UNIVERSITÉ RENNES 2

Programmation Orientée Objet : Héritage, surcharge et polymorphisme

Xavier André & Romain Tavenard

1 Rappel : organisation de votre code

Pour ce TD, vous créerez un nouveau fichier `td_poo_b.py`. Dans ce fichier, votre code sera organisé de la manière suivante :

```
# Imports
from abc import ABC, abstractmethod

# Classes et Fonctions
class [...]

# Tests
[...]
```

Notamment, vous définirez vos classes fonctions en début de fichier et les appels seront listés en fin de fichier. De cette manière, vous pourrez, d'une question à l'autre, réutiliser les classes et fonctions déjà codées au besoin.

2 Compte bancaire simple

Nous nous intéressons ici à un compte simple caractérisé par un solde exprimé en Euros qui peut être positif ou négatif.

1. Créez une classe `CompteSimple` respectant les caractéristiques ci-dessus.
2. Surchargez la méthode `__repr__()` afin d'obtenir l'affichage suivant :

> Le solde du compte est de XXX Euro(s).

3. Créez une méthode `enregistrerOperation()` qui permet de créditer le compte ou de le débiter.
4. Testez cette classe.

3 Compte bancaire courant

Une banque conserve pour chaque compte l'historique des opérations qui le concernent (on se limite ici aux opérations de crédits et de débits). On souhaite modéliser un tel compte qu'on appelle compte courant. En plus des méthodes d'un compte simple, un compte courant offre les méthodes suivantes :

— `afficherReleve` : affiche l'ensemble des opérations effectuées ;
— `afficherReleveCredits` : affiche seulement les opérations de crédit ;
— `afficherReleveDebits` : affiche seulement les opérations de débit.

Pour représenter l'historique, on utilisera une liste. Pour enregistrer une opération, on conservera simplement le montant de l'opération (Crédit : positif ; Débit : négatif).

1. Créez la classe `CompteCourant` correspondant aux spécifications ci-dessus.
2. Testez cette classe.
3. Ajoutez des attributs calculés `releveCredits` et `releveDebits` et mettez à jour les méthodes `afficherReleveCredits` et `afficherReleveDebits` pour qu'elles reposent sur ces attributs.

4 Classe abstraite

On reprend maintenant l'exemple de la classe `Intervalle` définie plus haut. On souhaite être capable de prendre en compte deux types d'intervalles : les intervalles ouverts (pour lesquels les bornes ne sont pas incluses dans l'intervalles) et les intervalles fermés.

1. Définissez une classe `IntervalleAbstrait` qui hérite de la classe `ABC` et contient deux méthodes :
   — une méthode `__init__` qui prend en entrée les deux bornes de l'intervalle,
   — une méthode `__repr__`,

— une méthode abstraite `__contains__` qui indique si une valeur passée en argument est considérée comme faisant partie de l'intervalle ou non.

2. Définissez une classe `IntervalleOuvert` qui hérite de `IntervalleAbstrait` en ne redéfinissant que la ou les méthodes strictement nécessaire(s).
3. Définissez une classe `IntervalleFerme` qui hérite de `IntervalleAbstrait` en ne redéfinissant que la ou les méthodes strictement nécessaire(s).