GLO-3004 — Spécification formelle et vérification de logiciels **Hiver 2019**

Travail pratique spécification formelle– Projet 2 (10%) : *Gestion de comptes bancaires*

À remettre *avant* 23h55, le mercredi **25 avril 2019**.
Pénalité de retard : 0.3 point / 10 par *heure* de retard.
Travail individuel.

Vous devez spécifier un système de *gestion de comptes bancaires* avec le langage Perfect. Un gabarit de spécification vous est fourni. Dans le présent énoncé, les besoins sont énoncés au complet, c'est la spécification en Perfect et la vérification de cette spécification qui sont demandées.

Si l'analyse syntaxique ou la vérification ne passent pas des pénalités importantes seront appliquées.

Définitions des besoins

La spécification fournie décrit une banque qui possède un ensemble de comptes protégés par des mots de passe, NIP, la banque identifie chaque compte de façon unique par un numéro compris entre un et un nombre maximal, `maxNum`, fixé d'avance. Ce nombre représente également le nombre maximal de comptes. C'est au moyen du numéro identifiant un compte donné que la banque accède directement à ce dernier. Un compte comporte un montant constituant le solde du compte, un NIP qui consiste en un entier sauvegardé de façon chiffrée, une date d'ouverture du compte et une date de fermeture valant *null* si le compte est ouvert. Pour la spécification complète vous pouvez vous baser sur le gabarit fourni. Certains choix qui y sont adoptés facilitent grandement la vérification.

Plus précisément, le gabarit fourni spécifie

1. les types et les constantes utilisés, le nombre maximal, le type du numéro de compte,
2. une classe Date qui comporte un jour, un mois et un an, tous des entiers ayant des contraintes. Il faut que la date soit valide. La spécification définit le nombre de jours pour chacun des mois. Pour le mois de février, on se base sur la définition des années bissextiles,
3. une classe Compte dont l'état comporte un solde, un NIP, une date d'ouverture et une date de fermeture (cf Annexe).
   (a) l'initialisation de la classe (le schéma "build"),
   (b) un schéma pour modifier le solde courant. Ce schéma prend en argument un nombre représentant le nouveau montant,
   (c) un schéma pour modifier le NIP. Ce schéma prend en argument un nombre représentant le nouveau NIP. Celui-là doit être différent de l'ancien NIP,
4. une classe Banque, qui a une variable d'état représentant une mappe qui associe à un numéro de compte compris entre un et le nombre maximal, `maxNum`, fixé d'avance, un compte,

   Cette classe comporte
   (a) la variable d'état de la classe,
   (b) un invariant stipulant que le nombre total des comptes est inférieur ou égal au nombre maximal, `maxNum`,
   (c) l'initialisation de la classe (le schéma "build"),
   (d) un schéma pour ouvrir un compte. Ce schéma prend en argument un numéro de compte qui ne doit correspondre à aucun compte existant, un solde qui doit être strictement positif et la date courante qui sera la date d'ouverture du compte. Notez qu'à l'ouverture du compte le NIP est 0 et c'est plus tard que le titulaire du compte pourra le modifier,

(e) un schéma pour fermer un compte. Ce schéma prend en argument un numéro de compte existant et une date différente de `null` (cf Annexe). Le compte doit avoir un solde de zéro et une date de fermeture égale à `null` (cf Annexe). Le compte reste tel quel sauf qu'on remplace la date de fermeture `null` par la date fournie en paramètre.

(f) un schéma pour supprimer un compte. Ce schéma prend en argument un numéro de compte existant et une date différente de `null` (cf Annexe). Le compte doit avoir un solde de zéro et la fermeture du compte doit être différente `null`. En plus la date fournie en paramètre doit être supérieure d'au moins deux ans que la date de fermeture indiquée dans le compte. Le compte est supprimé de la mappe de la banque,

(g) un schéma pour modifier le solde d'un compte donné. Ce schéma doit prendre en argument un numéro de compte existant et un montant qui doit remplacer l'ancien solde,

(h) un schéma pour modifier le NIP d'un compte donné. Ce schéma doit prendre en argument un numéro de compte existant et un NIP qui doit remplacer l'ancien. Le nouveau NIP doit être différent de l'ancien.

votre travail consiste à compléter cette spécification en introduisant les variables et contraintes nécessaires pour effectuer des vérifications financières. Pour le compte des particuliers et dans un souci de transparence financière, il est interdit de déposer plus d'un certain montant d'argent liquide. D'autres variables et contraintes seront rajoutées pour s'assurer qu'il n'y a pas de montants d'argent crées sans qu'ils ne soient bien répertoriés via des transactions enregistrées. Cela a pour objectif d'éviter la fraude incluant le blanchiment d'argent . Il faut s'assurer qu'il n'y a pas de perte indue ni de création d'argent sans source identifiée.

Notez, que j'ai inventé ces règles simples pour calibrer le travail. Il est clair que les banques doivent avoir des normes bien plus complexes que cela.

Ce qui est demandé :

1. Enrichir la classe `Compte` comme ce qui suit.

   (a) Ajoutez une constante, hors de la classe, spécifiant un montant maximal en argent liquide que chaque titulaire de compte a le droit de déposer entre deux exercices financiers de la banque. Ajoutez une variable d'état servant à comptabiliser l'argent liquide déposé et l'invariant qui s'y rapporte pour refléter le non dépassement de dépôt d'argent liquide. **(3 points)**

   (b) Ajoutez une constante, hors de la classe, spécifiant un solde minimal pour le solde d'un compte ouvert. Le solde d'un compte ouvert est toujours supérieur ou égal à ce seuil minimal. Un compte ouvert veut dire un compte dont la date de fermeture vaut `null`. Ajoutez l'invariant qui s'y rapporte pour refléter le respect de ce montant pour un compte ouvert. **(3 points)**

   (c) Modifier le schéma `build` pour prendre en compte ces ajouts. **(4 points)**

   (d) Enlever le schéma qui modifie le solde courant et le remplacer par 4 schémas `retirer`, `deposer` `viderFerme` et `deposerLiquide`. Le premier schéma prend en argument un entier naturel et le retire du solde du compte en veillant à ce que le solde après retrait ne baisse pas en deçà du montant minimal spécifié. et que le compte soit ouvert.

   Le deuxième schéma prend un entier naturel en entrée et le rajoute au solde du compte à condition que le compte ne soit pas fermé. Le 3ème schéma met le solde d'un compte fermé à zéro. Un compte est fermé quand la date de fermeture est une date qui ne vaut pas `null`. Le 4ème schéma prend en argument un entier naturel représentant un montant en argent liquide et le rajoute au solde seulement si le quota de dépôt en liquide n'est pas atteint et que le compte est ouvert. Ce schéma doit rajouter le montant liquide déposé à la variable servant à comptabiliser la totalité d'argent liquide déposé dans ce compte. **(8 points)**

   (e) Ajouter un schéma `changer_depotLiquide` qui modifie la valeur de la variable comptabilisant le cumul d'argent liquide déposé. Ce schéma est appelé dans la classe `Banque` au début de chaque

exercice financier pour remettre à 0 le cumul de dépôts en liquide pour tous les comptes. **(3 points)**

2. Modifier la classe `Banque`, pour respecter certaines règles financières. `soldeGlobal` représente la somme des soldes de tous les comptes à un moment donné. `soldeVerification` représente la somme des soldes de tous les comptes au début de l'exercice financier.

   La valeur de `soldeVerification` + les entrées - les sorties effectuées, jusqu'à un moment données, doit toujours être égal à `soldeGlobal`. Au début de l'exercice, `soldeGlobal` et `soldeVerification` sont égaux et valent la somme de tous les soldes des comptes, `soldeVerification` ne change pas durant l'exercice et `soldeGlobal`, ainsi que les entrées et les sorties varient chaque fois que le solde d'un compte est modifié par un retrait ou un dépôt. Les entrées et les sorties aux comptes de la banque doivent être remises à zéro au début de chaque exercice financier. En plus de ces variables, la banque doit comptabiliser dans une variable appelée `gains` tous les frais bancaires prélevées lors de transactions interbancaires. La vérification financière est planifiée à une date précisée a priori dans une variable d'état `dateBilan`. Le jour et le mois de cette date doivent toujours être égaux respectivement aux `jourDuBilan` et `moisDuBilan` qu'on pourrait mettre respectivement à 31 et 3.

   Plus précisément, modifiez la classe `Banque` ainsi :

   (a) Introduire les variables d'état nécessaires et l'invariant concernant les entrées sorties. Autrement dit le solde global doit être égal à tout moment au solde de vérification plus les entrées moins les sorties. Un autre invariant vérifie que la date du bilan se passe au jour et mois précisés dans les constantes `jourDuBilan` et `moisDuBilan`. Notez, qu'il n'y a pas besoin d'avoir un invariant sur la taille de la variable `comptes`, car le type des numéros de compte garantit qu'il n'y a pas dépassement de `maxNum`. **(4 points)**

   (b) Modifiez le schéma `build`, pour prendre en compte les nouvelles variables d'état. N'oubliez pas de donner en paramètre la date du bilan. La vérification doit s'effectuer à la même date tous les ans et c'est cela qui doit être vérifié. **(6 points)**

   (c) Modifiez les schémas `ouvrirCompte, fermerCompte` et `supprimerCompte` pour prendre en compte les nouvelles directives, s'il y a lieu. Notamment, le solde minimal, un compte ne peut être fermé que si le solde est inférieur ou égal au solde minimal et on ouvre un compte avec un solde supérieur ou égal au solde minimal. On ne peut supprimer un compte que s'il est fermé et que si son solde vaut 0 **(3 points)**

   (d) Ajouter un schéma `viderCompteFerme` qui sert à vider un compte fermé, c'est-à-dire mettre son solde à 0. On considère que ce solde sort de la banque puisqu'il est retiré par le titulaire du compte. **(4 points)**

   (e) À la place du schéma pour modifier le solde d'un compte donné. Spécifier 4 schémas, `retirerDuCompte, deposerDansCompte, deposerLiquideDansCompte`, et `virerEntreComptes` qui ont pour fonction, le retrait, le dépôt, le dépôt d'argent liquide et le virement d'argent entre deux comptes de la banque. Mettez les préconditions et postconditions nécessaires, notamment spécifiez comment on doit mettre à jour le solde du ou des comptes impliqués, le solde global, les entrées et les sorties de la banque. Notez que le virement entre deux comptes de la même banque n'implique pas de frais et consiste à enlever un montant du solde d'un compte et le rajouter au solde de l'autre. **(12 points)**

   (f) le schéma pour modifier le NIP d'un compte ne change pas.

   (g) Spécifiez un schéma `verifierBilan` qui prend en compte une date courante, vérifie que c'est bien la date du bilan et remet le solde global, le solde de vérification à jour en faisant la somme de tous les soldes des comptes de la banque comme ceci, par exemple,
   `(+ over (for i::comptes.dom yield comptes[i].solde))`.
   Ce schéma remet les entrées et sorties d'argent de la banque à zéro et met à zéro également

le montant de dépôt en liquide déjà effectué pour chaque compte et change la date du bilan de la banque en incrémentant l'année de 1, évidemment, il faut vérifier que c'est possible d'incrémenter cette année tout en respectant son type. **(10 points)**

(h) Spécifiez un schéma `virerInterBanques` qui effectue le résultat d'une transaction bancaire vers un compte à l'extérieur de la banque. Ce schéma prend en paramètre le numéro de compte source, qui doit appartenir à la banque, et un montant. Il déduit du solde du compte le montant. Le compte ne doit pas être fermé. Les entrées ou les sorties de la banque doivent être mises à jour ainsi que les gains qui doivent comptabiliser les frais de la transaction sortante prélevés par la banque. **(10 points)**

(i) Spécifiez un schéma `deposerInterBanques` qui effectue le résultat d'une transaction bancaire en provenance d'un compte à l'extérieur de la banque. Ce schéma prend en paramètre le numéro de compte destination de la transaction et le montant reçu. Il rajoute ce montant au solde du compte en question. Le compte destination doit être un compte de la banque. Il ne doit pas être fermé. Les entrées ou les sorties de la banque doivent être mises à jour ainsi que les gains qui doivent comptabiliser les frais de la transaction entrante prélevés par la banque. **(10 points)**

3. Spécifiez une classe `Transactions` pour gérer les transactions entre différentes banques. L'objectif est de gérer les transactions interbancaires en veillant à sauvegarder les montants d'argent qui circulent. Une transaction bancaire consiste en un virement d'un compte bancaire dans une banque vers un autre compte dans une autre banque. La banque source de la transaction doit prélever des frais de sortie du compte qui envoie et la banque qui reçoit prélève des frais d'entrée du compte qui reçoit la transaction. Ainsi, il faudrait prendre soin à ce que le solde global des montants qui sortent des banques soit le même que celui reçu par ces banques plus les frais.

   Vous allez avoir besoin de spécifier un type pour les numéros de banques. Cette classe utilise une mappe qui associe à des numéros de banques des banques. Une banque ne peut être associée à deux numéros différents. Lors des transactions, il faut identifier le numéro de la banque ainsi que le numéro du compte dans la banque.

   Il s'agit de compléter la spécification suivante :

```
const maxB : nat ^= 1000;
class NumB ^= those j: nat :- 1 <= j <= maxB;

final class Transactions ^=
abstract
 var
  banques : map of (NumB -> Banque),
  sortiesComptes, depotsComptes, frais: nat;

invariant

   sortiesComptes= depotsComptes +frais

interface
function banques,  sortiesComptes, depotsComptes, frais;

 build{}
 post
  banques! =  map of (NumB -> Banque){},
  sortiesComptes! = 0,
   depotsComptes! = 0,
   frais! = 0;
end;
```

(a) Spécifier un schéma `inscrireBanque` qui rajoute une banque non existante déjà aux banques pouvant faire des transactions dans ce système. Déterminez les paramètres, les préconditions et les postconditions nécessaires.**(5 points)**

(b) Spécifier un schéma `virer` qui représente une transaction bancaire consistant en un virement d'un compte d'une banque `banque1` vers un compte d'une banque `banque2`. Il faut que les banques soient différentes, que les comptes existent et soient ouverts. Sachant que les banques appliquent des frais, exprimez les préconditions pour que cette opération puisse être effectué ainsi les postconditions. Vous pouvez appeler les schémas déjà définis dans la classe `Banque`. **(15 points)**

Travail à faire

1. Écrire une spécification en Perfect du système de gestion de comptes bancaires qui respecte les demandes suivantes :
   — Pour chacune des opérations fournissez un schéma représentant une opération normale. Ce n'est pas exigée que ça soit une opération totale.
   — Il faut que le *Check* passe avec succès.
   — Il faut que le *Verify* passe sans erreurs, ni warnings ni de messages d'impossibilité de prouver.

Modalités de remise : Vous devez remettre ***par intranet*** les documents suivants dans un .zip. L'entête de votre fichier ".pd" 'doit contenir pour **le nom, le matricule et le nom du programme, (perte de 5% des points sinon)**. Votre .zip doit contenir :

1. un fichier .pdp représentant votre projet.
2. un fichier .pd. **Ce fichier doit compiler (passer la vérification syntaxique et la vérification de perfect developer) en utilisant les commande `sCheck` et `Verify` (perte de 50% sinon), les "warnings" ne sont pas acceptées.**

Autres contraintes :
— Ce travail se fait de façon individuelle.
— Vous devez prendre toutes les mesures nécessaires pour éviter le plagiat. Si deux personnes proposent des solutions trop semblables, des sanctions pourraient être prises conformément au règlement départemental.

Bon travail !

Annexe

La date de fermeture est de ce type `class DateFermeture ^= Date || void;` qui représente l'union de deux types et où `Date` est le même type que celui de la date d'ouverture. Le type `void` est prédéfini en `Perfect` et contient une seule valeur `null`. Celle-ci doit être donnée à la variable fermeture quand le compte est actif, sinon la variable doit prendre la valeur d'une date de type `Date`. Soit `f` une fermeture de type `DateFermeture`, on peut tester son appartenance à un type de l'union ainsi : `f within Date` qui est vraie si `f` est de type `Date` et fausse sinon. Si on veut utiliser `f` comme un des deux types de l'union dans une expression, on fait ceci `(f is Date)` ou `(f is void)`