TD 1 : Comptes bancaires

Rappels de notions : objet, `Java` et `UML`

— Gardez à l'esprit les différents principes de conception et programmation objet vues l'an dernier (encapsulation, polymorphisme, DRY, KISS etc.)

— Pensez à respecter les conventions de nommage `Java` :
- celles d'Oracle : https://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- une autre convention, par ex. https://google.github.io/styleguide/javaguide.html

Date limite de rendu de votre code sur le dépôt GitHub : **Dimanche 15 Septembre, 23h00**

Prise en main des outils

Tous les TDs de CPOA seront hébergés dans l'organisation GitHub du module :
https://github.com/IUTInfoMontp-M3105
Le schéma de travail sera le suivant :

0. Si ce n'est pas encore fait au semestre précédent, demandez le StudentPack de GitHub (https://education.github.com/pack). Vous obtiendrez la licence gratuite pour plusieurs outils payants et notamment la possibilité d'avoir des **projets privés** sur GitHub.
1. Dans le dépôt de chaque TP, un lien **GitHub Classroom** vous permet de créer un fork du projet et d'affecter automatiquement votre projet à l'organisation *IUTInfoMontp-M3105*.
   Ce qui permet aux enseignants d'être admins sur votre projet. L'adresse de votre fork sera :
   https://github.com/IUTInfoMontp-M3105/TP1-VotreLogin
   La commande pour le cloner en local (le télécharger sur votre machine) :
   ```
   ~/CPOA$ git clone https://github.com/IUTInfoMontp-M3105/TP1-VotreLogin
   ```
2. Ensuite, pour travailler **localement** vous allez utiliser **Git** pour suivre l'évolution de votre travail. Vérifiez d'abord que votre configuration locale est correcte en ouvrant le fichier `.gitconfig` de votre `$HOME`. Votre configuration devrait rassembler à cela :
   ```
   [user]
   username = prenom-nom
   name = Prenom Nom
   email = prenom.nom@etu-umontpellier.fr
   ```
3. Pour chaque changement dans votre dépôt local que vous compter enregistrer, vous ferrez :
   ```
   ~/CPOA/TP1-VotreLogin$ git add lEnsembleDeFichiersQueVousSouhaitezSuivre
   ~/CPOA/TP1-VotreLogin$ git commit -m "leMessagePourExpliquerLaSauvegarde"
   ```
   Conseil : Privilégiez des petits commits (un par fonctionnalité), plutôt que des gros commits. Un bon baromètre c'est la longueur du message de commit : plus la fonctionnalité est petite, plus courte est l'explication !
4. À la fin de votre travail, n'oubliez pas de pousser vos changements sur le dépôt distant.

Vous trouverez le regroupement de la documentation concernant Git, ainsi que les transparents du cours, ici : https://github.com/IUTInfoMontp-M3105/Ressources

Il est vivement recommandé d'utiliser au maximum les fonctionnalités de l'**IDE** pour réaliser les tâches courantes (renommage d'attributs/méthodes, génération des différentes méthodes : constructeurs, setters, getters, etc.).

Conseil : Afin de garder une trace de la progression de votre application, on vous conseille de travailler dans un package différent pour chaque exercice. Cela vous permettra de mieux comparer votre travail pour chaque exercice et également de mieux réviser plus tard..

Sujet

Dans la banque **BronzeManCrooks** un client souhaitant avoir un *compte* doit choisir parmi les différents types : *compte courant*, *livret*, *compte pro* etc. Il y a plusieurs types de livrets : *livret A*, *livret d'épargne PlusPlus*, *livret de spéculation*, etc.

Tous les comptes ont un solde (`double`), un IBAN (une donnée de type `String`), un nom de client et une adresse (des données de type `String`). Toutes ces données sont initialisées avec le constructeur. Le nom peut être modifié avec une méthode *modifieur* (*setter*). Une méthode *accesseur* `public double getSolde()` doit retourner le solde de chaque compte. Un compte pro possède en plus un numéro SIREN, alors que le compte courant enregistre le NoINSEE de la personne physique détenteur du compte.

Chaque livret possède un *taux d'intérêts* de type `double`. Vous pouvez supposer que cette valeur réelle est entre 0 et 1. Ainsi, la méthode `public double getSolde()` doit retourner le solde de compte + les intérêts (dans la vraie vie, ça serait trop beau, mais on va supposer pour le fun que c'est comme ça...). De plus le livret A a un *plafond* de dépôt maximum, le livre d'épargne a un *taux d'imposition* (valeur réelle entre 0 et 1) et le livret de spéculation a deux données : une *taxe* fixe qui s'appliquera à chaque transaction réalisé avec ce livret et le *nombre de transactions*.

Remarque : Dans ce qui suit il ne vous est pas demandé de modéliser le client, la seule information le concernant étant l'attribut `nom` évoqué ci-dessus.

Remarque : La plupart de questions sont assez simples en revanche on attend de votre part d'écrire du code propre respectant les différents principes objets : encapsulation, non-duplication de code, etc.

1. Proposez un diagramme de classes en y indiquant les relations entre les classes, les attributs, les méthodes, ainsi que leur visibilité. Votre solution doit permettre l'ajout facile d'autres types de comptes. Vous pouvez utiliser un logiciel de modélisation que vous souhaitez où bien le faire sur papier.

   Remarque : un compte ou un livret ne peuvent pas exister en tant que tels, ils doivent forcément être d'un type spécifique (compte courant ou livret A par exemple).

2. Écrivez le code `Java` correspondant et implémentez également la méthode `toString()` pour permettre l'affichage de l'intégralité des informations du compte. Vérifiez le bon fonctionnement de votre programme en implémentant la méthode principe `public static void main(String args[])` de la classe `App`. Pour ce faire, vous allez créer au moins un compte pour chacun des 5 types de comptes mentionnés ci-dessus et les initialiser avec des valeurs d'attributs **distinctes**.

3. On vous demande maintenant d'ajouter un plafond de découverte pour tous les comptes, qui est à initialiser avec un méthode *setter*. Combien d'ajout et de modifications devez-vous faire dans votre code ?

4. On vient vers vous avec une nouvelle précision : avec la mise en place du prélèvement à la source par le gouvernement, il faut que le solde de chaque type de livret tienne compte des différentes taxes. Ajoutez cette fonctionnalité **sans modifier** le programme précédemment écrit.

5. Moyennant un *prix* fixé par l'utilisateur (donnée de type `double`), il est possible de détenir un *compte groupé* : un regroupement de plusieurs comptes différents. Le nombre de "sous-comptes" autorisés dans un tel compte groupé est potentiellement illimité et il est toujours possible d'en ajouter (à travers une méthode). De plus, il est possible d'avoir plusieurs comptes groupés dans un compte groupé (pour chaque membre de sa famille par exemple, ou pour son entreprise...). Dans tous les cas, chaque compte groupé aura son propre prix.

   Complétez votre diagramme de classes et proposez une implémentation en `Java`. Assurez-vous que la méthode `public double getSolde()` retourne la somme des soldes de tous les "sous-comptes" du compte groupé correspondant.

   NB : à la création vous pouvez supposer que le solde d'un compte groupé est 0.

6. On souhaite pouvoir afficher l'ensemble des informations de chacun des comptes d'un compte groupé. En vous inspirant de l'exemple précédent, redéfinissez la méthode `toString()` dans les classes qui vous semblent appropriées afin que cette méthode retourne une chaîne de caractères contenant l'intégralité des informations du compte correspondant.

7. Dans la méthode principale `public static void main(String args[])` de la classe cliente (`App`) effectuez les actions suivantes dans l'ordre :
   — créer un compte groupé pour le client "Tintin Duchmolle" et lui ajouter un compte groupé contenant un compte courant A et un compte groupé B. Le compte groupé B devra contenir un *livret A*, un *livret PlusPlus* et un *livret de spéculation*.
   — afficher le solde total de tous les comptes
   — afficher l'ensemble des informations concernant chacun des comptes.

Gros Bonus

Pour ceux qui s'ennuient car ils ont tout fait très bien (mais pas que pour eux), observez que les constructeurs de vos classes ont beaucoup de paramètres. Certains d'entre vous avait rencontré dans le passé la notion de *Builder* (https://fr.wikipedia.org/wiki/Monteur_(patron_de_conception)), un modèle de conception destiné à proposer une construction flexible des objets complexes. Réfléchissez à une façon de réorganiser votre solution pour l'implémenter et apportez plus de souplesse à votre programme.