TP5 : PARADIGME DE PROGRAMMATION

Exercice1

- Définir une classe Rectangle avec un constructeur donnant des valeurs (longueur et largeur) par défaut et un attribut nom=’rectangle’, une méthode d’affichage et une méthode surface renvoyant la surface d’une instance
- Définir une classe Carre héritant de Rectangle et qui surcharge l’attribut d’instance :

Nom= « carré »

Dans le programme principal, instanciez un Rectangle et un carré et afficher les

- Définir une classe Point avec un constructeur fournissant les coordonnées par défaut d’un point du plan. (par exemple : x=0.0 ET y=0 .0).
- Définir une classe Segment dont le constructeur possède quatre paramètres : deux pour l’origine et deux pour l’extrémité. Ce constructeur définit de deux attributs : orig et exterem, instance de la classe Point. de cette manière, vous concevez une classe composite : la classe Segment est composée de deux instances de la classe Point
- Ajouter une méthode d’affichage
- Enfin écrire un auto-test qui affiche une instance de Segment initialisée par les valeurs 1, 2,3et4

Exercice2

On veut implémenter en python, une application pour la gestion d’une bibliothèque. Une conception orienté objet partielle du problème nous conduit vers la décomposition suivante :
Définir la classe **Date**, qui permet de représenter une date sous la forme jour, mois, et année (3entier naturels). Proposer une implémentation Python pour cette classe qui doit répondre aux spécifications suivantes :

1. **Classe Livre** qui permet de représenter la fiche d’un livre dans la bibliothèque.

Proposer une implémentation Python pour cette classe en respectant les directives suivantes :

- La classe doit contenir un attribut de classe **privé num_livres** initialisé à 0 et utilisé par le constructeur afin de générer un identifiant unique pour chaque livre (incrémenté après chaque appel du constructeur).
- Un constructeur qui reçoit deux paramètres (**t** et **nb_ex**) et qui permet d’initialiser une nouvelle instance de cette classe avec les attributs suivants :
    - **num**: attribut public initialisé à partir de l’attribut de classe **num_livres**
    - **titre** :un attribut d’instance public contenant le titre du livre initialisé par le paramètre **t**

- **nb_exemplaires** : un attribut d'instance **privé** indiquant le nombre d'exemplaires actuellement disponibles dans la bibliothèque initialisé par le paramètre **nb_ex** .

- La méthode magique**__str__** : permettant de représenter un livre sous forme d'un str par exemple :

« Livre N°5 ;intitulé : Apprendre à programmer avec Python 3;5exemplairedisponibles ».

- Une méthode **est_disponible(self)** qui retourne un booléen permettant d'indiquer s'il y a un exemplaire disponible en stock.
- Une méthode **retirer_un_exemplaire(self)** permettant de retirer un seul exemplaire du stock disponible. Prévoir une exception dans le cas où le stock est déjà épuisé.
- Une méthode **retour _exemplaire(self)** permettant d'ajouter un seul exemplaire au stock disponible

2. **Classe Abonné**, qui représente les fiches des abonnées de la bibliothèque. Proposer une implémentation Python pour cette classe en suivant les indications ci-dessous :

- Un attribut de **classe privé** de type **liste** contenant tous les numéros des cartes d'identités des abonnées déjà instanciés. Le constructeur doit utiliser cet attribut pour s'assurer de **l'unicité** du numéro de l'instance en cours de création et doit lever une exception si ce n'est pas le cas.
- Un constructeur qui reçoit **deux** paramètres **NCIN** : une chaine formée par **8 chiffres**, **Nom :** une chaine), le constructeur doit vérifier **l'unicité** de l'attribut **NCIN** par rapport aux instances déjà crée ensuite il créé les attributs d'instance suivants :
  - **Ncin :** l'identifiant unique de l'abonné(**privé**) initialisé par le paramétré NCIN
  - **Nom** : le nom de l'abonné initialisé par le paramètre Nom.
  - **Est_pénalisé** : un **attribut privé** de type booléen permettant d'indiquer si l'abonné est pénalisé ou pas. Un abonné initialement créé **n'est pas pénalisé**.
  - **Nb_emprunt** : un **attribut privé** de type entier indiquant le nombre de livres empruntés par l'abonné. Un abonné initialement créé n'a aucun livre en sa possession.
- la méthode magique__str__ : permettant de représenté un Abonné sous forme d'une chaine de caractère par exemple :

  « NCIN :05344430 ;Nom :Foulen Ben Foulen ; état : **non** pénalisé »
- Une méthode **est_pénalisé(self)** retournant un booléen indiquant si l'abonné est actuellement pénalisé ou pas
- Une méthode **pénaliser(self)** qui permet de pénaliser l'abonner.

- Une proprety **NCIN** permettant d'accéder en lecture seule à **l'attribut privé NCIN**
- Une méthode **marquer_retour**(self) permettant de marquer le retour d'un livre de la part de l'abonné en décrémentant le nombre de livres en sa possession.
- Une méthode **marquer_emprunt**(self) permettant de marquer l'emprunt d'un nouveau livre par l'abonné en incrémentant par conséquent le nombre de livres en sa possession.

3. Classe **Abonné_restreint** qui étend (**hérite d'Abonné**) et qui représente une catégorie spéciale d'abonné dont le nombre maximum d'emprunts est limité à une **borne_max_emprunt**, cette borne est passé au constructeur avec les autres informations nécessaires pour l'initialisation d'un abonné. Redéfinir les méthodes nécessaires afin de tenir compte de cette particularité (lever une exception si tentative d'emprunt dans le cas où le nombre maximum d'emprunts est déjà atteint

4. Classe **Emprunt**, une instance de cette classe représente une opération d'emprunt d'un seul exemplaire d'un livre par un abonné de la bibliothèque .proposer une implémentation Python pour cette classe en respectant les consignes suivantes :

Le constructeur de cette classe doit recevoir **trois** paramètres supposé valides (le numéro du livre emprunté, le CIN de l'abonné et la date de l'emprunt).ces 3 paramètres sont utilisés pour initialiser les **cinq** attributs de l'instances crée :

- **Num_livre :le numéro de livre emprunté**
- **Ncin_abonné : le NCIN de l'abonné**
- **De : la date de l'emprunt**
- **Dpr : la date prévu du retour= date de l'emprunt- 10jours**
- **Pénalité : un bool** qui indique si cette opération est pénalisée (dans le cas où la date prévus de retour est dépassée). Initialement, l'opération n'est pas pénalisée

- Une méthode **maj_pénalit(self,da)** : qui permet de mettre à jour la pénalité suivante la dpr et la date actuelle da

5. Classe **bibliothèque** qui représente l'application de la bibliothèque. Proposer une implémentation Python pour cette classe qui doit respecter les consignes suivantes :

- Le constructeur de cette classe est sans paramètres il doit initialiser les attributs d'instance suivants :
  - **Livres** : un dictionnaire (initialement vide) qui contient les fiches des titres disponibles ; les clés de ce dictionnaire sont les numéros des livres et les valeurs sont les instances de la classe livre qui encapsulent les informations associées

- **Emprunteurs** : un dictionnaire (initialement vide) contenant les fiches des abonnés de la bibliothèque. Les clés de ce dictionnaire sont les NCIN des abonnés et les valeurs associées sont des instances de la classe Abonné
- **Opérations** : un dictionnaire (initialement vide) contenant les opérations d'emprunt en cours. Les clés de ce dictionnaire sont des paires (des tuples de taille 2) contenant le ncin de l'abonnée et le numéro du titre emprunté, les valeurs associés sont des instances de la classe Emprunt

- Une méthode **ajouter_livre (self, l)** permettant d'ajouter l, une instance de la classe Livre à la bibliothèque (et de mettre à jour le dictionnaire concerné).
- Une méthode **ajouter_abonné (self,a)** permet d'ajouter un abonnée a et de mettre à jour le dictionnaire concerné
- Une méthode **magique _ _str_ _** qui permet de représenter l'inventaire de la bibliothèque sous format str
- Une méthode **livres_empruntés (self, ncin)** qui retourne l'ensemble (instance de la classe set) des numéros des livres empruntés par l'abonné ayant le numéro ncin
- Une méthode **ont_emprunté (self, num_liv)** qui retourne l'ensemble des ncin des abonnés qui ont emprunté le livre ayant le numéro **num_liv**
- Une méthode **emprunter (self, ncin, num_liv, date_emprunt)** permettant à l'abonné désigné par **ncin** (si ce dernier est présent dans le dictionnaire d'abonné, s'il n'est pas pénalisé et s'il ne possède pas déjà le même livre en sa possession) d'emprunter un exemplaire du livre identifié par **num_liv** si ce dernier est disponible en stock. L'opération est marquée (marquer_emprunt), ensuite le livre emprunté est déduit du stock et le dictionnaire des opérations est mis à jour.si le livre n'est pas en stock alors une erreur est déclenchée.