2. Objektorienterad programmering med Python

DA361A

Laboration 1

```python
# Dog, as a dictionary
dog = {
    "name": "Doug",
    "breed": "Pug",
    "age": 8,
    "colors": ["white", "black", "beige"]
}

def print_info(dog):
    """
    Prints out dog information
    """
    print "Woof! I'm %s the %s (%s years)." % (dog["name"], dog["breed"], dog["age"])

def print_fur_colors(dog):
    """
    Prints out all fur colors of the dog
    """
    print "%s has the following fur colors: %s" % (dog["name"], ", ".join(dog["colors"]))

```

```python
class Dog(object):
    """
    Represents a dog
    """

    def __init__(self, name, breed, age, colors):
        """
        Initialize class attributes
        """
        self.name = name
        self.breed = breed
        self.age = age
        self.colors = colors

# ex.
dog = Dog("Doug", "Pug", 8, ["white", "black", "beige"])
# => "dog" is an instance of the class "Dog"
```

python™

```python
class Dog(object):
    """
    Represents a dog
    """

    def __init__(self, name, breed, age, colors):
        """
        Initialize class attributes
        """
        self.name = name
        self.breed = breed
        self.age = age
        self.colors = colors

# ex.
dog = Dog("Doug", "Pug", 8, ["white", "black", "beige"])
# => "dog" is an instance of the class "Dog"
```

t.ex

```
print dog.name
# => “Doug”
```


```python
class Dog(object):

    def __init__(self, name, breed, age, colors):
        self.name = name
        self.breed = breed
        self.age = age
        self.colors = colors

    def print_fur_colors(self):
        """
        Prints out all fur colors of the dog
        """
        print "%s has the following fur colors: %s" % (self.name, ", ".join(self.colors))

    def __str__(self):
        """
        String representation of a dog
        """
        return "Woof! I'm %s the %s (%s years)." % (self.name, self.breed, self.age)

# ex
dog = Dog("Doug", "Pug", 8, ["white", "black", "beige"])

print dog # the method __str__ is called
# => Woof! I'm Doug the Pug (8 years).

dog.print_fur_colors()
# => Doug has the following fur colors: white, black, beige
```


```python
class Dog(object):

    def __init__(self, name, breed, age, colors):
        self.name = name
        self.breed = breed
        self.age = age
        self.colors = colors

    def print_fur_colors(self):
        """
        Prints out all fur colors of the dog
        """
        print "%s has the following fur colors: %s" % (self.name, ", ".join(self.colors))

    def __str__(self):
        """
        String representation of a dog
        """
        return "Woof! I'm %s the %s (%s years)." % (self.name, self.breed, self.age)

# ex
dog = Dog("Doug", "Pug", 8, ["white", "black", "beige"])

print dog # the method __str__ is called
# => Woof! I'm Doug the Pug (8 years).

dog.print_fur_colors()
# => Doug has the following fur colors: white, black, beige
```

```
# Dog 1
doug = Dog("Doug", "Pug", 8, ["white", "black", "beige"])
# Dog 2
watson = Dog("Watson", "Beagle", 12, ["brown", "white"])


class Dog(object):

    def __str__(doug):
        """
        String representation of a dog
        """
        return "Woof! I'm %s the %s (%s years)." % (doug.name,
                                                    doug.breed,
                                                    doug.age)


class Dog(object):

    def __str__(watson):
        """
        String representation of a dog
        """
        return "Woof! I'm %s the %s (%s years)." % (watson.name,
                                                    watson.breed,
                                                    watson.age)

```

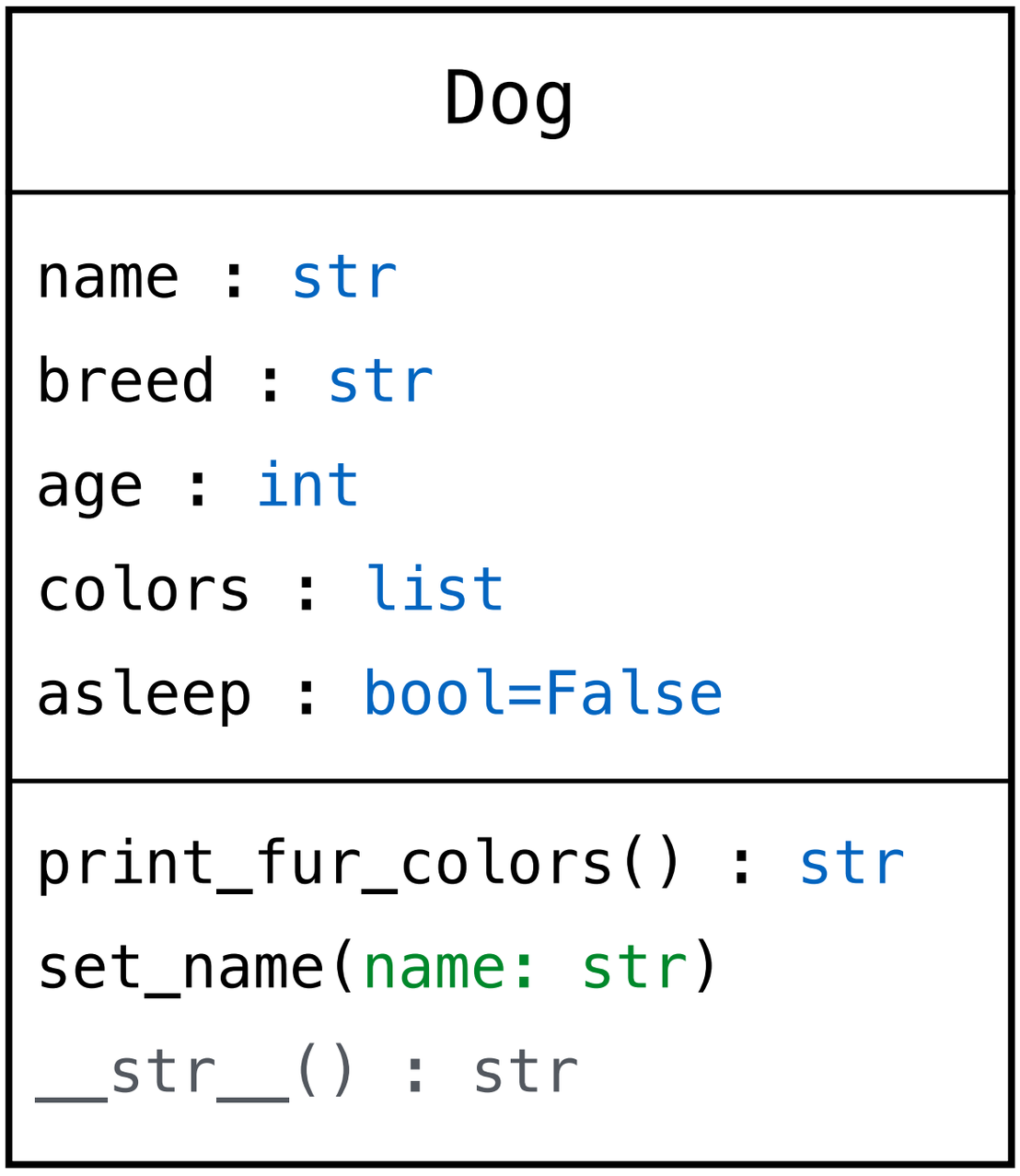
Klassdiagram

datatyp

attribut

metoder

Klassdiagram



datatyp

attribut

metoder

Arbetsflöde

- Identifiera vad som ska modelleras (substantiv)
- Skissa upp ett klassdiagram
- Implementera (stubb)
- Vidareutveckla klassdiagram och implementation

Klassdefinition

- Namn på klassen (dvs. vilken datatyp vi vill modellera)
- Initialisera attribut (så att dessa kan användas i våra metoder)
- Övriga metoder (samt - specialmetoder)

python™

```
1 # Class definition
2 class Dog(object):
3     """
4     Represents a dog
5     """
6     pass
7
```

datatyp

```python
# Class definition
class Dog(object):
    """
    Represents a dog
    """

    def __init__(self, name, breed, age):
        """
        Initialize class attributes
        """
        pass

```

datatyp

attribut

python™

```python
# Class definition
class Dog(object):
    """
    Represents a dog
    """

    def __init__(self, name, breed, age):
        """
        Initialize class attributes
        """
        pass

    # Metoder

    def print_fur_colors(self)
        pass

    def wake_up(self):
        pass

    def sleep(self):
        pass

```

datatyp

attribut

metoder

Specialmetoder

- Representera ett objekt som en sträng `__str__`
- Jämföra två objekt med operatorer som `==`, `<`, `>`
- OSV. => https://docs.python.org/2/reference/datamodel.html#basic-customization

Specialmetoder

- `__eq__`, equals, `==`
- `__lt__`, lesser then, `<`
- `__gt__`, greater then, `>`

```python
# Class definition
class Dog(object):

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

    def __eq__(self, other):
        """Equals"""
        return self.age == other.age

    def __lt__(self, other):
        """Lesser then"""
        return self.age < other.age

    def __gt__(self, other):
        """Greater then"""
        return self.age > other.age

doug = Dog("Doug", 8)
watson = Dog("Watson", 12)

print doug == watson
# => False
print doug > watson
# => False
print doug < watson
# => True
```

```python
# Class definition
class Dog(object):

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

    def __eq__(self, other):
        """Equals"""
        return self.age == other.age

    def __lt__(self, other):
        """Lesser then"""
        return self.age < other.age

    def __gt__(self, other):
        """Greater then"""
        return self.age > other.age

doug = Dog("Doug", 8)
watson = Dog("Watson", 12)

print doug == watson
# => False
print doug > watson
# => False
print doug < watson
# => True
```

Inkapsling

- Objektet har ett gränssnitt — en tydlig definition över **vad** som kan göras.
- Exakt **hur** saker och ting utförs spelar ingen roll utifrån.
- Men objektet måste ha **kontroll** över sitt tillstånd.

Inkapsling

```
# Class definition
class Dog(object):

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

    def set_name(self, name):
        self.name = name

    def get_name(self):
        return self.name

# Instance of Dog
dog = Dog("Watson")
# Bad
dog.name = "Sherlock"
# Good
dog.set_name("Sherlock")
# Bad
print dog.name
# Good
print dog.get_name()

```

Accessor and Mutator Methods

- The ***accessor*** (getter) method returns a value from a class's attribute but does not change it.
- The ***mutator*** (setter) method stores a value in a data attribute or changes the value of a data attribute in some other way.

Inkapsling & synlighet

Synlighet

En objektorienterad princip är att synligheten för egenskaper och operationer ska vara så liten som möjligt. Egenskaper bör ha synligheten ”privat”.

- framför en egenskap eller en operation betyder att endast klassen själv ser (kan använda) denna. Detta kallas för ”privat” synlighet.

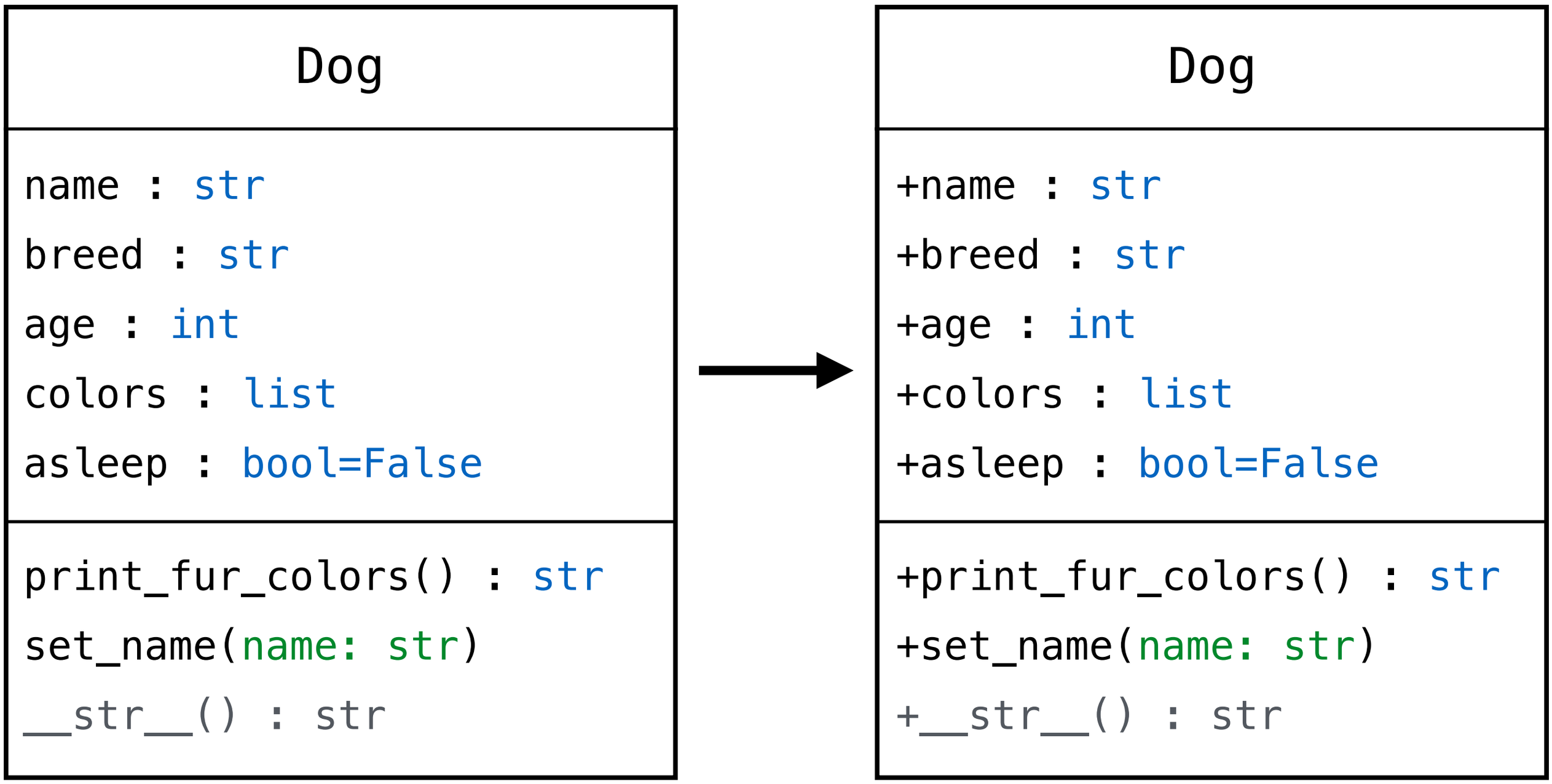
\# framför betyder ”protected” och gör att endast klassen och de klasser som ärver av klassen kan se egenskapen eller operationen.

~ framför betyder “package” och är en utökning av “protected” till att även klasser I samma paket kan se egenskapen eller operation.

\+ betyder “publik” och alla kan se egenskapen eller operationen.

Man kan säga att en klass publika gränssnitt är dess publika operationer (inga egenskaper bör vara publika).

MALMÖ HÖGSKOLA


Dog
name : str
breed : str
age : int
colors : list
asleep : bool=False
print_fur_colors() : str
set_name(name: str)
__str__() : str
Dog
+name : str
+breed : str
+age : int
+colors : list
+asleep : bool=False
+print_fur_colors() : str
+set_name(name: str)
+__str__() : str

Python does not support access protection as C++/ Java/C# does. Everything is public. The motto is, "We're all adults here." Document your classes, and insist that your collaborators read and follow the documentation. […]

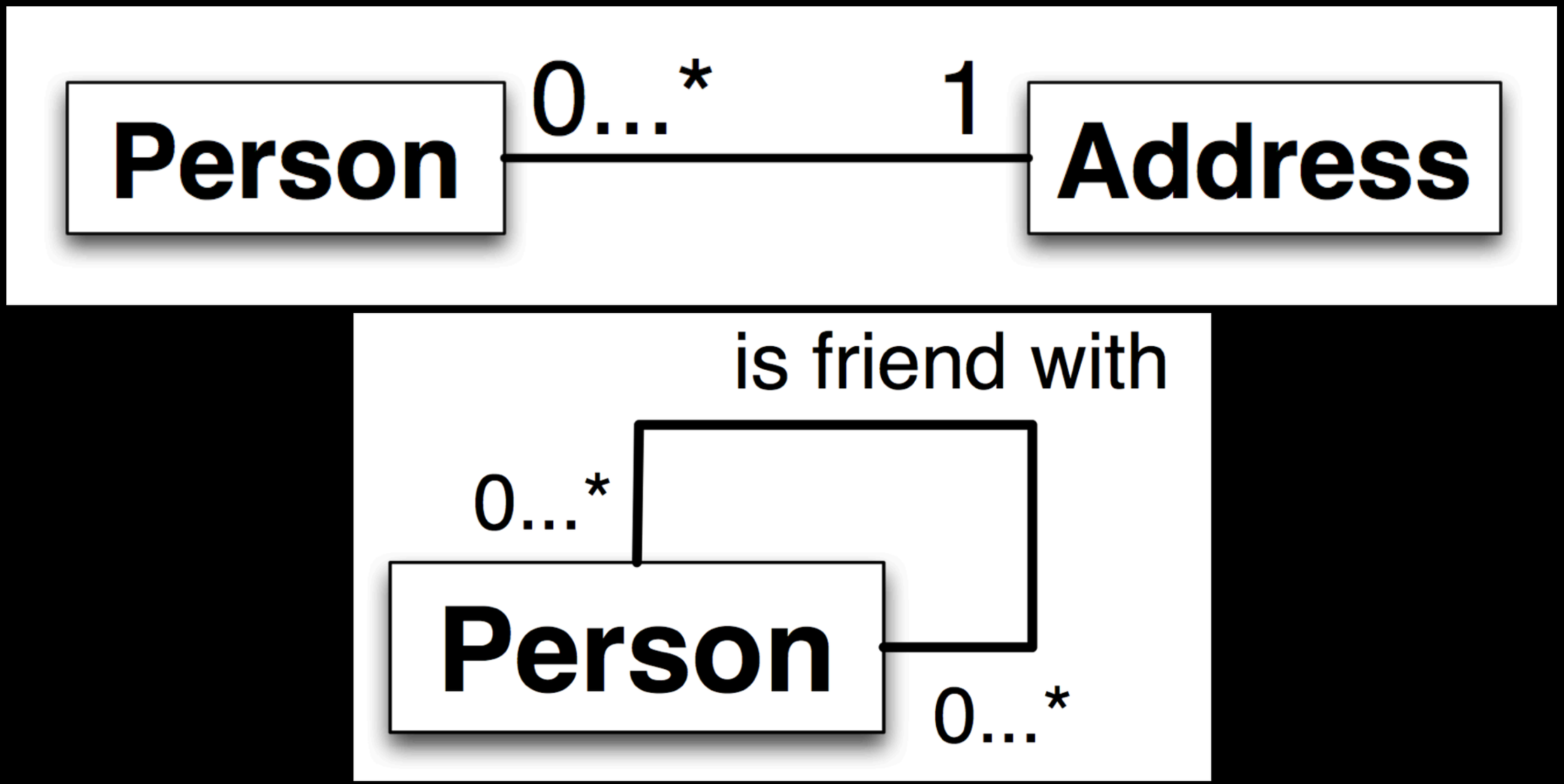
http://stackoverflow.com/a/11483397

Relationer & arv

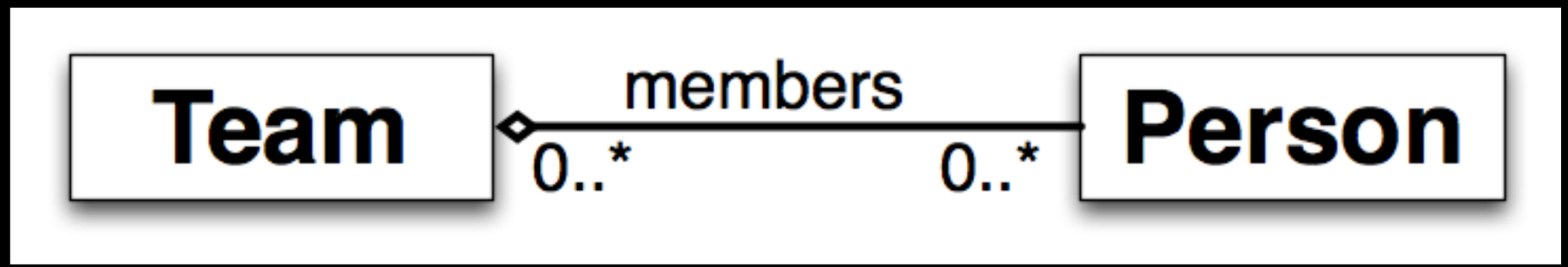
Olika typer av relationer

- Association, relation mellan objekt (1-1, 1-n, n-m)
- Aggregation, form av association, “has-a”
- Composition, form av aggregation, “owns-a”

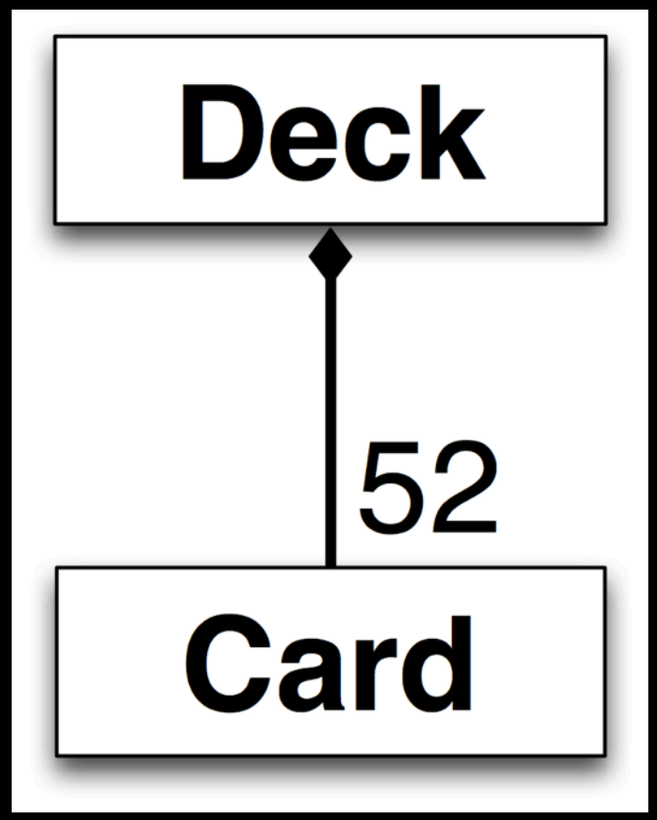
Association



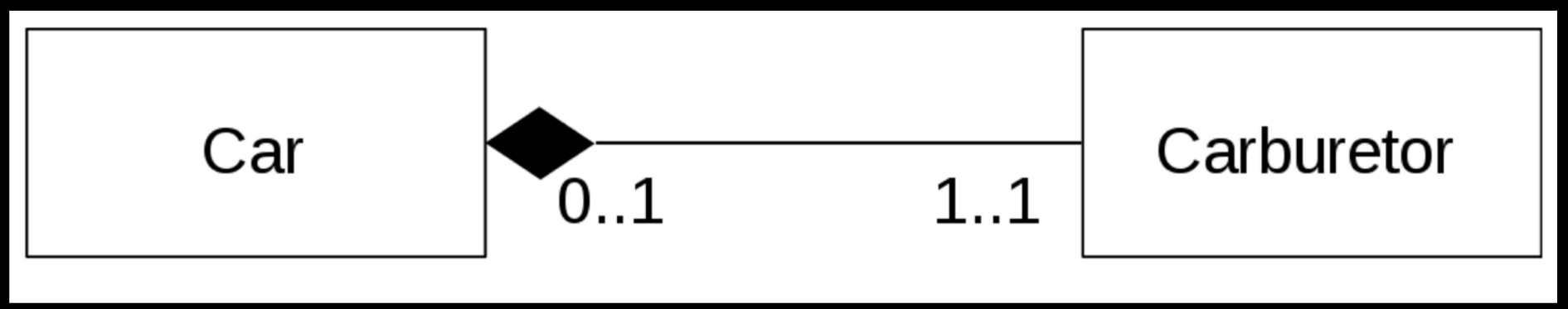
Aggregation



Composition



Implies a multiplicity of
1 or 0..1



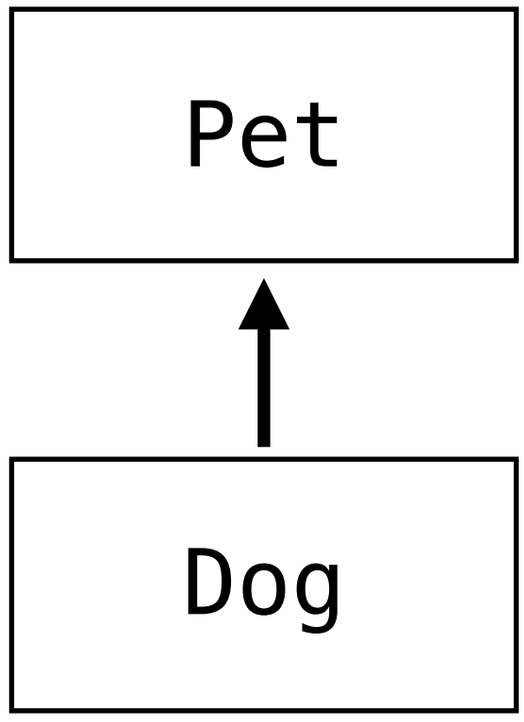
Arv

- En av de vanligaste formerna av relationer mellan objekt
- Kan beskrivas som “is-a” (kan ses som hierarkisk)
  - a cat is a kind of pet
  - a dog is a (different) kind of pet
  - Siberian huskies and poodles are both kinds of dogs

Hur implementerar vi relationer?

```python
class Pet(object):
    """
    Base class for all pets
    """
    pass

class Dog(Pet):
    """
    Subclass of pet
    """
    pass

```


```python
class Pet(object):
    """
    Base class for all pets
    """
    pass

class Dog(Pet):
    """
    Subclass of pet
    """
    pass

```



```python
class Pet(object):
    """
    Base class for all pets
    """
    number_of_legs = 0 # Available to all subclasses

class Dog(Pet):
    """
    Subclass of pet
    """
    pass


dog = Dog()
print dog.number_of_legs # => 0

```

python™

```python
class Pet(object):
    """
    Base class for all pets
    """
    number_of_legs = 0 # Available to all subclasses

class Dog(Pet):
    """
    Subclass of pet
    """

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


dog = Dog("Watson")

# Set number of legs
dog.number_of_legs = 4

print dog.name, dog.number_of_legs
# => Watson 4
```

python™

```python
class Pet(object):
    """
    Base class for all pets
    """
    number_of_legs = 0 # Available to all subclasses

class Dog(Pet):

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

class Cat(Pet):

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

# Instance of subclasses
dog = Dog("Watson")
cat = Cat("Garfield")

print dog.number_of_legs
print cat.number_of_legs

```

```python
class Pet(object):
    """
    Base class for pets
    """
    number_of_legs = 0 # Available to all subclasses

    def set_number_of_legs(self, n):
        self.number_of_legs = n

    def get_number_of_legs(self):
        return self.number_of_legs

    def sleep(self):
        print "zZzZzz"

class Dog(Pet):

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

    def bark(self):
        print "Woof!"

watson = Dog("Watson")
# Set amount of legs
watson.set_number_of_legs(4)
# Print amount of legs
print watson.get_number_of_legs() # => 4

watson.sleep() # => "zZzZzz"
watson.bark() # => "Woof!"
```

python™

```python
class Pet(object):
    """
    Base class for all pets
    """
    def sleep(self):
        print "zZzZzz"

class Dog(Pet):
    """
    Subclass of pet
    """

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

    def sleep(self):
        print "... don't think so!"


watson = Dog("Watson")
watson.sleep()
# => ?
```

```python
class Card(object):
    """
    Represents a card from a Deck
    """
    def __init__(self, label):
        self.label = label

    def __str__(self):
        return self.label

class Deck(object):
    """
    Represents a deck of cards
    """
    def __init__(self, cards):
        self.cards = cards

    def print_deck(self):
        for card in self.cards:
            print card

# List of cards (3)
cards = [Card("2 of Clubs"), Card("Ace of Spades"), Card("4 of Hearts")]
# Deck
deck = Deck(cards)
deck.print_deck()
# => ...
```

Projektstruktur

- Generellt är det bra att ha en klass per python-fil
- Importera klasser i andra filer (`import Dog`, osv.)
- Dokumentation (docstrings och övriga kommentarer)

Övrigt

- Pycco, generera dokumentation från kod
- Studera andras kod, t.ex ramverket Flask

Frågor?