Interfaces - Classes abstraites

Mathieu Bourgais
mathieu.bourgais@insa-rouen.fr

2023

Préambule

Vous pouvez utiliser l'éditeur de texte de votre choix. Vous pouvez aussi utiliser un IDE comme Eclipse, Visual Studio Code ou encore Intellij, qui présente de nombreux avantages (auto-complétion, par exemple). Dans Eclipse, vous devez toujours travailler dans un projet (à créer au début de chaque TP/mini-projet) dans lequel vous définirez vos classes.

Pour rappel, pour compiler et exécuter un code source Java, vous pouvez utiliser votre éditeur favori, ou ouvrir un terminal puis naviguer jusqu'au répertoire désiré, et taper :

— javac monFichier.java
— java monFichier

1 Exercice 1 - Interfaces

Une **interface** permet de lister un ensemble de méthodes, en spécifiant uniquement leurs signatures. Une interface peut donc être vue comme un prototype de classe : elle spécifie la signature des méthodes, et donc le comportement des classes construites à partir de ce prototype, sans pour autant les décrire précisément. On ne pourra alors pas instancier une interface. En Java, on utilisera le mot clé **interface**.

On dit qu'une classe construite à partir d'une interface implémente cette interface. Pour cela, la classe doit nécessairement définir l'ensemble des méthodes listées au sein de l'interface. En Java, on utilisera le mot clé **implements**.

Une interface définit alors un **type**, et les classes qui implémentent cette interface définissent donc des **sous-types** de ce type de base.

L'interface **Compte** fournie est incomplète. Modifiez-la afin de décrire convenablement le prototype d'un compte en banque. En outre, cette interface devra définir les signatures d'un ensemble de méthodes permettant de :

— Retourner et modifier le numéro du compte

— Retourner et modifier la **Personne** possédant le compte (vous pourrez réutiliser la classe Personne définie lors du TD1)
— Retourner et modifier le solde créditeur du compte
— Retourner et modifier le solde débiteur du compte
— Consulter le solde actuel du compte (à partir de ses soldes créditeur et débiteur)
— Créditer le compte d'un montant passé un paramètre
— Débiter le compte d'un montant passé en paramètre, en faisant attention à ce que le compte soit suffisamment provisionné. Dans le cas contraire, un message sera affiché.

Implémentez cette interface au sein d'une classe **CompteBancaire**. Comme d'habitude, vous définirez deux constructeurs, dont un sans argument, et redéfinirez la méthode **toString()**, en séparant les valeurs des différents attributs par des tabulations.

Compilez vos différentes classes, ainsi que la classe **TestCompte** fournie. Exécutez le programme afin de tester le bon fonctionnement de votre code.

2 Exercice 2 - Classes abstraites

Lors de la création d'une hiérarchie de classes, certaines méthodes peuvent être difficiles à décrire dans des classes mères trop générales. Par exemple, au sein d'une classe **Animal**, comment pourrait-on définir la méthode crier() ? Il serait plus pertinent de la définir indépendamment dans chacune des classes héritant de la classe **Animal**, pour que chaque animal puisse avoir un cri spécifique.

Dans un tel cas, on peut alors définir une méthode abstraite au sein d'une classe mère. En Java, on utilisera le mot clé **abstract** au début de la déclaration de la méthode. Une classe contenant une méthode abstraite est, par définition, également abstraite, et doit donc être déclarée comme telle. En Java, comme pour les méthodes, on utilisera le mot clé **abstract** lors de la déclaration de la classe. Une classe abstraite peut cependant définir des méthodes non abstraites (on parlera alors de méthodes concrètes). Une méthode abstraite doit alors obligatoirement être redéfinie dans les classes filles, afin que ces classes puissent être concrètes.

Les classes abstraites sont principalement utilisées dans des objectifs de factorisation de code, lorsque des méthodes concrètes de la classe mère vont être implémentées de la même manière, quelles que soient les classes filles. Cela permet, en effet, d'éviter de dupliquer le code commun au sein de chacune des classes filles.

Au cours de cet exercice, vous allez raffiner les définition des comptes données précédemment. En particulier, vous allez définir deux nouvelles classes concrètes, **CompteCourant** et **CompteEpargne**, héritant toutes deux de la classe abstraite **CompteBancaire**. La classe **CompteCourant** autorisera un certain découvert maximal, tandis que la classe **CompteEpargne** offrira un certain taux d'intérêts, mais n'autorisera aucun découvert.

Avec ces nouvelles définitions, un compte pourra désormais uniquement être soit un **CompteCourant**, soit un **CompteEpargne**. En particulier, un compte ne pourra donc plus être un simple **CompteBancaire**. Cependant, la classe **CompteBancaire** peut tout de même être conservée afin de factoriser le code commun à **CompteCourant** et à **CompteEpargne**.

2.1 Abstraction

Modifiez le code de la classe **CompteBancaire** afin de rendre cette classe abstraite. Compilez cette classe modifié, et compilez de nouveau la classe **TestCompte**. Exécutez le programme. Que se passe-t-il ? Pourquoi ?

Modifiez à nouveau le code de la classe **CompteBancaire** afin d'y spécifier uniquement le code des méthodes communes aux deux classes **CompteCourant** et **CompteEpargne**. En particulier, vous définirez comme abstraite la méthode **debiter**, devant être implémentée différemment dans chacune des deux classes, et ne spécifierez donc pas son implémentation.

2.2 Création du compte courant

Créez la classe **CompteCourant**, héritant de **CompteBancaire**. Cette classe comportera un nouvel attribut représentant le découvert maximal autorisé. Vous penserez également à définir les accesseurs et mutateurs de ce nouvel attribut. La méthode **debiter** devra alors être redéfinie, afin de prendre en compte ce découvert maximal. Comme toujours, vous définirez également deux constructeurs, dont un sans argument, et redéfinirez la méthode **toString()**.

2.3 Création du compte épargne

Créez la classe **CompteEpargne**, héritant de **CompteBancaire**. Cette classe comportera un nouvel attribut représentant le taux d'intérêts. Vous penserez également à définir les accesseurs et mutateurs de ce nouvel attribut. La méthode **debiter** devra à nouveau être redéfinie, afin de prendre en compte l'absence de découvert autorisé. Vous définirez également une nouvelle méthode **double calculerInterets()** permettant de calculer les intérêts générés par un compte, en fonction de son solde et de son taux d'intérêts. Une fois encore, vous définirez également deux constructeurs, dont un sans argument, et redéfinirez la méthode **toString()**.

2.4 Tests

Compilez vos différentes classes, ainsi que la classe **TestComptes** fournie. Exécutez le programme afin de tester le bon fonctionnement de votre code.

Au sein de la classe **TestComptes**, essayez de modifier le découvert maximal du compte c2 et le taux d'intérêts du compte c4. Compilez et exécutez le programme. Que se passe-t-il ? Pourquoi ?