R308 : Consolidation de la programmation

TPD 2 : Héritage et polymorphisme

1 Des villes et des capitales :

Vous allez réaliser un programme qui permet de gérer des villes, certaines pouvant être des capitales.
Une ville sera repérée par son nom et son nombre d'habitants.
Une capitale est une ville qui sera située dans un pays.

1.1 Réalisez la représentation UML de la classe Ville à partir des informations suivantes :

1. La classe Ville comportera un constructeur qui permet d'initialiser les attributs
2. La classe comportera également des observateurs :

   get_attributs() qui retournera les attributs sous la forme d'un tuple.
   get() qui retournera une chaîné de caractères décrivant tous les attributs d'une ville.
   get_dict() qui retournera les attributs sous la forme d'un dictionnaire.
   get_nom() qui retourne le nom de la ville
   get_nb_habitants() qui retourne le nombre d'habitants de la ville

1.2 Réalisez la représentation UML de la classe Capitale à partir des informations suivantes :

1. La classe Capitale hérite de la classe Ville
2. La classe comportera constructeur qui permet d'initialiser l'ensemble des attributs
3. La classe comportera également des observateurs :

   get_attributs() qui retournera les attributs sous la forme d'un tuple.
   get() qui retournera une chaîné de caractères décrivant tous les attributs d'une capitale.
   get_dict() qui retournera les attributs sous la forme d'un dictionnaire.
   get_pays() qui retourne le nom du pays

1.3 Écrivez les définitions des classes Ville et Capitale

1.4 Complétez le programme principale aux endroits indiqués :

```
print("----------objets simples-----------")
# déclarer les références de 2 villes et 1 capitale
...
# instancier les 3 objets
      # ROSPEZ, 1681 habitants
      # PERROS_GUIREC, 7440 habitants
      # LANNION 19920, habitants, capitale du TREGOR
...
# afficher tous les attributs des villes et des capitales
...
# afficher les noms des villes et capitales
...
print("----------liste de villes et capitale------------")
# déclarer la référence d'un liste de Ville appelée liste_Villes
...
# instanciez la liste des villes
...
# ajouter les 3 villes déclarées individuellement à la liste des villes
...
# ajouter une ville à la liste
# RENNES 220 488 habitants, capitale de la BRETAGNE
...
print("----------affichez attributs des villes et capitales : ")
...
print("----------affichez les noms des villes et capitales : ")
...
print("----------affichez les attributs des villes uniquement: ")
...
print("----------affichez population complète-----------")
```

2 Étudiants et crédits ETCS :

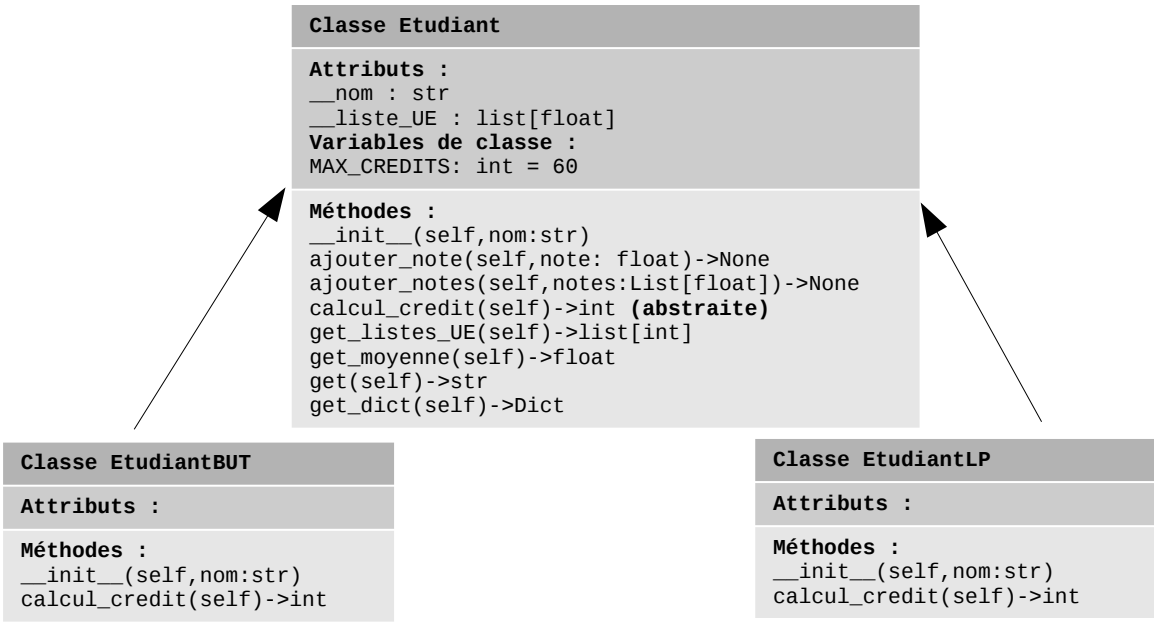
2.1 Présentation :

Selon la formation suivie par un étudiant, le mode de calcul du nombre de crédit ECTS obtenus est différent :

- Pour les étudiants en BUT, si la moyenne générale est supérieure à 10, le nombre de crédit vaut 60.
- Pour un étudiant en LP, seule les notes des UE > 10 permettent d'obtenir des crédits ECTS. Le résultat est normalisé au prorata du nombre d'UE.
  exemple : 5 UE dont 3 supérieures à 10
  ECTS = 3/5 * 60= 36 crédits

2.2 Graphe d'héritage :

Le graphe d'héritage est le suivant :



la méthode calcul_credit() de la classe mère est abstraite, les méthodes sont définies dans les classes dérivées.

2.3 Soit la classe Etudiant dont voici les caractéristiques :

La classe Etudiant possède deux attributs :

- son nom, nommé nom de type chaîne de caractères
- une liste de notes d'UE nommée liste_UE, chaque note est un réel

La classe Etudiant possède un constructeur permettant d'initialiser le nom de étudiant et d'instancier la liste des notes.

La classe Etudiant possède plusieurs méthodes :

- Le modificateur ajouter_note() qui permet d'ajouter une note à la liste.
- Le modificateur ajouter_note**s**() permettra d'ajouter plusieurs notes à partir d'une liste.
- Le modificateur clacul_credit() est abstraite.
- L'observateur get_liste_UE() qui retourne la référence de la liste des notes d'UE. Cette méthodes permettra de calculer le nombre de crédits ECTS d'un étudiant en LP.
- L'observateur get_moyenne() qui retourne la moyenne des UE.
- L'observateur get() retourne l'ensemble des information sous la forme d'une chaîne de caractères, par exemple :

```
BOB (Etudiant_But)
BILL (Etudiant_LP) notes UE : 12.0 15.0 18.0  moyenne : 15.00 ECTS : 60.0
CHUCK (Etudiant_LP) notes UE : 7.0 12.0 15.0  moyenne : 11.33 ECTS : 40.0
```

- L'observateur get_dict() retourne l'ensemble des attributs (y comprit les crédits ECTS) sous la forme d'un dictionnaire.

2.4 Définition des classes :

Écrivez la définition des 3 classes Etudiant, EtudiantBUT et EtudiantLP.

2.5 Complétez le programme principal aux endroits indiqués :

```
# Déclarer une variable de type liste d'étudiants appelée liste_etudiants
...
# Instancier le liste
...
# Ajouter 3 étudiants à la liste
# BOB, étudiant en BUT, "BILL" étudiant en LP et CHUCK étudiant en LP
...
# afficher les étudiants
...
# Ajouter 3 notes d'UE à BOB : 10, 15 et 9
...
# Ajouter 3 notes d'UE à BIL : 12, 15, 18
...
# Ajouter 3 note d'UE également à CHUCK : 7, 12 et 15
...
# afficher les résultats de tous les étudiants en utilisant la méthode get()
...
# afficher le dictionnaire associé à chaque étudiant
…
```

3 Affranchissement de lettres et de colis

Cet exercice se propose de calculer le montant des affranchissements de lettres et de colis déposés dans une boîte aux lettres de l'opérateur de distribution du courrier.

3.1 Proposer un graphe d'héritage.

3.2 La classe Courrier

Les objets déposés (courrier) dans la boîte aux lettres seront modélisés par une classe Courrier.

- Un courrier se définira par son poids (en grammes) et son adresse.
- Un constructeur permettra d'initialiser tous ses attributs.
- Une méthode valide() retournera vrai si l'adresse n'est pas vide.
- La méthode __str__() retournera une chaîne indiquant si le courrier est valide, son poids, sa destination, son prix.
- Une méthode abstraite affranchir() qui ne peut pas être définie dans la classe mère car les règles sont différentes pour les colis et les lettres.
- Un observateur get_poids() retournera le poids.

1. Ecrire le modèle UML de la classe Courrier
2. Ecrire la définition de cette classe

3.3 Classe Lettre :

- Une lettre se distingue par son format (A4 ou A3).
- Un constructeur permettra d'initialiser tous ses attributs.
- La méthode __str__() retournera une chaîne contenant les éléments d'une lettre.
- Une méthode affranchir calculera le prix d'une lettre selon les critères suivants :
  taille A4 : 1€, taille A3 : 2€, on ajoutera au montant le poids (g) / 1000 (g).

1. Ecrire le modèle UML de la classe Lettre
2. Ecrire la définition de cette classe

3.4 Classe Colis

- Un colis se distingue par son volume en litres.
- Un constructeur permettra d'initialiser tous ses attributs.
- La méthode __str__() retournera une chaîne contenant les éléments du colis.
- Une méthode affranchir calculera le prix d'un colis selon la formule suivante :
  0,25* volume + le poids (g) / 1000 (g)

- La méthode valide() qui retourne un booléen, sachant qu'un colis n'est pas valide s'il dépasse 50 litres en plus des règles déjà définies.
  1. Ecrire le modèle UML de la classe Colis
  2. Ecrire la définition de cette classe

3.5 La classe Boite :

La classe Boite permettra de définir la boite aux lettres dans laquelle les courriers sont déposés par les usagers.

- Les différents objets seront ajoutés dans une liste avec une méthode ajouter_courrier().
- Une méthode affranchissement() retournera le montant total des affranchissements des courriers.
- Une méthode courriers_invalides() retournera le nombre de courriers invalides.
- la méthode __str__() permettra d'afficher à l'écran les informations de tous les courriers.
  1. Ecrire le modèle UML de la classe Boite

3.6 L'affichage des caractéristique de chaque courrier sera de la forme suivante :

```
Lettre
      (Courrier invalide)
      Poids : 800.0 grammes
      Destination :
      Prix : 1.8 €
      Format : A4
Colis
      Poids : 5000.0 grammes
      Destination : Place des Mouettes 29670 Taulé
      Prix : 12.5 €
      Volume : 30.0 litres
Colis
      (Courrier invalide)
      Poids : 3000.0 grammes
      Destination : Stade Jean Dauger 64100 Bayonne
      Prix : 20.5 €
      Volume : 70.0 litres
```

3.7 Compléter le programme principal aux endroits indiques :

```
# declarer trois références d'objets de type Courrier
...
# instancier les trois courriers :
      # Lettre de 800 grammes sans adresse au format A4
      # Colis de 5 kg dont l'adresse est "IUT de Lannion" avec un volume de 30 litres
      # Colis de 3 kg dont l'adresse est "Place des Mouettes" avec un volume est 70 litres
...
# afficher les caractéristiques des trois courriers
...
# seconde partie
print("-------------gestion d'un boite-----------------------")
# déclarer une référence de boite de courriers
...
# instancier le boite
...
# ajouter les 3 courriers à la boite
...
# ajouter un nouveau courrier
...
#afficher les caractéristiques des courriers de la boite
...
# afficher le nombre de courriers invalides de la boite
...
```