Table des matières

~~stoggle_buttons~~

Notions de base sur la programmation Objet

Vocabulaire Objet

- Classe : équivalent à un type (plus ou moins complexe). Elle contient essentiellement la définition de méthodes qui pourront s'appliquer sur un objet dont le type sera "de cette classe".
- responsabilité d'une classe
- objet ou instance : on crée un objet (ou une instance) à partir d'une définition (un type) écrite dans une classe. Un objet existe en mémoire vive.
- méthode. Fonction ou procédure interne à une classe qui s'appliquera sur un objet de cette classe et travaillera à partir des attributs de l'objet, de variables locales à la méthode et des paramètres de la méthode ;
- Variable Objet : un pointeur qui permet "d'accéder à un objet".
- constructeur : méthode particulière d'une classe qui permet d'initialiser les attributs de l'objet (**init(self, ...)** en Python) ;
- propriété, attribut, "data instance", "data attribute"
- encapsulation : on ne peut intervenir sur l'état (les attributs ou propriétés) qu'en utilisant des méthodes.
- héritage
- association, rôle, navigabilité
- polymorphisme
- surcharge de méthodes ou d'opérateurs
- redéfinition de méthode
- interface ou API : la notion d'interface qui existe dans UML n'apparaît pas en Python car elle ne semble pas nécessaire.
- transtypage
- design patterns ou patrons de conception
- ...

Créer une classe

Cas 1

EditeurCas1.py

```
class Editeur:
    # Le constructeur des objets
    def __init__(self, nom):
        ## On définit les (nouveaux) attributs de l'objet et on leur affecte une valeur
        self.nom = nom
        self.num = 0
```

```
    # Une méthode de type fonction
    def toString(self):
        return "Editeur : " + str(self.num) + " " + self.nom
```

LivreCas1.py

```
class Livre:

    def __init__(self, prmNum, prmTitre, prmEditeur):
        self.num = 0
        self.titre= prmTitre
        self.sonEditeur=prmEditeur
    def toString(self):
        return "Livre : " + str(self.num) + " " + self.titre + " " +
self.sonEditeur.nom
```

Cas 2

Editeur.py

```
# ======================================
class Editeur:

    def __init__(self, nom):
        self.nom = nom
        self.num = 0
        self.sesLivres=[]

    def toString(self):
        return "Editeur : " + str(self.num) + " " + self.nom

    def toString(self):
        return "Editeur : " + str(self.num) + " " + self.nom
    def toStringAvecSesLivres(self):
        ch=self.toString()
        for livre in self.sesLivres:
            ch = ch + livre.toString()
        return ch

    def ajouterLivre(self,prmLivre):
        self.sesLivres.append(prmLivre)
    def getSesLivres():
        return self.sesLivres
```

Livre.py

```
# ====================================
class Livre:

    def __init__(self, prmNum, prmTitre):
        self.num = 0
        self.titre= prmTitre

    def toString(self):
        return "Livre : " + str(self.num) + " " + self.titre
```

Cas 3

Cet exemple correspond à l'association bidirectionnelle exposée ci-dessous.

file classes_du_domaine.py

```
class Editeur:

    def __init__(self, nom):
        self.nom = nom
        self.num = 0
        self.sesLivres=[]

    def toString(self):
        return "Editeur : " + str(self.num) + " " + self.nom

    def toString(self):
        return "Editeur : " + str(self.num) + " " + self.nom

    def toStringAvecSesLivres(self):
        ch=self.toString()
        for livre in self.sesLivres:
            ch = ch + livre.toString()
        return ch

    def ajouterLivre(self,prmLivre):
        self.sesLivres.append(prmLivre)

    def getSesLivres():
        return self.sesLivres
# ============================================================
class Livre:

    def __init__(self, prmNum, prmTitre, prmEditeur):
        self.num = 0
        self.titre= prmTitre
```

```
        self.sonEditeur=prmEditeur
        self.sonEditeur.ajouterLivre(self)

    def toString(self):
        return "Livre : " + str(self.num) + " " + self.titre + " " +
self.sonEditeur.nom

    def toStringAvecSonEditeur(self):
        return "Livre : " + str(self.num) + " " + self.titre + " " +
self.sonEditeur.nom
```

Créer un objet

Les classes utilisées dans ce code sont celles décrites ci-dessus (Cas 3).

```
if __name__=='__main__':
    try:
        e1 = Editeur("")
        e2=Editeur("Dunod")
        print(e1.toString())
        print(e2.toString())
        e1.num=2
        e1.nom="Eyrolles"
        print(e1.toString())
        l1 = Livre(1,"Python 3",e1)
        l2=Livre(2,"Uml",e2)
        l3 = Livre(3,"Python 2",e1)
        print(l1.toString())
        print(l2.toString())
        print(l1.toStringAvecSonEditeur())
        print(l2.toStringAvecSonEditeur())

        print(e1.toStringAvecSesLivres())
        print(e2.toStringAvecSesLivres())
    except ValueError:
        # traceback.print_exc()
        print("erreur")
```

Association

Association avec Navigabilité de Livre vers Editeur

Un objet de classe `Livre` pourra obtenir une information sur "son éditeur". Un objet de classe `Editeur` ne peut pas connaître (ou fournir) d'informations sur "ses livres".

Association avec Navigabilité de Editeur vers Livre

Un objet de classe `Editeur` pouura obtenir une information sur "ses livres". Un objet de classe `Livre` ne peut pas connaître (ou fournir) d'informations sur son éditeur.

Association avec Navigabilité bidirectionnelle

Notions de base

Un ouvrage qui parcourt les différentes notions abordées par UML et, à partir de la page 364, vous fournit des exemples de code (certains en Python) sur le passage "diagramme des classes" ←→ Programme.

Un exemple avec graphisme

```
import tkinter as tk
import sys
sys.path.append("F:\\__2012_2013\\_PREPAS\\jeudi\\python\\livres\\appli")
from classes_du_domaine import *
from classes_dao import *
from classes_services import *


class TopLevelTest3:
    def creer_fenetre_ajout_editeur(self):
        self.creer_fenetre_ajout_editeur = self.grid = tk.Toplevel()
        self.creer_fenetre_ajout_editeur.title("Ajouter un Editeur")
#         self.canvas3 = tk.Canvas(self.root2, width=400, height=200)

        self.lblNom = tk.Label(self.creer_fenetre_ajout_editeur,text="Nom
```

```
Editeur")

        self.txtNom = tk.Entry(self.creer_fenetre_ajout_editeur)
        self.lblNom.grid(row=0, column=0)
        self.txtNom.grid(row=0, column=1)
self.btnAjouter=tk.Button(self.creer_fenetre_ajout_editeur,text="Ajouter
l'Editeur" , command=self.ajouterEditeur)
        self.btnAjouter.grid(row=1, column=1)
#         self.enText = Entry(self)
#         self.enText.grid(row=0, column=1, columnspan=3)
        self.creer_fenetre_ajout_editeur.mainloop()

    def ajouterEditeur(self):
        """"""""
        self.service.ajouter(self.txtNom.get)

    def creer_fenetre_liste_editeurs(self):
        self.root3 = self.grid = tk.Toplevel()
        self.canvas3 = tk.Canvas(self.root3, width=400, height=200)

        self.canvas3.pack()
        self.root3.title("Liste des Editeurs")
        self.grid = tk.Frame(self.root3)

        self.root3.mainloop()

    def creer_fenetre_liste_editeurs(self):
        self.root3 = self.grid = tk.Toplevel()
        self.canvas3 = tk.Canvas(self.root3, width=400, height=200)

        self.canvas3.pack()
        self.root3.title("Liste des Editeurs")
        self.grid = tk.Frame(self.root3)

        self.root3.mainloop()

    def __init__(self, prmtrService):
        self.service=prmtrService
        self.root1 = tk.Tk()
        self.grid = tk.Frame(self.root1)
        self.grid.pack()
        self.canvas1 = tk.Canvas(self.root1, width=200, height=200)
        self.canvas1.grid(in_=self.grid,row=0,column=0)
        self.canvas2 = tk.Canvas(self.root1, width=200, height=200)
        self.canvas2.grid(in_=self.grid,row=0,column=1)

        self.root1.title("TopLevelTest")
        self.menubar = tk.Menu(self.root1)
        self.menubar.add_command(label="Ajouter un
```

```
Editeur",command=self.creer_fenetre_ajout_editeur)
        self.menubar.add_command(label="Liste des
Editeurs",command=self.creer_fenetre_liste_editeurs)
        self.menubar.add_command(label="Liste des livres d'un
Editeur",command=self.creer_fenetre_liste_livres_un_editeur)
        self.root1.config(menu=self.menubar)
        self.root1.mainloop()

laDao = DaoEditeurEnMemoire()
leService= ServiceEditeur(laDao)
TopLevelTest3(leService)
```