Object Oriented Programming

Classes and instances

Jolan Philippe

February 26th, 2024

IMT Atlantique

UML

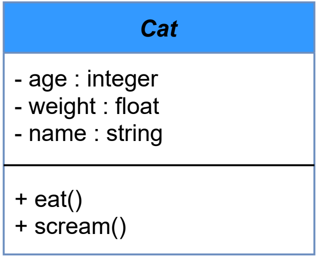
Representing objects with UML

The Unified Modeling Language (UML)

- Standard way to visualize a system
- Visual representation of objects

Let's represent an object for **cat**

- The class "Cat"
  - to represent all the cats
  - an instance: a cat
- Attributes
  - age (integer)
  - weight (float)
  - name (string)
- Functions
  - eat()
  - scream()



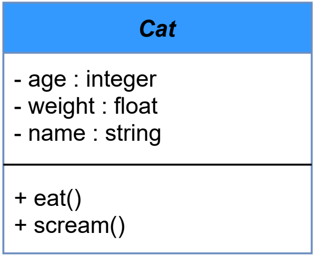
Cat

- age : integer
- weight : float
- name : string

+ eat()
+ scream()

More objects

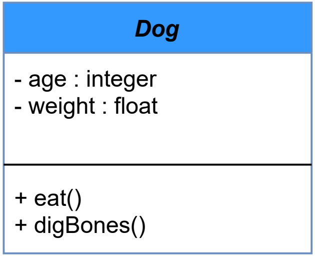
- The class “Cat”
  - to represent all the cats
- The class “Dog”
  - to represent all the dogs



Cat

- age : integer
- weight : float
- name : string

+ eat()
+ scream()

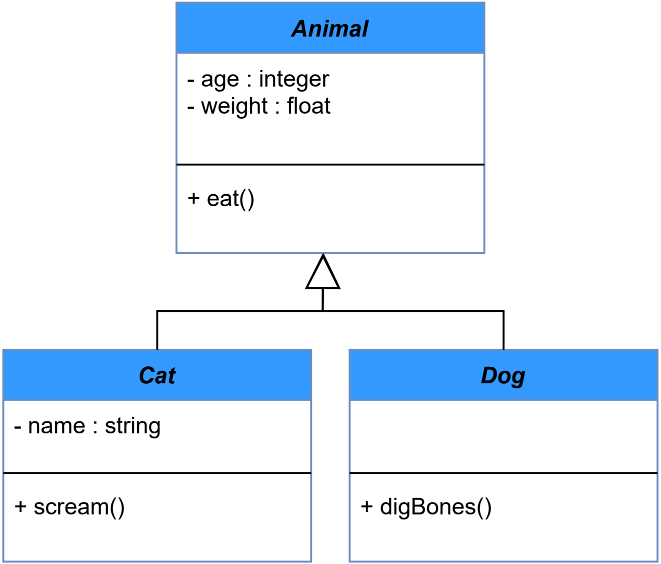


Dog

- age : integer
- weight : float

+ eat()
+ digBones()

Code reuse with inheritance



Cats and Dogs are Animals

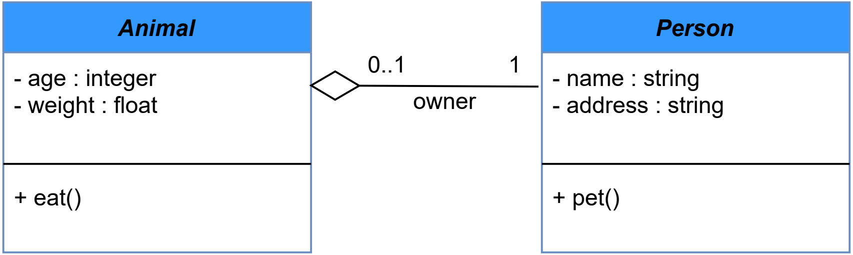
- A Cat is an Animal
- A Dog is an Animal
- Cat and Dog **inherit** from animals

⇒ A subclass inherits all attributes and functions from super class

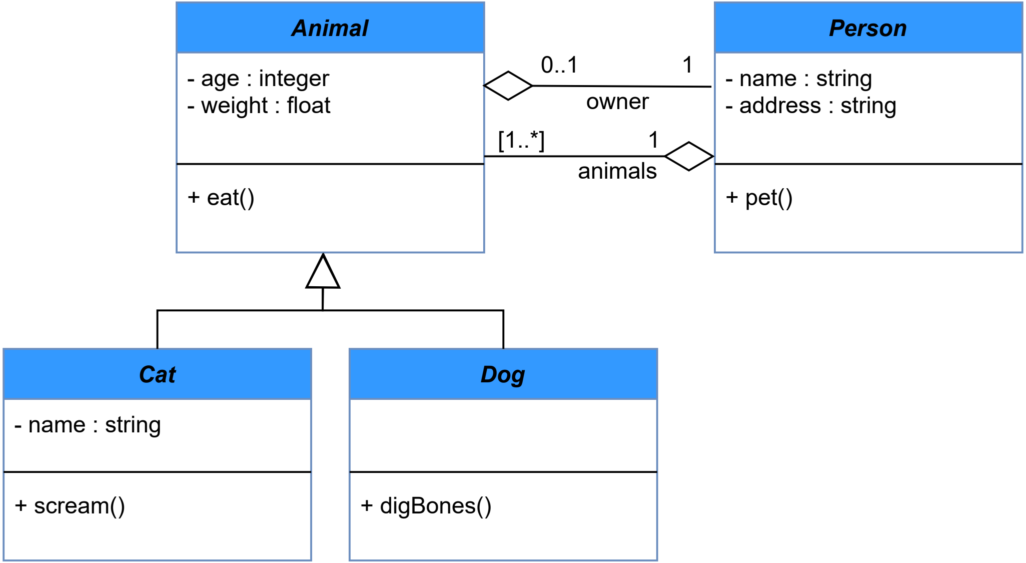
Composition and aggregation

All animal has an Owner

- The class “Animal” as a relation of **aggregation** with the class “Owner”



Full example



Exercise

Specification

There is two type of cars, A and B. The cars A have 4 wheels but the car B have 6 wheels. All the cars have an engine that can be or electric or thermal.

Write the UML class diagram corresponding to this example

Python

Using Python

Python files: “name.py”

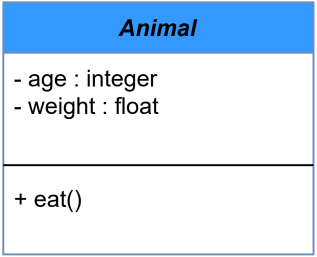
Integrated development environement

- Text editor + CLI
- Spyder: `https://www.spyder-ide.org/`
- PyCharm: `https://www.jetbrains.com/pycharm`

Code example:

```python
if __name__ == "__main__":
    print("Hello world")
```

Class definition



Animal

- age : integer
- weight : float

+ eat()

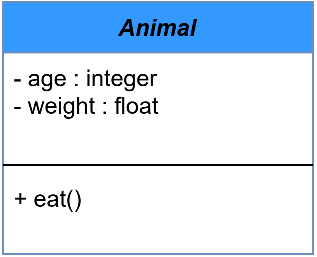
```python
class Animal:

    def __init__(self, ...):
        # Constructor
        ...

    def __str__(self):
        # String to print
        ...

    def eat():
        # Behavior
        ...
```

Class definition



Animal

- age : integer
- weight : float

+ eat()

```
class Animal:

    def __init__(self, age, weight):
        # Constructor
        self.age = age
        self.weight = weight

    def __str__(self):
        print(f"I am {self.age} years old, and weigh {self.weight} kg")

    def eat():
        print("Eat")
```

Class usage

```python
class Animal:
    ...

if __name__ == "__main__":
    my_animal = Animal(14, 8.0)
    your_animal = Animal(weight=4.5, age=3)
    print(my_animal.age)
    my_animal.eat()
    print(my_animal)
```

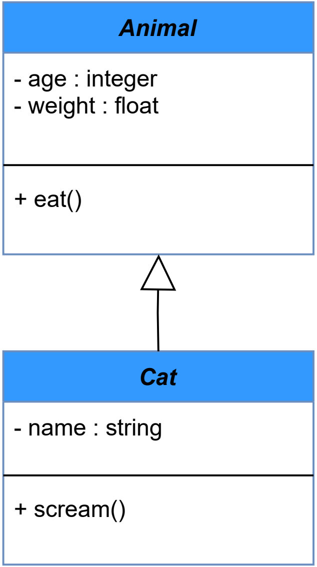
Output:

```
14

Eat

I am 14 years old, and weigh 4.5 kg
```

Class definition



Animal

- age : integer
- weight : float

+ eat()

Cat

- name : string

+ scream()

```python
class Animal:
    ...

class Cat(Animal):

    def __init__(self, name, age, weight):
        Animal.__init__(self, age, weight)
        self.name = name

    def __str__(self):
        print(f"I am {self.name} the cat")

    def scream():
        print("Grrrrr")
```

Class usage

```python
class Animal:
    ...

class Cat(Animal):
    ...

if __name__ == "__main__":
    lila = Cat("Lila", 14, 8.0)
    my_animal.eat() # From Animal
    my_animal.scream()  # From Cat
    print(lila)  # From ?
```

Exceptions

Managing errors

- rising exceptions to control unwanted behavior or definition
- management of exceptions when calling the function

```python
class Animal:

    def __init__(self, age):
        if (age >= 0):
            self.age = age
        else:
            raise Exception('Negative␣age␣are␣not␣allowed')

if __name__ == "__main__":
    try:
        Animal(-1)
    except Exception as e:
        print(e)
```

Static functions

We can define function for a given object or for the whole class

⇒ For an object

```python
def purr(self):
    print("ronron")
lila.purr() # To call
```

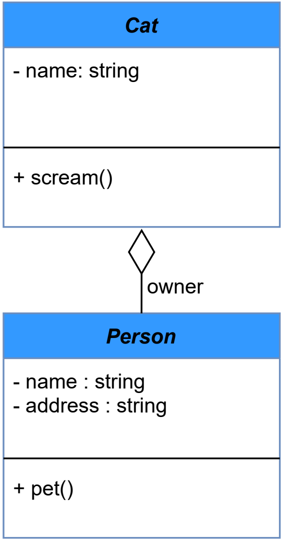
⇒ For a class

```python
def hugs(a1, a2):
    print(f"{a1.name}␣hugs␣{a2.name}")
Cat.hugs(lila ,garfield) # To call
```

Exercise

1. Create a class **Point** with attribute x and y corresponding to its coordinate. Write the code for the function __**str**__ and __**init**__. Test these methods.
2. Create a function **cartesianDistance** to compute the cartesian distance between two points. Reminder: cartesian distance of $p1 = (x_1, y_1)$ and $p2 = (x_2, y_2)$ is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)}$
3. Test your function over the two points (0, 5) and (-1, 9)
4. Create a sub-class of Point, named **City**. The cities have a name and a number of inhabitant. Write the code for the function __**str**__ and __**init**__.
5. Create **a function to add** a number of inhabitant in the city.

Aggregation



Cat

- name: string

+ scream()

owner

Person

- name : string
- address : string

+ pet()

```python
class Person:

    def __init__(self, name, address):
        self.name = name
        self.address

class Cat(Animal):

    def __init__(self, name, age, weight, person):
        Animal.__init__(self, age, weight)
        self.name = name
        self.owner = person
```

Exercise

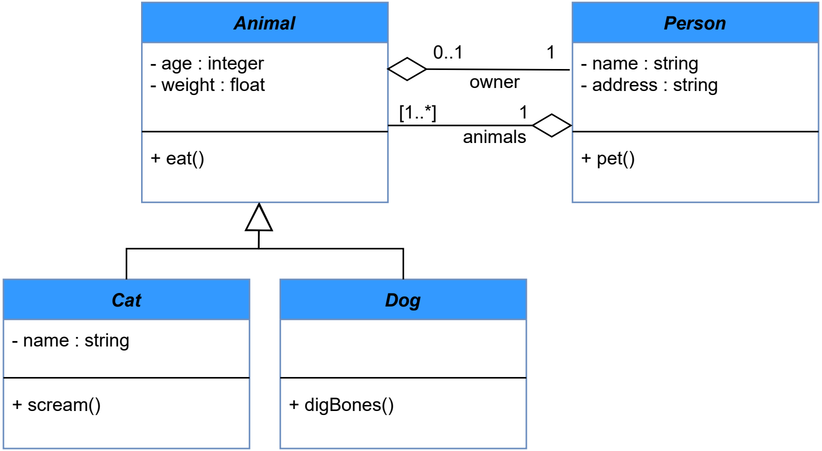
1. Create a class **City** which has a name and a number of inhabitant.
2. Create a class **Country**. This class has a capital city and has a list of cities.
3. Create two functions for Country: One to **add** a City, and one to **remove** a City.
4. Write a function that calculates the total **number of inhabitant** in a Country.
5. Add a static function **isACapitalCity** to the class City, that returns True if the City is the capital of a country, False otherwise.
6. Test all your functions.

How to use list in Python, and how to iterate on it.

```
my_list = []
my_list.append(a)
my_list.remove(a)
for element in my_list:
    print(element)
```

Exercise

Write the Python code of the following UML diagram.



Animal
- age : integer
- weight : float
+ eat()
0..1
1
owner
[1..*]
1
animals
Person
- name : string
- address : string
+ pet()
Cat
- name : string
+ scream()
Dog
+ digBones()