Object-Oriented Programming

Object-Oriented Programming (OOP)

- OOP is a programming technique organized based on using objects to design and develop applications.
- OOP combines data and computation for processing the data into encapsulated objects.
- In Object Oriented Programming we are trying to model either real world entities or processes and represent them in software.

Why we build software models?

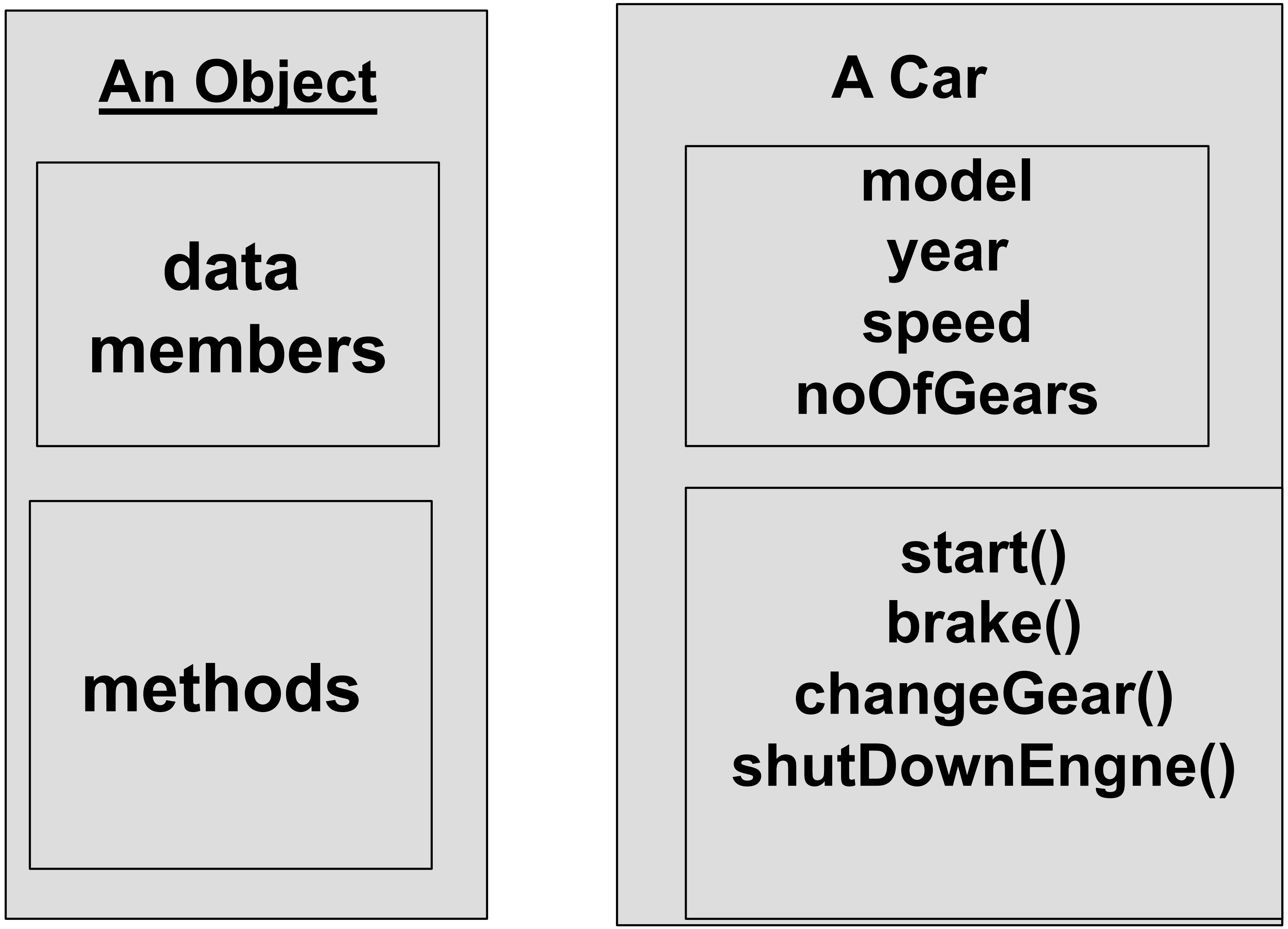
- A model is a simplification of reality. We model because we cannot comprehend the complexity of a system in its entirety.
- We model to visualize, specify, construct, and document the structure and behavior of a system's architecture.
- A model is a complete description of a system from a particular perspective.

Principles of Modeling

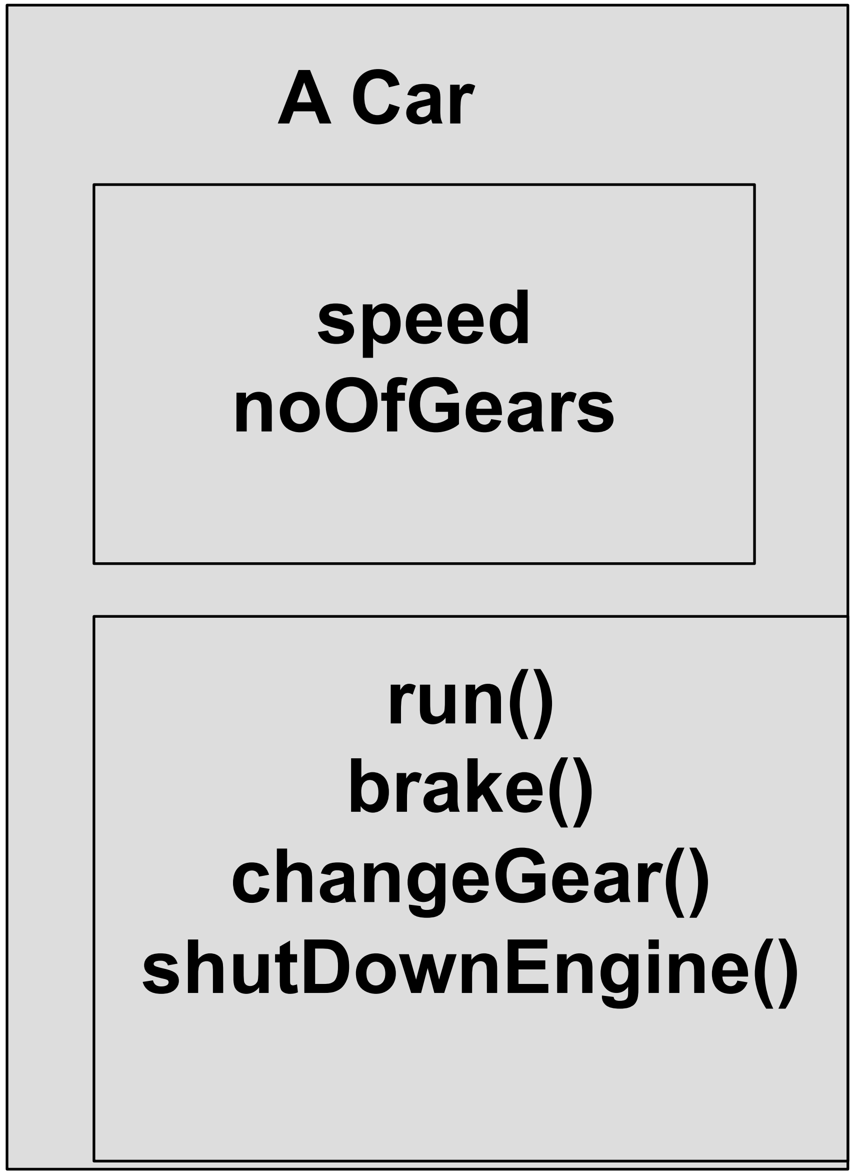
The model that we create is dependent on the problem that we are trying to solve and the entities in the scope of the problem.

- The **choice of what models** to create has a profound influence on how the problem is attacked and how a solution is shaped.
- Every model maybe expressed at **different levels of precision.**
- The best models are **connected to reality**.
- **No single model is sufficient.** Every non-trivial system is best approached through a small set of nearly independent models.

Classes and Objects



A Class and an object of it

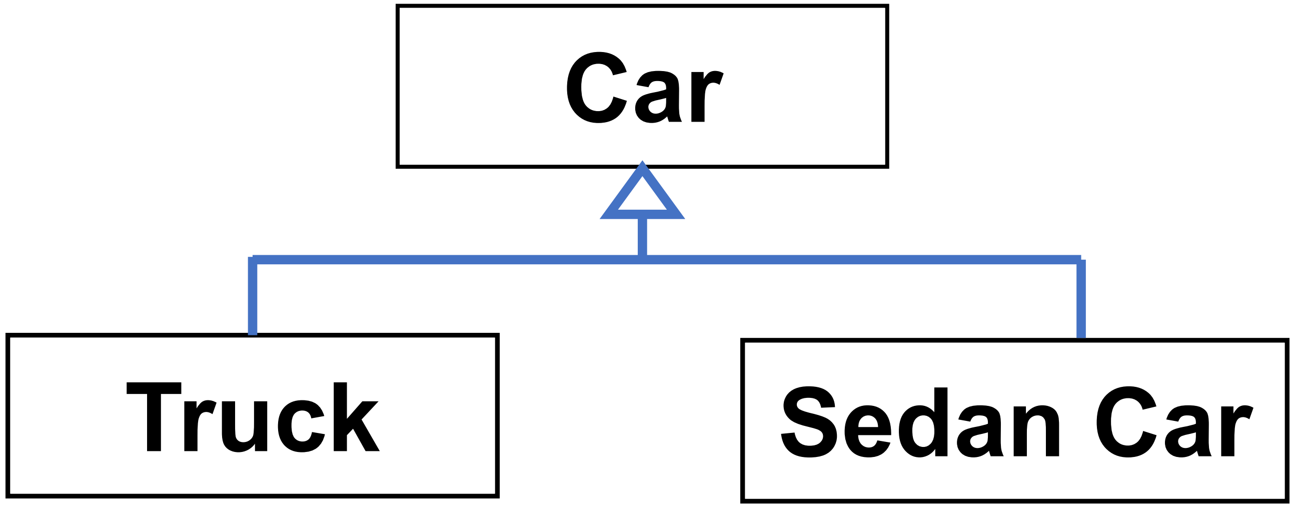


```python
class Car:
    '''This class defines a basic car'''
    def __init__(self, cspeed, ngears):
        self.speed = cspeed
        self.gears = ngears

    def run(self, speed):
        self.speed = speed
        print("Running with speed ", self.speed)

    def full_brake(self):
        while(self.speed > 0):
            self.speed -=1
            print("Braking the car", self.speed)
```

Object-Oriented Features

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction

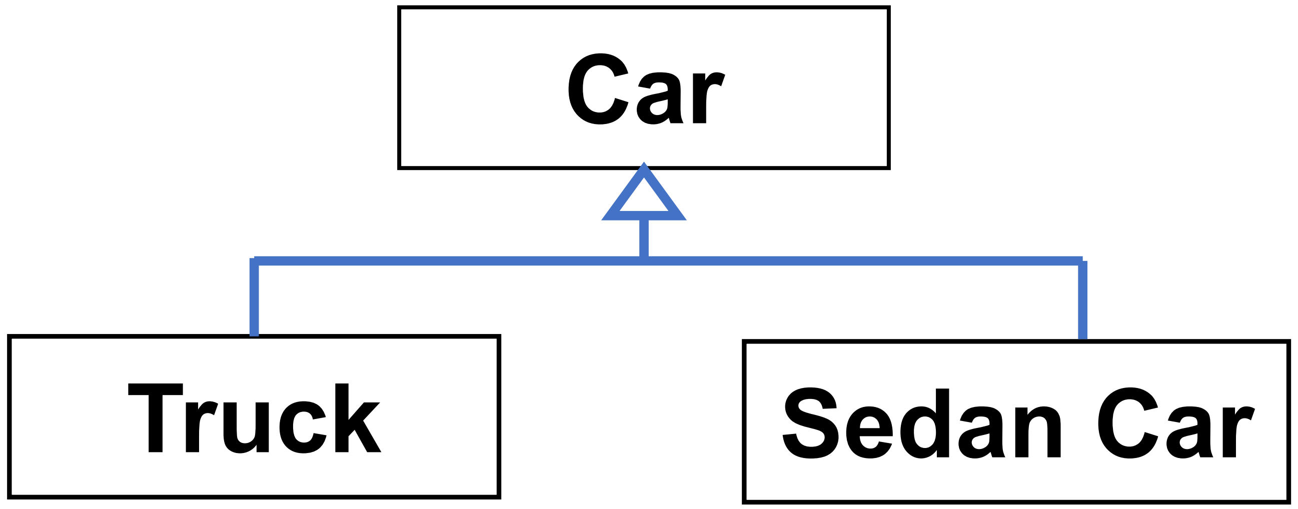
Inheritance



Organization of abstractions according to some order (e.g. complexity, responsibility, etc.).

Is-A-Type-Of Relation

Inheritance

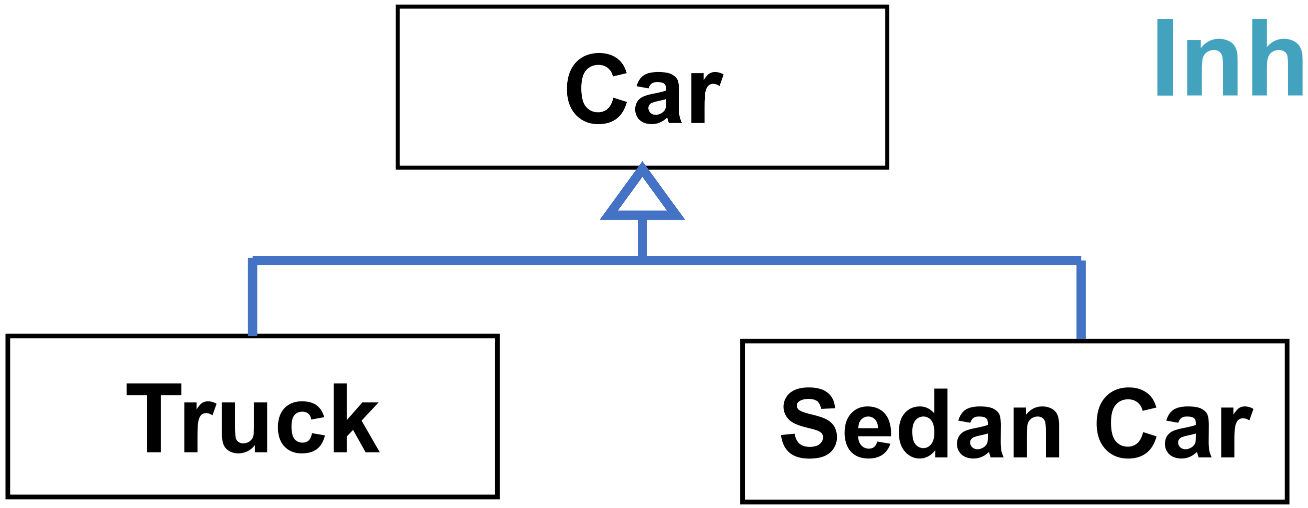


- Is-A-Type-Of Relation

A Truck is type of a Car

A Sedan is a type of a Car

Inheritance



```
class Car:
    def __init(self, cspeed):
        self.speed=cspeed

class Truck(Car):
    pass

class SedanCar(Car):
    pass
```

Polymorphism

- Closely related to Inheritance
- Substitute variables or objects of one type with variables or objects of another type.
- Polymorphism gives us the ability to switch components without loss of functionality.
- Polymorphism is when two or more objects have the same method name, but with different implementations.

```
m_toyota = Sedan_Car(0, 4);
m_truck = Truck(0, 4);

m_toyota.full_brake()
m_truck.full_brake()
```

Encapsulation

- Goal is to bind the data with the computation that manipulates it.
- Restrict the access to Object’s data from external interference.
- We can control and check the input values

Encapsulation

```
class Car():
    def __init__(self):
        self.__build = 0  # private attribute member

    def set_year(self, year):

        if(year > 1885 and year < 2021):
            self.__build = year
        else:
            print("Year must be between 1885 and 2021")
            self.__build = 0


    @property
    def year(self):
        return self.__build
```

Abstraction

- Hiding the implementation complexity

- Offering computation services over Application Programing Interfaces (API)

Abstraction - Example

```
m_toyota = Sedan_Car(0, 4);
m_truck = Truck(0, 4);

m_car_shop = Car_Shop(4);

# Now we bring our cars to the shop.
m_car_shop.repair_car(m_toyota)
m_car_shop.repair_car(m_truck)
```

Object-Oriented Features

- Inheritance
- Polymorphism
- Encapsulation
- Abstraction