Notes Importantes – TP2

Ce document explique les concepts clés utilisés dans TP2 et pourquoi ils sont importants.

1. Utilisation de BigDecimal pour les montants monétaires

Pourquoi utiliser BigDecimal au lieu de double ou float ?

Les types primitifs `double` et `float` en Java utilisent une représentation en virgule flottante qui peut conduire à des erreurs d'arrondis. Ces erreurs, bien que minimes, sont inacceptables dans des applications financières où la précision est primordiale.

Problèmes avec double/float :

- **Imprécision des calculs** : 0.1 + 0.2 ne donne pas exactement 0.3 avec les doubles
- **Erreurs d'arrondi cumulatives** : de petites erreurs peuvent s'accumuler dans des calculs complexes
- **Incapacité à représenter exactement certaines fractions décimales** : dû à la conversion binaire

BigDecimal offre plusieurs avantages :

- **Précision arbitraire** : peut représenter n'importe quel nombre décimal avec la précision souhaitée
- **Opérations exactes** : addition, soustraction, multiplication et division sans perte de précision
- **Contrôle de l'arrondi** : différentes stratégies d'arrondi disponibles

Exemples concrets dans notre application bancaire

Dans le code du TP :

1. **Initialisation des soldes** : `BigDecimal.ZERO` pour avoir un solde initial précisément à zéro
2. **Dépôts et retraits** : méthodes `add()` et `subtract()` pour des opérations exactes
3. **Comparaisons** : méthode `compareTo()` pour comparer des montants (au lieu des opérateurs <, >, ==)
4. **Formatage** : méthode `setScale()` pour afficher correctement les montants avec 2 décimales

2. L'importance de la méthode toString()

Pourquoi redéfinir toString() ?

La méthode `toString()` est héritée de la classe `Object` et, par défaut, renvoie une chaîne contenant le nom de la classe suivi de l'adresse mémoire de l'objet (ex: `CompteBancaire@15db9742`).

Redéfinir cette méthode présente plusieurs avantages :

- **Débogage facilité** : affichage clair des informations importantes de l'objet
- **Intégration avec d'autres méthodes** : utilisée implicitement par `System.out.println()` et lors de la concaténation avec des chaînes

Pourquoi ne pas simplement créer une méthode normale ?

On pourrait créer une méthode comme `afficherInfos()` qui ferait la même chose, mais :

1. **Standardisation** : `toString()` est une convention universelle en Java
2. **Utilisation implicite** : appelée automatiquement dans de nombreux contextes
3. **Polymorphisme** : permet un traitement uniforme des différentes classes
4. **Intégration avec les bibliothèques Java** : de nombreuses classes Java utilisent `toString()`

3. Utilisation des variables statiques

Compteur de comptes débiteurs

Dans la classe `CompteBancaire`, nous avons utilisé une variable statique `nombreComptesDebiteurs` pour compter le nombre de comptes ayant un solde négatif.

Caractéristiques d'une variable statique :

- **Appartient à la classe et non aux instances** : une seule copie partagée par tous les objets
- **Accessible via le nom de la classe** : `CompteBancaire.getNombreComptesDebiteurs()`
- **Persiste pendant toute la durée de vie du programme** : conserve sa valeur tant que l'application s'exécute

Cette approche permet de maintenir facilement une statistique globale qui concerne tous les comptes bancaires, sans avoir besoin de parcourir tous les comptes existants.

4. Bonnes pratiques de programmation

Encapsulation

Les attributs des classes sont déclarés `private` pour :

- Protéger les données contre les modifications non contrôlées

- Permettre la validation des données (ex: empêcher un solde négatif)
- Masquer l'implémentation interne

Réutilisation de code via this()

Dans le constructeur qui accepte un solde initial, nous utilisons `this(code, titulaire)` pour :

- Éviter la duplication de code
- Centraliser l'initialisation des attributs communs
- Améliorer la maintenabilité (une modification dans un seul endroit)

Validation des données

Nombreuses validations pour garantir l'intégrité des données :

- Vérification que le montant du dépôt est positif
- Vérification que le retrait ne dépasse pas le solde + découvert autorisé
- Vérification que le découvert autorisé est positif