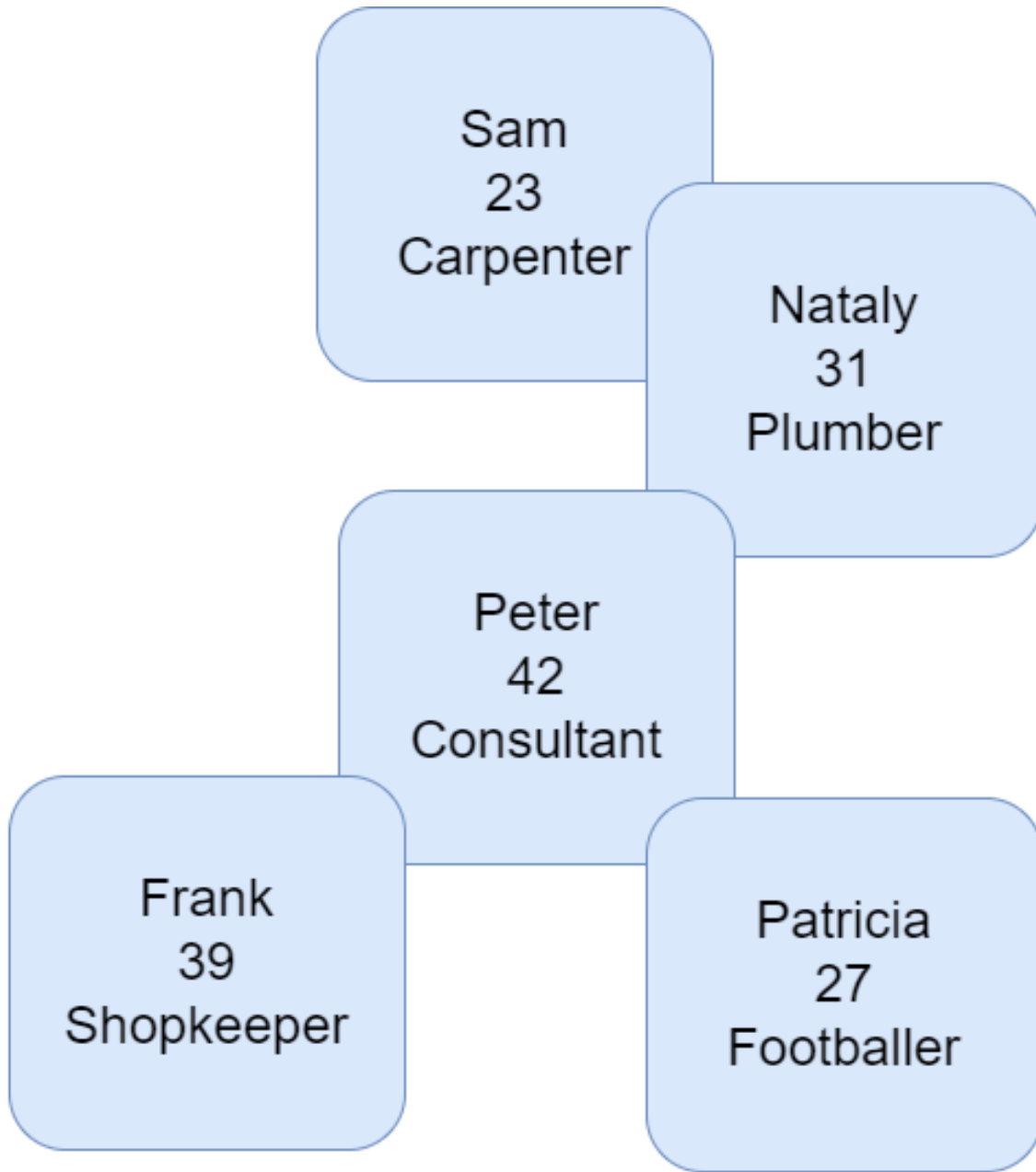
# Class inheritance

## Class

Blueprint or template for user-defined data.

Defines data and functionality to be associated with each instance.

Does not yet reserve any memory space for data.

**PERSON**

Name
Age
Job

Sam
23
Carpenter

Nataly
31
Plumber

Peter
42
Consultant

Frank
39
Shopkeeper

Patricia
27
Footballer

Sam
23
Carpenter

Nataly
31
Plumber

Peter
42
Consultant

Frank
39
Shopkeeper

Patricia
27
Footballer

# Object

Instance of a specific object, based on the class definition.
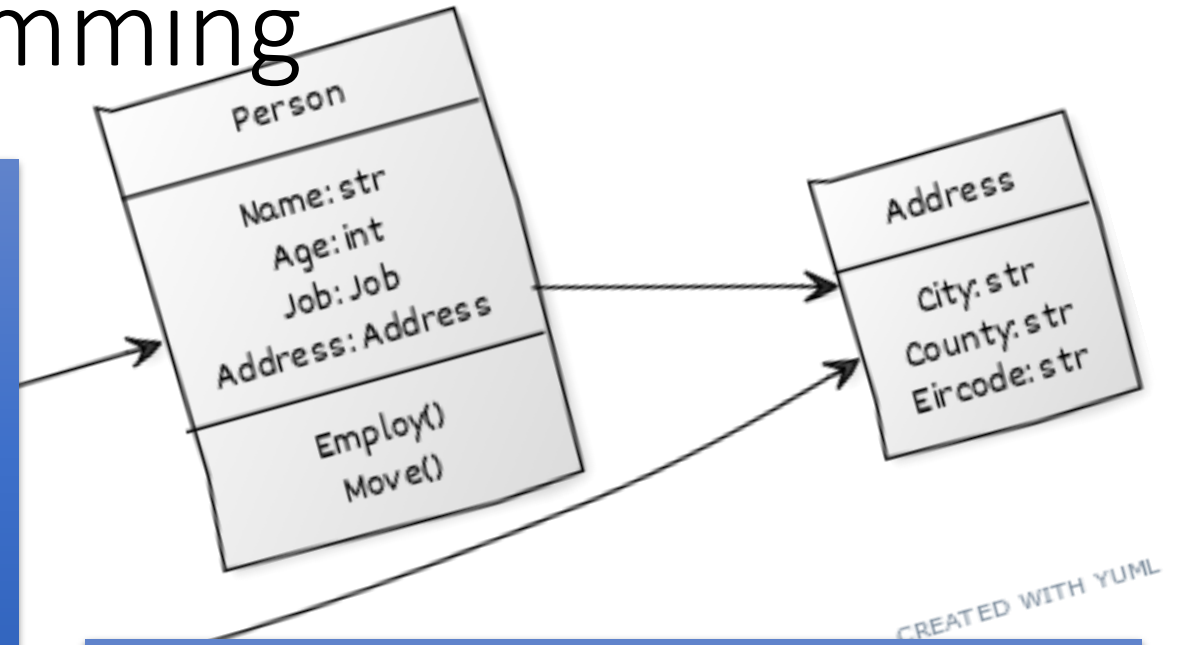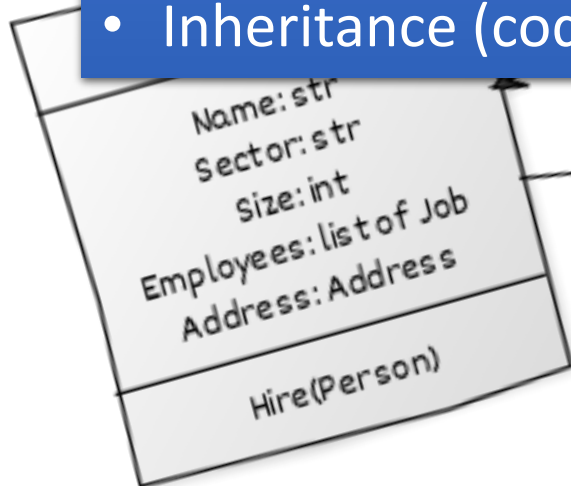
Reserves specific memory space for the data, as any other variable type.

# Object-Oriented Programming

**Key principles**

- Encapsulation (bundling data & methods)
- Abstraction (hiding details)
- Polymorphism (one interface, multiple implementations)
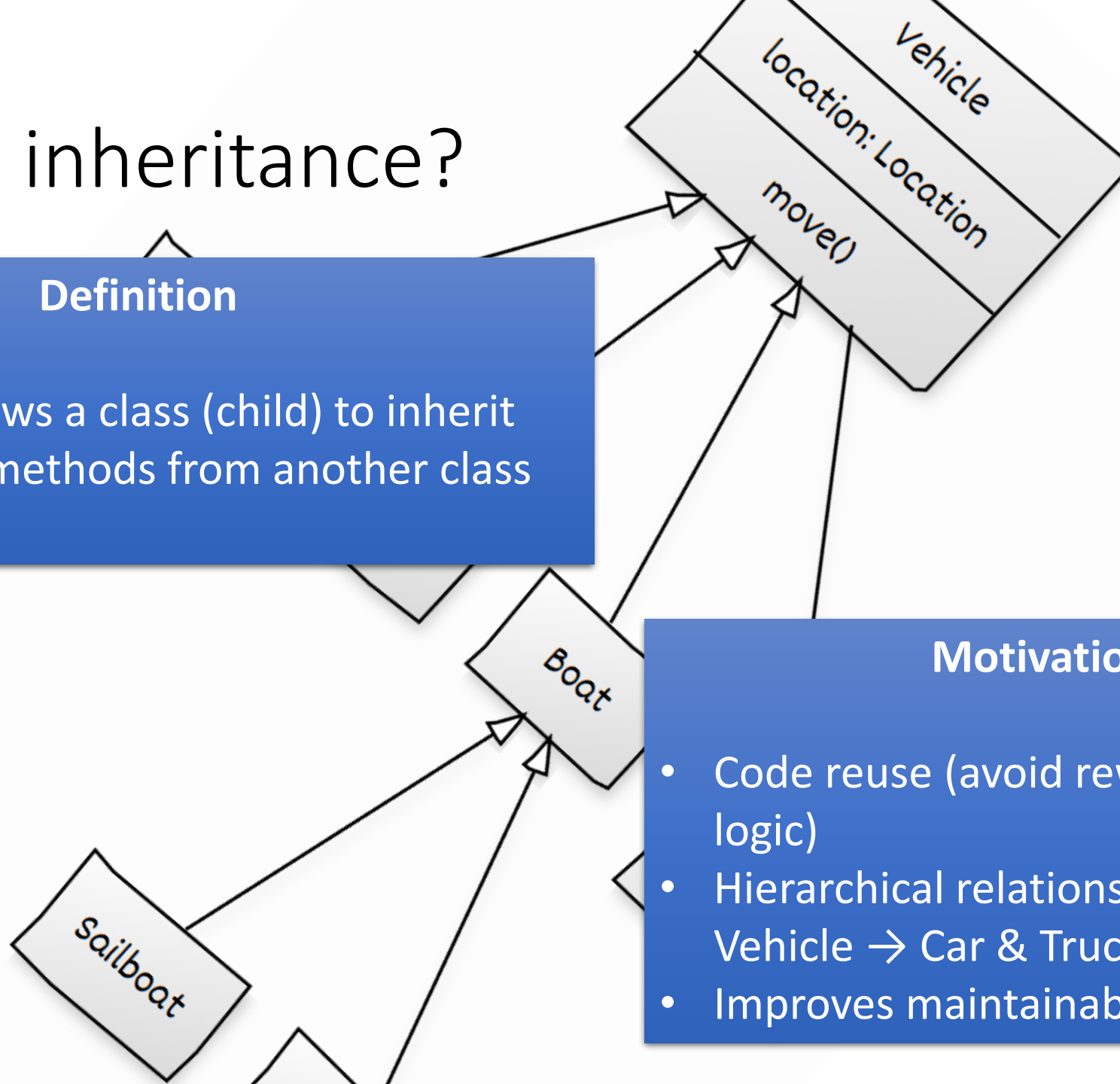- Inheritance (code reuse & relationships)

**Person**

Name:str
Age:int
Job:Job
Address:Address

Employ()
Move()

**Address**

City:str
County:str
Eircode:str

Name:str
Sector:str
Size:int
Employees:list of Job
Address:Address

Hire(Person)

CREATED WITH YUML

**Motivation**

- Organizes code efficiently
- Makes maintenance easier
- Allows for scalable and modular design

# What is inheritance?

**Vehicle**

location: Location

move()

## Definition

Inheritance allows a class (child) to inherit attributes and methods from another class (parent).
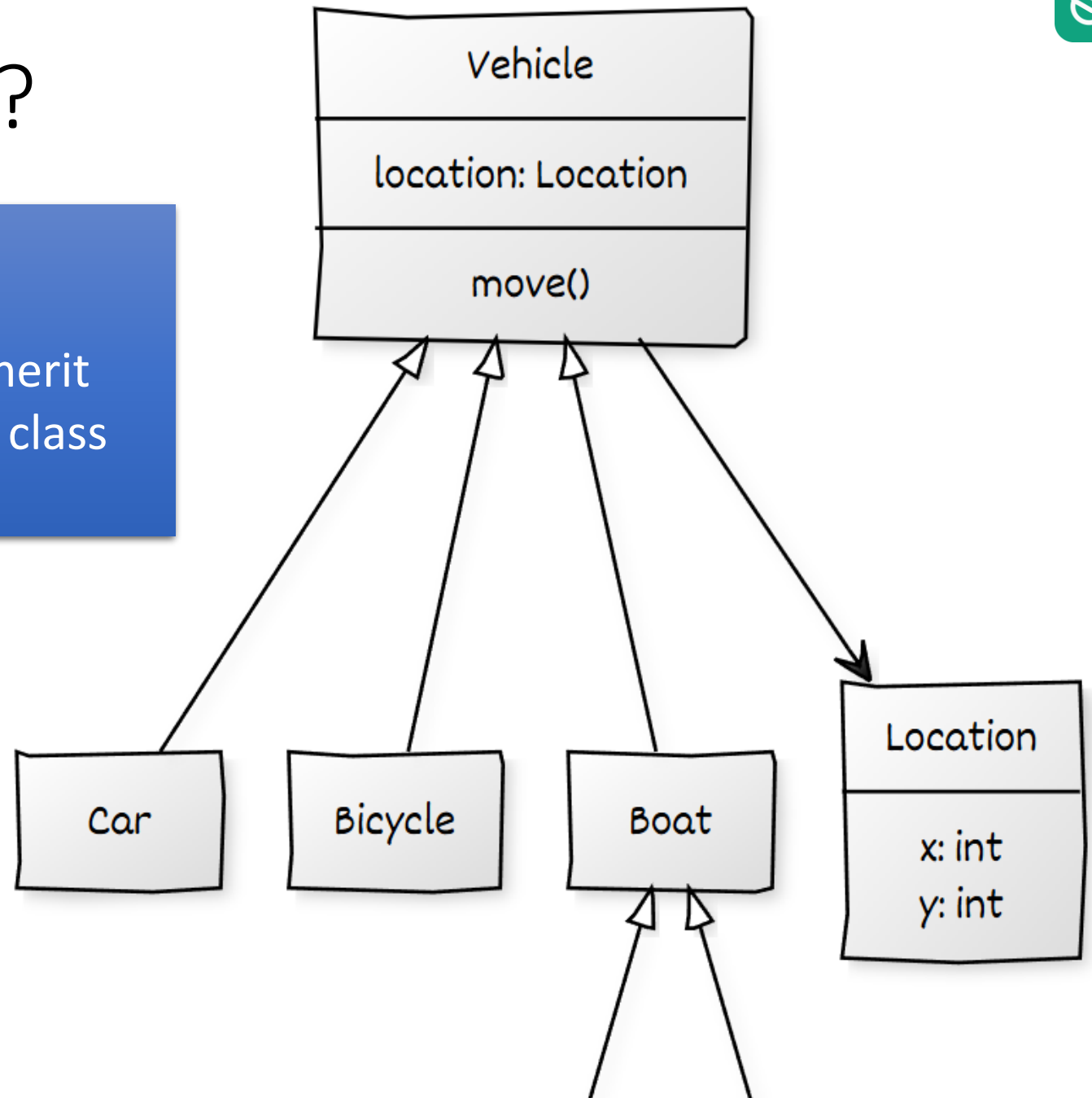
Boat

Sailboat

## Motivation

- Code reuse (avoid rewriting existing logic)
- Hierarchical relationships (e.g., Vehicle → Car & Truck)
- Improves maintainability

# What is inheritance?

**Definition**

Inheritance allows a class (child) to inherit attributes and methods from another class (parent).

Vehicle

location: Location

move()

Car

Bicycle

Boat

Location
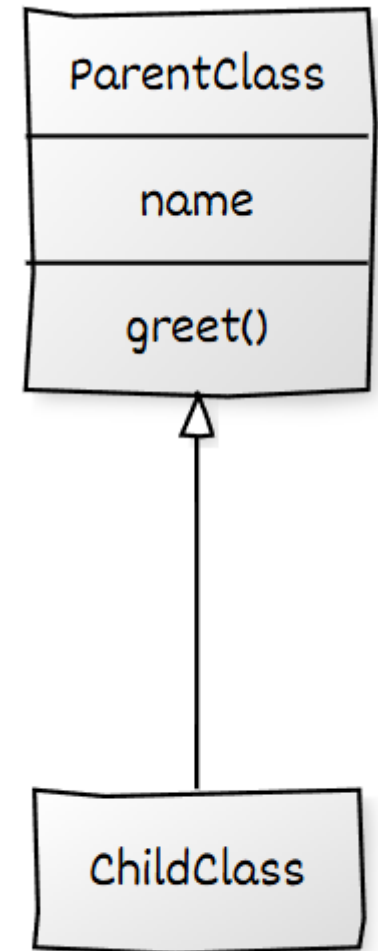
x: int

y: int

# Inheritance in Python

```python
class ParentClass:

    def __init__(self, name):

        self.name = name


    def greet(self):

        print(f"Hello, my name is {self.name}")



class ChildClass(ParentClass):
    pass


obj = ChildClass("Alice")

obj.greet()  # Inherits greet() from ParentClass
```
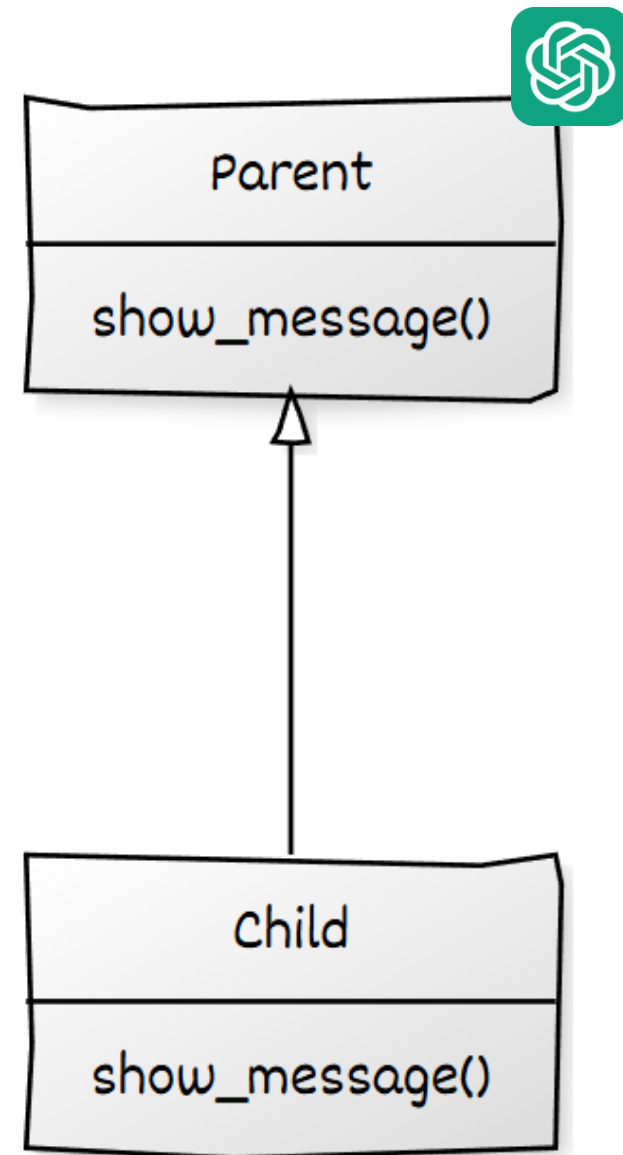


CREATED WITH YUML

# Overriding methods

```python
class Parent:

    def show_message(self):

        print("This is a message from the Parent class.")


class Child(Parent):

    def show_message(self):  # Overriding

        print("This is a message from the Child class.")


obj = Child()

obj.show_message()  # Calls the overridden method
```
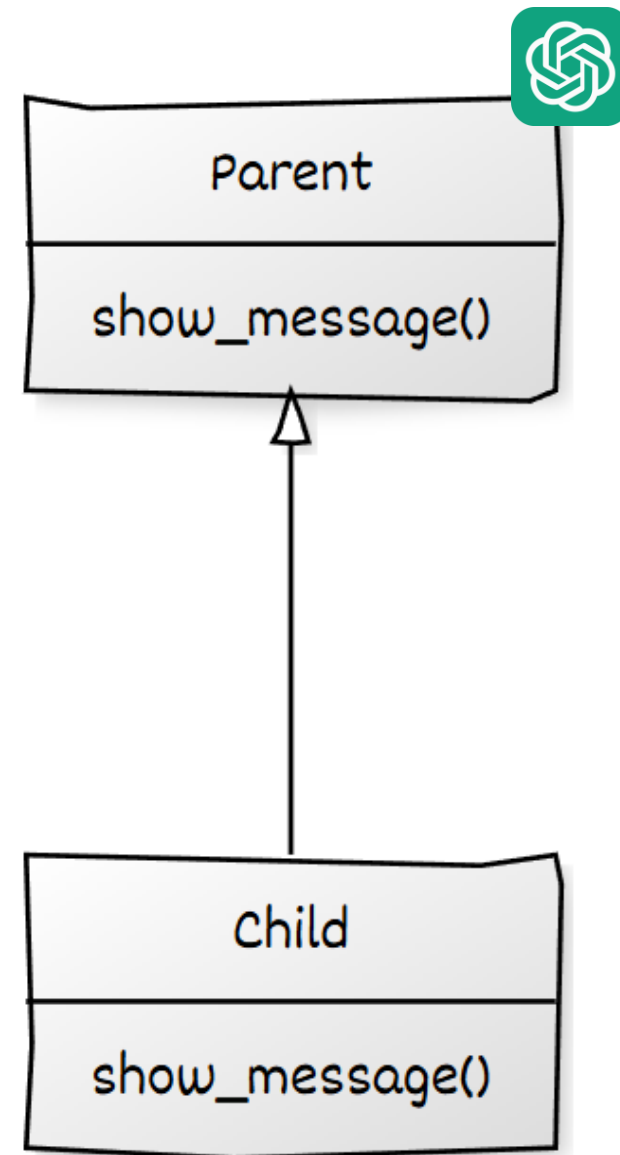
Parent

show_message()

Child

show_message()

CREATED WITH YUML

# Using super()

```python
class Parent:

    def show_message(self):

        print("This is a message from the Parent class.")


class Child(Parent):

    def show_message(self):

        super().show_message()   # Calls Parent's method

        print("This is a message from the Child class.")


obj = Child()
obj.show_message()
```
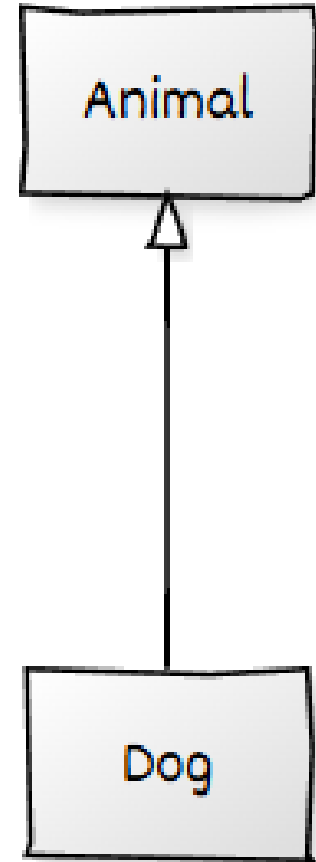


CREATED WITH YUML

# Using super() in the constructor

```python
class Animal:
    def __init__(self, name):
        self.name = name


class Dog(Animal):
    def __init__(self, name, breed):
        super().__init__(name)  # Call parent's __init__
        self.breed = breed

dog = Dog("Buddy", "Golden Retriever")
print(dog.name)   # Inherited from Animal
print(dog.breed)  # Defined in Dog
```



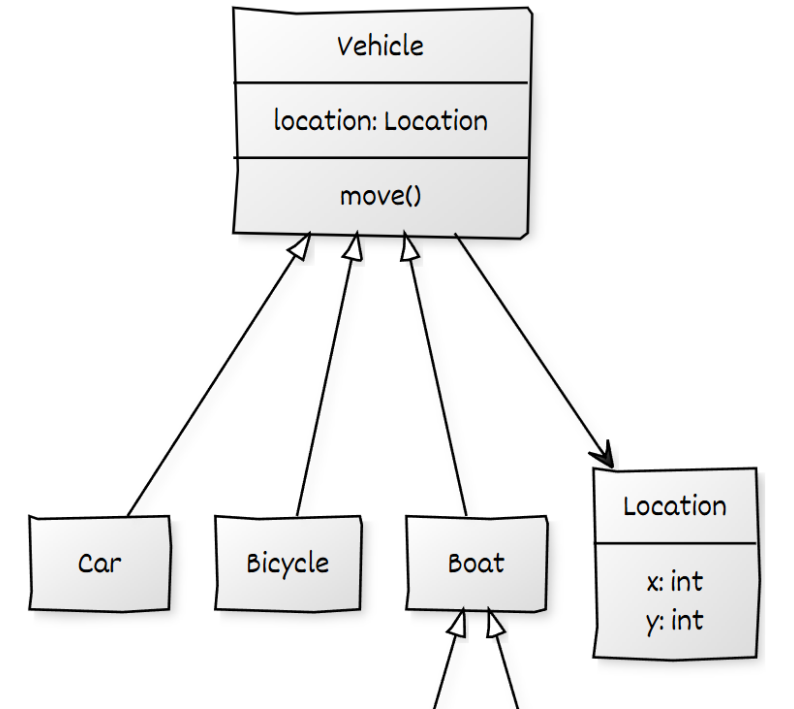CREATED WITH YUML

# Example in Python

```python
class Location:

    def __init__(self):
        self.x = 0
        self.y = 0
```

```python
class Vehicle:

    def __init__(self):
        self.location = Location()
        self.speed = 1
        self.direction = 0

    def move(self):
        self.location.x += self.speed * math.cos(self.direction)
        self.location.y += self.speed * math.sin(self.direction)
```

# Example in Python

```python
class Car(Vehicle):

    def __init__(self):
        super().__init__()
        self.fuel = 100

    def drive(self):
        if self.fuel > 0:
            self.move()
            self.fuel -= 1
        else:
            print("Out of fuel!")
```
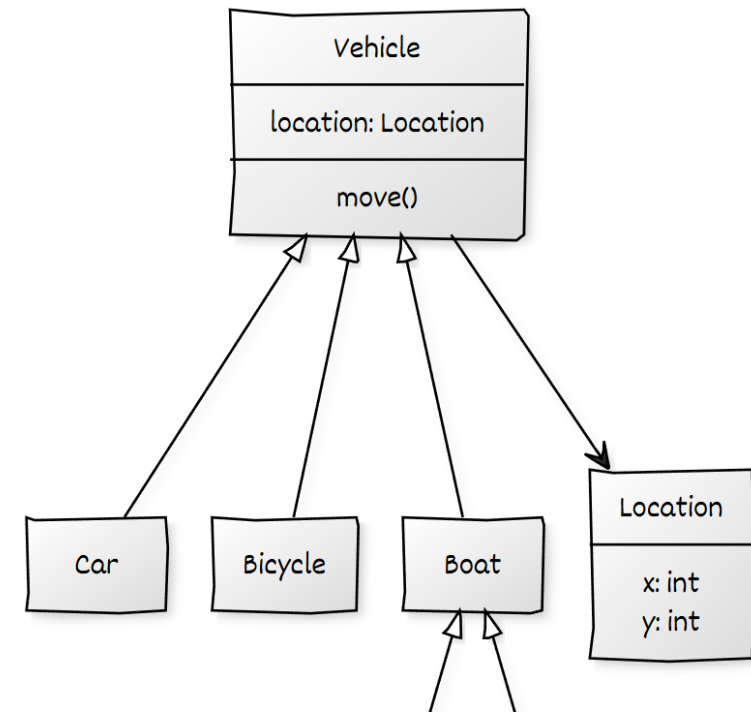
```python
class Vehicle:

    def __init__(self):
        self.location = Lo
        self.speed = 1
        self.direction = 0
```
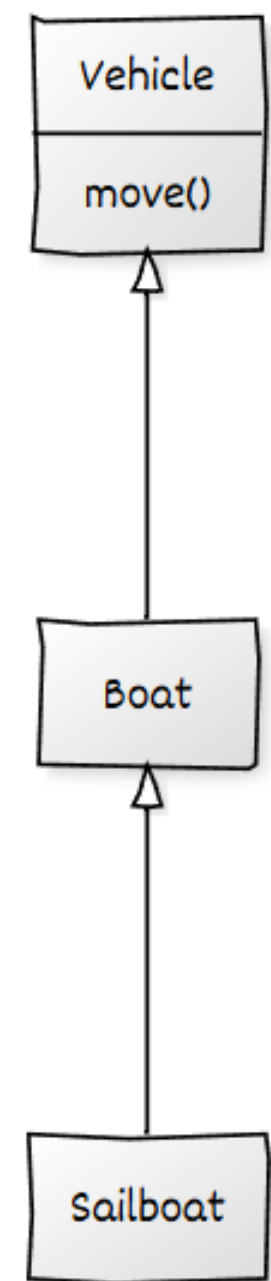
# Multi-level inheritance

```python
class Vehicle:
    def move(self, distance):
        print(f"Vehicle moved {distance} meters")


class Boat(Vehicle):
    pass


class Sailboat(Boat):
    pass


mySailboat = Sailboat()
mySailboat.move(100)
# Output: Vehicle moved 100 meters
```



CREATED WITH YUML

```python
class Party:

    def __init__(self, name, strategy):
        self.name = name
        self.strategy = strategy
        self.voters = []
        self.location = Location()

    def update(self):
        self.strategy.update(self)
```