TD10: Programmation Orientée Objet en python

Notions sur l'encapsulation et l'héritage :

- *Encapsulation :* L'encapsulation consiste à masquer des données de l'état interne pour protéger l'intégrité de l'objet. Les opérations (getters et setters) sont les seules moyens de manipuler l'état interne d'un objet. Pour encapsuler un attribut on le précède par « __ ». Exemple « self.__titre = titre ». Il est possible aussi de masquer une méthode, exemple « def __ methode_privee(self) ».

- *Héritage :* L'héritage permet de définir l'implémentation d'une classe à partir de l'implémentation d'une autre. Ce mécanisme permet, lors de la définition d'une nouvelle classe, de ne préciser que ce qui change par rapport à une classe existante. Une hiérarchie de classes permet de gérer la complexité, en ordonnant les classes au sein d'arborescences d'abstraction croissante. Si B hérite de A, on dit que B est une classe fille (sous-classe, classe dérivée) et que A est une classe mère (super-classe, classe de base). B hérite de tous les attributs et toutes les méthodes de A. Python (contrairement à JAVA) supporte l'héritage multiple. Donc une classe peut hériter de plusieurs classes et peut être la super-classe de plusieurs autres classes.

Exo1 : Application des notions

Dans ce qui suit, les classes doivent respecter le principe de l'encapsulation.

1) Créez une classe ***« Titre »*** avec l'attribut « titre ». Cette classe doit contenir un constructeur, une méthode «getTitre(self) » puis une méthode « Afficher(self) ». Créez une deuxième classe ***« Annee »*** avec l'attribut « annee », cette classe doit contenir un constructeur et deux méthodes (comme la classe « Titre »).

2) Créez une classe ***« CD »*** qui hérite de la classe « Titre » et de la classe « Annee ». Cette classe, en plus de l'attribut année et titre qu'elle hérite des superclasses « Titre » et « Annee », doit contenir un 3 ème attribut « nomArtiste » puis un 4 ème « NbreTitres ». Ajoutez les getters des deux nouveaux attribut et une méthode « Afficher(self) » qui afficher tous les attributs d'un CD.

3) Créez une classe ***« DVD »*** qui hérite de la classe « Titre » et de la classe « Annee ». Cette classe, en plus de l'attribut année et titre qu'elle hérite des superclasses « Titre » et « Annee », doit contenir un 3 ème attribut « realisateur». Rajoutez un getter du réalisateur puis une méthode « Afficher(self) » qui affiche le DVD.

4) Créez une classe ***« Collection ».*** Cette classe nous permettra de stocker plusieurs CDs ou DVDs. L'attribut de la classe est une liste que vous initialisez à « vide » dans le constructeur.

Dans ce qui suit, les méthodes sont à implémenter dans la classe « Collection »

5) Créez une méthode « Ajout(self,objet) » qui ajoute un objet qui peut être soit un CD soit un DVD. L'objet n'est ajouté que s'il n'est pas déjà dans la liste (on suppose que les objets ont un « titre » unique). Créez une méthode « Afficher(self) » qui affiche tous les objets d'une collection.

6) Créez une méthode «Rechercher_Titre(self,titre:str) » qui affiche l'objet dans la collection possédant comme titre le titre passé en argument. Utilisez la méthode « liste.index(titre) », et traitez l'exception « ValueError» levée dans le cas où « titre » n'est pas dans la liste.

7) Créez une méthode «Rechercher_Annee(self,annee:int) » qui affiche tous les objets dont l'année de sortie est l'année passé en argument.

8) Créez une méthode « Rechercher_Real_Annee(self, real:str, annee:int) » qui affiche tous les DVDs du réalisateur « real » qui ont comme année de sortie « annee ».

9) Test du programme : Créez deux CDs et trois DVDs, assurez-vous d'avoir un réalisateur avec au minimum deux DVDs la même année.