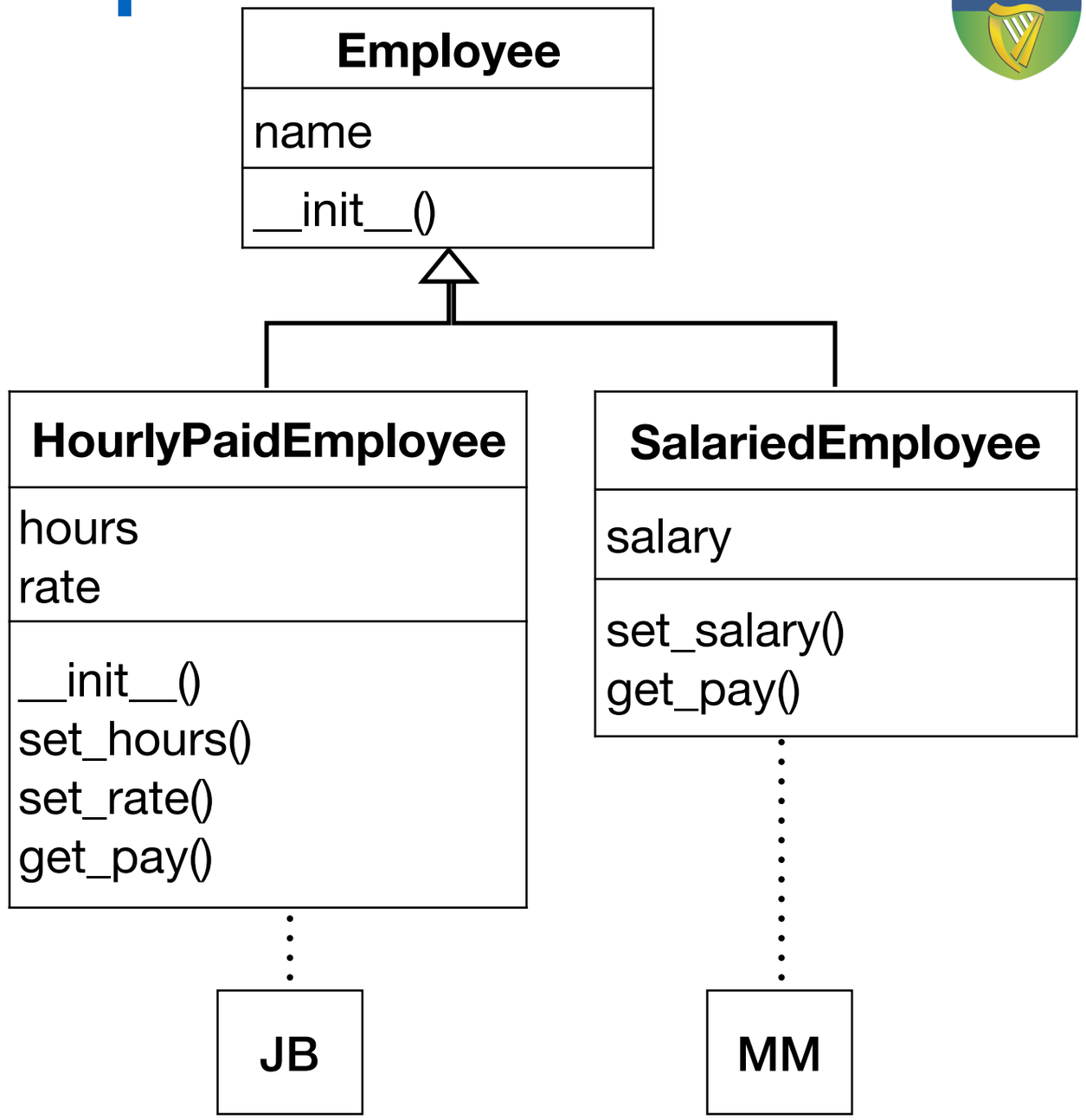
Classes, Instances & Methods

- Classes are categories
- Instances are individuals
- Variables
  - Class variables
    - Same value for all instances of a class
  - Instance variables
    - Each instance can have a different value
- Methods
  - class methods
  - instance methods

Remember the First Example

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

class HourlyPaidEmployee(Employee):

    def __init__(self, name):
        Employee.__init__(self, name)
        self.hours = 0
        self.rate =

    def set_hours(self, hours):
        self.hours = hours

    def set_rate(self, r):
        self.rate = r

    def get_pay(self):
        return self.rate * self.hours

class SalariedEmployee(Employee):

    def set_salary(self, sal):
        self.salary = sal

    def get_pay(self):
        return self.salary / 12
```



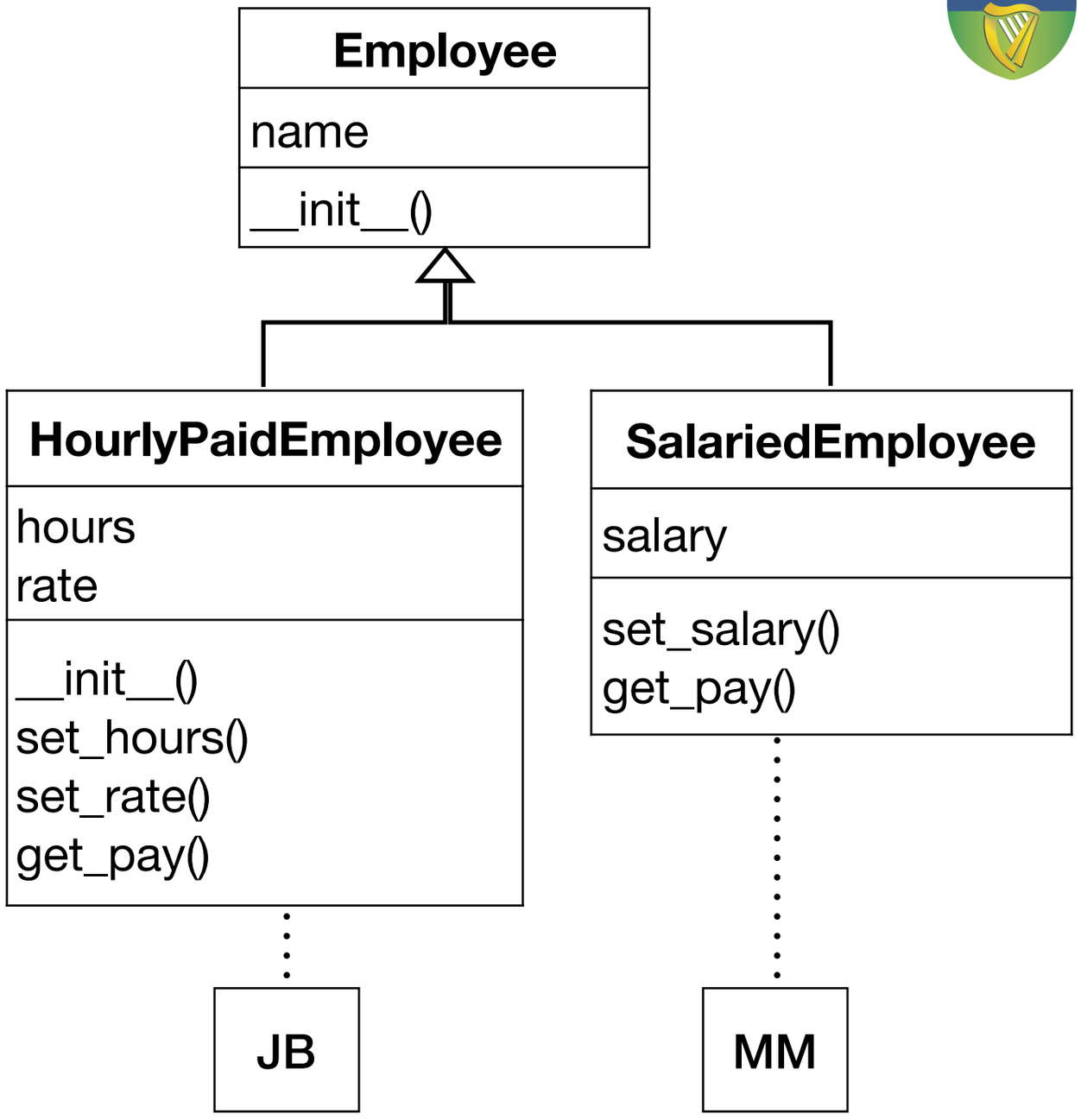

```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

Classes & Instances

- All relationships in the tree are ‘is-a’ relations

- 3 classes
  - HourlyPaidEmployee & SalariedEmployee **subclasses** of Employee

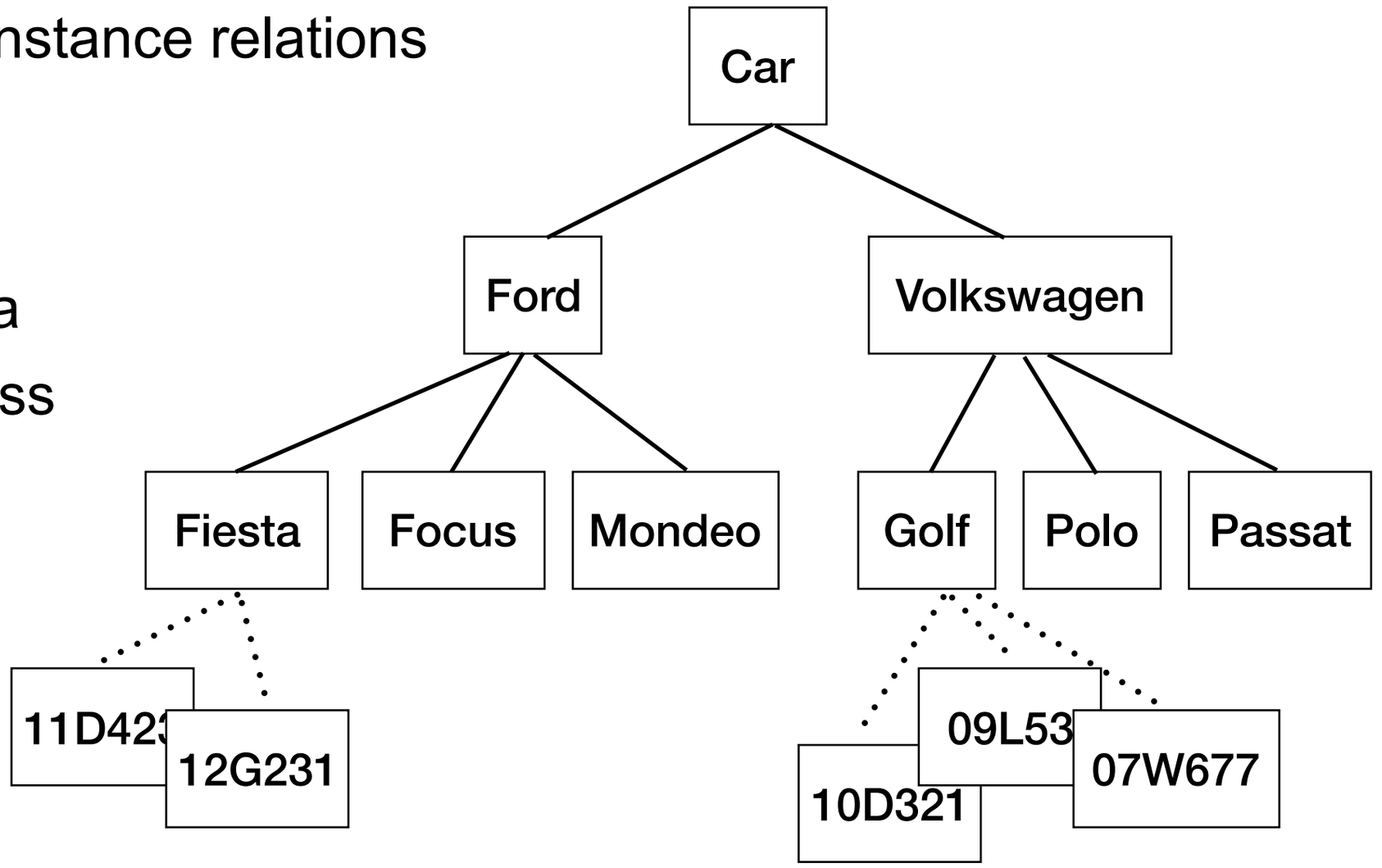
- 2 instances
  - JB **instance** of HourlyPaidEmployee
  - MM **instance** of SalariedEmployee




```
JB = HourlyPaidEmployee("Joe Bloggs")
MM = SalariedEmployee("Marvelous Mary")
JB.set_hours(121)
JB.set_rate(10.50)
MM.set_salary(45000)
```

Sub-Classes & Instances

- An tree of is-a relations can have
  - A few levels of sub-class relations
  - Only one level of instance relations
- Cars
  - Fiesta is-a Ford
  - 12G231 is-a Fiesta
  - Fiesta is a sub-class of Car
  - 12G231 is an instance of Fiesta



Class Variables & Instance Variables

- Class variables
  - true for all cars
- Instance variables
  - instance specific

Class variable

Instance variables

```
class Car(object):
    wheels = 4

    def __init__(self, make, reg):
        self.make = make
        self.reg = reg

nl = Car("Nissan Leaf","11D4324")
print ("My new car is a %s" % nl.make)
print ("The reg is", nl.reg)
print ("My car, like all cars, has %d wheels" % Car.wheels)
print ("My car has %d wheels" % nl.wheels)
```


```
My new car is a Nissan Leaf
The reg is 11D4324
My car, like all cars, has 4 wheels
My car has 4 wheels
```

Class Variables

- `nl.wheels = 3`
  - creates an instance variable wheels
  - overrides Class instance variable
  - Car.wheels remains unchanged
- `Car.wheels = 3`
  - would change the class variable

Exercise

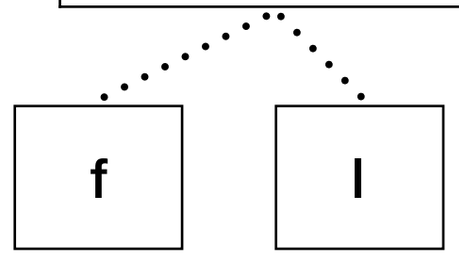
- Create a Class `Student` and sub-classes `FT_Student` and `PT Student`
  - Each should have an Class variable `credits`
    - For `FT_Student: credits = 60`
    - For `PT_Student: credits = 30`
  - Create instances
    - `FT_Student: Joe & Mary`
    - `PT_Student: Ann & Fred`
  - Change the value of `credits` for `PT_Student` to 20
- Does this change for `Ann & Fred`?

Class Variables

- species is a class variable
  - same value for all instances

```
class Dog:
    species = 'Canidae'

    def __init__(self, name):
        self.name = name
        self.tricks = []


    def add_trick(self, trick):
        self.tricks.append(trick)
In [2]:
f = Dog('Fido')
l = Dog('Lassie')
```

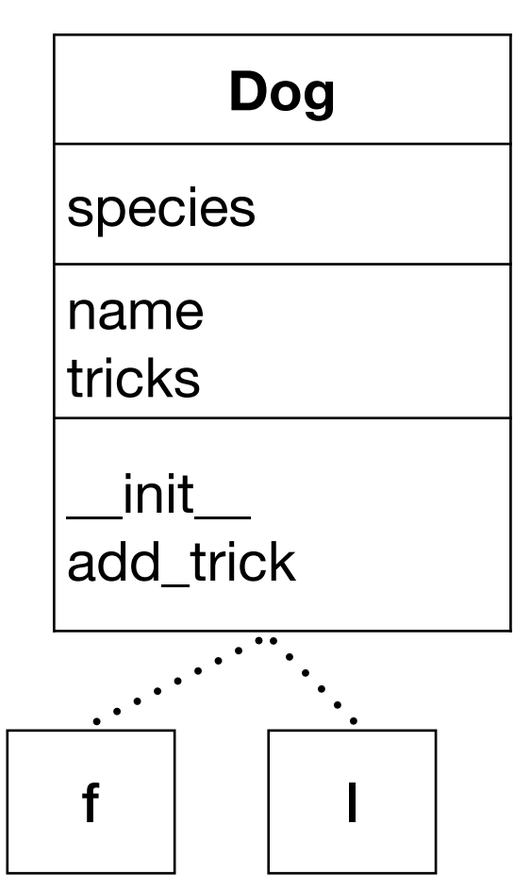
Dog
species
name tricks
__init__ add_trick



self a unique handle for each instance

Class & Instance Variables

- Class
  - species
- Instance
  - name
  - tricks




```
f = Dog('Fido')
l = Dog('Lassie')
In [3]:
f.name, l.name
Out[3]:
('Fido', 'Lassie')
In [4]:
f.add_trick("Play Dead")
f.add_trick('Fetch')
In [5]:
print(f.tricks)
print(l.tricks)

['Play Dead', 'Fetch']
[]
In [6]:
print(f.species)
print(l.species)

Canidae
Canidae
In [7]:
f.__dict__
Out[7]:
{'name': 'Fido', 'tricks': ['Play Dead', 'Fetch']}
```

Changing Class Variables

- The right way:

```
Dog.species = 'Dog-like'
In [24]:
l.species
Out[24]:
'Dog-like'
```

- The wrong way

Creates a new instance variable

```
l.__dict__
Out[26]:
{'name': 'Lassie', 'tricks': []}
In [27]:
l.species = 'All dogs'
In [28]:
l.__dict__
Out[28]:
{'name': 'Lassie', 'species': 'All dogs', 'tricks': []}
```

Methods

- Functions associated with a class
- Inherited by instances of a class
  - __init__ is a special constructor method
    - invoked when an instance is created
- Static methods
  - bound to the class rather than to instances

Class Methods

```
class Dog:
    species = 'Canidae'
    dog_count = 0
    dogs = []

    def __init__(self,name):
        self.name = name
        self.tricks = []
        Dog.dog_count += 1
        Dog.dogs.append(name)

    def rollcall(self):
        print("We have", Dog.dog_count,"dogs.")
        for name in Dog.dogs:
            print(name)

    def add_trick(self, trick):
        self.tricks.append(trick)
```

```
l = Dog("Lassie")
print("After", l.name,"dog_count = ",Dog.dog_count)
r = Dog("Rover")
print("After", r.name,"dog_count = ",Dog.dog_count)

After Lassie dog_count =  1
After Rover dog_count =  2
In [39]:
l.rollcall()

We have 2 dogs.
Lassie
Rover
```

This method should not belong to the instance

Class Methods

self - a handle for the instance

m1 inherited by all instances of c1

cls - a handle for the class

```
class c1:
    def m1(self):
        print("This is method m1 from instance ",self)
    @classmethod
    def m2(cls):
        print("This is method m2 from class ",cls)

i = c1()
i.m1()
c1.m2()

This is method m1 from instance  <__main__.c1 object at 0x10c568400>
This is method m2 from class  <class '__main__.c1'>

i,c1
Out[69]:
(<__main__.c1 at 0x10c568400>, __main__.c1)
```

Class Methods

```
class Dog:
    species = 'Canidae'
    dog_count = 0
    dogs = []

    def __init__(self,name):
        self.name = name
        self.tricks = []
        Dog.dog_count += 1
        Dog.dogs.append(name)

    @classcmethod
    def rollcall(cls):
        print("We have", cls.dog_count,"dogs.")
        for name in cls.dogs:
            print(name)

    def add_trick(self, trick):
        self.tricks.append(trick)
```

```
l = Dog("Sailor")
print("After", l.name, "dog_count = ",Dog.dog_count)
r = Dog("Captain")
print("After", r.name, "dog_count = ",Dog.dog_count)

After Sailor dog_count =  1
After Captain dog_count =  2
In [54]:
Dog.rollcall()

We have 2 dogs.
Sailor
Captain
```

Class method bound to class, not instance

Class Methods

```
l = Dog("Sailor")
print("After", l.name, "dog_count = ",Dog.dog_count)
r = Dog("Captain")
print("After", r.name, "dog_count = ",Dog.dog_count)

After Sailor dog_count =  1
After Captain dog_count =  2
In [61]:
Dog.rollcall()

We have 2 dogs.
Sailor
Captain
In [62]:
l.rollcall()

We have 2 dogs.
Sailor
Captain
```

- **`rollcall(), dogs[] & dog_count`** are class-level rather than instance-level

Exercise

- Create a new class called Cat with
  - Instance variables
    - `name`
    - `fav_food`
    - `staff`
  - Class variables
    - `disposition = 'Not as nice as dogs'`
    - `cat_count = 0`
  - Methods
    - `__init__`
    - `add_staff`
- Fix `add_staff` so that if won't add someone who is already staff
- Fix `__init__` so that it updates `cat_count` each time a cat is created

Classes, Instances & Methods

- Classes are categories
- Instances are individuals
- Variables
  - Class variables
    - Same value for all instances of a class
  - Instance variables
    - Each instance can have a different value
- Methods
  - class methods
  - instance methods