ING 1 - POO Java année 2015 -2016

TD-TP n°6

Application à la gestion bancaire – Partie sans héritage

Nous allons commencer par définir une classe Compte et une classe Client, version améliorée du TD/TP précédent.

La classe Client

Classe Client :

Client
- numero : long - nom : String
Client(numero : long, nom : String) + getNumero() : long + getNom() : String + setNom(nom : String) : void + toString() + main(args : String [])

A réaliser

Créer la classe 'Client' en utilisant le diagramme de classes et les précisions concernant les méthodes :

1. le constructeur mémorise le numéro et le nom
2. les méthodes accesseurs renvoient et modifient les valeurs des attributs correspondants
3. la méthode toString() renvoie une représentation textuelle d'un client (*cf.* exemple de résultat plus bas)
4. la méthode main définit les instructions suivantes :
    - instancier un client de numéro 1 et de nom "Gold"
    - instancier un client de numéro 2 et de nom "Silver"
    - afficher les 2 clients

- modifier le nom du client de la 2ème instance avec "Platinum" (au lieu de Silver)
- afficher la 2ème instance de Client

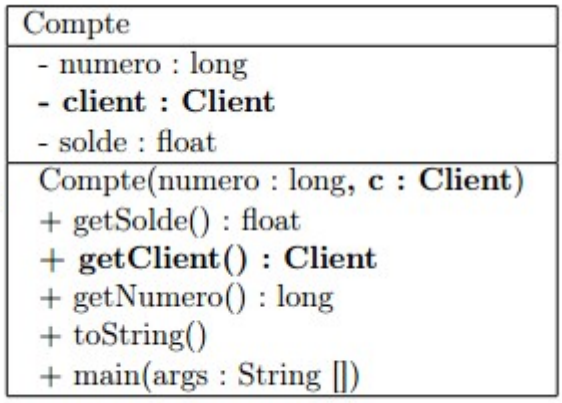
Compiler et tester

L'exécution produit le résultat suivant :
Client 1 - Gold
Client 2 - Silver
Client 2 – Platinum

la classe Compte

Vous définirez la classe Compte à partir du schéma ci-dessous et des informations qui vous sont données.

Classe Compte :



A réaliser

Créer la classe 'Compte' en utilisant le diagramme de classes et les précisions concernant les méthodes :

1. ajouter un attribut privé qui mémorise le client du compte
2. modifier le constructeur pour mémoriser le client à la création d'un compte
3. ajouter une méthode accesseur getClient() qui renvoie le client du compte
4. modifier la méthode toString() pour ajouter l'affichage du nom du client

Compiler et tester

Utiliser le programme principal suivant pour tester votre application

```
public static void main(String [] args){
      Client c1 =  new Client("Gold",1);
      Client c2 =  new Client("Silver",2);
```

```
Compte com1 = new Compte(10,c1);
Compte com2 = new Compte(20,c2);
System.out.println(com1);
System.out.println(com2);
}
```

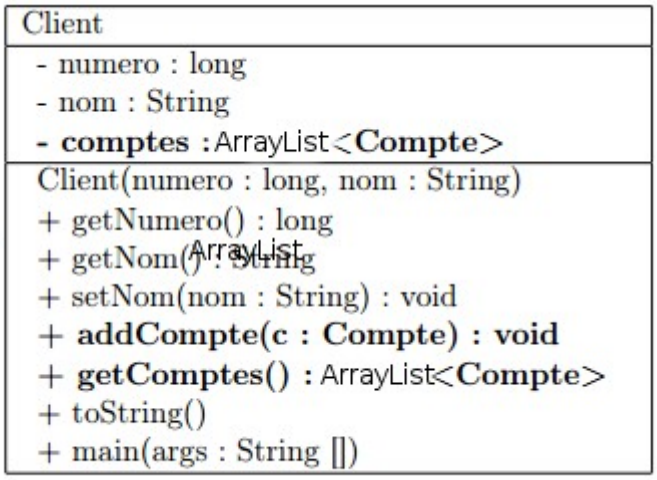
java Compte
Compte 10 - client Gold - Solde : 0.0
Compte 20 - client Silver - Solde : 0.0

Dernière amélioration

Un client pouvant posséder plusieurs compte, modifier la classe Client en conséquence en ajoutant un attribut de type ArrayList<Compte> à chaque client.
Vous ajouterez les méthodes de la figure suivante permettant de rajouter un compte à un client et d'obtenir la liste de tous ses comptes.

La méthode toString() sera surchargée afin d'afficher la liste de tous les comptes d'un client et leur solde (*cf* exemple ci-dessous)

Classe Client :



Compiler et tester

Utiliser le programme principal suivant pour tester votre application

```
public static void main(String [] args){
    Client c1 =  new Client("Gold",1);
    System.out.println(c1);


    Compte com1 = new Compte(10,c1);
    Compte com2 = new Compte(20,c1);


    c1.addCompte(com1);
    c1.addCompte(com2);

    System.out.println(c1);
```

```
com1.credit(500.0);

System.out.println(c1);

}
```

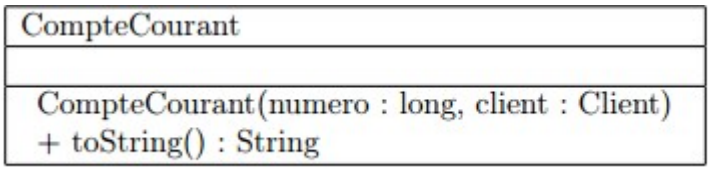
Client 1 - Gold
 pas de compte rattaché à ce client
Client 1 - Gold
 Compte n°10 - solde : 0.0
 Compte n°20 - solde : 0.0
Client 1 - Gold
 Compte n°10 - solde : 500.0
 Compte n°20 - solde : 0.0

HERITAGE

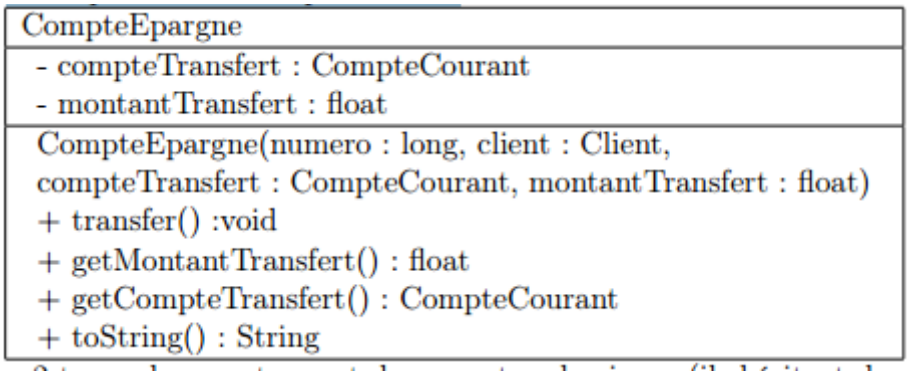
La notion de compte a été décrite précédemment.

On peut cependant distinguer 2 types de comptes spécialisés :

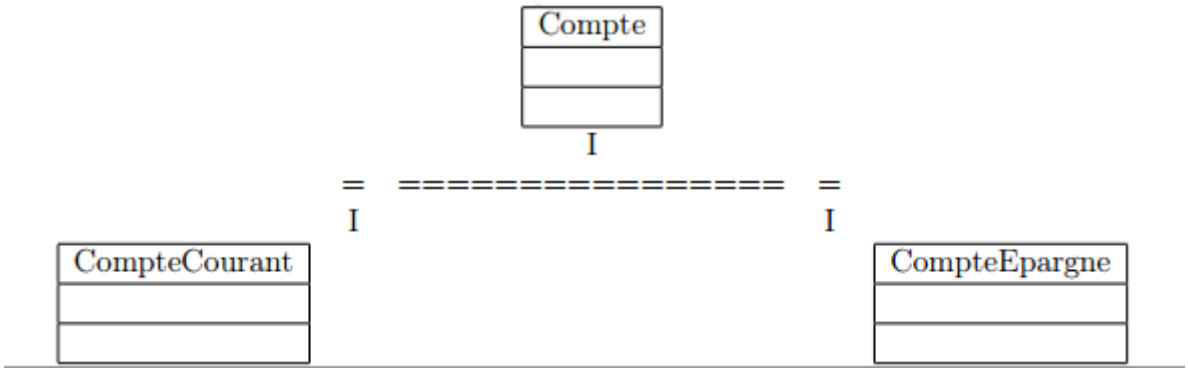
- le compte courant, qui sert aux opérations quotidiennes.



- le compte épargne, vers lequel on effectue périodiquement des opérations de transfert d'un montant fixe à partir d'un compte courant



Ces 2 types de comptes sont des comptes classiques (ils héritent de compte) avec en plus des particularités dans chacune de leurs classes respectives



A réaliser

Créer la classe ’CompteCourant’ en utilisant le diagramme de classes et les précisions concernant les méthodes :

1. la classe ’CompteCourant’ hérite de Compte
2. elle définit un constructeur qui attend un numéro de compte et un client : ce constructeur appellera le constructeur de la classe Compte en lui passant les valeurs de numéro et Client (méthode ’super’)
3. elle redéfinit la méthode toString() pour afficher "Compte courant no " numéro du compte et solde (cf. exemple plus bas)

Compiler et tester

A réaliser

Créer la classe ’CompteEpargne’ en utilisant le diagramme de classes et les précisions concernant les méthodes :

1. la classe ’CompteEpargne’ hérite de Compte
2. elle définit les attributs permettant la mémorisation du compte courant de transfert et du montant de transfert
3. elle définit un constructeur qui attend un numéro de compte, un montant de transfert, un client et un compte courant à partir duquel des transferts seront réalisés : ce constructeur appellera le constructeur de la classe Compte en lui passant les valeurs de numéro et Client (méthode ’super’)
4. elle définit une méthode ’transfert()’ qui crédite le compte et débite le compte courant en utilisant le montant de transfert
5. elle redéfinit la méthode toString() pour afficher "Compte épargne no " numéro du compte et solde (cf. exemple plus bas)

Compiler et tester

A réaliser

Créer la classe ’BanqueApp à partir de la classe ’BanqueApp’ et modifier le déroulement dans la méthode main :

1. la méthode ’main’ :

- créer une ArrayList de Client
- ajouter 4 clients à la collection des clients de la banque (numéro 1 à 4 et nom "Client" suivi du numéro : Client1, Client2, etc.)
- lister les clients de la banque
- dans une boucle, parcourir les clients pour leur ajouter des comptes :
  - récupérer le client
  - créer un compte courant pour le client (numéro de compte : numéro du client * 1000 + 1)
  - créer un compte épargne pour le client (numéro de compte : numéro du client * 1000 + 2, référence au compte courant (pour les transferts) et montant de transfert de 50
  - ajouter les 2 comptes au client
- lister les clients de la banque

Si vous êtes arrivé jusqu'ici et vous avez réussi tous les exemples précédents, bravo !

Compiler et tester

L'exécution produit le résultat suivant :

```
Banque DuTresor
======================
Liste initiale des clients:
Client 1 - client1
 pas de compte rattaché à ce client
Client 2 - client2
 pas de compte rattaché à ce client
Client 3 - client3
 pas de compte rattaché à ce client
Client 4 - client4
 pas de compte rattaché à ce client
Client 1 - client1
 Compte n°1001 - solde : 0.0
 Compte n°1002 - solde : 0.0
Client 2 - client2
 Compte n°2001 - solde : 0.0
 Compte n°2002 - solde : 0.0
Client 3 - client3
 Compte n°3001 - solde : 0.0
```

Compte n°3002 - solde : 0.0

Client 4 - client4

Compte n°4001 - solde : 0.0

Compte n°4002 - solde : 0.0

```java
import java.util.ArrayList;

public class Client {

    private long numéro;
    private String nom;
    private ArrayList<Compte> comptes;

    public Client(long numéro, String nom){
        this.numéro = numéro;
        this.nom = nom;
        this.comptes = new ArrayList<Compte>();
    }

    public String getNom(){
        return nom;
    }

    public long getNumero(){
        return numéro;
    }

    public void setNom(String n){
        nom = n;
    }
    public void addCompte(Compte c){
        comptes.add(c);
    }


    public String toString(){
        String res = "";

        if (comptes.isEmpty()) // vrai si l'ArrayList des clients est vide
                return "client " + nom + "\t numéro " + numéro + "\n" + "pas de
compte associé";

        for(int i = 0; i < comptes.size(); i++)
            res = res + comptes.get(i).toString() + "\n";
        return "client " + nom + "\t numéro " + numéro + "\n" + res;
    }

    public static void main(String [] args){

        Client c1 = new Client(1,"Gold");
        Client c2 =new Client(2,"Silver");

        Compte com1 = new Compte(10,c1);
        Compte com2 = new Compte(20,c2);

        c1.addCompte(com1);
        c1.addCompte(com2);
```

```
        System.out.println(c1);
        com1.credit(500.0);

        System.out.println(c1);
    }

}

public class Compte {

    private long numéro;
    private Client client;
    private double solde;


    public Compte(long numéro, Client c){
        this.numéro = numéro;
        client = c;
        solde = 0.0;
    }

    public double getSolde(){
        return solde;
    }

    public Client getClient(){
        return client;
    }

    public long getNumero(){
        return numéro;
    }

    public void credit(double m){
        solde += m;
    }

    public void debit(double m){
        solde -= m;
    }


    public String toString(){
        String res;

        res = "Compte " + numéro + " - client " + client.getNom() + " - " +
solde;
        return res;
    }

    public static void main(String [] args){
        Client c1 = new Client(1,"Gold");
        Client c2 = new Client(2,"Silver");

    }
}

public class CompteCourant extends Compte {

    public CompteCourant(long numero, Client cl){
        super(numero,cl);
    }
```

```
    public String toString(){
        return "Compte courant numéro " + getNumero() + " - solde " +
getSolde();
    }
}
```

```
public class CompteEpargne extends Compte {

    private CompteCourant compteTransfert;
    private double montantTransfert;

    public CompteEpargne(long numéro, Client cl,
                         CompteCourant compteTransfert, double
montantTransfert){
        super(numéro, cl);
        this.compteTransfert = compteTransfert;
        this.montantTransfert = montantTransfert;
    }

    public void transfert(){
        credit(montantTransfert);
        compteTransfert.debit(montantTransfert);

    }

    public String toString(){
        return "Compte épargne numéro " + getNumero() + " -
solde " + getSolde();
    }

}
```