Comptes bancaires : une vue simplifiée

Objectif : Comprendre au travers d'un exemple simplifié de comptes bancaires, les concepts d'encapsulation, de propriétés d'instances et de classes, d'héritage, de surcharge et de redéfinition de méthodes, de polymorphisme et de liaison dynamique.

Exercice 1 : Compte bancaire simple

Nous nous intéressons à un compte simple caractérisé par un solde exprimé en euros, positif ou négatif, et son titulaire. [1] Il est possible de créditer ce compte ou de le débiter d'un certain montant.

1.1 Spécifier la classe `CompteSimple`. Sur le diagramme de classe, on fera apparaître aussi la classe `Personne` (dont la documentation [2] est fournie dans le listing 1).

1.2 Proposer un programme de test de la classe CompteSimple.

1.3 Écrire la classe CompteSimple.

Exercice 2 : Compte courant

En fait, une banque conserve, pour chaque compte, l'historique des opérations qui le concernent (on se limite ici aux opérations de crédit et de débit). On souhaite modéliser un tel compte qu'on appelle compte courant. En plus des méthodes d'un compte simple, un compte courant offre des méthodes pour afficher l'ensemble des opérations effectuées (`éditerRelevé`) ou seulement les opérations de crédit (`afficherRelevéCrédits`) ou de débit (`afficherRelevéDébits`).

Pour représenter l'historique, on utilisera la classe `Historique` fournie (listing 2). Pour enregistrer une opération, on conservera simplement le montant de l'opération précédé d'un signe indiquant s'il s'agit d'un crédit (montant positif) ou d'un débit (montant négatif).

2.1 Indiquer quelle est la relation entre un compte simple et un compte courant et compléter le diagramme de classes UML.

2.2 Écrire la classe `CompteCourant` correspondant à un compte courant.

2.3 Le listing 3 propose un exemple d'utilisation des comptes bancaires. Exécuter et commenter ce programme.

2.4 Quels tests supplémentaires faudrait-il faire ?

Exercice 3 : La banque

Une banque est bien entendu un organisme qui gère un grand nombre de comptes, qu'ils soient simples ou courants.

1. Nous simplifions le problème en considérant que tout compte a un et un seul titulaire correspondant à une personne physique modélisée par une classe Personne fournie.
2. Il s'agit de la documentation telle qu'elle pourrait être engendrée par `javadoc` mais présentée sous la forme d'une classe.

Listing 1 – Documentation de la classe Personne

```
/**
 * Une personne est simplement définie par son nom, son prénom et son sexe qui
 * sont les informations nécessaires pour pouvoir la construire.
 */
public class Personne {

   /** Construire une personne à partir de nom, son prénom et son sexe.
    * @param prenom_ le prénom de la personne
    * @param nom_ le nom de la personne
    * @param masculin_ est-ce un homme ?
    */
   public Personne(String prenom_, String nom_, boolean masculin_);

   /** Le nom de la personne.
    * @return le nom de la personne
    */
   public String getNom();

   /** Le prénom de la personne.
    * @return le prénom de la personne
    */
   public String getPrenom();

   /** La personne est-elle un homme ?
    * @return la personne est-elle un homme ?
    */
   public boolean estHomme();

   /** La personne est-elle une femme ?
    * @return la personne est-elle une femme ?
    */
   public boolean estFemme();

   /** Afficher le nom et le prénom. */
   public void afficher();
}
```

Listing 2 – Documentation de la classe `Historique`

```
/** Historique gère un historique chronologique des entiers enregistrés.
*/
public class Historique {

   /** Construire un historique vide. */
   public Historique();

   /** Enregistrer une nouvelle information dans l'historique
    * @param info l'information à enregistrer dans l'historique
    */
   public void enregistrer(double info);

   /** La i<SUP>è</SUP> valeur de l'historique, 1 correspond à la plus
    * ancienne, getNbValeurs() à la plus récente (la dernière).
    *
    * <b>Attention :</b> Cette convention est différente de celle
    * traditionnellement adoptée en Java pour les tableaux et vecteurs !
    *
    * @param i indice de l'opération compris en 1 et getNbValeurs().
    */
   public double getValeur(int i);

   /** Le nombre d'entiers enregistrés dans l'historique
    * @return le nombre d'entiers dans l'historique
    */
   public int getNbValeurs();
}
```

Listing 3 – Programme utilisant les comptes

```
/** Programme utilisant les comptes bancaires. */
public class ExempleComptes {
   public static void main (String argv []) {
      Personne p1 = new Personne("Xavier", "Crégut", true);
      CompteSimple cs1 = new CompteSimple(p1, 0);
      CompteCourant cc1 = new CompteCourant(p1, 100);

      cs1.crediter(1000);
      System.out.println("Solde de cs1 = " + cs1.getSolde());

      cc1.crediter(1000);
      System.out.println("Solde de cc1 = " + cc1.getSolde());
      cc1.editerReleve();

      CompteSimple cs = cc1;
      cs.debiter(500);
      System.out.println("Solde de cs = " + cs.getSolde());
      System.out.println("Solde de cc1 = " + cc1.getSolde());
      cc1.editerReleve();
   }
}
```


3.1 *État de la classe `Banque`.* Sachant que tous les comptes sont rangés dans un unique tableau, indiquer quels sont les attributs de la classe `Banque`. Donner les constructeurs de la classe `Banque`.

3.2 *Ouverture des comptes.* Définir sur la classe `Banque` une méthode pour ouvrir un compte simple et une méthode pour ouvrir un compte courant.

3.3 *Cumul des soldes des comptes.* Définir une méthode sur la banque qui donne le cumul des soldes disponibles sur chacun des comptes. D'autres informations pourraient être fournies telles que le nombre de comptes débiteurs, la somme des débits, etc.

3.4 *Frais de tenue de compte.* Nous considérons que la banque prélève périodiquement des frais de tenue de compte (par exemple pour l'envoi des relevés de compte). Écrire une méthode qui débite sur tous les comptes la somme de 2 euros.

3.5 *Édition des relevés.* Écrire une opération qui édite (à l'écran) les relevés de tous les comptes courants (et seulement des comptes courants, le relevé n'a pas de sens pour un compte simple).

3.6 *Autre organisation des comptes.* Proposer une autre organisation des comptes qui faciliterait l'édition des relevés. Discuter cette solution.

Exercice 4 : Numéro de compte

Pour les différencier, chaque compte possède un numéro unique. On suppose que les numéros sont des entiers et qu'ils sont attribués par ordre croissant en commençant à 10001. Les numéros de compte sont donc 10001, 10002, 10003, etc. Dans la suite, nous envisageons deux solutions pour attribuer les numéros de compte.

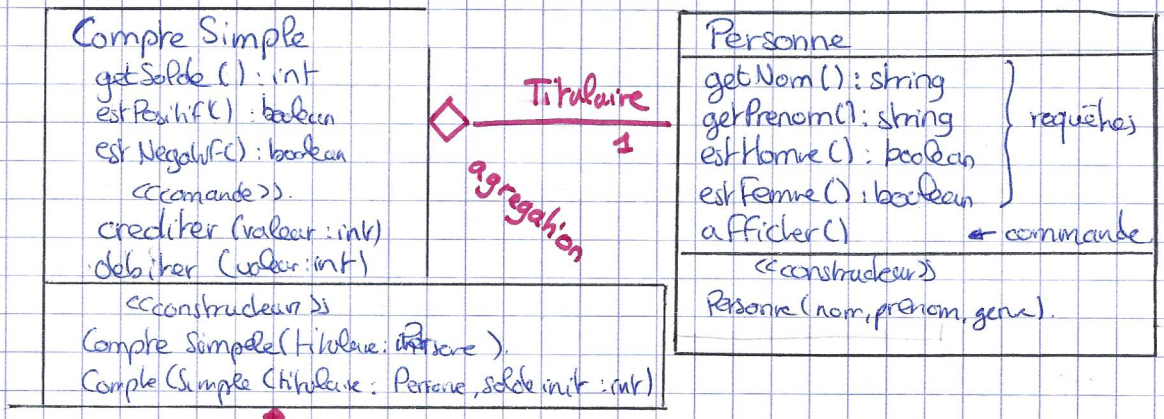
4.1 On souhaite que l'attribution du numéro de compte soit de la responsabilité des classes `CompteSimple` et `CompteCourant`. Indiquer les modifications à apporter à ces classes.

4.2 On suppose maintenant que c'est la banque qui gère et attribue les numéros de compte. Indiquer les modifications à apporter aux classes `CompteSimple` et `CompteCourant`.

TD3.1
TOB

Comptes bancaires : une vue simplifiée

Exercice 1 : Compte bancaire simple

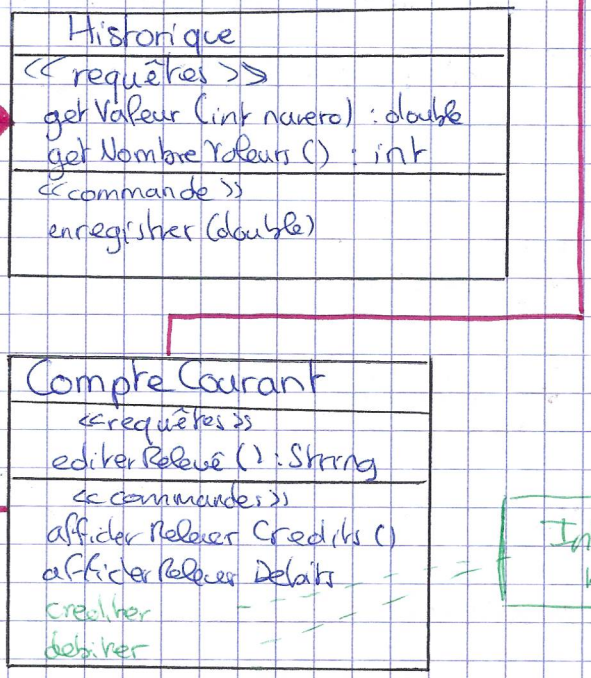


Exercice 2 : Compte Courant

relation d'Héritage

Protected : droit de l'utiliser dans les sous classes
Private : Pas le droit de l'utiliser dans les sous classes → passer par get...

composition



Si des méthodes ont des comportements similaires on ne les appelle pas de la même manière.

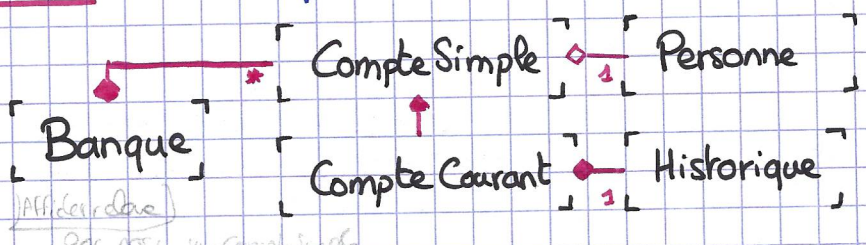
Introduire gestion historique

```
public class CompteCourant extends CompteSimple {

    private Historique historique;

    public CompteCourant (Personne titulaire) {
        this (titulaire, 0);
    }

    public CompteCourant (Personne titulaire, double solde) {
        super (titulaire, solde);
        this.historique = new Historique();
        if (solde > 0) { this.historique.enregistrer(solde); }
    }
```

```
public void crediter (double valeur){
        super.crediter (valeur );
        this.historique.enregistrer (valeur );
}
public void debiter (double valeur ){
        super.debiter ( valeur );
        this.historique.enregistrer (- valeur );
}
void afficherReleve (){
    for (int indice =0; indice < this.historique.getNombreValeur () ) {
                                                          indice++ ) }.
        System.out.println ( ((this.historique .getValeur ( indice ) >=0)?
                "Credit : ":"Debit : ") + this.historique.getValeur (indice)
    }.
}.
```

Devrait utiliser un itérateur fourni par historique

Exercice 3 : La banque



Composition / Agrégation ...

```
if ( compte instance of CompteCourant){
        ((CompteCourant) compte). afficherReleve ()
```

Il vaut mieux mettre la création des comptes de la banque.

pour pouvoir savoir différencier