TP n° 1
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

Version avec éléments de correction

1 Représentation d'un système bancaire (simplifié)

Pour cet exercice, portant sur le domaine bancaire, chaque classe développée possédera une méthode `toString()`. Sa réalisation comporte plusieurs étapes :

1. Créer les classes `CompteBancaire` et `Client` conformément à la documentation fournie sur Moodle [1] veillant au principe d'encapsulation et en vous aidant de la classe `TestCompteBancaire` fournie.
2. Puis définir une nouvelle classe `CompteBancaireRémunéré` qui héritera de la classe `CompteBancaire`. Cette classe possédera un attribut représentant le taux de rémunération et un attribut de classe indiquant un taux d'intérêt par défaut (par exemple 3 %) à utiliser lorsque celui-ci n'est pas précisé lors de la création du compte. Prévoir trois constructeurs :
   — un constructeur permettant d'initialiser le solde sans préciser le taux d'intérêt, la valeur prise sera alors la valeur précisée par l'attribut de classe ;
   — un constructeur permettant d'initialiser le solde et le taux d'intérêt ;
   — un constructeur initialisant le solde à zéro et le taux d'intérêt à la valeur par défaut fixée arbitrairement à 3 %.
3. Définir une méthode `crediterInteretMensuel()` permettant de créditer le compte des intérêts mensuels générés par la valeur du solde (pour rendre la méthode plus simple, mais inexacte, elle créditera le compte du douzième des intérêts annuels).
4. Tester votre implémentation de la classe `CompteBancaireRémunéré` avec la classe `TestCompteBancaireRémunéré`.
5. Créer deux nouvelles classes `Particulier` et `Entreprise` héritant de `Client` en définissant un attribut spécifique à chacune de ces sous-classes (ex. prénom pour un particulier, numéro de SIRET pour une entreprise).
6. Concernant la classe `CompteBancaire`, pour que le numéro de compte soit attribué automatiquement (sans devoir être fourni en paramètre), vous utiliserez une variable de classe, qui sera incrémentée à chaque création d'un compte et dont la valeur servira à initialiser le numéro de compte.
7. Puis créer une classe `Banque` permettant de gérer à la fois les clients et leurs comptes (ajout et suppression).

1. Consulter la documentation en ouvrant le fichier `index.html` en premier.

Pour éviter de devoir associer systématiquement les clients à la banque ainsi que les comptes créés, la classe `Banque` offrira des méthodes qui prendront en charge la création des clients et des comptes ainsi que leur association à la banque (pour les noms de méthodes se reporter à la classe `TestBanque` fournie).

La figure 1 ci-dessous montre les différentes classes ainsi que leurs relations.

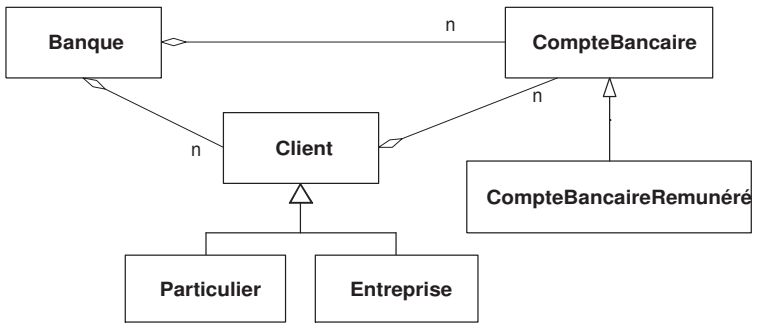

FIGURE 1 – Les différentes classes du système bancaire

Les associations débutant par un « losange » sont des relations dites d'agrégation et signifient « peut contenir » ou « peut posséder », la cardinalité s'exprimant à l'extrémité de l'association (ici `n`). Ainsi on peut déduire de ce diagramme qu'une banque peut posséder plusieurs (`n`) clients et plusieurs comptes et qu'un client peut posséder plusieurs comptes.

La cardinalité est importante pour déterminer le type de structure de données à utiliser. Ainsi une cardinalité `n` se traduira par la présence d'une liste (ou d'un tableau) dans la classe « portant » le losange. Le type de la liste correspondra à la classe située à l'autre extrémité de l'association. Par exemple, on peut déduire de ce diagramme que la classe `Banque` possédera une liste de type `Client`.

8. Compléter la classe `TestBanque` fournie pour vérifier les contraintes suivantes :
   — un client ne peut pas ouvrir plus de trois comptes bancaires au sein de la même banque ;
   — un client peut être supprimé que s'il n'a aucun compte associé ;
   — un transfert entre deux comptes appartenant à deux établissements bancaires différents sera facturé 5 euros au débiteur (cette somme devra donc être débitée en plus du montant à transférer). Il faudra ici tester la valeur des deux comptes après l'opération.

   Les tests seront toujours sur le même modèle : on effectue un appel à une méthode puis on compare le résultat obtenu à celui attendu. Si les deux diffèrent alors un message d'erreur sera affiché.

Éléments de correction

Quelques indications :

— *Penser à utiliser les constructeurs de la super-classe (vous êtes normalement obligé car les attributs de la super-classe doivent être qualifiés* `private`*)...*
— *Penser à réutiliser les constructeurs d'une classe via* `this(...)`*...*
— *Pour la gestion automatique des numéros de compte, il est possible d'utiliser un attribut de classe qui mémorise le numéro du dernier compte créé.*
— *Dans le code proposé il y a utilisation d'itérations appelées* for_each *: lorsqu'il n'est pas nécessaire de manipuler un indice mais de parcourir une collection*

d'objets en lecture (pas de modifications) il est plus sûr d'utiliser ce type d'itération. Je mets sur Moodle un document `Rappels.pdf` *donnant quelques explications plus détaillés à ce sujet.*

— *Concernant la mise en œuvre du diagramme de classes présenté sur la figure 1 ne pas se tromper de sens concernant les structures de données représentant les cardinalités. Par exemple, sur le diagramme, nous avons 3 cardinalités 1-N :*
  — *association* `Banque` - `CompteBancaire` *: une banque gère un ensemble de comptes bancaires;*
  — *association* `Banque` - `Client` *: une banque gère un ensemble de clients;*
  — *association* `Client` - `CompteBancaire` *: un client possède un ensemble de comptes bancaires;*

  La cardinalité 1 représente une référence et la cardinalité N représente une collection de références représentée la plupart du temps (pour son aspect pratique) par une liste. La particularité de la notation UML est que la cardinalité est à l'opposé de la classe qui va contenir la structure de données correspondant à cette cardinalité : ainsi, dans le cas de l'association `Banque` - `CompteBancaire`*, la cardinalité N est à côté de* `CompteBancaire` *mais la liste (de* `CompteBancaire`*) se situera dans la classe* `Banque`*; dans la classe* `CompteBancaire` *il y aura simplement une référence de type* `Banque` *correspondant à la banque à laquelle est rattaché le compte.*

```
/** Repr&eacute;sente un compte bancaire (simplifi&eacute;) */

public class CompteBancaire
{
    // numero du compte
    private int numero;
    // solde du compte
    private double solde;
    // detenteur du compte
    private Client detenteur;

    // gestion automatique du numero de compte
    private static int compteur = 1;

    /**
     * Permet de cr&eacute;er un compte en pr&eacute;cisant le num&eacute;ro,
     * le solde intial et le d&eacute;tenteur
     * @param numero numero du compte
     * @param soldeInitial solde initial du compte
     * @param client le client d&eacute;tenteur du compte
     */
    public CompteBancaire(int numero, double soldeInitial, Client client)
    {
        this.numero = numero;
        this.solde = soldeInitial;
        this.detenteur = client;
    }

    /**
     * Permet de cr&eacute;er un compte en pr&eacute;cisant le num&eacute;ro
     * et le d&eacute;tenteur
     * @param numero numero du compte
     * @param client le client propri&eacute;taire du compte
     */
    public CompteBancaire(int numero, Client client)
    {
        this(numero, 0, client);
    }

    /** Permet de cr&eacute;er un compte en lui attribuant automatiquement un
     * num&eacute;ro.
     * A implementer pour la question 6 (le numero de compte
     *  est attribu&eacute; automatiquement).
     * @param soldeInitial montant initial du compte
     * @param client       client auquel est associ&eacute;e le compte
     */
    public CompteBancaire(double soldeInitial, Client client)
    {
        this(CompteBancaire.compteur,soldeInitial,client);
        CompteBancaire.compteur++;
```

```
    }
    /**
     * Retourne le num&eacute;ro du compte
     * @return numero du compte
     */
    public int donneNumero()
    {
        return this.numero;
    }

    /**
     * Retourne le d&eacute;tenteur du compte
     * @return le client d&eacute;tenteur du compte
     */
    public Client donneDetenteur()
    {
        return this.detenteur;
    }

    /**
     * Cr&eacute;dite le compte du montant indiqu&eacute;
     * @param montant montant &agrave; cr&eacute;diter
     * @return le solde du compte apres cr&eacute;dit
     */
    public double crediter(double montant)
    {
        this.solde = this.solde + montant;
        return this.solde;
    }

    /**
     * D&eacute;bite le compte du montant indiqu&eacute;
     * @param montant montant &agrave; d&eacute;biter
     * @return le solde du compte apres d&eacute;bit
     */
    public double debiter(double montant)
    {
        this.solde = this.solde - montant;
        return montant;
    }

    /**
     * Retourne le solde courant du compte
     * @return le solde du compte
     */
    public double consulter()
    {
        return this.solde;
    }

    /**
     * Transfert le montant vers le compte indiqu&eacute;. Le transfert ne doit
     * pas g&eacute;n&eacute;rer un d&eacute;couvert. Si tel est le cas, il n'est
     * pas accept&eacute;.
     * @param unCompte compte &agrave; cr&eacute;diter
     * @param montantATransferer montant &agrave; d&eacute;biter
     * @return <code>true</code> si le transfert a pu s'effectuer, <code>false</code> sinon
     */
    public boolean transferer(CompteBancaire unCompte, double montantATransferer)
    {
        if (this.consulter() - montantATransferer >= 0)
        {
            this.debiter(montantATransferer);
            unCompte.crediter(montantATransferer);
            return true;
        }
        else
        {
            return false;
        }
    }

    public String toString()
    {
        return "Compte de " + this.detenteur + ", Solde : " + this.solde;
    }
}
```

```
import java.util.ArrayList;

/** Repr&eacute;sente un client d'un banque */

public class Client {

    private String nom, adresse;
    private ArrayList<CompteBancaire> comptes;

    /**
     * Cr&eacute;e un client avec des informations personnelles.
     *
     * @param nom nom du client
     * @param adresse adresse du client
     */
    public Client(String nom, String adresse) {
        this.nom = nom;
        this.adresse = adresse;
        this.comptes = new ArrayList<>();
    }

    /**
     * Retourne le nom du client.
     *
     * @return le nom du client
     */
    public String donneNom() {
        return nom;
    }

    /**
     * Retourne l'adresse du client
     *
     * @return l'adresse du client
     */
    public String donneAdresse() {
        return adresse;
    }

    /**
     * Modifie l'adresse du client
     *
     * @param adresse la nouvelle adresse
     */
    public void changeAdresse(String adresse) {
        this.adresse = adresse;
    }

    /**
     * Attribue un compte a un client.
     *
     * @param compte compte &agrave; attribuer
     * @return <code>false</code> si le compte est deja attribu&eacute;,
     * <code>true</code> sinon.
     */
    public boolean ajouteCompte(CompteBancaire compte) {
        for (CompteBancaire c : this.comptes) {
            if (c.donneNumero() == compte.donneNumero()) {
                return false;
            }
        }
        this.comptes.add(compte);
        return true;
    }

    /**
     * Supprime le compte du client
     *
     * @param numCompte numero du compte a supprimer
     * @return <code>true</code> si le compte existe <code>false</code> sinon.
     */
    public boolean supprimeCompte(int numCompte) {
        for (CompteBancaire c : this.comptes) {
            if (c.donneNumero() == numCompte) {
                this.comptes.remove(c);
                return true;
            }
        }
        return false;
    }
```

```
    /**
     * Retourne l'ensemble des comptes du client
     *
     * @return la liste des comptes du client
     */
    public ArrayList<CompteBancaire> donneComptes() {
        return this.comptes;
    }

    public String toString() {
        return " Nom : " + this.nom + " Adresse : " + this.adresse;
    }
}
```

```
/** Represente un compte bancaire r&eacute;mun&eacute;r&eacute;. */
public class CompteBancaireRemunere extends CompteBancaire
{
    /** taux d'int&eacute;ret par defaut commun &agrave; tous les comptes */
    public static final double interetParDefaut = 3.0;
    // taux d'interet du compte
    private double interet;

    /**
     * Cr&eacute;e un compte bancaire r&eacute;mun&eacute;r&eacute;.
     * @param numero le num&eacute;ro du compte
     * @param soldeInitial valeur du solde initial
     * @param tauxInteret taux d'int&eacute;ret de la r&eacute;muneration
     * @param c client auquel le compte est rattach&eacute;
     */
    public CompteBancaireRemunere(int numero, double soldeInitial, double tauxInteret, Client c)
    {
        super(numero, soldeInitial, c);
        this.interet = tauxInteret;
    }

    /**
     * Cr&eacute;e un compte bancaire r&eacute;mun&eacute;r&eacute;
     * dont le taux d'int&eacute;ret est celui par defaut.
     * @param numero le numero du compte
     * @param soldeInitial valeur du solde initial
     * @param c client auquel le compte est rattach&eacute;
     */
    public CompteBancaireRemunere(int numero, double soldeInitial, Client c)
    {
        super(numero, soldeInitial, c);
        this.interet = interetParDefaut;
    }

    /**
     * Cr&eacute;e un compte bancaire r&eacute;mun&eacute;r&eacute;
     * dont le taux d'interet est celui par defaut
     * et dont le solde initial est &agrave; z&eacute;ro.
     * @param numero le numero du compte
     * @param c client auquel le compte est rattach&eacute;
     */
    public CompteBancaireRemunere(int numero, Client c)
    {
        super(numero, c);
        this.interet = interetParDefaut;
    }

    /** Permet de cr&eacute;er un compte bancaire r&eacute;mun&eacute;r&eacute;
     *  en lui attribuant automatiquement un num&eacute;ro.
     *  A implementer pour la question 6 (le numero de compte
     *  est attribu&eacute; automatiquement).
     *  @param soldeInitial montant initial du compte
     *  @param tauxInteret  taux d'int&eacute;ret du compte
     *  @param c            client auquel est associ&eacute;e le compte

     */
    public CompteBancaireRemunere(double soldeInitial, double tauxInteret, Client c)
    {
        super(soldeInitial, c);
        this.interet = tauxInteret;
    }

    /**
```

```
     * Cr&eacute;dite le compte des int&eacute;rets mensuels.
     */
    public void crediterInteretMensuel()
    {
        // formule simplifiee mais fausse...
        this.crediter((this.consulter() * this.interet / 100) / 12);
    }

    public String toString()
    {
        return super.toString() + ", Taux = " + this.interet;
    }
}
```

```
/** Repr&eacute;sente une particulier client d'une banque */

public class Particulier extends Client {

    private String prenom;

    /**
     * Cr&eacute;e un particulier avec des informations personnelles.
     *
     * @param nom nom du particulier
     * @param prenom pr&eacute;nom du particulier
     * @param adresse adresse du particulier
     */
    public Particulier(String nom, String prenom, String adresse) {
        super(nom, adresse);
        this.prenom = prenom;
    }

    /**
     * Retourne le pr&eacute;nom du particulier
     *
     * @return le pr&eacute;nom du particulier
     */
    public String donnePrenom() {
        return prenom;
    }

    public String toString() {
        return "Prenom : " + this.prenom + super.toString();
    }
}
```

```
/** Repr&eacute;sente une entreprise cliente d'une banque */

public class Entreprise extends Client
{
    private long numSIRET;

    /**
     * Cr&eacute;e une entreprise avec des informations.
     * @param nom nom de l'entreprise
     * @param numSIRET numero de SIRET de l'entreprise
     * @param adresse adresse nom de l'entreprise
     */
    public Entreprise(String nom, int numSIRET, String adresse)
    {
        super(nom, adresse);
        this.numSIRET = numSIRET;
    }

    /**
     * Retourne le numero de SIRET de l'entreprise
     * @return le numero de SIRET de l'entreprise
     */
    public long donneNumSIRET()
    {
        return this.numSIRET;
    }

    public String toString()
    {
        return "Numero de SIRET : " + this.numSIRET + super.toString();
    }
```

```
}
```

```
import java.util.ArrayList;
import java.util.Iterator;

public class Banque {

    private String nom;
    // permet d'acceder aux comptes directement sans passer par des clients
    private ArrayList<CompteBancaire> comptes;
    private ArrayList<Client> clients;
    private final int coutTransfert = 5;

    /**
     * Creation d'une banque identifi&eacute;e par son nom
     * @param nomBanque nom de la banque
     */
    public Banque(String nomBanque) {
        this.nom = nomBanque;
        this.comptes = new ArrayList<CompteBancaire>();
        this.clients = new ArrayList<Client>();
    }

    /**
     * Cr&eacute;eation d'un client de type Particulier
     * @param nomParticulier nom du particulier
     * @param prenom prenom du particulier
     * @param adresse adresse de particulier
     * @return Une instance de <code>Particulier</code> ou <code>null</code> si
     * un client de meme nom existe deja.
     */
    public Particulier creerParticulier(String nomParticulier, String prenom, String adresse) {
        Particulier p = null;
        if (this.rechercheClient(nomParticulier) == null) {
            p = new Particulier(nomParticulier, prenom, adresse);
            this.clients.add(p);
        }
        return p;
    }

    /**
     * Cr&eacute;eation d'un client de type Entreprise
     * @param nomEntreprise nom de l&eacute;ntreprise
     * @param numSIRET numero de SIRET de l&eacute;ntreprise
     * @param adresse adresse de l&eacute;ntreprise
     * @return Une instance de <code>Entreprise</code> ou <code>null</code> si le nom
     *         du client existe deja.
     */
    public Entreprise creerEntreprise(String nomEntreprise, int numSIRET, String adresse) {
        Entreprise e = null;
        if (this.rechercheClient(nomEntreprise) == null) {
            e = new Entreprise(nomEntreprise, numSIRET, adresse);
            this.clients.add(e);
        }
        return e;
    }

    /**
     * Cr&eacute;eation d'un compte bancaire au sein de la banque.
     * Si le nom du client correspond a un nom de client deja existant alors
     * seul le compte est cr&eacute;e puis associ&eacute;e au client
     * Si le nom du client ne correspond pas &agrave; un nom de client deja existant alors
     * le compte n&eacute;st pas cr&eacute;e.
     * @param soldeInitial solde initial du compte
     * @param nomClient nom du client posscr&eacute;edant le compte
     * @return l'instance de <code>CompteBancaire</code> cr&eacute;e ou
     * <code>null</code> en cas de probl&egrave;me.
     */
    public CompteBancaire creerCompteBancaire(double soldeInitial, String nomClient) {
        Client client = this.rechercheClient(nomClient);
        if (soldeInitial >= 0 && client != null) {
            // pas plus de 3 comptes dans la meme banque
            if (client.donneComptes().size() < 3) {
                CompteBancaire cb = new CompteBancaire(soldeInitial, client);
                this.comptes.add(cb);
                client.ajouteCompte(cb);
                return cb;
            }
```

```
        }
        return null;
    }

    /**
     * Cr&eacute;ation d'un compte bancaire r&eacute;mun&eacute;r&eacute;
     * au sein de la banque.
     * Le nom du client doit correspondre a un nom de client deja existant.
     * @param soldeInitial solde initial du compte
     * @param tauxInteret taux d'interet du compte
     * @param nomClient nom du client poss&eacute;dant le compte
     * @return l'instance de <code>CompteBancaireRemunere</code> cr&eacute;&eacute;e ou
     * <code>null</code> en cas de probl&egrave;me.
     */
    public CompteBancaireRemunere creerCompteRemunere(double soldeInitial,
            double tauxInteret, String nomClient) {
        Client client = this.rechercheClient(nomClient);
        if (soldeInitial >= 0 && client != null && tauxInteret > 0) {
            // pas plus de 3 comptes dans la meme banque
            if (client.donneComptes().size() < 3) {
                CompteBancaireRemunere cbr
                        = new CompteBancaireRemunere(soldeInitial, tauxInteret, client);
                this.comptes.add(cbr);
                client.ajouteCompte(cbr);
                return cbr;
            }
        }
        return null;
    }

    /**
     * Retourne l'ensemble des comptes associ&eacute;s &agrave; un client identifi&eacute; par son
     * nom.
     * @param nomClient nom du client
     * @return une liste des comptes appartenant au client
     */
    public ArrayList<CompteBancaire> rechercheCompte(String nomClient) {
        ArrayList <CompteBancaire> cptes = new ArrayList<CompteBancaire>();
        Client client = this.rechercheClient(nomClient);
        if (client != null) {
            cptes.addAll(client.donneComptes());
        }
        return cptes;
    }

    /**
     * Retourne le compte bancaire, s'il existe, poss&eacute;dant le num&eacute;ro de compte
     * donn&eacute; en parametre.
     * @param numeroCpte num&eacute;ro du compte recherch&eacute;
     * @return l'instance de <code>CompteBancaire</code> recherch&eacute;e ou
     * <code>null</code> si elle n'existe pas.
     */
    public CompteBancaire rechercheCompte(int numeroCpte) {
        for (CompteBancaire cb : this.comptes) {
            if (cb.donneNumero() == numeroCpte) {
                return cb;
            }
        }
        return null;
    }

    /**
     * Supprime un compte bancaire, s'il existe, poss&eacute;dant le num&eacute;ro de compte
     * donn&eacute; en parametre.
     * @param numeroCpte numero du compte
     * @return <code>true</code> si la suppression a &eacute;t&eacute; effectu&eacute;e
     *         <code>false</code> si il y a un probleme
     */
    public boolean supprimerCompte(int numeroCpte) {
        CompteBancaire cb = null;
        for (Iterator<CompteBancaire> it = this.comptes.iterator();
                it.hasNext();) {
            cb = it.next();
            if (cb.donneNumero() == numeroCpte) {
                // suppression du compte pour la banque
                it.remove();
                // suppression du compte pour le client
                cb.donneDetenteur().supprimeCompte(numeroCpte);
                return true;
            }
```

```
        }
        return false;
    }

    /**
     * Recherche un client de la banque &agrave; partir de son nom.
     * @param nom nom du client &agrave; rechercher
     * @return le client correspondant ou <code>null</code> s'il n'existe pas.
     */
    public Client rechercheClient(String nom) {
        for (Client c : this.clients) {
            if (c.donneNom().equals(nom)) {
                return c;
            }
        }
        return null;
    }

    /**
     * Supprime un client de la banque. La suppression s'effectue uniquement
     * si le client ne possede plus de compte.
     * @param nomClient nom du client
     * @return <code>true</code> si la suppression a &eacute;t&eacute; effectu&eacute;e
     *         <code>false</code> si il y a un probleme
     */
    public boolean supprimerClient(String nomClient) {
        Client client = this.rechercheClient(nomClient);
        if (client == null) {
            return false;
        } else {
            // on supprime le client que s'il n'a plus de comptes
            if (client.donneComptes().isEmpty()) {
                this.clients.remove(client);
                return true;
            } else {
                return false;
            }
        }
    }

    /**
     * Transfert un montant d'un compte vers un autre compte.
     * @param numeroCpteDebiteur numero du compte d&eacute;bit&eacute;
     * @param banqueCrediteur banque du compte cr&eacute;dit&eacute;
     * @param numeroCpteCrediteur numero du compte cr&eacute;dit&eacute;
     * @param montant montant du transfert
     * @return <code>true</code> si le transfert a &eacute;t&eacute; effectu&eacute;
     *         <code>false</code> s'il y a un probleme
     */
    public boolean transfertInterBancaire(
            int numeroCpteDebiteur,
            Banque banqueCrediteur,
            int numeroCpteCrediteur,
            double montant) {
        CompteBancaire cbDebiteur = this.rechercheCompte(numeroCpteDebiteur);
        if (cbDebiteur == null || montant < 0) {
            return false;
        }
        if (banqueCrediteur != null) {
            CompteBancaire cbCrediteur = banqueCrediteur.rechercheCompte(numeroCpteCrediteur);
            if (cbCrediteur != null) {
                cbDebiteur.debiter(montant + this.coutTransfert);
                cbCrediteur.crediter(montant);
                return true;
            }
        }
        return false;
    }

    public int getCoutTransfert() {
        return this.coutTransfert;
    }

    public String toString() {
        return "Banque " + this.nom + " " + this.comptes;
    }
}
```