420-F16-RO Programmation 2 : Concepts de la programmation orient´ee objet

TP3 – Gestion des comptes bancaires

Dini Ahamada - Coll`ege de Rosemont

Hiver 2021

**Ce travail peut être fait seul ou en équipe deux étudiants (0 points si plagiat découvert). Dans le cas projet où est fait en équipe, chacun doit spécifier sa contribution dans le projet**

* Date de remise avant : 29 Mai 2021, 23h55 sur LEA
* Vous déposez dans un dossier portant votre nom et votre numéro étudiant et le programme Java dans LEA.
* Dans le cas où vous envisagez de faire le TP en équipe, vous m’envoyez un MIO en avance les noms des étudiants participants dans le projet et les mentionnez dans la remise
* Pour les questions théoriques posées, vous répondez directement sur cette feuille et déposez-la sur LEA.
* Pour la partie programmation, vous devez utiliser les fichiers de départ fournis dans le projet « TP3CompteBancaireFichierDepart » et ainsi que le fichier « Compt.dat »
* Total  /100
* Pondération 12% de la note globale
* **Toute remise du travail après ces date et heure de remise entraînera une pénalité de 10% par jour de retard.**
* **Votre TP doit être fonctionnel et exécutable. Tout travail remis sans être testé ou qu’il contiendrait du code en rouge, ne sera pas évalué.**
* Vous devez utiliser exclusivement les instructions de Java vues en classe.
* Les variables et les fonctions doivent avoir des noms significatifs qui doivent respecter les normes de programmation Java telles qu’énoncées dans le document « les exigences à respecter ».
* Vous devrez documenter vos classes et ainsi que votre programme principal de teste
* Vous devez mettre des commentaires pertinents dans chacune de vos fonctions et code.
* La pertinence des messages à l’utilisateur sera évaluée.

Objectif :

L’objectif de ce TP3 consiste à travailler sur projet existant, en le modifiant pour ajouter des nouvelles fonctionnalités. Il s’agit d’un projet que nous avons travaillé tout le long de ce cours. Ce TP vous permet de mettre en œuvre la plupart des notions importantes abordées dans le cours telles que les classes, héritage de classe, le Polymorphisme, la gestion des fichiers et des exceptions.

Rappel de l’énoncé du départ :

Cahier des charges

Il s’agit d’écrire une application interactive qui permet de gérer l’ensemble des comptes

bancaires d’une personne. Les fonctionnalités fournies par le programme de gestion de

comptes bancaires sont les suivantes :

Création, Suppression d'un compte

Affichage d'un compte donné

Saisie d'une ligne comptable pour un compte donné

Calcul de statistiques

Sauvegarde des données (n° de compte, lignes comptables)

***Niveau 1 : programme interactif sous forme de choix dans un menu***

L’exécution du programme affiche le menu suivant :

1. Créer un compte

2. Afficher un compte

3. Créer une ligne comptable

4. Sortir

5. De l'aide

Votre choix :

L’utilisateur choisit une valeur pour exécuter l’opération souhaitée (les zones grisées correspondent à des valeurs choisies par l’utilisateur).

* Si l’utilisateur choisit l’option 1, les informations à fournir concernent :

Le type du compte [Types possibles : Compte courant, joint, épargne] :

Le numéro du compte :

La première valeur créditée :

Le taux de placement dans le cas d'un compte épargne :

* Si l’utilisateur choisit l’option 2, le programme affiche les caractéristiques d’un compte (type, valeur courante, taux), ainsi que les dix dernières opérations comptables dans l’ordre des dates où ont été effectuées les opérations.
* Pour l’option 3, il s’agit de fournir des informations pour créer une ligne comptable.

Ces informations sont les suivantes :

Le numéro du compte concerné (avec vérification de son existence) :

La somme à créditer (valeur positive) ou à débiter (valeur négative):

La date de l'opération :

Le motif de l'achat ou de la vente [thèmes possibles : Salaire,

Loyer, Alimentation, Divers] :

Le mode de paiement[Types possibles : CB, n° du Chèque, Virement]:

L’option 4. Sortir du menu général permet de sortir du programme.

L’option 5. Aide du menu général affiche une information relative à chaque option du

menu.

***Niveau 2 : structure de données optimisée en termes d’utilisation de la mémoire***

* Le programme doit pouvoir gérer autant de comptes que nécessaire. Pour chaque compte le nombre d’opérations comptables doit être infini et est donc indéterminable au moment de l’écriture du programme.
* En conséquence, la réservation des cases mémoire ne peut pas être réalisée de façon définitive en tout début de programme. À chaque ligne comptable et à chaque nouveau compte créé, le programme doit être capable de réserver lui-même le nombre suffisant d’emplacements mémoire pour la bonne marche du programme. Lorsque le programme gère lui même la réservation des emplacements mémoire, on dit qu’il gère sa mémoire de manière dynamique.
* L’option permettant la suppression d’un compte est dépendante de la façon dont est stockée l’information. Cette option ne peut être abordée avant d’avoir choisi le mode de gestion des emplacements mémoire.
* L’option 5. Sortir du menu général doit contrôler la sauvegarde de l’information.
* Les données sont sauvegardées sur disque sous forme d’un fichier portant le nom

compte.dat.

Travail à faire

1. **Structure de données par la programmation orienté objet avec la classe Compte ( 20 points )**

En utilisant les classes fournies, apporter les modifications suivantes

La **classe Compte** a les attributs suivants :

* **typeCpte** : String // pour identifier le type de compte
* **val\_courante** : double // la somme courant
* **numéroCpte** : String // numero de compte
* **ligne []:** LigneComptable // tableau ligne de compte qui permet contenir dix lignes comptables.
* **nbLigneRéel** : int // nombre de ligne
* Les lignes comptables étant créées au fur et à mesure des opérations réalisées par l’utilisateur, il est nécessaire de définir une variable (nbLigneRéel), qui compte le nombre de lignes comptables effectivement créées en cours d’exécution du programme.
* **NBLigne** : int // est un constant statique qui vaut 10 et qui représente le nombre maximal de lignes comptables à traiter

La **classe Compte** a les méthodes suivantes :

* **Les constructeurs** :

1. Le type de compte est saisies au clavier

Le constructeur par défaut permet de

saisir le numero de compte,

determiner le type de compte grâce à la méthode contrôleType(), initilialiser la somme courante en utilisant la méthode contrôleValinit() , créer en mémoire la donnée ligne, sous forme d’un tableau de dix lignes comptables.

Initialiser la variable nbLigneRéel à –1, puisqu’aucune ligne n’a encore

1. Le type de compte est passé en paramètre :

Le constructeur qui prend en paramètre le type de compte,

vérifier si le type de compte passé en paramètre de est un compte « epargne » et fait la même chose que le constructeur par défaut.

1. Quelle est la notion est en évidence pour construire des objets ?

**Il est important de pouvoir créer les objets dont les constructeurs afin d’appeler les attributs dans les autres classes pour que notre programme fonctionne correctement.**

* La méthode **contrôleType()** : String

Un compte bancaire ne peut être que de trois types : Epargne, Courant ou Joint.

Il est donc nécessaire, au moment de la saisie du type du compte, de contrôler l’exactitude du type entré.

La méthode **contrôleType()** doit réaliser ce contrôle : a la sortie de la fonction, nous devons être certains que le type retourné correspond aux types autorisés par le cahier des charges. Elle affiche les options possibles et demande à l’utilisateur de choisir l’une d’entre d’entre elles

Est-ce que pourquoi la visibilité de la méthode doit être privée ?

**Selon moi, je suppose que la visibilité de cette méthode est en privée parce que c’est comme dans la vraie vie lorsque l’on saisit nos données bancaires. Il faut que nos informations soient sécurisées. C’est la même chose en programmation.**

* La méthode **controleValinit()** : double permet de retourne la valeur initiale, elle invite à l’utilisateur à la saisir. Sachant que la valeur initiale ne peut être négative à la création d’un compte, chaque fois qu’une valeur négative est saisie, vous devez inviter l’utilisateur à réasaisir.
* La méthode **créerLigne()** réalise ce traitement de la façon suivante :

1. Incrémente nbLigneRéel de 1.
2. Si le nombre de lignes créées est inférieur à NBLigne, crée en mémoire une ligne comptable grâce au constructeur de la classe LigneComptable et stocke en mémoire son adresse dans le tableau ligne.
3. Si le nombre de lignes est supérieur à NBLigne, décale toutes les lignes vers le haut, grâce à la méthode décalerLesLignes() décrite ci-dessous, et stocke la nouvelle ligne comptable en dernière position (NBLigne – 1) du tableau ligne.

**private** **void** **décalerLesLignes()** {

**for**(**int** i = 1; i < NBLigne ; i++)

ligne[i-1] = ligne[i];

}

1. Met à jour la valeur courante du compte en fonction du crédit ou débit réalisé par la nouvelle ligne comptable.

* La méthode **afficherCpte ()** : réalise l’affichage de la facon suivante :

1. le numéro de compte
2. le type de compte
3. l’ensemble des lignes saisies en cours d’exécution du programme s’il en existe,
4. la valeur courante, dans le cas que cette valeur est négative et affiche un message d’avertissement.

* Completer les différents accesseurs selon les difféentes attributs que chaque doit retourner

La **classe LigneComptable** a les attributs suivants (**15 points ):**

* valeur : double
* date : String
* motif :String
* mode : String

La classe LigneComptable a les méthodes suivantes :

* Le **constructeur** initialise :

1. la valeur selon le montant saisi par l’utilisateur de précédé (+ pour le montant à créditer et - pour le montant à débiter ) .
2. le motif, le mode et la date respectivement grâce aux méthodes contrôleMotif (), contrôleMode () et contrôleDate()

* La méthodes **contrôleMotif ()** : vérifie et retourne le motif de paiement pour une ligne comptable. Le motif ne peut être que le Salaire, Loyer, Alimentation ou Divers.Si l’utilisateur venait à saisir autre option que celles mentionnées, vous devez l’inviter à saisir.
* La méthode **contrôleMode ()** : vérifie et retourne le mode de paiement pour une ligne comptable. Le mode ne peut être que CB, Virement ou Chèque. Si l’utilisateur venait à saisir autre option que celles mentionnées, vous devez l’inviter à saisir.
* La méthode **contrôleDate()** permet le contrôle du format de la date est le suivant.
* La méthode **afficherLigne()** réalise l’affichage de la facon suivante **:**

1. le montant à créditer, la date, le motif et le mode.
2. S’il s’agit d’un montant à debiter, on affiche seulement le montant.
3. **Comprendre l’héritage avec la classe CpteEpargne (12 points ):**

**Protection des données héritées**

* Sachant qu’un compte d’épargne est un compte bancaire ayant un **taux** de rémunération,
* Complétez **la classe CpteEpargne** avec la nouvelle donnée.
* Quel mode de protection doit-on appliquer à cette nouvelle donnée ainsi que ses accesseurs / mutateurs ?

**Ça serait recommandé de mettre le mode protected parce que les accesseurs/ mutateurs seront seulement accessibles dans le même package.**

* Quel est le concept qui est mis en œuvre ?

**C’est le concept du Polymorphisme, c’est-à-dire d’être capable de pouvoir travailler dans plusieurs classes en même temps.**

* Un compte d’épargne modifie la valeur courante par le calcul des intérêts, en fonction du taux d’épargne. Il ne peut ni modifier son numéro, ni son type.
* Quels modes de protection doit-on appliquer aux différentes données héritées de la classe Compte ?

**Il serait idéal que certaines méthodes qui demandent les données bancaires de l’utilisateur soient en mode private puisque c’est pour protéger les informations personnelles.**

**Le constructeur d’une classe dérivée**

* En supposant que le constructeur de la classe CpteEpargne s’écrive de la façon suivante :

**public** CpteEpargne() {

**super**("Epargne");

taux = contrôleTaux();

}

* Recherchez à quel constructeur de la classe Compte fait appel CpteEpargne (). Pourquoi?

**Il recherche le constructeur défini qui ne reçoit pas de paramètres de retour. Cependant, le constructeur doit savoir que le super doit retourner le type du Compte Épargne afin d’accéder à cette classe.**

* Modifiez ce constructeur de façon que la donnée typeCpte prenne la valeur Epargne.
* Dans la classe CpteEpargne, écrivez la méthode **afficherCpte (),** sachant qu’afficher les données d’un compte d’épargne revient à afficher les données d’un compte, suivi du taux d’épargne.
* Quel est le concept qui est mis en œuvre ?

**On utilise le concept de l’héritage parce qu’on va réutiliser les constructeurs de la classe Compte. La classe fille (CpteEpargne) va reprendre les attributs et méthodes de la classe mère (Compte) en ayant déjà les données enregistrées.**

* La méthode **controleTaux()** : double permet de retourne le taux d’emplacement, elle invite à l’utilisateur à le saisir. Sachant que le taux d’emplacement ne peut être négative à la création d’un compte, chaque fois qu’une valeur négative est saisie, vous devez inviter l’utilisateur à réasaisir.

1. **Les comptes sous forme de dictionnaire avec la classe ListeCompte (15 points)**

La classe **ListeCompte** est composée des méthodes suivantes :

* Le constructeur **ListeCompte (),** qui fait appel au constructeur de la classe Hashtable et permet de créer une liste de compte en utilisant HashMap
* La méthode **ajouteUnCompte (String t)** permet la création d’un compte courant, joint ou d’épargne de la facon suivante :

1. Lorsque le paramètre de la méthode vaut "E", un compte d’épargne est créé, alors que s’il vaut "A" (comme Autre), un compte ordinaire est créé.
2. Lorsque le compte est créé, insérez-le dans le dictionnaire, en prenant comme clé d’association son numéro de compte.
3. S’il le compte existe déjà, on doit afficher un message pour signifier que le compte existe

* La méthode **ajouteUneLigne (**String n**),** ajoute une ligne au compte dont le numéro est spécifié en paramètre de la méthode existe, s’il n’existe pas on affiche un message suivante « Le systeme ne connait pas le compte ». Pour cela, faites appel à la méthode créerLigne() de la classe Compte.
* La **rechercheUnCompte(**String n**),** permet de rechercher un compte selon sa clé, c’est-à-dire son numéro de compte et de l’afficher.
* La méthode **quelCompte**(String n) : permet d’obtenir un compte selon sa clé, c’est-à-dire son numéro de compte
* La méthode **supprimeUnCompte(**String n**)** **,** permet de supprimer un compte selon sa clé, c’est-à-dire son numéro de compte.
* La méthode **afficheLesComptes()** **,** permet d’afficher l’ensemble des comptes

Remarque un compte doit être existé avant de l’afficher ou de le supprimer, puisque ce sont les méthodes de la classe ListeCompte qui s’en chargent directement.

1. **La sauvegarde des comptes bancaires avec la classe FichierCompte (15 points)**

Les données de **la classe FichierCompte** sont deux objets représentant des flux d’écriture (ofW), de lecture (ofR) d’objets, ainsi qu’un caractère (mode) représentant le type d’ouverture du fichier et une chaîne de caractères (nomDuFichier), où se trouve mémorisé le nom de fichier de sauvegarde des données (**Compte.dat**).

Pour sauvegarder les données saisies pour chaque compte, complétez les méthodes suivantes :

**Ouverture du flux (entrant ou sortant)**

* La méthode **ouvrir (String s)**  est un méthode boolean qui permet de vérifier si l’ouverture du flux que l’on soit en mode ecriture ( E ou W) ou lecture ( L ou R) s’est bien passé. Le paramètre s peut-être ( E ou W) en écriture ou ( L ou R) en lecture.

1. L’ouverture du fichier « Compte.dat » en lecture est réalisée grâce aux constructeurs des classes ***FileInputstream*** et ***ObjectInputstream***,
2. alors que l’ouverture en écriture est effectuée par les constructeurs ***ObjectOutputstream()*** et ***FileOutputstream().***
3. On doit vérifier le type de mode avant lecture/ecriture du fichier « Compte.dat »

En résultat, les flux ofW et **ofR** contiennent les adresses de début de fichier.

**Traitement du fichier**

L’objectif est d’archiver l’ensemble des données relatives à une classe de Compte.

* La méthode **ecrire()** prend en paramètre un objet **tmp** de type **ListeCompte**
* L’objet transmis est alors archivé grâce à la méthode **writeObject**(tmp).
* Inversement, la méthode **lire()** lit l’objet stocké dans le fichier Compte.dat
* L’objet retourné est lu grâce à la méthode **readObject().**

**Fermeture du flux**

* La méthode **fermer()** permet la fermeture du flux que l’on soit en mode ecriture ( E ou W) ou lecture ( L ou R) . On doit vérifier le type de mode avant la fermeture
* La fermeture d’un flux est réalisée par la méthode close(), de la même façon qu’un flux de fichier texte.

1. **Gérer les exceptions (8 points)**

Les méthodes **ouvrir()** , **ecrire()** , **lire()** et **fermer()** doivent capturer et traiter elle-même les erreurs éventuelles lors de l’ouverture, écriture, lecture et fermeture du fichier Compt.dat. Afficher un message proprié si le fichier n’existe pas

1. **Test de l’application (15 points)**

En vous inspirant de l’application ProjetV3 et du nouveau menu dans la fonction menuPrincipal(),dans l’application ProjetV6, déclarer l’objet C comme étant du type ListeCompte et l’objet F comme étant du type F. Puis :

1. Lire le fichier « Compte.dat » avant de proposer l’ajout, la suppression ou l’affichage des comptes. Assurez que vous êtes bien en mode lecture (L) en utilisant la méthode ouvrir() avant la lecture du fichier.

le fichier « Compte.dat » contient des comptes d’existants clients. Le client dont le numéro de compte est « err435 » n’existe plus, vous devez le supprimer du fichier quand vous rendez votre travail. Les comptes des autres clients doivent y rester, vous pouvez ajouter des nouveaux clients.

1. Dans chaque option du menu, faire appel aux méthodes de la classe ListeCompte.
2. Modifiez l’option 1, de façon à demander à l’utilisateur s’il souhaite créer un compte simple ou un compte d’épargne. Selon la réponse, lors de l’ajout d’un compte, ne pas omettre de spécifier en paramètre le type du compte ("A", ou "E").
3. Ajoutez l’option de suppression d’un compte (option 5) et
4. L’affichage de la liste de tous les comptes (option 3).
5. Modifier l’affichage du menu et le switch de façon à tenir compte de ces nouvelles options. Remarquez qu’il n’est plus besoin de tester l’existence du compte avant de l’afficher ou de le supprimer, puisque ce sont les méthodes de la classe ListeCompte qui s’en chargent directement.
6. Réaliser une sauvegarde automatique à la sortie du programme (option 6).
7. Afficher l’aide dans l’option 7