

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 comptez enregistrer, vous ferez :

```
~/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 ouvrir 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*.

Tous les comptes ont un solde, un IBAN et un nom de client. Toutes ces données sont initialisées avec le constructeur. Le nom peut être modifié avec une méthode *modifieur* (*setter*). Une méthode *accesseur* 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*. Vous pouvez supposer que cette valeur réelle est entre 0 et 1. Ainsi, la méthode *accesseur* doit retourner le solde du compte + les intérêts. 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 réalisée avec ce livret et le *nombre de transactions*.

Remarque : La plupart des questions étant assez simples, on attend de vous des diagrammes et un code propres, respectant les différents principes UML et objets : héritage, encapsulation, non-duplication de code, etc.

1. Modéliser à l'aide de StarUML la vue des cas d'utilisations.
2. Proposer un diagramme de classes avec les relations, les attributs, les méthodes, ainsi que les visibilité. Votre solution doit permettre l'ajout facile d'autres types de comptes.

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).

3. Illustrez chaque classe du diagramme de classes à l'aide d'un diagramme d'objet.
4. Utilisez le générateur automatique de StarUML pour récupérer le code Java de la structure du système. Complétez / corrigez votre code si besoin.
5. Implémentez 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 principale 'main' de la classe *App*. Appuyez-vous sur votre diagramme d'objet.
6. Ajoutez un plafond de découvert autorisé pour tous les comptes, qui est à initialiser avec une méthode *setter*. Donnez le nombre d'ajouts et le nombre de modifications apportés.
7. Nouveau besoin : 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 revoir* le code précédemment écrit.

8. Moyennant un *prix* fixé par l'utilisateur, 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 d'autres (à l'aide d'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 '*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.

9. 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.
10. Dans la méthode principale 'main' de la classe ('*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.