IUT2 Université Grenoble Alpes
Département STID

Programmation Objet

R3.VCOD.10

Martin LENTSCHAT

martin.lentschat@univ-grenoble-alpes.fr

Les principales notions de la POO

- Classe (avec ses Attributs et Méthodes) et Objet/Instance
- Envoi de messages
- Encapsulation
- **Héritage**
- **Polymorphisme**
- Associations entre classes
- Agrégation
- Composition

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes

Reprenons notre classe Etudiant

- Comme une personne, un étudiant a un nom, un prénom, une adresse… De plus on gère aussi les informations sur son âge…

- 1ère solution :
  - On redéfinit les attributs et les méthodes qu'il a déjà en tant que personne (pas terrible ☹)

On ajoute les attributs et méthodes qui lui sont propres en tant qu'étudiant

```
class Personne:

    def __init__(self, nom : str, prenom : str, adresse : str):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse

    def get_firstname(self):
        return self.firstname

    def get_lastname(self):
        return self.lastname

    def get_address(self):
```

```
class Etudiant:

    moyenne_age : int

    def __init__(self, nom : str, prenom : str, adresse : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse
        self.age = age


    def get_age(self):
        return self.age

    def set_moyenne_age(ma : float):
        Etudiant.moyenne_age = ma

    def get_firstname(self):
        return self.firstname

    def get_lastname(self):
```

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes

Reprenons notre classe Etudiant :

comme un étudiant est avant tout une personne (il a les mêmes attributs et les mêmes méthodes), on dit qu'il **hérite** de la classe Personne (relation '*Est-Un*')

Vocabulaire équivalent :

classe/sous-classe ou **classe parent/classe enfant**

- 2ème solution : utilisation de l'héritage avec la syntaxe **class Enfant(Parent)**
  - Inutile de redéfinir les attributs et méthodes qu'il *hérite de la classe Personne*
  - On redéfinit le constructeur pour initialiser les *attributs qu'il hérite de la classe Personne, et ceux qui lui sont propres*
  - On peut enrichir une classe en ajoutant dans sa (ou ses) sous-classes des attributs et méthodes qui lui (leur) sont propres (**enrichissement**)

```python
class Personne:

    def __init__(self, nom : str, prenom : str, adresse : str):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse

    def get_firstname(self):
        return self.firstname

    def get_lastname(self):
        return self.lastname

    def get_address(self):
        return self.address

    def change_firstname(self, nouv_prenom : str):
        self.firstname = nouv_prenom

    def change_lastname(self, nouv_nom : str):
        self.lastname = nouv_nom

    def change_address(self, nouv_adresse : str):
        self.address = nouv_adresse


class Etudiant(Personne):

    def __init__(self, nom : str, prenom : str, adresse : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse
        self.age = age


    def get_age(self):
        return self.age

    def change_age(self, nouv_age : int):
        self.age = nouv_age
```

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes

```python
class Personne:

    def __init__(self, nom : str, prenom : str, adresse : str):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse

    def get_firstname(self):
        return self.firstname

    def get_lastname(self):
        return self.lastname

    def get_address(self):
        return self.address

    def change_firstname(self, nouv_prenom : str):
        self.firstname = nouv_prenom

    def change_lastname(self, nouv_nom : str):
        self.lastname = nouv_nom

    def change_address(self, nouv_adresse : str):
        self.address = nouv_adresse


class Etudiant(Personne):

    def __init__(self, nom : str, prenom : str, adresse : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse
        self.age = age


    def get_age(self):
        return self.age

    def change_age(self, nouv_age : int):
        self.age = nouv_age
```

etudiantHeritage.py

Appel au constructeur de Etudiant, avec les attributs hérités de la classe Personne, et ses attributs propres

```python
import etudiantHeritage

e1 = etudiantHeritage.Etudiant("Durand", "Théo", "GNB", 18)

print("Nom de e1: ", e1.get_lastname())
print("Prénom de e1: ", e1.get_firstname())
print("Age de e1: ", e1.get_age())
```

main.py

Appel aux méthodes de Etudiant, qu'il s'agisse des méthodes héritées de la classe Personne, ou de ses propres méthodes

```
Nom de e1:  Durand
Prénom de e1:  Théo
Age de e1:  18
```

À l'exécution

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes : avantages

- L'**héritage** permet :
  - La **propagation** des attributs, des opérations et des associations d'une classe vers ses sous-classes
  - **L'héritage** dans une sous-classe de **tous** les attributs et de **toutes** les opérations de sa super-classe (héritage **complet**)

- Avantages :
  - Réutilisabilité : réutiliser le code d'une classe
  - Extensibilité : étendre la fonctionnalité d'une classe *sans* avoir à modifier le code existant

Héritage entre classes

- Une classe peut avoir plusieurs sous-classes

```python
class Etudiant(Personne):

    def __init__(self, nom : str, prenom : str, adresse : str, age : int):
        self.firstname = prenom
        self.lastname = nom
        self.address = adresse
        self.age = age


    def get_age(self):
        return self.age

    def change_age(self, nouv_age : int):
        self.age = nouv_age


class Enseignant(Personne):

            def __init__(self, nom : str, prenom : str, adresse : str, stat : str):
                self.firstname = prenom
                self.lastname = nom
                self.address = adresse
                self.statut = stat


            def get_statut(self):
                return self.statut

            def change_age(self, nouv_statut : str):
                self.statut = nouv_statut
```

etudianEnseignant.py

Création d'un étudiant et d'un enseignant, et appel de leurs méthodes

```python
import etudiantEnseignant

etu1 = etudiantEnseignant.Etudiant("Durand", "Théo", "GNB", 18)
ens1 = etudiantEnseignant.Enseignant("Dupont", "Jeannine", "SMH", "Maitre de conférences")

print("Nom de etu1: ", etu1.get_lastname())
print("Prénom de etu1: ", etu1.get_firstname())
print("Age de etu1: ", etu1.get_age())

print("Nom de ens1: ", ens1.get_lastname())
print("Prénom de ens1: ", ens1.get_firstname())
print("Statut de ens1: ", ens1.get_statut())
```

main.py

```
Nom de etu1:  Durand
Prénom de etu1:  Théo
Age de etu1:  18
Nom de ens1:  Dupont
Prénom de ens1:  Jeannine
Statut de ens1:  Maitre de conférences
```

À l'exécution

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes

- Grâce à la méthode *super()*, il n'est pas nécessaire de redéfinir les attributs hérités de la super-classe

- Fonctionne différemment en cas d'héritage multiples

```python
class Personne:
    def __init__(self, nom: str, prenom: str, age: int):
        self.first_name = prenom
        self.last_name = nom
        self.age = age

    def get_first_name(self):
        return self.first_name

    def get_last_name(self):
        return self.last_name

    def set_first_name(self, new_val: str):
        self.first_name = new_val

    def set_last_name(self, new_val: str):
        self.last_name = new_val

    def set_age(self, new_val: int):
        self.age = new_val

    def get_age(self):
        return self.age

    def to_string(self):
        return self.first_name + ' ' + self.last_name


class Etudiant(Personne):
    def __init__(self,  nom: str, prenom: str, age: int, etablissement: str):
        super().__init__(nom, prenom, age)
        self.school = etablissement


class Enseignant(Personne):
    def __init__(self,  nom: str, prenom: str, age: int, etablissement: str):
        super().__init__(nom, prenom, age)
        self.school = etablissement

etu1 = Etudiant('Freddie', 'Mercury', 45, 'Queen College')
print(etu1.to_string())
```

IUT2 Université Grenoble Alpes
Département STID

Exercice

1. Définissez une classe *Animal* avec des attributs d'objet *nom* et *age* (0 par défaut), les méthodes *get* et *set* que vous jugerez nécessaires, et une méthode *vieillir* augmentant l'âge de 1

2. Définissez une sous-classe *Reptile*, héritière complète de *Animal* avec un attribut *nb_ecailles* (1000 par défaut) et une méthode *vieillir* augmentant l'âge du reptile et diminuant le nombre d'écailles

   a) Comment faire pour ne pas avoir à modifier l'*age* de *Reptile* dans sa méthode *vieillir* ?

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes : Polymorphisme des méthodes

Le **Polymorphisme** *(ou substitution, redéfinition)* désigne le fait qu'une sous-classe puisse redéfinir une méthode, cad crée une *nouvelle implémentation d'une méthode héritée*.

- Lorsque deux méthodes ont le même nom on parle de **surcharge de méthodes**.
- Il faut que la méthode dans la sous-classe ait le *même nom* que la méthode dans la super-classe.
- Dans ce cas, la méthode de la sous-classe l'emporte si l'objet qui reçoit le message est une sous-classe.
- Il est possible d'appeler la méthode parent dans la méthode redéfinie dans la sous classe

IUT2 Université Grenoble Alpes
Département STID

Héritage entre classes : **Surcharge** de méthodes

- Surcharge = un opérateur a différents comportements selon les opérandes sur lequel il s'applique
  - Cela permet d'utiliser le polymorphisme pour redéfinir les méthodes prédéfinies ou héritées

Nom	Méthode spéciale	Utilisation
opposé	`__neg__`	`-obj1`
addition	`__add__`	`obj1 + obj2`
soustraction	`__sub__`	`obj1 - obj2`
multiplication	`__mul__`	`obj1 * obj2`
division	`__div__`	`obj1 / obj2`
division entière	`__floordiv__`	`obj1 // obj2`

Exemple – redéfinition du Destructor

- Les Destructor
  - Methode :

    def __del__(self)
  - Appel :

    del ref_obj
  - Fait partis des méthodes définies par défaut

```python
class Etudiant:
    def __init__(self, nom: str, prenom: str):
        self.first_name = prenom
        self.last_name = nom
        self.notes = []

    def __del__(self):
        print('etudiant éliminé')


etu1 = Etudiant('Curie', 'Marie')
etu2 = Etudiant('Einstein', 'Albert')
etu1b = etu1


del etu1

print(etu1)
print(etu1b)
```