TP8 - Exceptions

Lire un entier à partir du clavier

Créer une méthode `int nextInt(String message)` permettant de lire, à partir du clavier, un entier. Le message passé en paramètre est affiché pour inviter l'utilisateur à taper l'entier. L'entier est d'abord lu sous la forme d'une chaîne de caractères qui est ensuite transformée en un entier.
Écrire un algorithme utilisant cette méthode, et permettant de récupérer et d'afficher un nombre saisi au clavier. Si l'utilisateur n'a pas saisi un entier, une nouvelle saisie est demandée.
On utilisera la méthode `nextLine()` de la classe `Scanner` pour lire une chaîne, `parseInt(String)` de la classe `Integer` pour transformer la chaîne en entier, ainsi que la méthode `flush()` de la classe `PrintStream` si besoin.

Comptes bancaires

L'objectif de cet exercice est d'utiliser les exceptions pour exprimer les responsabilités de la classe `SimpleAccount`, et de voir les conséquences sur un programme de test disponible ici.
On réutilisera la classe définie dans le TP6 (faîtes un copier-coller).

1. Un compte simple est créé à partir d'un titulaire et d'un dépôt initial. Le titulaire doit exister (différent de `null`) et le dépôt initial doit être positif ou nul.
   On créera pour cela deux exceptions `InvalidOwnerException` et `InitialBalanceException` pour détecter ces deux cas d'erreurs.
   1. Dans un premier temps, on utilisera des exceptions non contrôlées par le compilateur Java. Quelles sont les modifications à apporter au programme de test ?
   2. Dans un second temps, on utilisera des exceptions contrôlées par le compilateur Java. Quelles sont les modifications à apporter au programme de test ?
2. Définir une exception `AmountException` qui signale le montant passé en paramètre des méthodes `credit()` et `withdraw()` est incorrect.
   (Quelles sont les informations à associer à cette exception pour qu'elle puisse être traitée dans le gestionnaire d'exception qui décidera de la récupérer ?)
   Modifier le programme de test pour qu'il prenne en compte cette nouvelle exception.
3. Si l'utilisateur des méthodes ne souhaite pas traiter les exceptions mais les propager, il devra alors énumérer tous les cas d'erreurs possibles (ce qui peut-être ennuyeux).
   Proposer et implanter une solution à ce problème.
   Indiquer ses inconvénients.

Maintenant que la classe `SimpleAccount` a été modifiée pour lever des exceptions, nous allons voir l'impact sur la classe `CheckingAccount` (copier-coller celle du TP6).
Essayer de compiler la classe `CheckingAccount`. Quelles sont les erreurs signalées ?
Corriger la classe.

Nous allons maintenant définir la notion de Livret comme étant une spécialisation d'un compte courant avec comme seul ajout le fait que le solde d'un livret ne peut pas dépasser un certain montant (disons 22.950 €). Cela permet de réutiliser la gestion du titulaire, le solde ainsi que l'historique du compte courant.

1. Écrire la classe `SavingsAccount` sans tenir compte du plafond.
2. Implémenter la contrainte de plafond en définissant une exception sous contrôle `LimitExceededException`.
   Quelles sont les erreurs signalées par le compilateur lors de la compilation de `SavingsAccount` ?
   Pourquoi ces erreurs apparaissent-elles ?
   Que fait-il faire pour corriger ces erreurs ?
3. L'utilisation de l'héritage entre `SavingsAccount` et `CheckingAccount` conduit à modifier la sémantique des classes `CheckingAccount` et `SimpleAccount`.
   Si ceci n'est pas souhaitable, il ne faut pas utiliser la relation de spécialisation (et donc l'héritage).
   Comment faire alors pour écrire la classe `SavingsAccount` ?