IUT d'Orsay - BUT Informatique - Jean-Claude MARTIN – jean-claude.martin@universite-paris-saclay.fr

# R2.01 - Développement Orienté Objet (DOO)
# EXERCICES 3

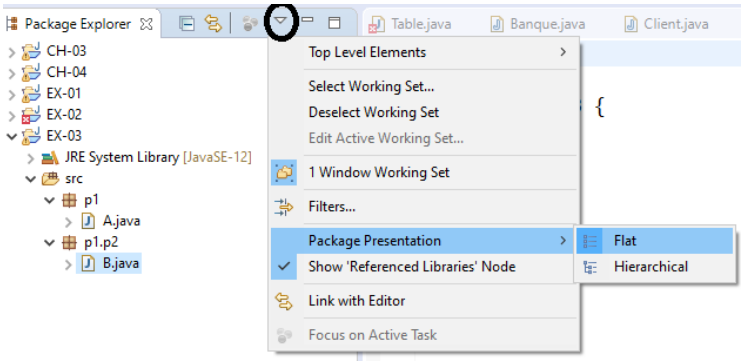
## A- Créer et visualiser une structure de paquetages et sous-paquetages

Créez la structure suivante dans Eclipse dans un projet EX-03.
Les mots commençant par une minuscule sont des paquetages.
Les mots commençant par une majuscule sont des classes.
Attention : Le nom d'un **sous-paquetage** p2 devant être inclus dans un paquetage p1 est : **p1.p2**

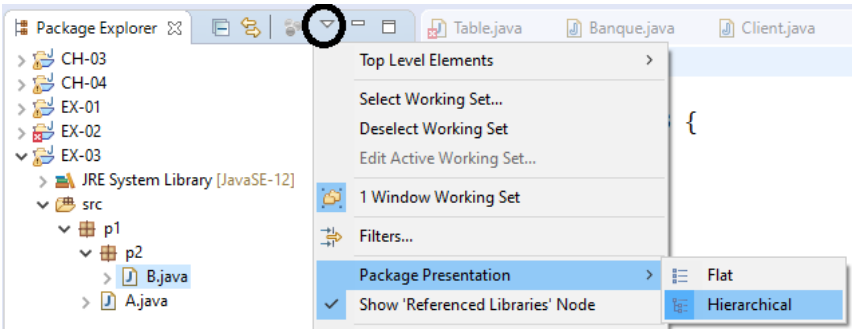
```
modeles
     Voyage
     transport
          Train
          Avion
vues
     Voyage
     Radar
     Texte

controleurs
     PanneauControle
```

Dans le mode de présentation par défaut d'Eclipse (Flat), tous les sous-paquetages apparaissent au même niveau : ils n'apparaissent pas comme étant imbriqués (le menu s'affiche quand on clique sur la petite flèche vers le bas dans le package explorer) :




Pour voir les sous packages imbriqués dans leur package englobant,
sélectionner "Package presentation > hierarchical":




# B - Conception et programmation d'un projet banque

Objectifs de cet exercice :

- Savoir dans quelle classe doit être définie une méthode
- Savoir déléguer une partie des instructions à une autre classe
- Comprendre les bénéfices de l'encapsulation (possibilité de changer l'implémentation sans changer les entêtes des méthodes appelées par les autres développeurs)
- Comprendre les liens entre associations UML et attributs de type tableau en Java
- Bien commenter et structurer, présenter son programme

Vous travaillez dans une société de services informatiques.
Vous devez écrire un programme pour gérer des banques.

*Le responsable informatique du réseau de banques a exprimé les besoins suivants :*
*Une banque a un nom.*
*Il y a plusieurs clients et plusieurs comptes courants dans une banque (au début il n'y a bien sûr aucun client et aucun compte lors de la création de la banque).*
*Un client a un numéro, un nom et une adresse.*
*Un compte courant est représenté par un numéro, un solde et un seuil de découvert autorisé.*
*Chaque compte courant a un propriétaire qui est un client de la banque.*
*Un compte courant appartient à un seul client.*
*On pourra créditer ou débiter un montant sur un compte courant.*
*Un client peut avoir plusieurs comptes courants.*
*Il faut pouvoir ajouter un compte courant à un client, ajouter un compte courant à une banque, et ajouter un client à une banque.*
*Un client est dans une seule banque.*

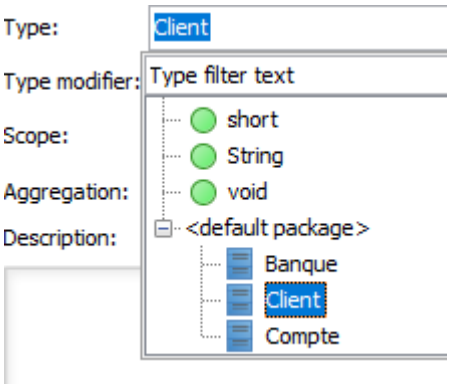
<u>ETAPE 1 : Conception orientée objet</u>

1. Le responsable informatique des banques vous commande dans un premier temps un schéma des classes avant de voir s'il vous commande ensuite la programmation de ce projet.
   Sous Visual Paradigm, créez un nouveau projet "banque".
   Créer un nouveau schéma des classes.

   Créer un paquetage "banque" (dans la version française de visual paradigm il faut sélectionner "Ensemble").

Créeer les classes Banque, Client et CompteCourant.
Ces classes seront encapsulées (c'est-à-dire que leurs attributs auront comme visibilité (modificateur d'accès) "private" et que si c'est nécessaire vous écrirez des méthodes, appelées accesseurs, permettant d'y accéder).

Créer les associations entre les classes comme indiquées dans l'énoncé
(de la même manière que vous avez indiqué les associations entre Table et Magasin dans EX 02).
Comme rôle pour une association, utiliser des noms comme "saBanque" pour le côté Banque d'une association et sesClients pour le rôle associé à la classe Client.
Dans la spécification des associations, dans la liste déroulante "Visibilité" sélectionnez "privée" : cela permettra de mettre automatiquement en private les attributs correspondant à l'association lors de la génération du code Java depuis Visual Paradigm.




ETAPE 2 : PROGRAMMATION

Vous avez présenté votre schéma des classes et convaincu le responsable informatique des banques.
Il vous passe maintenant commande de la programmation complète de ce projet en Java.
Ouvrez Eclipse et créer un paquetage banque dans votre projet EX-03.

Copier coller l'énoncé de chaque question sous la forme de commentaire dans votre code :
// Q2 : Définir un constructeur public CompteCourant avec paramètres (un des paramètres aura le même nom qu'un des attributs)
…CompteCourant (...

1. Depuis l'explorateur Windows ouvert sur votre dossier de projets Visual Paradigm, faites glisser - coller les sources Java des classes vers votre package banque dans Eclipse.
   Ouvrez la classe *CompteCourant*. Vérifiez bien que les attributs de la classe *CompteCourant* sont private. Utilisez le menu d'Eclipse Source > generate Getters and Setters pour générer automatiquement les accesseurs pour tous les attributs de la classe *CompteCourant*.
   Implémenter la méthode créditer
   et la méthode débiter qui vérifie que le solde (- le montant) est supérieur au seuil de découvert autorisé avant de débiter.
2. Définir un constructeur public *CompteCourant* avec paramètres (un des paramètres aura le même nom qu'un des attributs)

3. Ajouter un test dans le setter de l'attribut solde : si on essaye d'y mettre une nouvelle valeur qui est supérieure à une constante static final SEUIL_SECURITE (que vous fixerez à 1000), afficher un message « ATTENTION tentative d'affectation suspecte d'un nouveau solde : compte no … » et ne modifiez alors pas cet attribut. Ce test doit aussi être fait lors de l'initialisation d'un compte.
4. a) Complétez la classe Client générée depuis le diagramme des classes de Visual Paradigm. Ajouter un constructeur allouant le tableau de comptes de ce client et une méthode ajoutant un compte à ce client (on en profitera pour mémoriser que le propriétaire de ce compte est justement ce client!).
   b) Complétez la classe *Banque* dans laquelle vous ajouterez un main (). Dans ce main(), déclarez et créez quelques instances des classes *Client* et *CompteCourant*. Initialisez les attributs de ces comptes et de ces clients à l'aide de leurs constructeurs.
   c) Programmez la méthode toString() dans chacune de vos classes comme indiqué à la question 14. Cela vous permettra facilement d'afficher vos objets pour vérifier que les opérations que vous avez effectuées dessus se sont bien passées.
5. Dans le main() de la classe Banque, essayez de modifier le solde de ces comptes en accédant directement à leurs attributs.
   Est-ce possible ? ……...
   Si ce n'est pas possible : Est-ce une erreur de compilation ou d'exécution ? ……..
   Quel est le message d'erreur ? ……..
6. Vous devez maintenant gérer le fait qu'une banque a plusieurs comptes bancaires (à l'aide d'un tableau). Pensez à encapsuler la classe Banque. Vous devez par exemple pouvoir ajouter un nouveau compte aux comptes existants.

7. Dans la méthode main () de Banque, créez deux banques, des comptes que vous ajoutez à l'une ou l'autre banque.

8. Ecrire une méthode qui parcourt tous les comptes d'une banque et affiche les informations sur chaque compte. Est-ce mieux d'écrire une méthode de classe (static) ou une méthode d'objet ?
   ……
   ……

9. Ecrire les méthodes qui permettent d'afficher les informations sur tous les comptes d'un client spécifié par son nom (on supposera pour simplifier qu'il n'y a pas d'homonymes) : numéro du compte et solde.
   ATTENTION : pour tester si 2 chaînes de caractères sont égales, il faut utiliser la méthode .equals:
   if (s1.equals(s2)) …
   en effet l'opérateur == teste si les deux chaînes de caractères sont à la même adresse (ce qui n'est pas toujours le cas, par exemple si une des deux chaînes est lue au clavier)

10. Ecrire une méthode qui, à partir d'un numéro de compte, affiche toutes les informations sur le propriétaire de ce compte.

11. Utilisez maintenant un attribut static pour initialiser automatiquement le numéro des comptes à l'aide d'un compteur.

12. Les attributs static peuvent servir à représenter une variable partagée par toutes les instances d'une même classe pour autre chose que de compter le nombre d'instances créées.
    Définissez un **attribut static tauxRemuneration** dont la valeur dépend des bénéfices **de toutes les banques au niveau national**. Fixez le à 1% au départ.
    Lors d'un dépôt (méthode crediter), ajouter ce pourcentage multiplié par le montant déposé.
    Prévoyez une méthode qui permet de changer ce taux.
    Dans le main() de la classe Banque, créez quelques comptes et faites des opérations en appelant la méthode.
    Ensuite augmenter le taux de rémunération à 2%. Faites de nouveau quelques opérations et vérifiez que ce nouveau taux est bien pris en compte.

13. Un compte d'épargne est représenté par un numéro, un solde, un taux d'intérêt (qui varie selon chaque compte) et une référence vers le client propriétaire de ce compte. Il n'y a pas de découvert autorisé pour les comptes d'épargne. Créez la classe CompteEpargne.
    Dans la méthode créditer, pour chaque versement ajouter le taux d'intérêt appliqué au montant versé.
    Dans la méthode débiter, il n'y a pas de découvert autorisé pour les comptes d'épargne.
    Pas non plus de seuil de sécurité quand on affecte un solde.
    Ajoutez la gestion d'un tableau de comptes d'épargne dans la banque.
    Dans le main de Banque, créer quelques comptes d'épargne et les ajouter à la banque.

14. Prévoir dans chacune de ces classes une méthode toString() :

```
// Méthode retournant une chaîne de caractères décrivant la valeur des attributs de cet objet
public String toString () {
 String s = "" ;
 s = s + attribut ;
 …
 return s ;
}
```

    Cette méthode toString() est appelée ensuite de manière implicite lorsque vous affichez un objet avec println comme : System.out.println (clientDurant) ; // À UTILISER
    est équivalent à : System.out.println (clientDurant.toString()) ; // A NE PAS UTILISER

15. Dupliquez votre package banque en un package banqueAL. Nous allons maintenant changer **l'implémentation** (les déclarations d'attributs et les instructions du corps des méthodes) de la classe Banque sans en changer l'**interface** (les entêtes des méthodes). => Remplacez les tableaux par des instances de la classe ArrayList (http://docs.oracle.com/javase/7/docs/api/index.html)

Est-ce que les entêtes des méthodes de la classe Banque ont changé ?……..

Est-ce que le main() de la classe Banque a changé ?

……..

Est-ce que les développeurs qui utilisaient la version précédente de la classe Banque doivent changer leurs classes (à part récupérer la nouvelle version de Banque.class) ?

……..