OOP

Bank Account

Bank Account Deposit

- Download the Back Account File
- Add a function `deposit()` to deposit some money into your account

- Sample Usage:

```
>>> myAcc = BankAccount('Alan',1000)
>>> myAcc.showBalance()
Your balance is $1000
>>> myAcc.deposit(200)
>>> myAcc.deposit(400)
>>> myAcc.showBalance()
Your balance is $1600
```

Bank Account Secure Withdraw

- Add a control measure when you withdraw
- You must provide your name when you withdraw and it must match your name in the account

```
>>> myAcc = BankAccount('Alan',1000)
>>> myAcc.withdraw('Mary',100)
You are not authorized for this account
>>> myAcc.withdraw('Alan',10000)
Money not enough! You do not have $10000
0
>>> myAcc.withdraw('Alan',100)
100
>>> myAcc.showBalance()
Your balance is $900
```

Bank Account Compute Interest

- Add an attribute for interest rate
    - And initialize it at the constructor
- Implement a function “oneYearHasPass()” such that you gain the interest in your balance:

```
>>> myAcc = BankAccount('Alan',1000,0.04)
>>> myAcc.showBalance()
Your balance is $1000
>>> myAcc.oneYearHasPass()
>>> myAcc.showBalance()
Your balance is $1040.0
>>> myAcc.oneYearHasPass()
>>> myAcc.showBalance()
Your balance is $1081.60
```

Overall Solution

```
class BankAccount():
    def __init__(self,name,balance,interestrate):
        self.name = name
        self.balance = balance
        self.ir = interestrate
    def withdraw(self,name,amount):
        if name != self.name:
            print("You are not authorized for this account")
            return
        if self.balance < amount:
            print(f"Money not enough! You do not have ${amount}")
            return 0
        else:
            self.balance -= amount
            return amount
    def deposit(self,amount):
        self.balance += amount
    def oneYearHasPass(self):
        self.balance *= 1 + self.ir
    def showBalance(self):
        print(f'Your balance is ${self.balance}')
```

Minimal Account

- Define a new class of bank account called `MinimalAccount`
- This class will be the same as the normal `BankAccount`, except
  - If one year has pass, and your account is less than \$1000, \$20 dollars of administration fee will be deducted from your account.
    - Unless the balance will be less than zero, then reset to zero
  - The fee will be deducted BEFORE the calculation of interest

Sample Run

```
>>> mySonAcc = MinimalAccount('John',40,0.04)
>>> mySonAcc.oneYearHasPass()
>>> mySonAcc.showBalance()
Your balance is $20.8
>>> mySonAcc.oneYearHasPass()
>>> mySonAcc.showBalance()
Your balance is $0.8320000000000007
>>> mySonAcc.oneYearHasPass()
>>> mySonAcc.showBalance()
Your balance is $0.0
```

Minimal Account

- Define a new class of bank account called `MinimalAccount`
- This class will be the same as the normal `BankAccount`, except
  - If one year has pass, and your account is less than $1000, $20 dollars of administration fee will be deducted from your account.
    - Unless the balance will be less than zero, then reset to zero
  - The fee will be deducted BEFORE the calculation of interest
- Discuss with your neighbor, how will you design this class?
  - **Direct modification** to `BankAccount`? Or
  - **Duplicate** `BankAccount` and modify it? Or…
  - What else?

Home Challenge

- method `TransferTo()` in class `BankAccount`
  - Given another account, you can transfer your money to another, e.g.

```
>>> myAcc.transferTo(myWifeAcc,500)
```

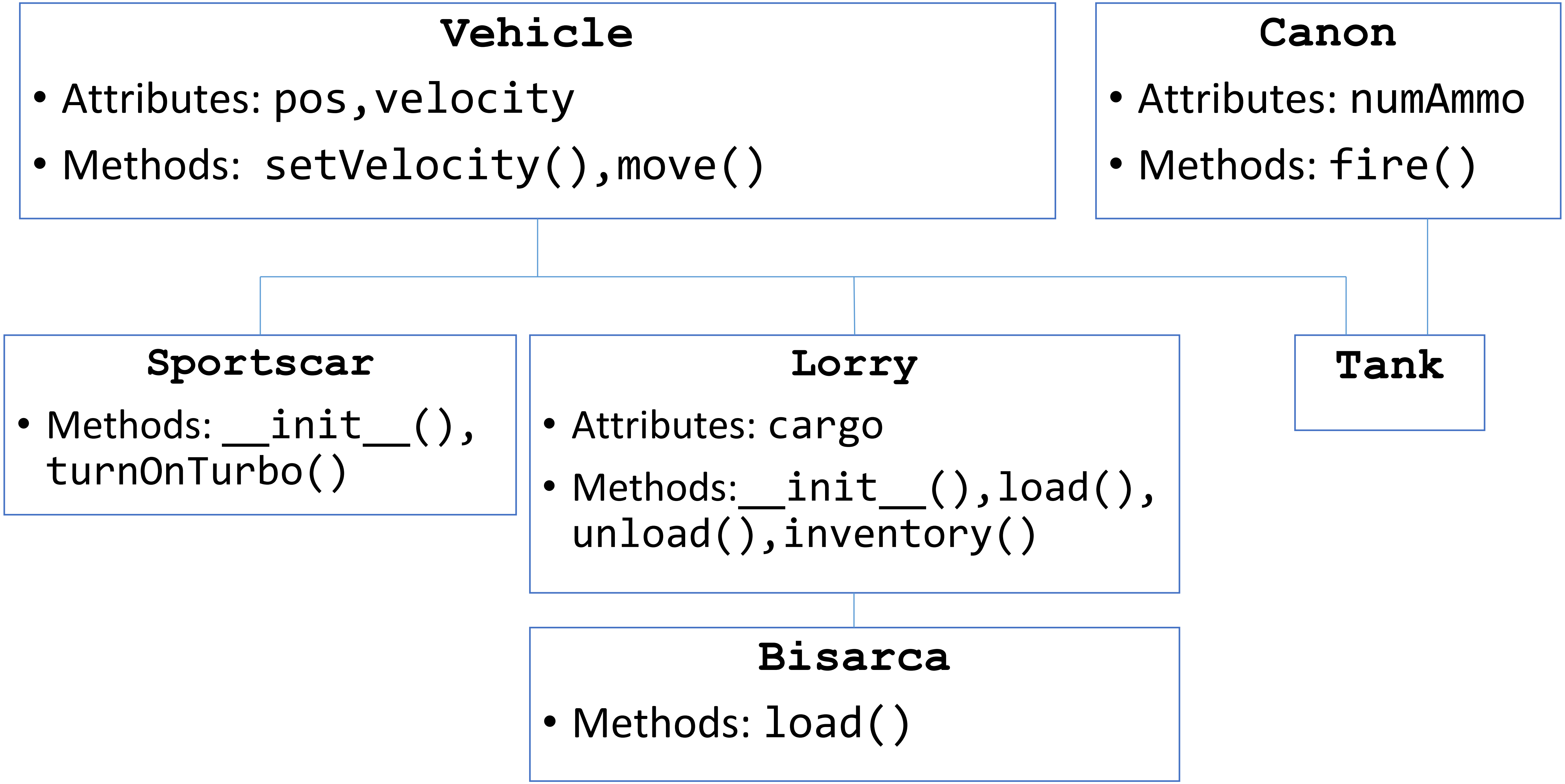
- method `setupGiro()` in class `BankAccount`
  - Money will be deducted every year before interest

```
>>> myAcc = BankAccount('Alan',1100,0.04)
>>> myAcc.setupGiro(40)
>>> myAcc.setupGiro(60)
>>> myAcc.oneYearHasPass()
>>> myAcc.showBalance()
Your balance is $1040
```

- A new class `JointAccount`
  - An account has two names, anyone of them can withdraw

Vehicles

Recap: Lecture



Vehicle
- Attributes: `pos`,`velocity`
- Methods: `setVelocity()`,`move()`

Canon
- Attributes: `numAmmo`
- Methods: `fire()`

Sportscar
- Methods: `__init__()`, `turnOnTurbo()`

Lorry
- Attributes: `cargo`
- Methods: `__init__()`,`load()`, `unload()`,`inventory()`

Tank

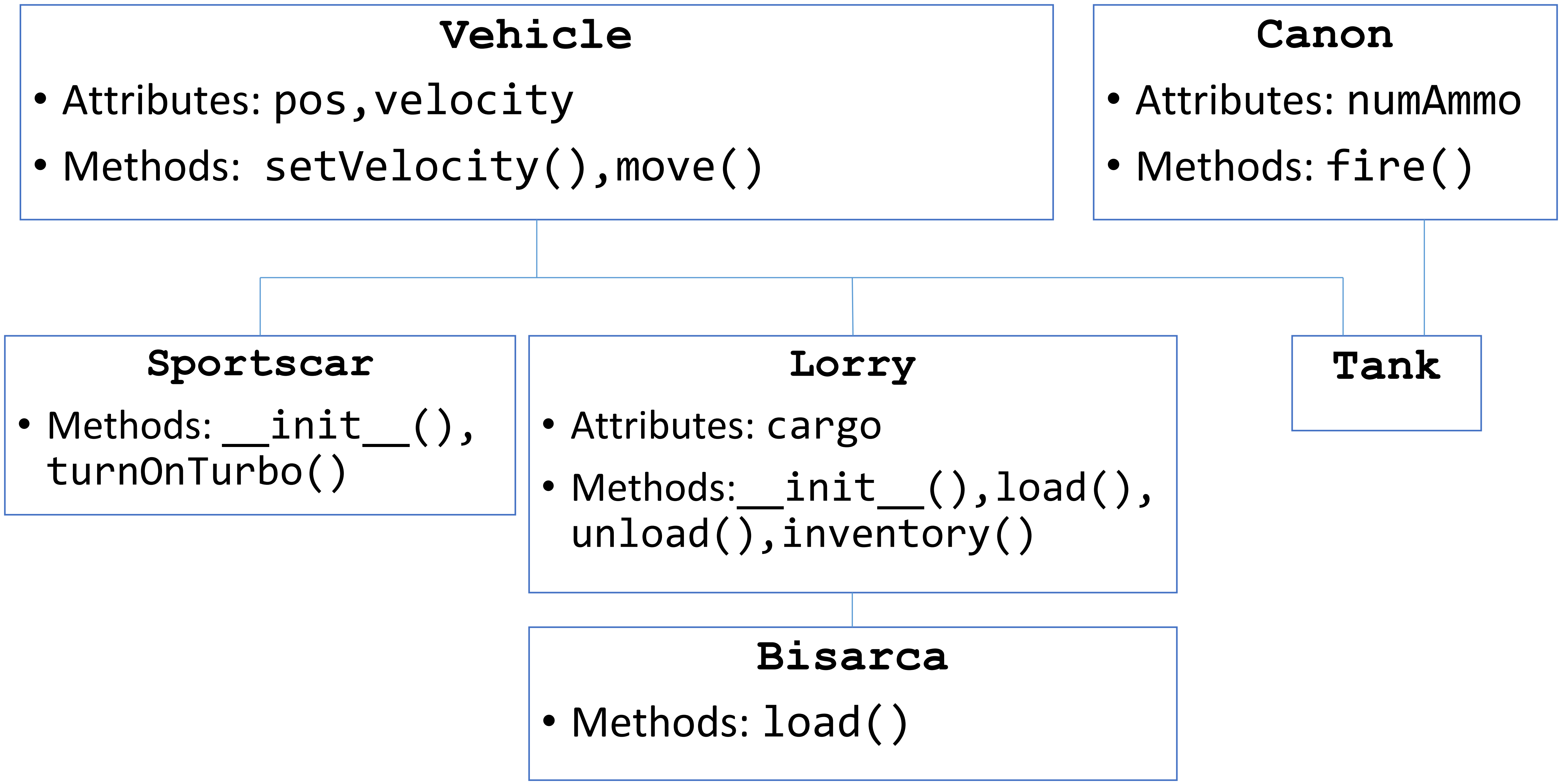
Bisarca
- Methods: `load()`

More Realistic

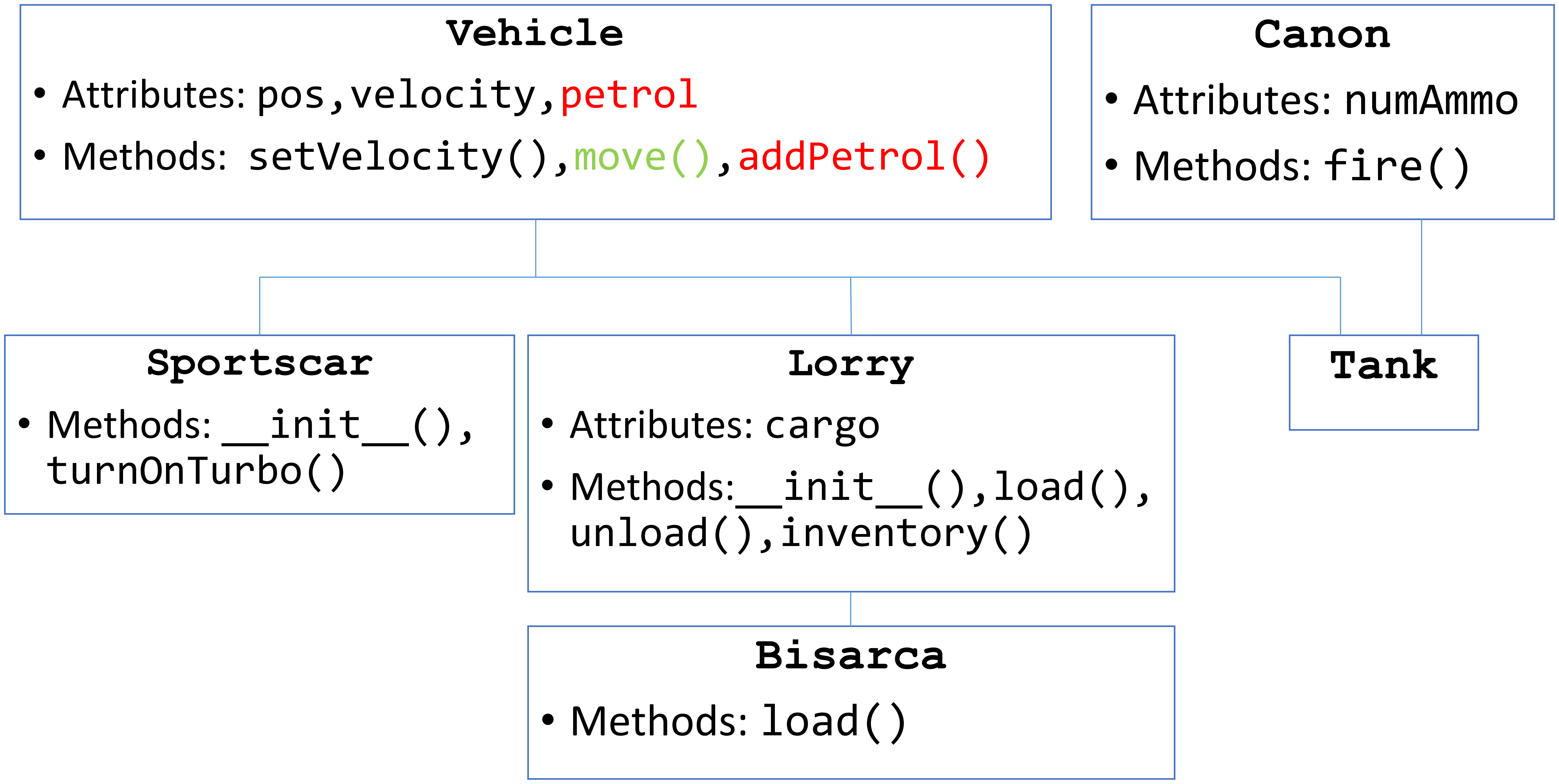
- Let’s try to be more realistic
- Every vehicle need some petrol
    - Sportscar, Lorry, etc.
- A new method called `addPetrol(n)` will add n liters of petrol into a vehicle
- And for every “move”, the vehicle will use 1 liter of petrol
- What attribute do you need to add? And where?

```
>>> myCar.addPetrol(2)
>>> myCar.move()
Move to (0, 80)
>>> myCar.move()
Move to (0, 160)
>>> myCar.move()
Out of petrol. Cannot Move.
>>> myCar.addPetrol(1)
>>> myCar.move()
Move to (0, 240)
>>> myCar.move()
Out of petrol. Cannot Move.
```

Add where?


Vehicle
• Attributes: pos,velocity
• Methods: setVelocity(),move()
Canon
• Attributes: numAmmo
• Methods: fire()
Sportscar
• Methods: __init__(), turnOnTurbo()
Lorry
• Attributes: cargo
• Methods:__init__(),load(), unload(),inventory()
Tank
Bisarca
• Methods: load()

Add Red and Modify Green



Vehicle
- Attributes: pos,velocity,petrol
- Methods: setVelocity(),move(),addPetrol()

Canon
- Attributes: numAmmo
- Methods: fire()

Sportscar
- Methods: __init__(), turnOnTurbo()

Lorry
- Attributes: cargo
- Methods:__init__(),load(), unload(),inventory()

Tank

Bisarca
- Methods: load()

Try To Implement the Petrol Feature

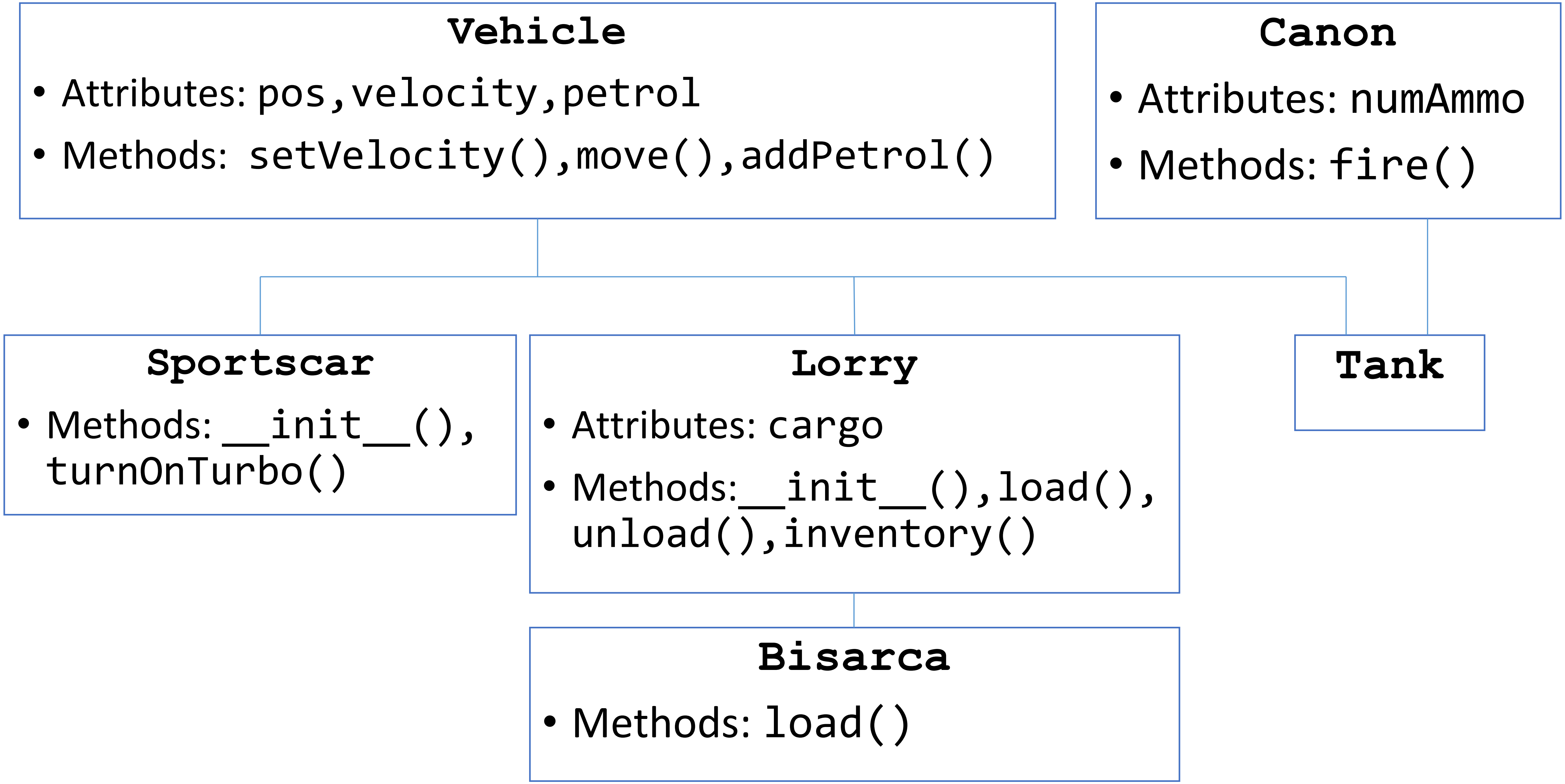
Vehicle That Needs Petrol

```
class Vehicle:
    def __init__(self,pos):
        self.petrol = 0
        self.pos = pos
        self.velocity = (0,0)
    def addPetrol(self,l):
        self.petrol += l
    def setVelocity(self,vx,vy):
        self.velocity = (vx,vy)
    def move(self):
        if self.petrol == 0:
            print("Out of petrol. Cannot Move.")
            return
        self.petrol -= 1
        self.pos = (self.pos[0]+self.velocity[0],self.pos[1]+self.velocity[1])
        print(f"Move to {self.pos}")
```

Design Issue

- How about a Tank that can survive on solar power?
  - Don’t need petrol

How to Design a Solar Tank?



Vehicle

- Attributes: `pos,velocity,petrol`
- Methods: `setVelocity(),move(),addPetrol()`

Canon

- Attributes: `numAmmo`
- Methods: `fire()`

Sportscar

- Methods: `__init__(), turnOnTurbo()`

Lorry

- Attributes: `cargo`
- Methods: `__init__(),load(), unload(),inventory()`

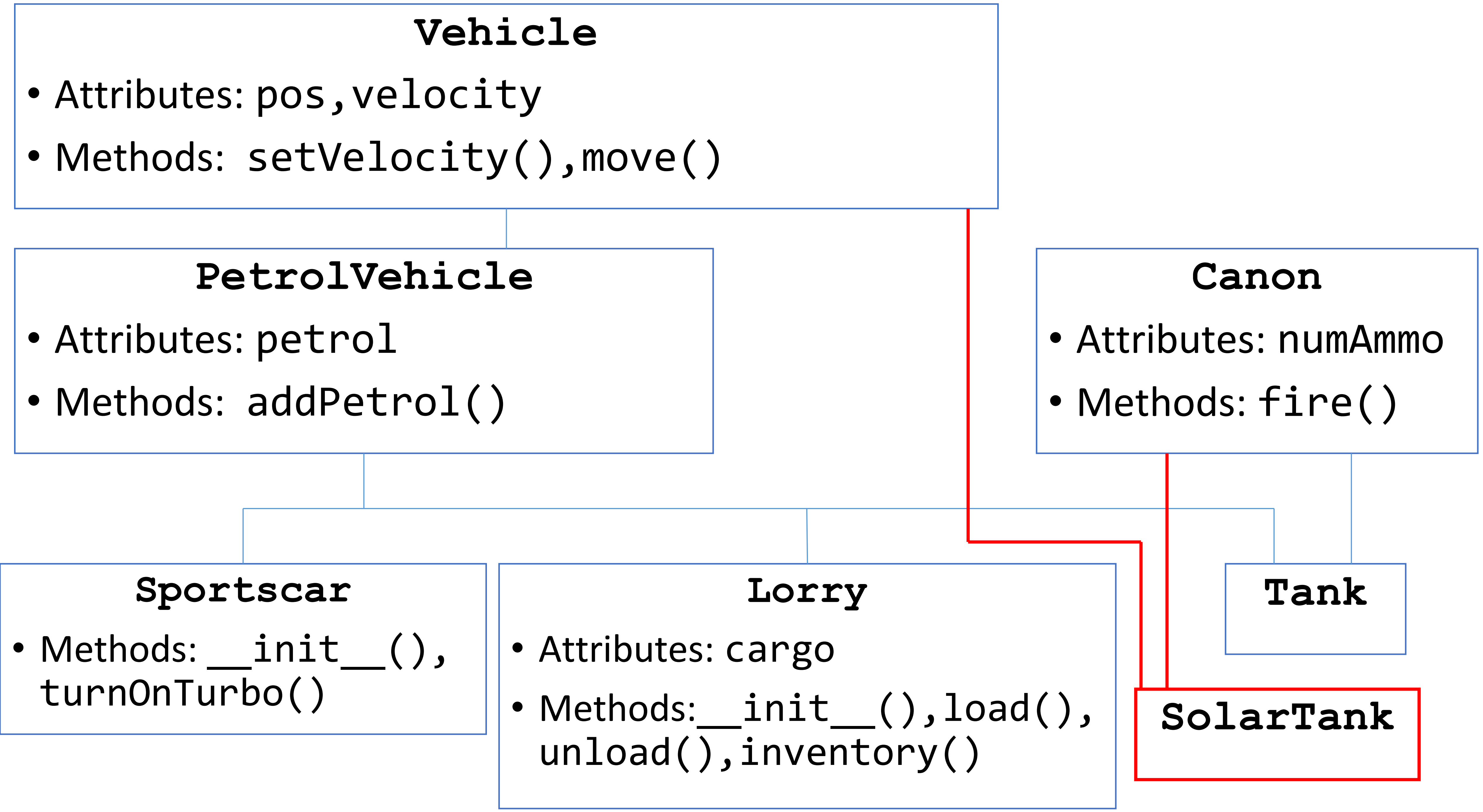
Tank

Bisarca

- Methods: `load()`

Solution?

- Separate the current “petrol” vehicle into
    - A superclass Vehicle and a Subclass PetrolVehicle
    - Then the solar tank will be a subclass of both Vehicle and Cannon

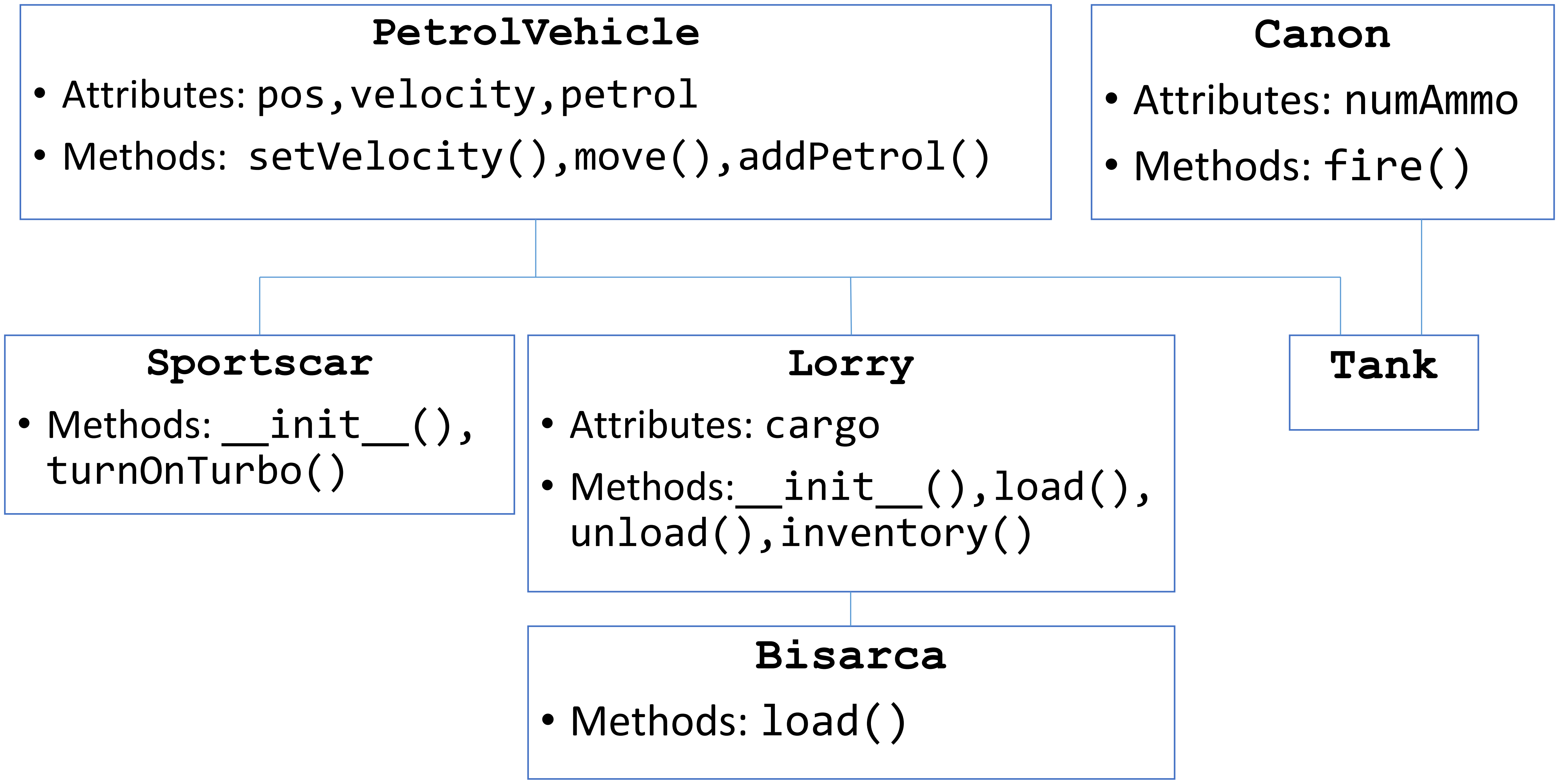

Vehicle
• Attributes: pos,velocity
• Methods: setVelocity(),move()
PetrolVehicle
• Attributes: petrol
• Methods: addPetrol()
Canon
• Attributes: numAmmo
• Methods: fire()
Sportscar
• Methods: __init__(), turnOnTurbo()
Lorry
• Attributes: cargo
• Methods: __init__(),load(), unload(),inventory()
Tank
SolarTank

Try To Implement the SolarTank after PetrolVehicle

Solution?

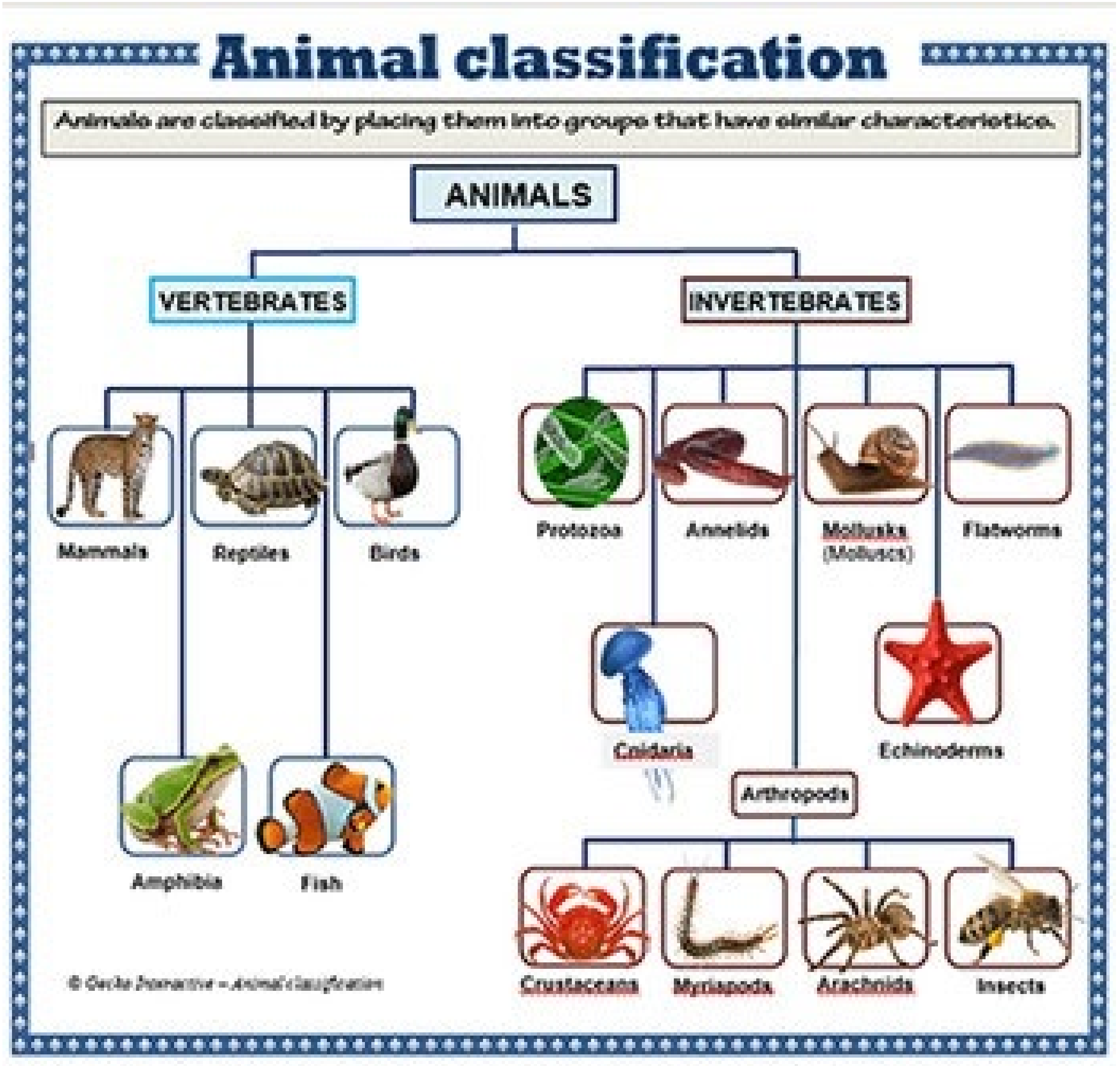
- Separate the current “petrol” vehicle into
  - A superclass Vehicle and a Subclass PetrolVehicle
  - Then the solar tank will be a subclass of both Vehicle and Cannon
- **[CHALLENGE]** Get into Trouble with
  - SolarBattleBisarca
  - You are forced to re-implement a SolarBisarca first? or…?

You want the “load()” in Bisarca but don’t want petrol


PetrolVehicle
• Attributes: pos,velocity,petrol
• Methods: setVelocity(),move(),addPetrol()
Canon
• Attributes: numAmmo
• Methods: fire()
Sportscar
• Methods: __init__(), turnOnTurbo()
Lorry
• Attributes: cargo
• Methods: __init__(),load(), unload(),inventory()
Tank
Bisarca
• Methods: load()

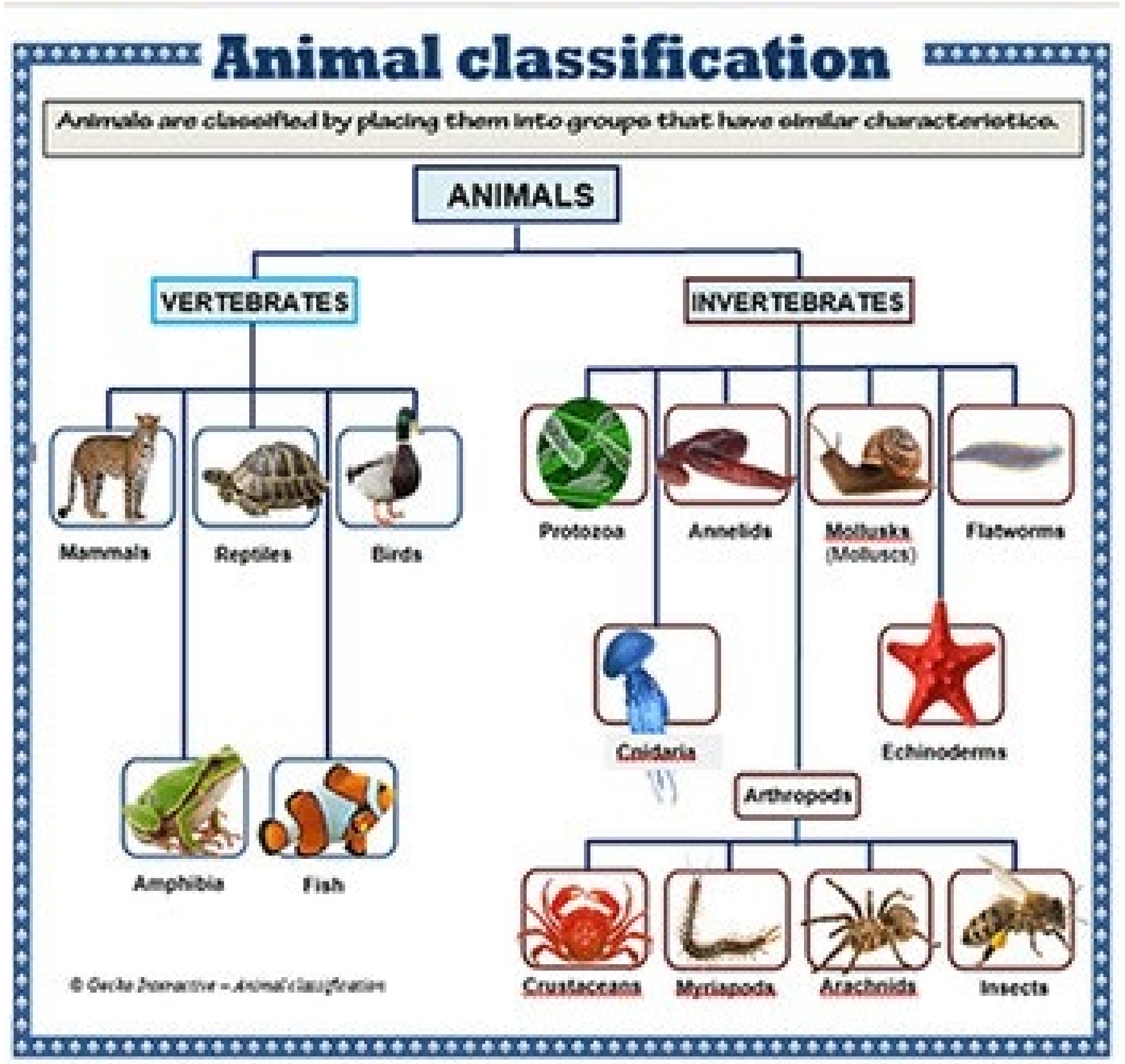
Design Issue

- If every class can be classified nicely, the world is beautiful
  - Every subclass is a subset of its superclass
  - Every subclass in the same level is distinct
  - Not like …



Design Issue

- Where will you fit platypus into the classification?


Animals are classified by placing them into groups that have similar characteristics.

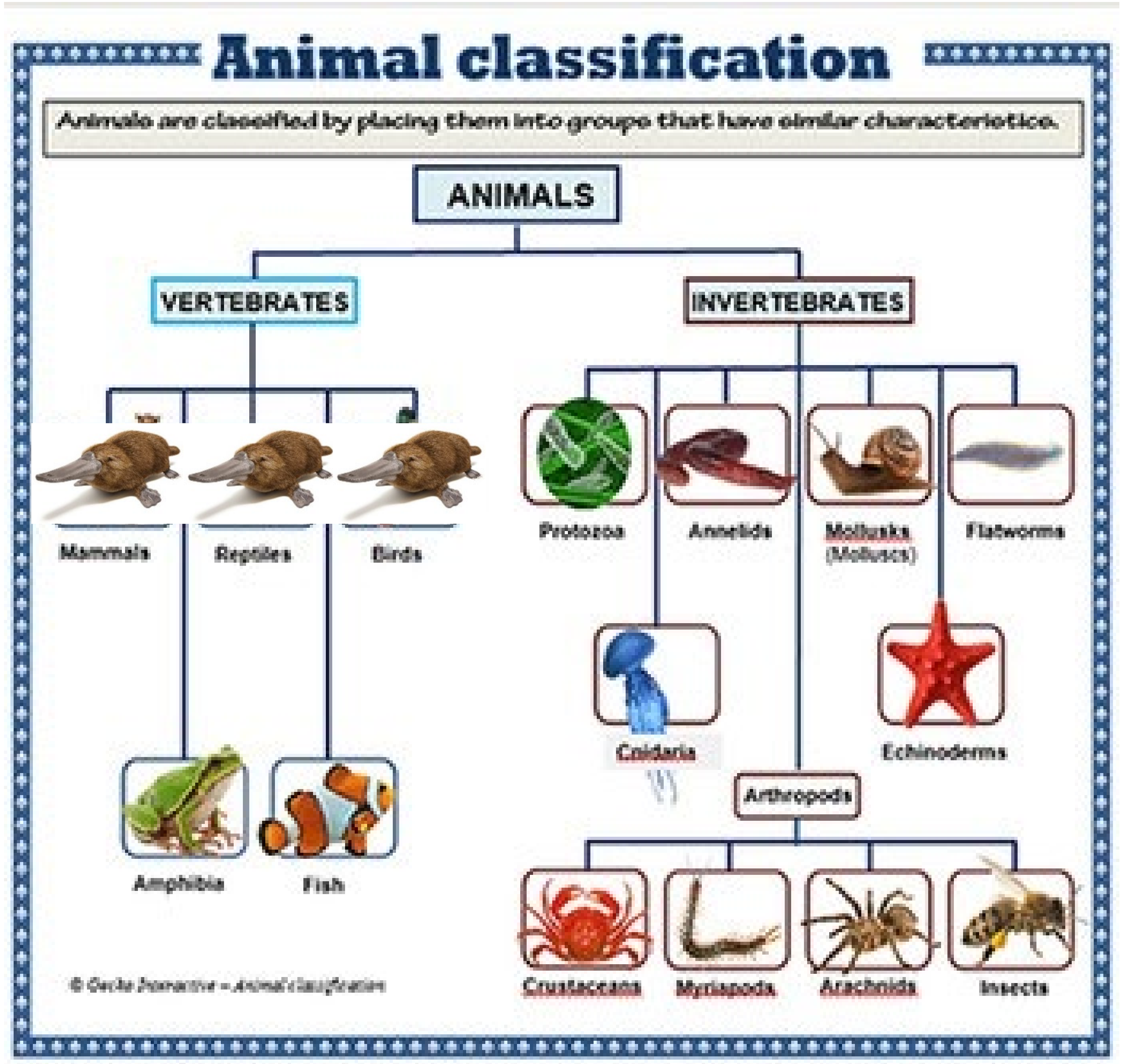
Where will you fit platypus ?

- Platypus
  - Got venom like reptiles
  - Lay eggs like birds
  - Milk like Mammals

REPTILES
BIRDS
MAMMALS
+
+
VENOM
EGGS
LACTATION

Design Issue

- Where will you fit platypus into the classification?


Animal classification
Animals are classified by placing them into groups that have similar characteristics.
ANIMALS
VERTEBRATES
INVERTEBRATES
Mammals
Reptiles
Birds
Amphibia
Fish
Protozoa
Annelids
Mollusks (Molluscs)
Flatworms
Cnidaria
Echinoderms
Arthropods
Crustaceans
Myriapods
Arachnids
Insects
© Gecko Interactive – Animal classification