CNAM
Ecole d'ingénieur spécialité Systèmes électroniques - SETI2

Java et Objet – USSE09

TP La programmation orientée objet

Exercice 1 — *Gestion de comptes bancaires*

Une agence commande un logiciel de gestion de ses comptes bancaires. Chaque `Compte` est caractérisé par le nom de son `titulaire`, son `solde`, et son `identifiant` unique. A la création d'un compte, son `identifiant` est généré automatiquement et correspond au nombre total de comptes créés. Les opérations possibles sur les comptes sont : `débiter`, `créditer` et obtenir le solde du compte. On ajoutera une opération permettant de représenter un compte sous la forme d'une chaîne de caractères. La méthode `toString()` de la classe `Object` fournit une représentation de tout objet (instance) sous la forme d'une chaîne de caractères. Il suffit de la redéfinir dans chaque classe représentant un compte pour obtenir la représentation d'une instance de compte sous la forme d'une chaîne de caractères. Plus précisément, on souhaite que la méthode `toString()` retourne une chaîne de caractères d'une instance de `Compte` qui aura la forme suivante :

```
*************************
compte n° : xxxxxx
titulaire     :  Biloute
solde        :  521 euro
*************************
```

Les comptes peuvent être de plusieurs types : `Courant`, `Securise`, `Remunere` et `RemunereSecurise`, dont les caractéristiques spécifiques sont listées ci-après :

- Un compte `Courant` peut être crédité ou débité.
- Un compte `Securise` peut également être crédité, mais ne peut être débité que si le solde est suffisant.
- Un compte `Remunere` peut être crédité ou débité, et propose une opération supplémentaire de calcul d'intérêts sur la base d'un taux propre à chaque compte rémunéré qui, lorsqu'elle est invoquée, ajoute les intérêts au solde courant. On se contentera d'un calcul simple non réaliste des intérêts effectué en multipliant le solde actuel avec le taux.
- Un compte `RemunereSecurise` possède les caractéristiques d'un compte Remunere, mais ne peut être débité que si le solde est suffisant.

On devine ici la relation d'héritage qui lie les différents comptes et qui est décrite dans la Figure 1.

En structurant ainsi les classes représentant les comptes bancaire, il sera ensuite facile de concevoir une agence bancaire qui sera structurée comme suit :

- Une agence regroupe l'ensemble des comptes créés (de tous types) afin d'en assurer la gestion. Elle aura donc un attribut qui sera une tableau de `Compte`. Chaque élément de ce tableau pourra ensuite être instancié sous forme d'un compte `Courant`, `Securise`, `Remunere` ou `RemunereSecurise`.
- Une agence est identifiée par un `nom`.
- Une agence est en mesure d'`ajouter`, de `supprimer` et d'`extraire` un compte de son tableau de `Compte` en connaissant le numéro du compte à extraire.

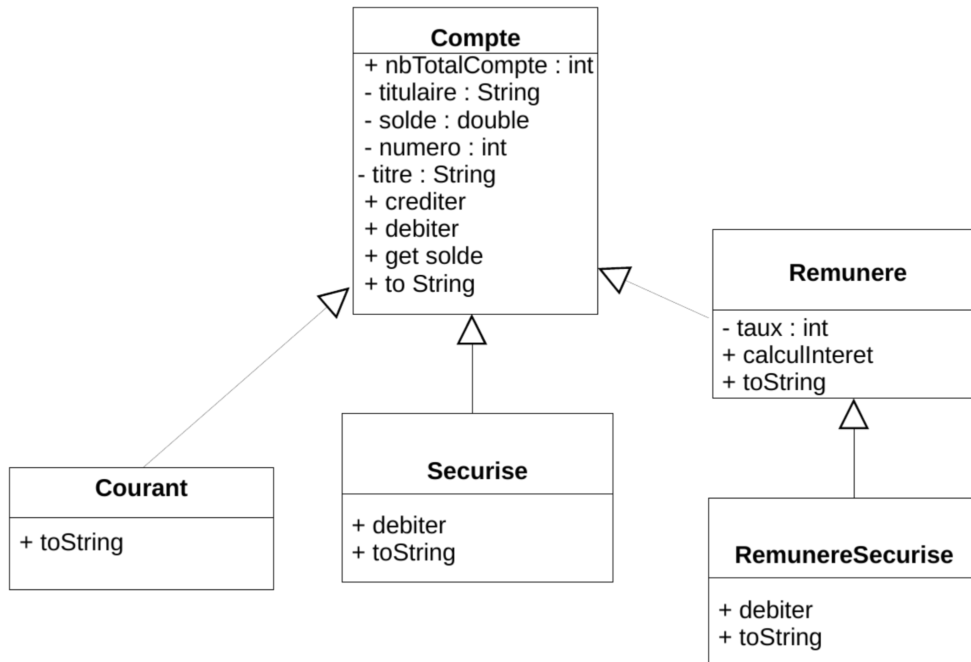

FIGURE 1 – Diagramme de classe de la Banque

— La classe `Agence` aura également une méthode `toString()` pour obtenir la représentation d'une instance d'agence (et donc de l'ensemble de comptes qu'elle détient) sous la forme d'une chaîne de caractères.

(1) Implanter les différents comptes sous la forme de classes Java et tester chaque classe de manière unitaire.
(2) Implanter la classe `Agence`. Modifier la classe principale (`main`) pour tester la classe `Agence`.

Exercice 2 — *Star Wars*

On souhaite programmer un jeu vidéo sur le thème de Star Wars, où les héros sont des objets de la classe `Personnage` qui sont spécialisés soit en `Jedi`, soit en `SoldatClone`. Ces personnages seront équipés en armes selon leur type : un `Sabre` pour un `Jedi`, et un `Blaster` pour un `SoldatClone`, et pourront réaliser des combats. Les relations entre les personnages et les armes sont décrites dans la Figure 2.

Nous décrivons les armes comme suit :
— Une `Arme` est soit allumée, soit éteinte (`boolean allume`) et possède une puissance de frappe (`puissance`). Il est possible de l'allumer ou de l'éteindre via les méthodes `void allumer()` et `void eteindre()`.
— Un `Sabre` est une arme spécialisée qui a une puissance de 10 et une couleur (on peut utiliser la classe `java.awt.Color`).
— Un `Blaster` est une arme spécialisée qui a une puissance de 5.

Pour les personnages, on a :
— Un `Personnage` possède un `nom`, une quantité de points de vie (`vie`), une position indiquée par son abscisse et son ordonnée (`x` et `y`), une vitesse de déplacement (`vitesse`), et une force pour blesser un autre personnage (`force`).
— Il peut se blesser avec la méthode `void seBlesse(Arme a, Personnage p)` qui décrémente le nombre de points de vie du personnage blessé de la puissance de l'arme `a` par la force du personnage `p` qui le blesse.

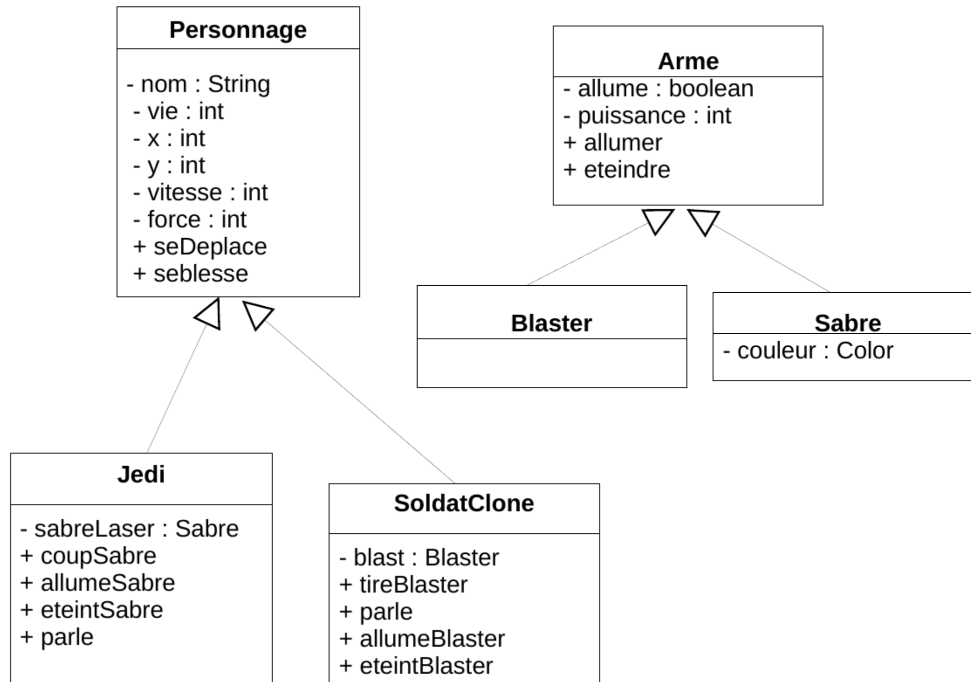

FIGURE 2 – Diagramme de classe de Star Wars

— La méthode `String toString ()` construit une chaine de caractère qui indique l'état d'un personnage.
— ***(Bonus)*** Il peut se déplacer dans le plan grâce à la méthode `void seDeplace(int dx, int dy, int dt)` qui déplace le personnage dans la direction (`dx`, `dy`) pendant l'intervalle de temps `dt`.

Les personnages se spécialisent en :
— `Jedi` qui sont des sortes de personnages qui parlent (`void parler()`) en disant "Que la force soit avec vous" et qui ont tous une vitesse de $5km/h$. Un `Jedi` a toujours un `Sabre` laser qu'il peut allumer ou éteindre. Il peut donner un coup de Sabre via la méthode `void coupSabre(Personnage p)` s'il a pensé à allumer son sabre laser.
— `SoldatClone` qui sont des sortes de personnages qui parlent (`void parler()`) en disant "Je m'appelle" et leur `nom`. Ils ont toujours 100 points de vie, une vitesse de $4km/h$ et une force de 2. Un `SoldatClone` a toujours un `Blaster` qu'il peut allumer ou éteindre. Il peut tirer avec son `Blaster` via la méthode `void tireBlaster(Personnage p)` s'il a pensé à allumer son blaster.

1. Implémentez les différentes classes présentées.
2. Ecrivez une classe `Principale` qui après avoir créé des personnages teste les méthodes implémentées.