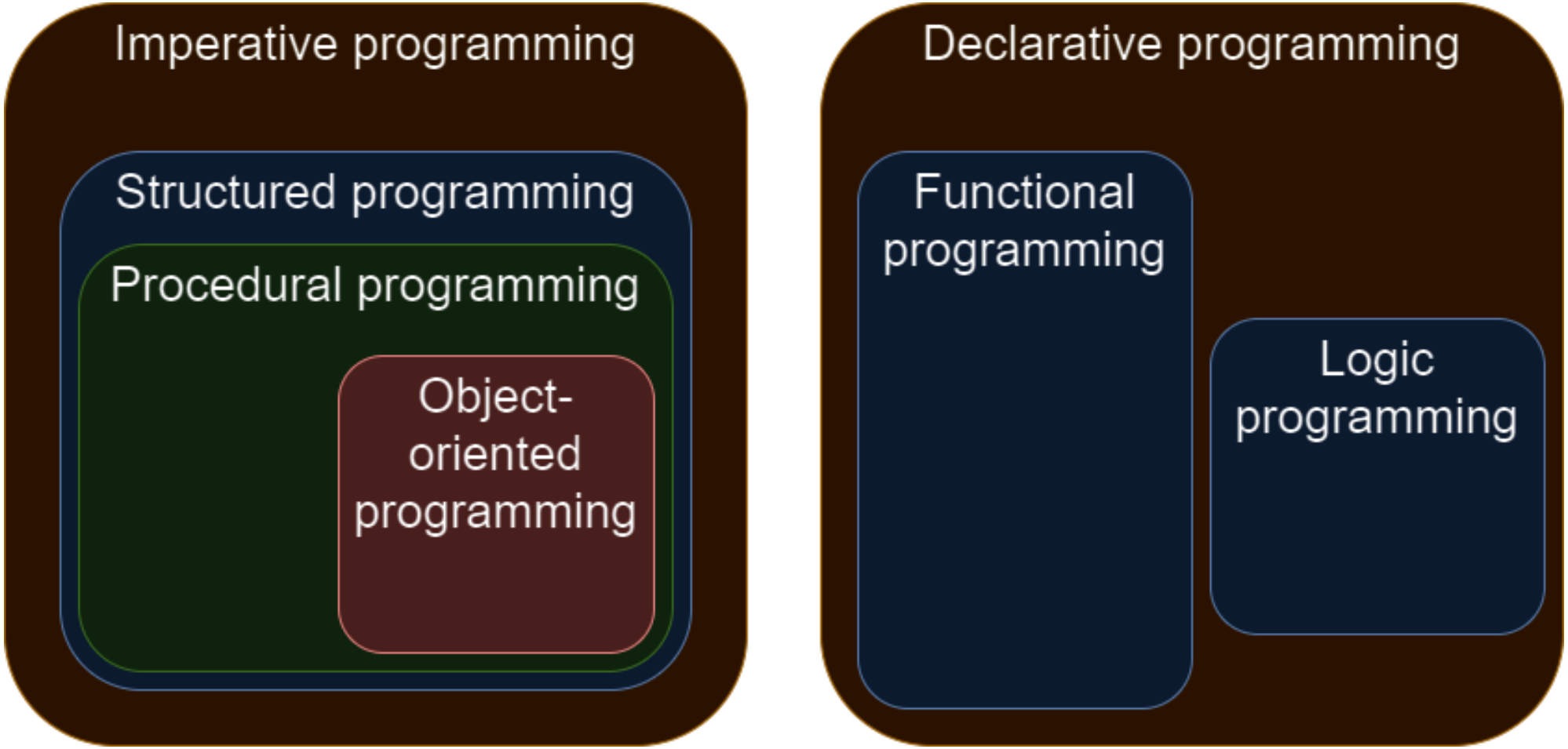
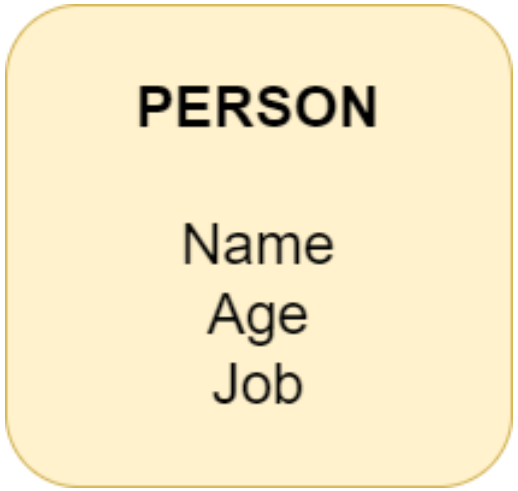
Programming for Social Scientists

Object-oriented programming

Johan A. Dornschneider-Elkink



Programming paradigms

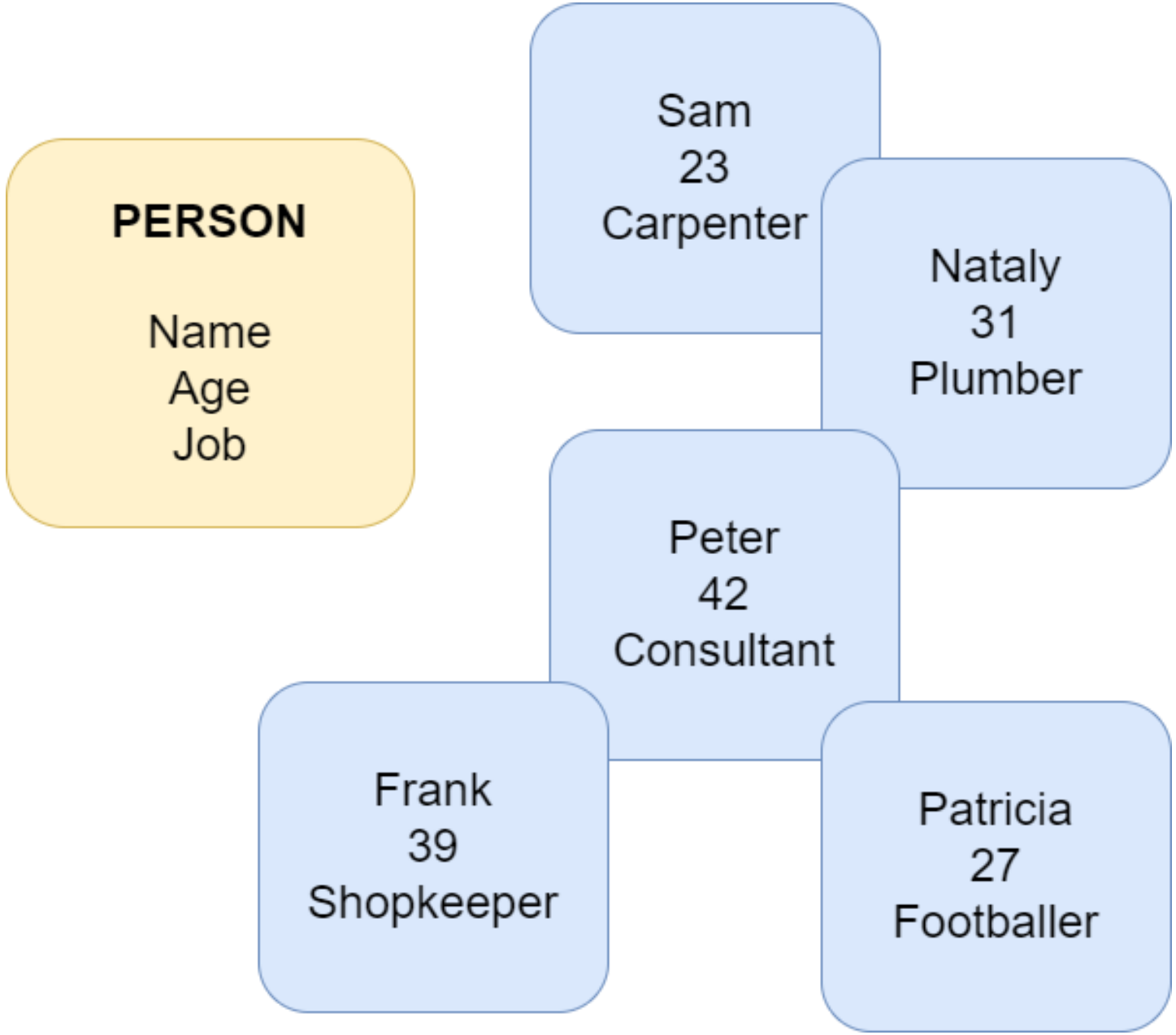


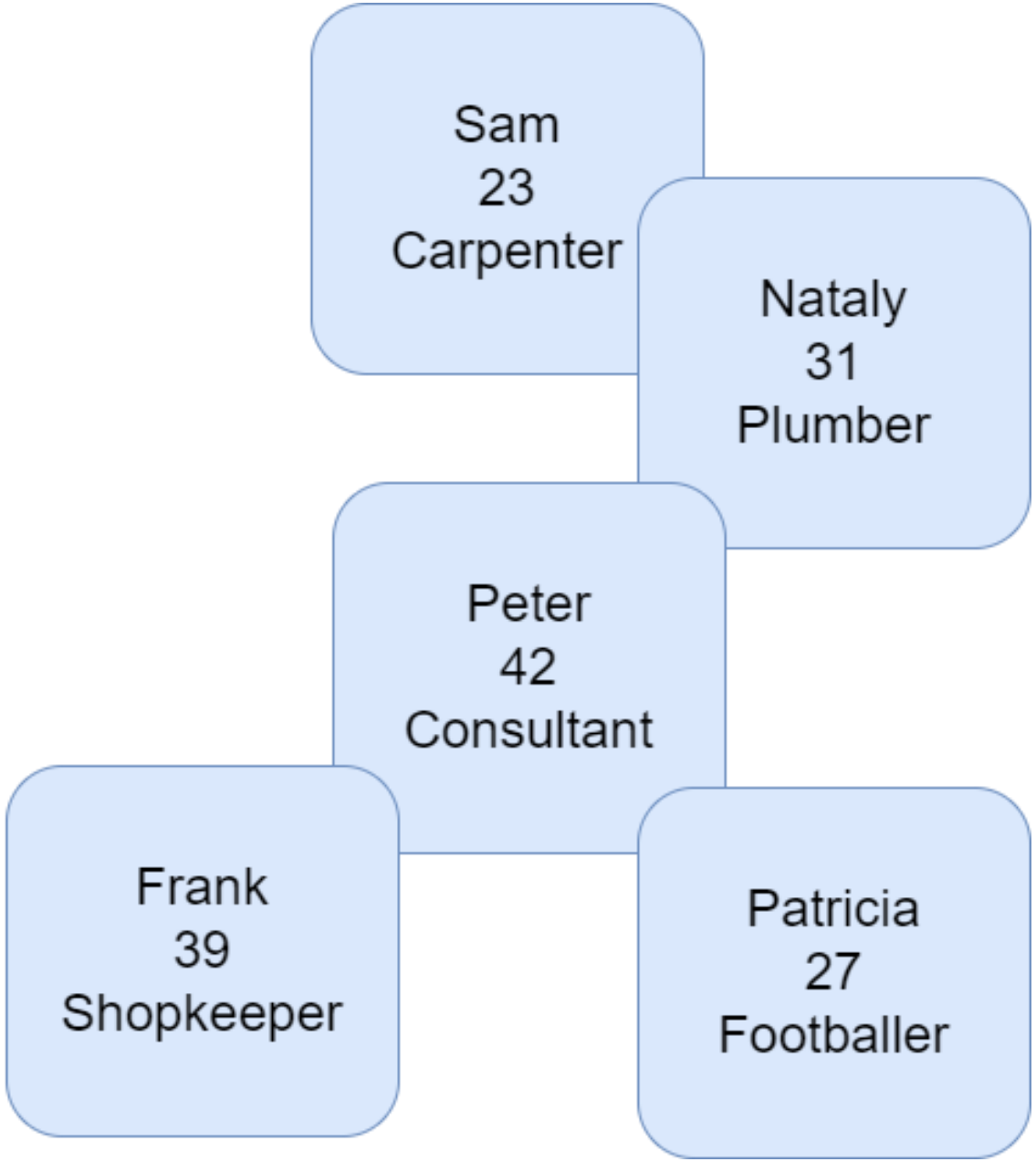
Class

Blueprint or template for user-defined data.

Defines data and functionality to be associated with each instance.

Does not yet reserve any memory space for data.


PERSON
Name
Age
Job
Sam
23
Carpenter
Nataly
31
Plumber
Peter
42
Consultant
Frank
39
Shopkeeper
Patricia
27
Footballer



Object

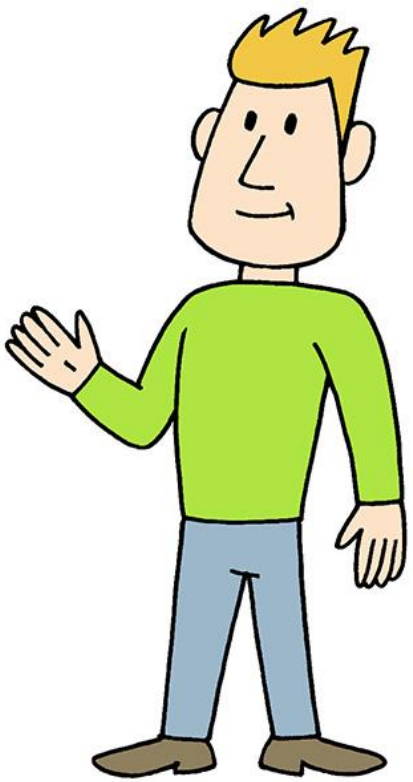
Instance of a specific object, based on the class definition.

Reserves specific memory space for the data, as any other variable type.

```python
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.age))
```

Constructor

Person
Name Age
Print



```python
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.age))
```

Defining a class

```python
john = Person("John", 42)
peter = Person("Peter", 30)
```

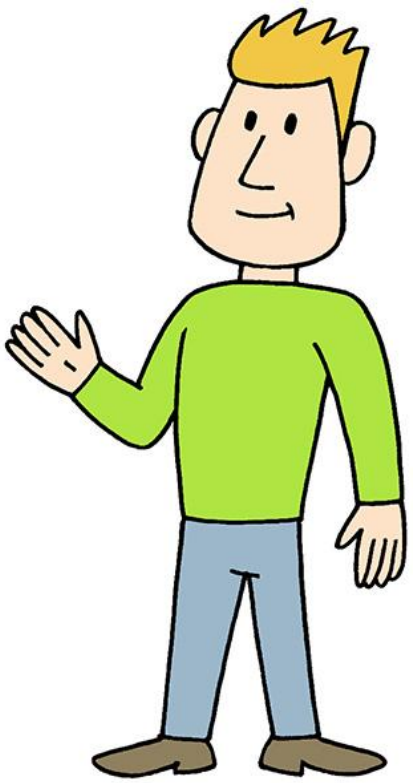
Creating objects

```python
john.print()

peter.print()
```

Calling methods

```python
print(type(john))
```

Person
Name Age
Print



```python
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.age))
```

Instance variables

```python
p = Person("Jos", 48)
print(p)

print(p.age)
```

Instance variables are publicly accessible

```
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.age))

  def getAge(self):
    return self.age

  def setAge(self, age):
    self.age = age
```

Getter- and setter-methods

```
p = Person("Jos", 48)
print(p)

print(p.getAge())
```

```python
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.__age))

  def getAge(self):
    return self.__age

  def setAge(self, age):
    self.__age = age




p = Person("Jos", 48)
print(p)

print(p.getAge())

print(p.__age)
```

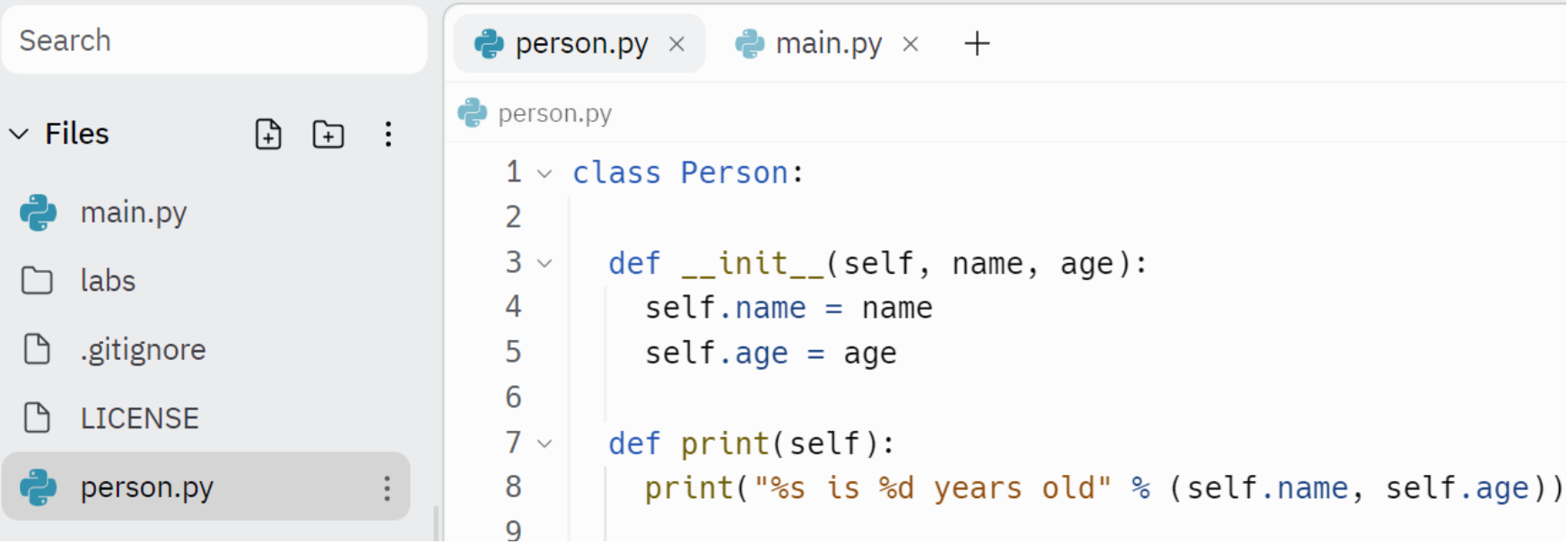
Age is now a private instance variable

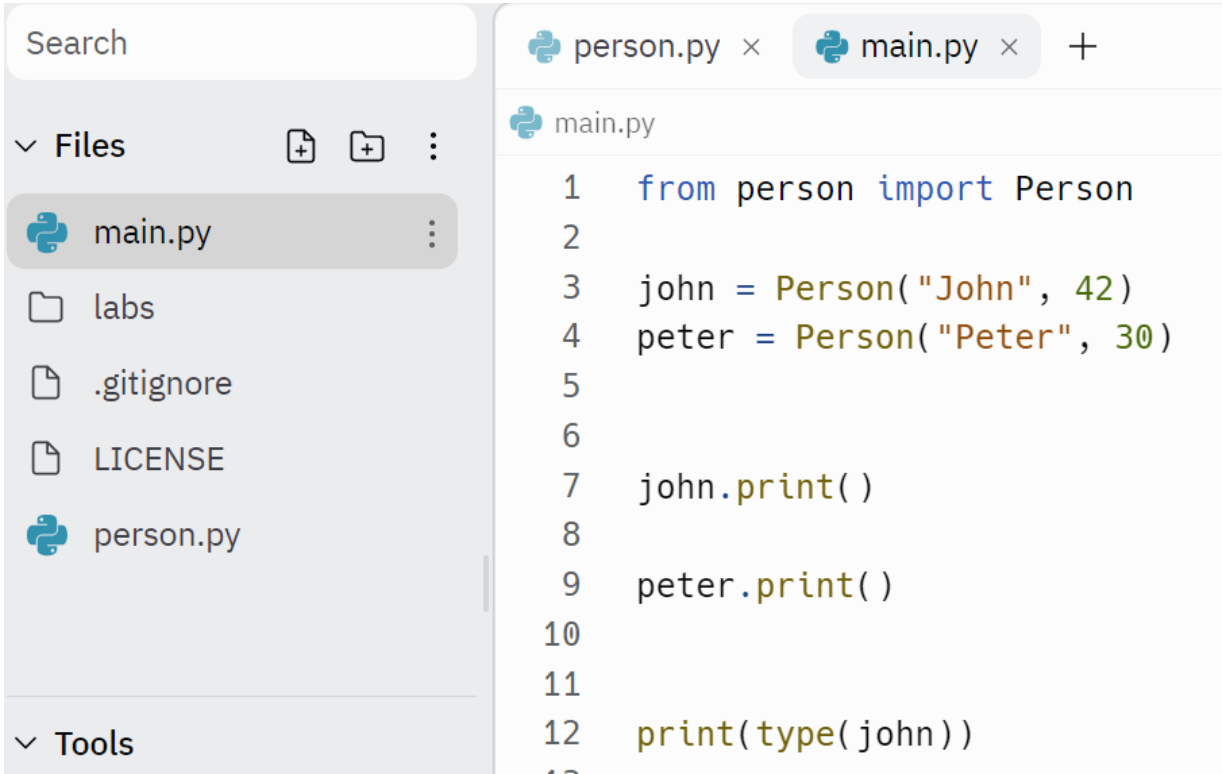
Now you need getter- and setter-methods
Direct access generates error

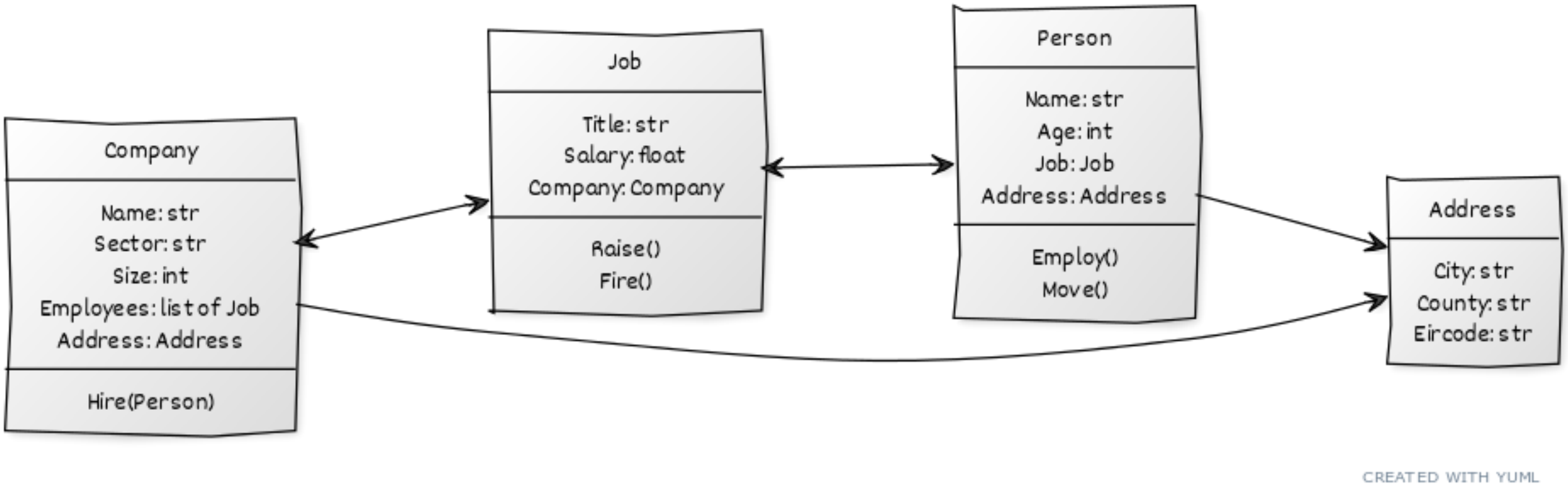
```python
class Person:

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

  def print(self):
    print("%s is %d years old" % (self.name, self.__age))

  def getAge(self):
    return self.__age

  def setAge(self, age):
    self.__age = age




p = Person("Jos", 48)
print(p)

print(p.getAge())

print(p.__age)
```

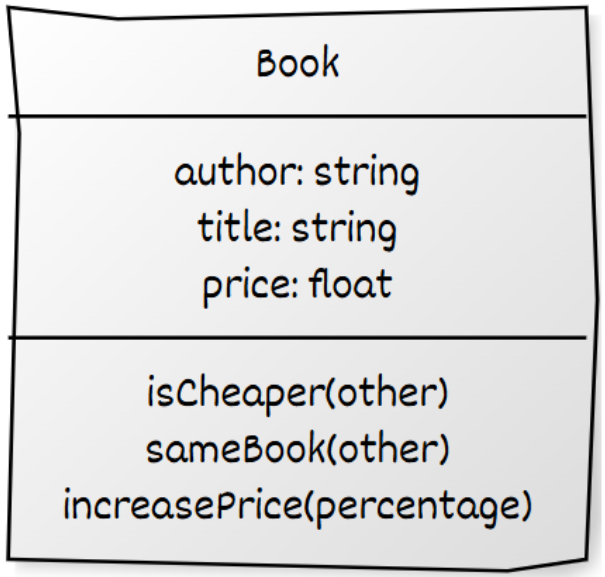
Age is now a private instance variable

Now you need getter- and setter-methods
Direct access generates error


Search
Files
main.py
labs
.gitignore
LICENSE
person.py
person.py
main.py
person.py
1 class Person:
2
3 def __init__(self, name, age):
4 self.name = name
5 self.age = age
6
7 def print(self):
8 print("%s is %d years old" % (self.name, self.age))
9


Search
Files
main.py
labs
.gitignore
LICENSE
person.py
Tools
person.py
main.py
main.py
1 from person import Person
2
3 john = Person("John", 42)
4 peter = Person("Peter", 30)
5
6
7 john.print()
8
9 peter.print()
10
11
12 print(type(john))


Company
Name: str
Sector: str
Size: int
Employees: list of Job
Address: Address
Hire(Person)
Job
Title: str
Salary: float
Company: Company
Raise()
Fire()
Person
Name: str
Age: int
Job: Job
Address: Address
Employ()
Move()
Address
City: str
County: str
Eircode: str
CREATED WITH YUML



CREATED WITH YUML

American Political Science Review

Policy and the Dynamics of Poli

MICHAEL LAVER *New York Universi*

This paper proposes a model that takes the competition into a multiparty environmen switch parties to increase their expectatio shifting affiliations of voters. Different algorithm gator" (adapt party policy to the ideal policy positi that were rewarded; otherwise make random moves), position of the largest party), and "Sticker" (never change of parties using different methods of adaptation are explored a series of experiments to the dynamics of a real party system, described in the time series. This paper reports first steps toward endogenizing key features o cess, including the birth and death of parties, internal party decision rules, and voter ideal poin

Thinking about our simulation, what are some classes (types of objects) that come to mind?