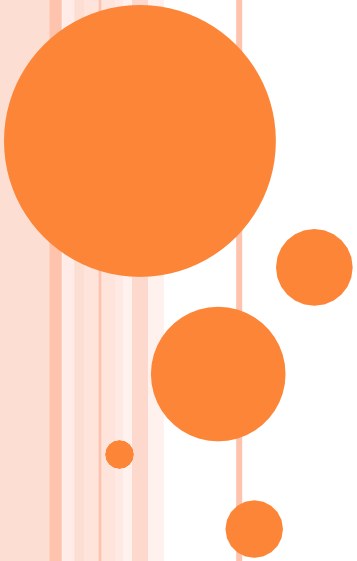


Files in Python

Object oriented programming in Python

Lab 6

FILES

- f = open ('*filename '[, mode [, buffersize])*
  - the mode can be "r", "w", "a" (as C stdio); default "r"
  - add "b" to the text translation mode
  - add "+" to read /write openbuffersize0 =unbuffered; 1 = line-buffered; buffered
- methods:
  - read ([*nbytes]) readline(),readlines()*
  - write (*string), writelines(list)*
  - seek (*pos [, how]), tell ()*
  - flush (), close ()
  - fileno()

Opening of file

- The mode is given as a string. The main modes:
  - 'R': Read.
  - 'W': Write. The file content is overwritten. If the file does not exist, it is created.
  - 'A': opening for write appending (Append). File at the end is written without overwriting the old contents of the file. If the file does not exist, it is created.
  - We can add to all these modes the signebpour open the file in binary mode.

```
>>> my_file = open ( "file.txt", "r")
>>> contents = my_file.read()
>>> print(contents)
The file contents.
>>> my_file.close()
```

To write a chain

```
>>> mon_fichier = open("fichier.txt", "w")
>>> mon_fichier.write("SE")
>>> mon_fichier.close()
```

The method tell () returns an integer indicating the current position in the file measured in bytes from the beginning of the file when the file is opened in binary mode or a dark number in text mode.

To change the position in the file, use f.seek(lag, from).

```
>>> f = open ( 'workfile''rb+ ')
>>> f.write(B'0123456789abcdef ')
16
>>> f.seek(5) # Go to the 6th byte in the file
5
>>> f.read(1)
b'5 '
>>> f.seek(-3, 2) # Go to the 3rd byte before the end
13
>>> f.read(1)
B'D'
```

List items recursively

```
import os
folder_path = “/Documents/SE"
for path, dirs, files in os.walk(folder_path):
    for filename in files:
        print(filename)
```

Exercise

- What will we find in the characteristics of a person?
  - the name, surname, age, place of residence => four attributes
- To set the attributes of our object, define a constructor in class. A constructor is a method of the object to create the charging attributes.

object.attribute = value

```
class Project(object):
    def __init__(self,name="Proj",budget=100):
        self.Name=name
        self.budget=budget

    def getBudget(self):
        return self.budget

    def toString(self):
        return self.name+str(self.budget)
```

Inheritance

- All attributes and methods of Python classes public.
- All methods take a self variable as the first argument. This variable is a reference to the object being manipulated.

```
class Employee(object):
    def __init__(self,name="Emp",projects=[]):
        self.name=name
        self.nrOfProjects= len(projects)
        self.projects=projects

    def getNrOfProjects(self):
        return self.nrOfProjects

    def addProject(self,project):
        self.projects.append(project)

    def calculateBudgetTotal(self):
        sum=0
        for proj in self.projects:
            sum+=proj.getBudget()
        return sum

    def calculateSalary(self):
        return 0


class Manager(Employee):
    def __init__(self,**kwargs):
        super(Manager,self).__init__(**kwargs)

    def calculateSalary(self):
        return 4000+(self.calculateBudgetTotal()*10)/100
```

*ARGS AGAINST **KWARGS

- you use *args when you are not sure of the number of arguments that can be passed to your function. It allows you to pass an arbitrary number of arguments to your function, as in Java.
- **kwargs you can manage named arguments that you have not defined in advance, such as attributes Class Person.

```
class Programmer(Employee):
    def __init__(self, **kwargs):
        super(Programmer, self).__init__(**kwargs)

    def calculateSalary(self):
        return 3000 + self.getNrOfProjects()*500
```

Overriding

- Inheritance allows you to rewrite some methods of the parent in the child => the overriding.

```
class Parent(object):

    def truc(self):
        print ('foo')

class Enfant1(Parent):
    pass # pas d'overriding

class Enfant2(Parent):

    def truc(self):
        print ('bar') # overriding !


Enfant1().truc()
Enfant2().truc()
```

```
foo
bar
```

OVERLOADING

- Python does not support overloaded methods, but you can do this:

```
class TestSurcharge(object):
    def surcharge_function(self, *args, **kwargs):
        # Appelez la fonction qui a le même nombre d'arguments non-mot-clé.
        getattr(self, "_surcharge_function_impl_" + str(len(args)))(*args, **kwargs)

    def _surcharge_function_impl_3(self, nom, debut, direction, **kwargs):
        print ("Surcharge 3")
        print ("Nom: %s" % str(nom))
        print ("Debut: %s" % str(debut))
        print ("Direction: %s" % str(direction))

    def _surcharge_function_impl_2(self, nom, description):
        print ("Surcharge 2")
        print ("Nom: %s" % str(nom))
        print ("Description: %s" % str(description))
        print (description)

test = TestSurcharge()

test.surcharge_function("S1", 0, "NE")
test.surcharge_function("S2", "Étudiez autant que possible")
```

```
Surcharge 3
Nom: S1
Debut: 0
Direction: NE
Surcharge 2
Nom: S2
Description: Étudiez autant que possible
Étudiez autant que possible
```

Class dependencies

- The Person class has an array of hobbies and an array of children belonging to the class Children

```
class Personne(object):
    def __init__(self, nom, prenom, age="", lieu_residence=""):
        self.enfants = []
        self.loisirs = []
        self.nom = nom
        self.prenom = prenom
        self.age = age
        self.lieu_residence = lieu_residence

    def ajouter_enfant(self, enfant):
        self.enfants.append(enfant)
    def ajouter_loisir(self, sport):
        self.loisirs.append(sport)
    def __repr__(self):
        return str(self)
    def __str__(self):
         return "member of Personne"


class Enfant(Personne):
    def __init__(self, numero, **kwargs):
        super(Enfant, self).__init__(**kwargs)
        self.numero = numero
    def __repr__(self):
        return str(self)
    def __str__(self):
        return "member of Enfant"
```

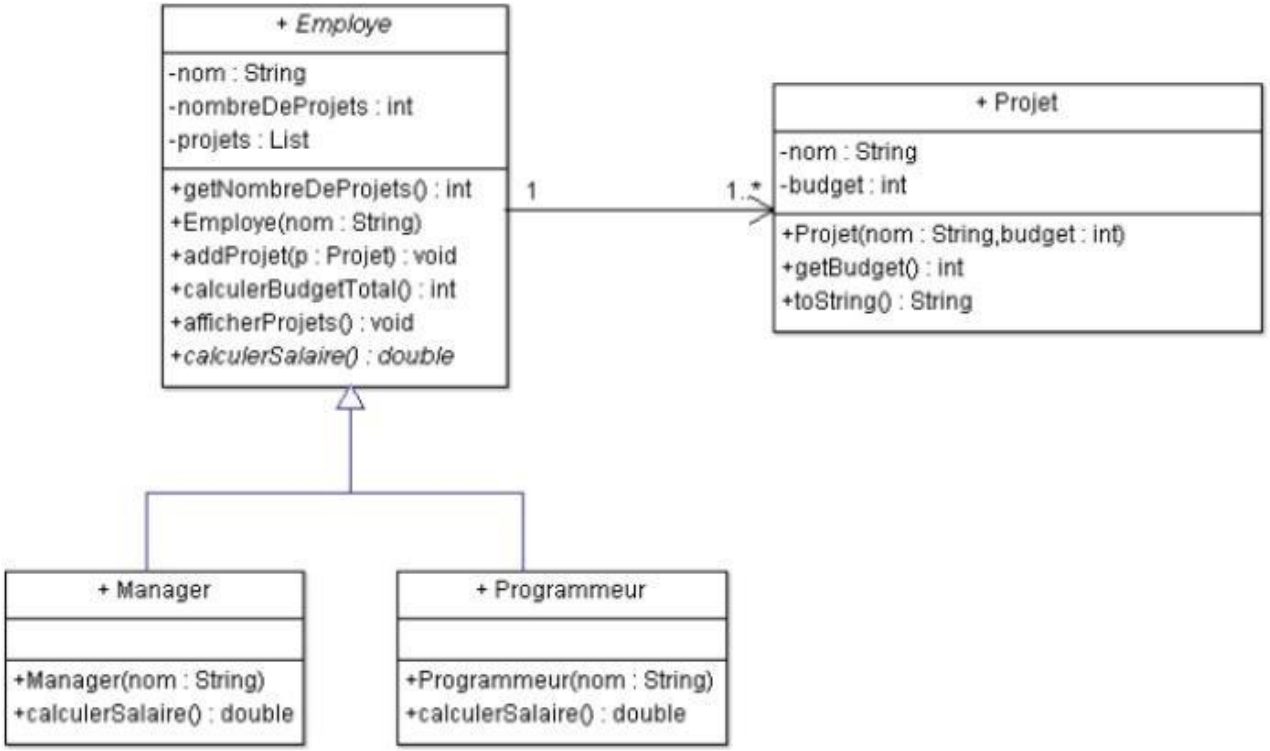
```
p = Personne(nom='Piaf', prenom='Edith')

e = Enfant(nom='Popescu', prenom="Ion", numero='1220F_20')
print (e.nom, e.prenom, e.age, e.numero)
p.ajouter_enfant(e)
p.ajouter_loisir("natation")
p.ajouter_loisir("ski")
print(p.loisirs)
print ([e.nom for e in p.enfants])
```

```
Popescu Ion  1220F_20
['natation', 'ski']
['Popescu']
```

Example

Implement the class diagram in Python.

- To calculate the salary, the following is known:
  - - manager: 4000 + computeTotalBudget() * 10 /100;
  - - programmer: 3000 + getNoProjects() * 500



Example - solution

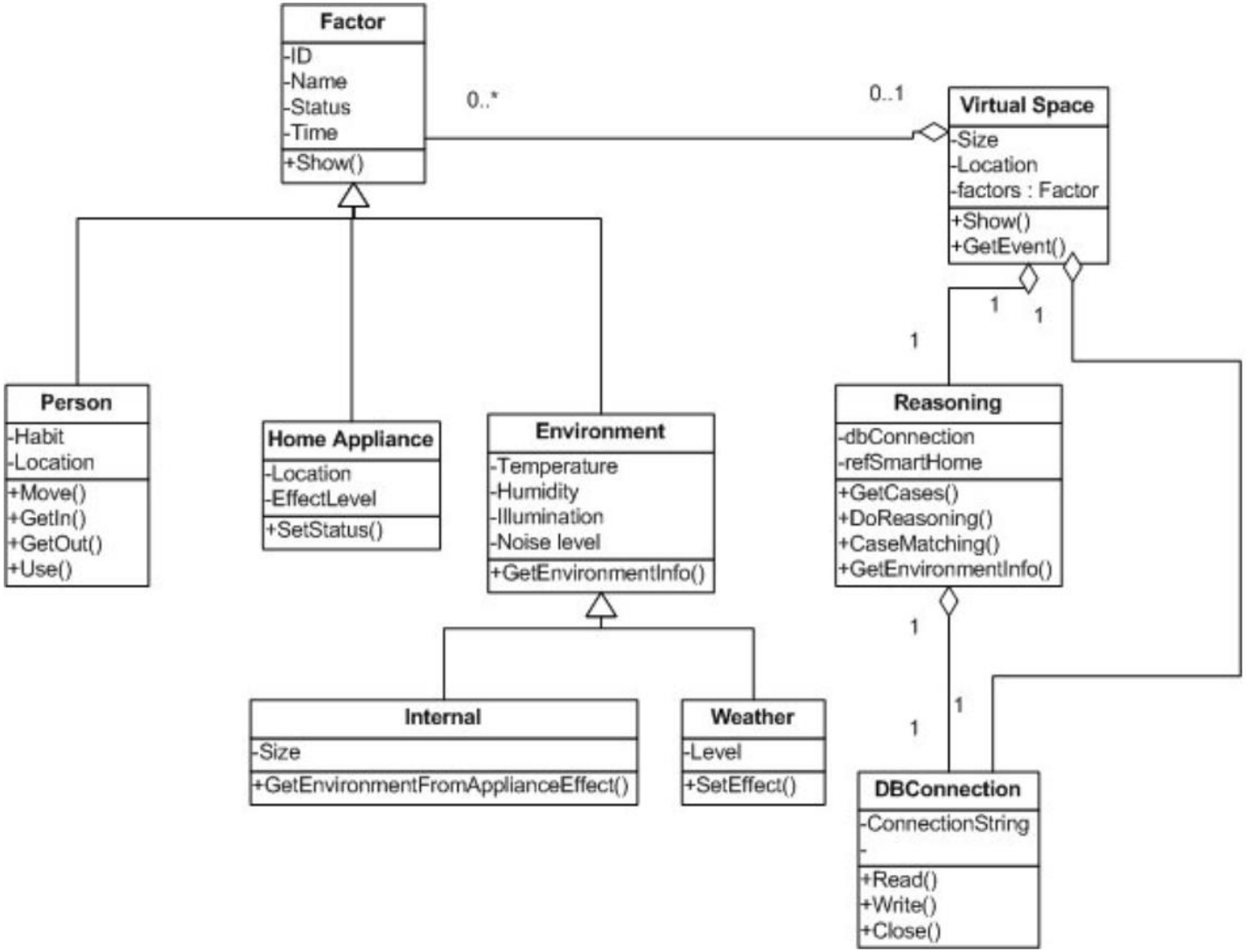
```
class Project(object):
    def __init__(self,name="Proj",budget=100):
        self.Name=name
        self.budget=budget

    def getBudget(self):
        return self.budget

    def toString(self):
        return self.name+str(self.budget)


class Employee(object):
    def __init__(self,name="Emp",projects=[]):
        self.name=name
        self.nrOfProjects= len(projects)
        self.projects=projects

    def getNrOfProjects(self):
        return self.nrOfProjects

    def addProject(self,project):
        self.projects.append(project)

    def calculateBudgetTotal(self):
        sum=0
        for proj in self.projects:
            sum+=proj.getBudget()
        return sum

    def calculateSalary(self):
        return 0
```

```
class Manager(Employee):
    def __init__(self,**kwargs):
        super(Manager,self).__init__(**kwargs)

    def calculateSalary(self):
        return 4000+(self.calculateBudgetTotal()*10)/100


class Programmer(Employee):
    def __init__(self, **kwargs):
        super(Programmer, self).__init__(**kwargs)

    def calculateSalary(self):
        return 3000 + self.getNrOfProjects()*500


if __name__=="__main__":

projects=[Project(name="Proj1",budget=100),Project(name="Proj2",bud
get=200)]
    man=Manager(name="manager1",projects=projects)
    pro=Programmer(name="programmer1",projects=projects)
    print("Manager budget:",man.calculateSalary()," Programmer
budget:",pro.calculateSalary())
    man.addProject(Project('Proj 3',budget=500))
    print("Project added to manager")
    print("New Manager budget:", man.calculateSalary(), " New
Programmer budget:", pro.calculateSalary())
```

```
C:\Users\Iuliana\PycharmProjects\ex1\venv\Scripts\python.
Manager budget: 4030.0  Programmer budget: 4000
Project added to manager
New Manager budget: 4080.0  New Programmer budget: 4000
```

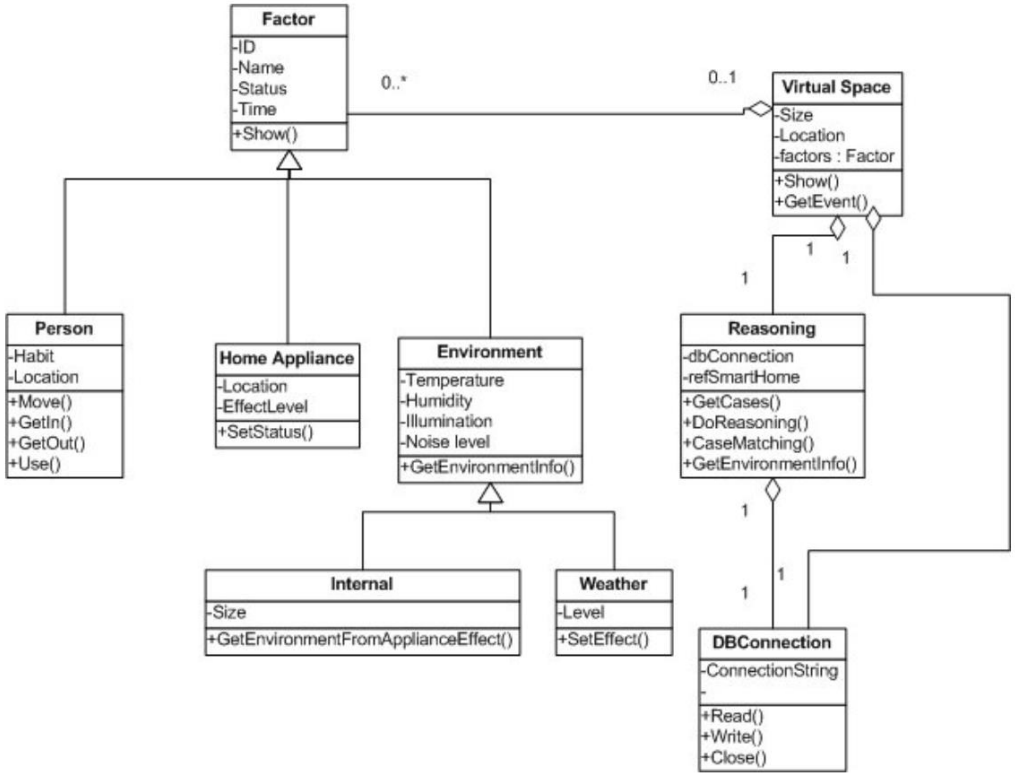
EXERCISES

1. Implement model classes:



EXERCISES

2. Enviromental factors affect persons according to the illnesses which they have. Think of 10 diseases and the parameters which influence them, per category (emergency, warning, normal, below normal). Code the reasoning rules. Test the rules.



Homework

Read the 10 diseases and the parameters which influence them, per category (emergency, warning, normal, below normal) from a CSV file.

Create an interaction scenario between the user and the program, like in the case of bots.