**TRAVAUX DIRIGES : n°3**

**Implémentation du type abstrait des Comptes**

**MARQUEZ Lucas & ROBIN Florian**

**I – PROBLEMATIQUE :**

Le but de ce TP est de concevoir une implémentation complète pour le type abstrait des Comptes en répondant au cahier des charges imposé par un responsable du secteur Gestion Clients d’une banque. Faisant suite aux cahiers des charges la spécification CASL est également fournie, ce langage est universel et par conséquent adapté à ce type de travail. Cette implémentation va permettre au robot superviseur de contrôler en temps réel des fonctionnalités en ligne qui seront proposées aux clients d’une banque.

**II – REALISATION :**

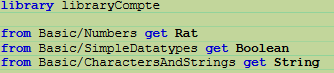
Pour chaque implémentation à réaliser il faut procéder en suivant 4 étapes définies de la manière suivante :

ETAPE 1 : Spécification du type abstrait.

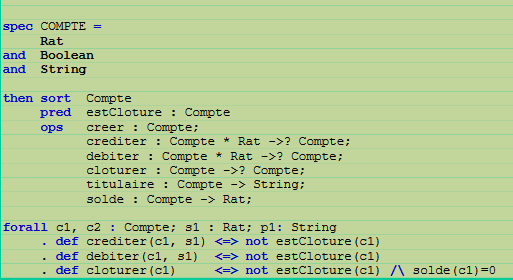
Celle-ci est composée de 3 parties bien distinctes : l’en tête, la signature et la sémantique.

La signature correspond à l’interface de la spécification, c’est-à-dire la partie visible auquel toute personne a accès. Elle permet de déclarer le nom du module de spécification, les types utilisés, les types abstraits à définir ainsi que les opérations du type avec leur signature propre. Ensuite la sémantique est un ensemble de propriétés appelés axiomes qui permettent d’observer, modifier ou de créer l’état d’un objet. Pour finir l’en-tête permet de définir les librairies qui seront utilisés dans la spécification.

***Spécification en CASL des types messages :***

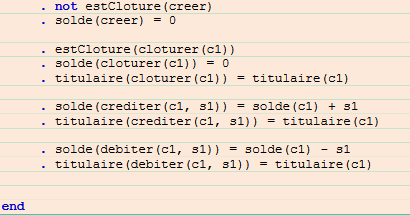


**EN-TETE**



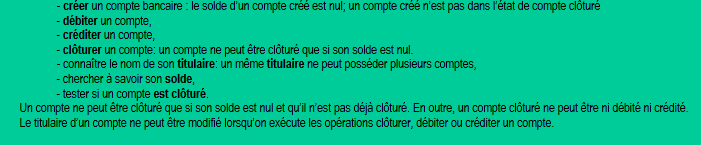
**SIGNATURE**

**SEMANTIQUE**



**SEMANTIQUE**

Afin d’être conforme au cycle de développement du type abstrait, il faut étudier le cahier des charges fourni pour bien respecter la demande. Nous devons donc respecter les fonctionnalités et conditions définies ci-dessous :



Il faut également détailler les mots clés de la spécification afin de ne pas commettre d’erreurs.

**Mots clés du CASL** :   
Library : Librairie(s)

From : Depuis

Get : Prendre (« from Basic/Numbers get Rat  » = Importer Rat (rationnel) depuis la librairie définie avant avec Library)

Spec … : Spécification du type abstrait de …

And : Et, permet de définir les types abstraits utilisés

Then : Puis/ensuite

Sort : Type abstrait défini par la spécification

Pred : Prédicat

Ops : (donner la signature des) Opérations (du type)

« : » :  
Forall … : Pour tout …

. def … : Axiomes définissant les opérations par leurs propriétés

… ⬄ … : … si et seulement si …

Not … : Non …

… /\ … : … et … (et logique)

Pour finir nous allons commenter tous les groupes d’axiomes de la spécification dans le but de les relier au cahier des charges.



Ce groupe d’axiomes fait référence à 3 fonctions différentes, ici on nous impose des préconditions à exprimer afin d’utiliser ces fonctions.

* Def crediter (c1, s1) : Un compte ne peut être crédité que si et seulement s’il n’est pas clôturé.
* Def debiter (c1, s1) : Un compte ne peut être débité que si et seulement s’il n’est pas clôturé.
* Def cloturer (c1) : Un compte ne peut être clôturé que si et seulement s’il n’est pas déjà clôturé et que le solde du compte vaut 0.



Ce groupe d’axiomes fait référence à la fonction créer un nouveau compte, avec un solde nul et un état de compte non clôturé.

* Not estCloture (créer) : On initialise le compte en imposant que le compte ne soit pas clôturé.
* Solde(créer) : A la création du compte le solde doit avoir une valeur de 0.



Ici le groupe d’axiomes fait référence à la fonction clôturer du cahier des charges.

* estCloturer(cloturer(c1)) : Un compte déjà clôturé ne peut pas être clôturé une fois de plus.
* Solde(cloturer(c1)) = 0 : Un compte ne peut être clôturé que si son solde est à 0.
* Titulaire(cloturer(c1)) = titulaire(c1) : Le titulaire du compte doit rester inchangé afin de pouvoir clôturer le compte.



Ce groupe d’axiomes fait référence à la fonction créditer du cahier des charges.

* Solde(crediter(c1, s1)) = solde(c1)+s1 : Un compte peut avoir un solde s1 quelconque ajouté au solde précédent afin de créer un nouveau solde (somme de c1 et s1).
* Titulaire(crediter(c1,s1))= titulaire(c1) : Un compte est crédité que si son titulaire reste inchangé.



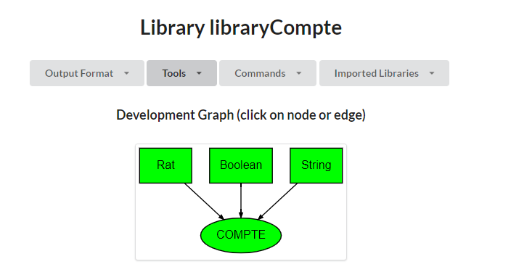
Pour finir la fonction débiter possède ces axiomes.

* Solde(debiter(c1,s1)) = solde(c1)-s1  : Un compte peut avoir un solde s1 quelconque déduit du solde précédent afin de créer un nouveau solde (soustraction de c1 et s1).
* Titulaire(debiter(c1,s1)) = titulaire(c1) : Un compte est débité que si son titulaire reste inchangé.

ETAPE 2 : Validation de la spécification

Pour déterminer si la spécification est correcte il existe différents outils, nous utiliseront ici HETS (Heterogeneous ToolSet) qui est l’un des plus performants. Il a été développé par CoFi. L’analyseur HETS est disponible sur n’importe quel navigateur web à partir de DOLiator : ([*http://rest.hets.eu/*](http://rest.hets.eu/)*)*.

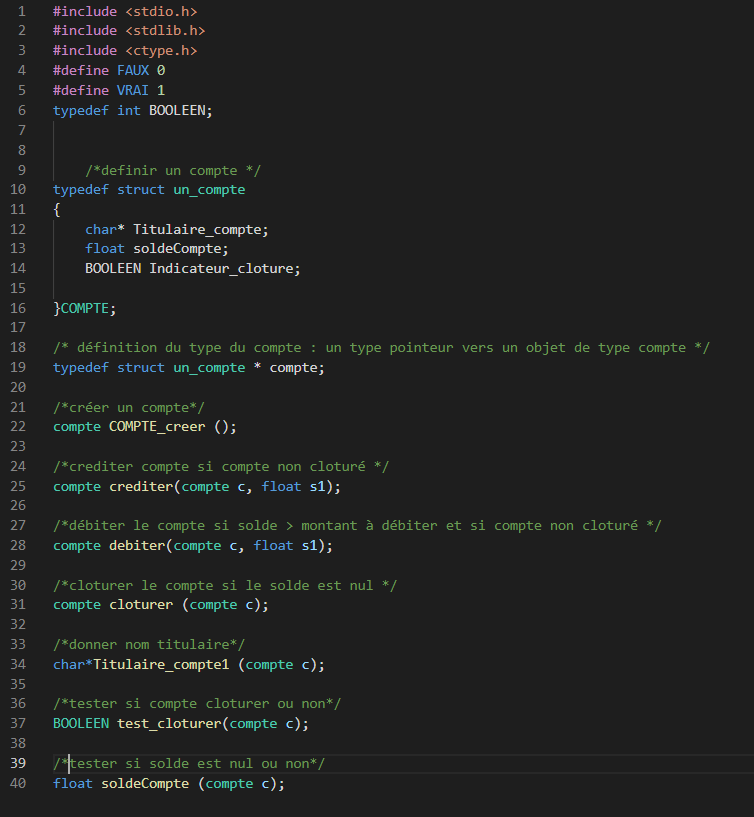
Ainsi valider une spécification c’est prouver formellement que cette dernière satisfait la consistance (pas d’axiomes exprimant des propriétés contradictoires) et la complétude (suffisamment d’axiomes pour savoir si une certaine propriété est vraie ou fausse). Vérifier une implémentation revient à prouver que cette dernière satisfait toutes les propriétés énoncées en spécification.



Voici le résultat de la spécification, on remarque que les types Nat, Boolean et String sont définis, la spécification est donc correcte.

ETAPE 3 : Implémentation de la spécification

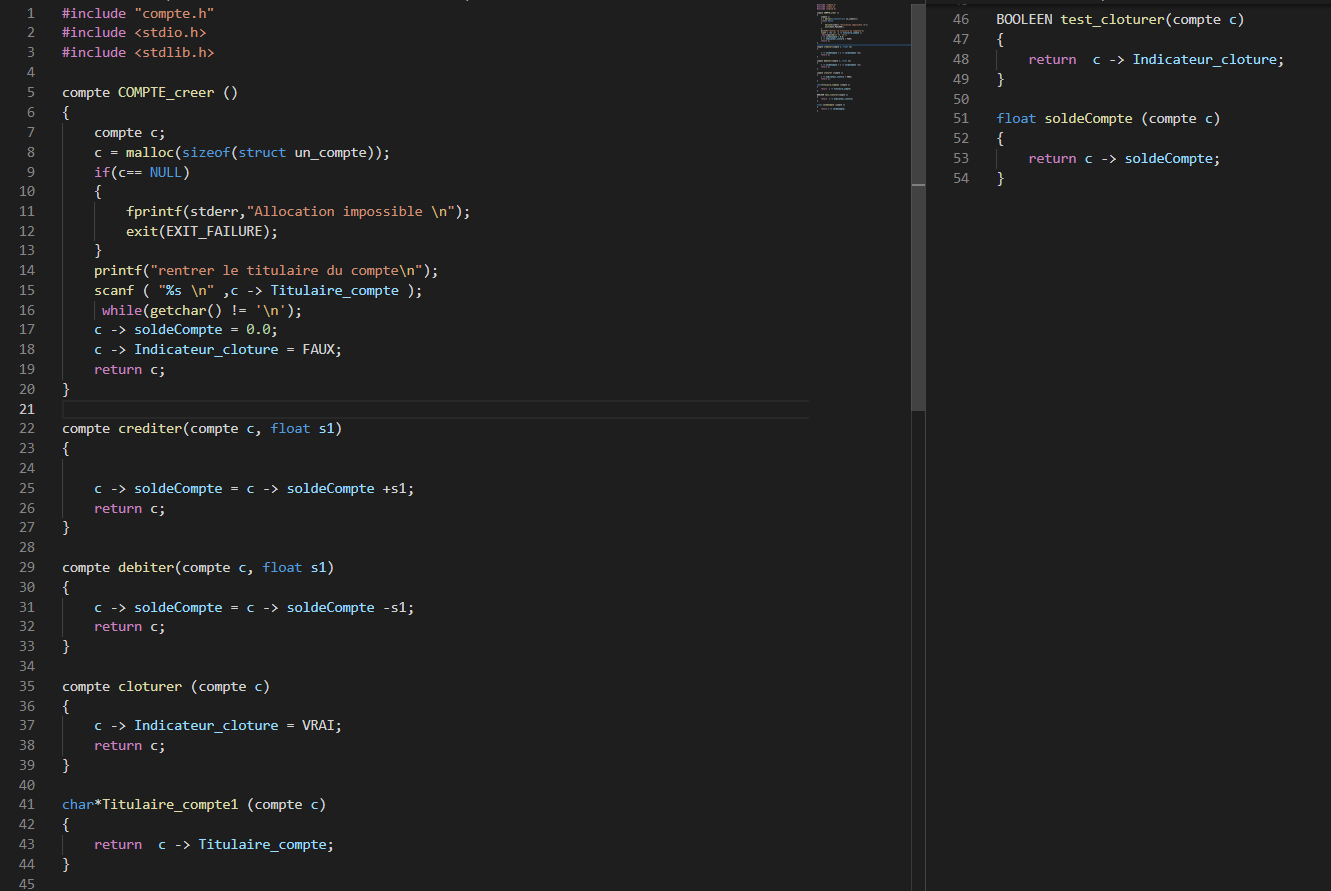
Nous avons donc commencé l’implémentation par le fichier d’interface, le header, dans ce fichier toutes les fonctions seront définies ainsi que le type abstrait Compte.



Définition de l’en-tête des fonctions

Définition du type Compte

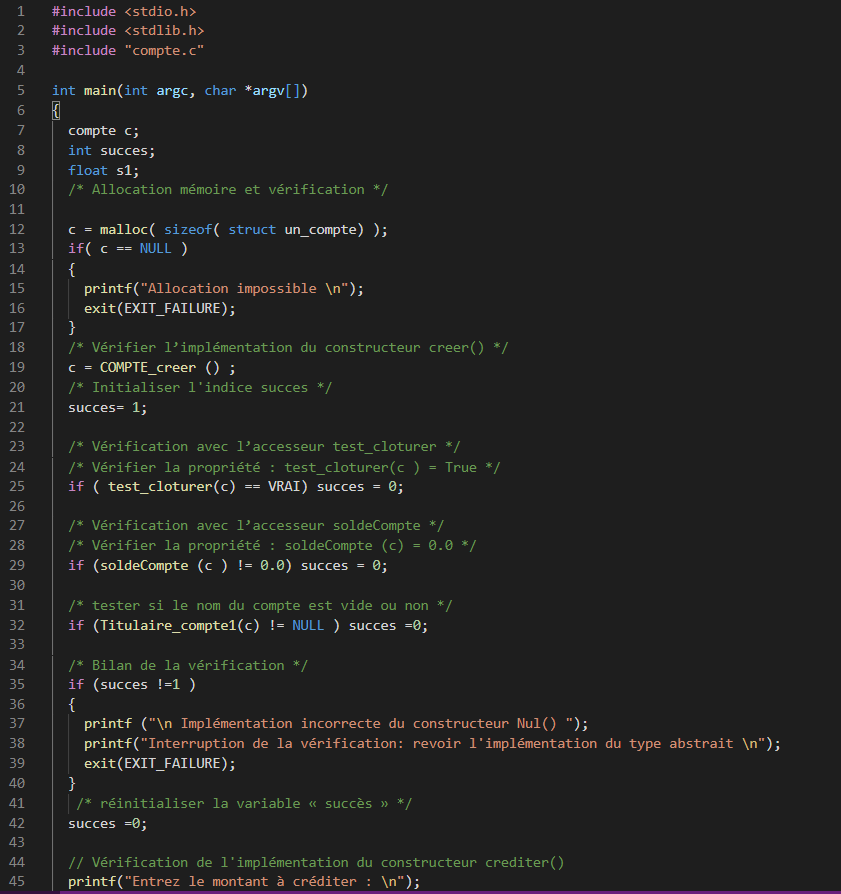
Ensuite nous avons réalisé le fichier « compte.c » où toutes les opérations sur les fonctions y sont décrites. Ce fichier est non consultable et utilise le header car les en-têtes de fonctions sont déjà déclarés.

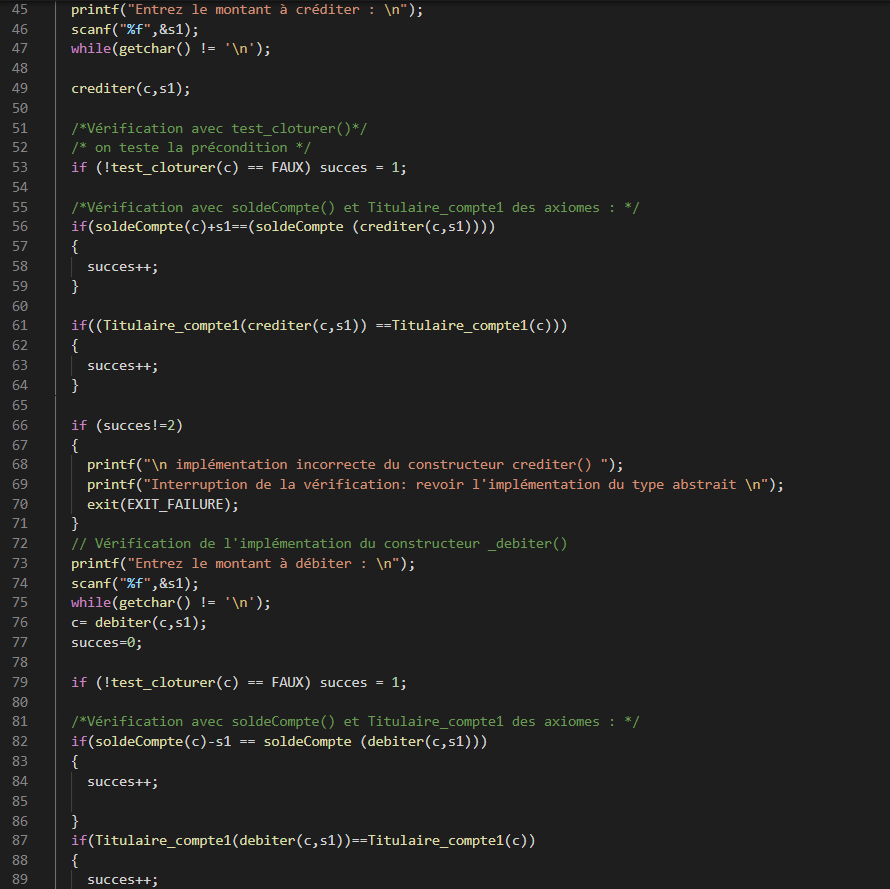


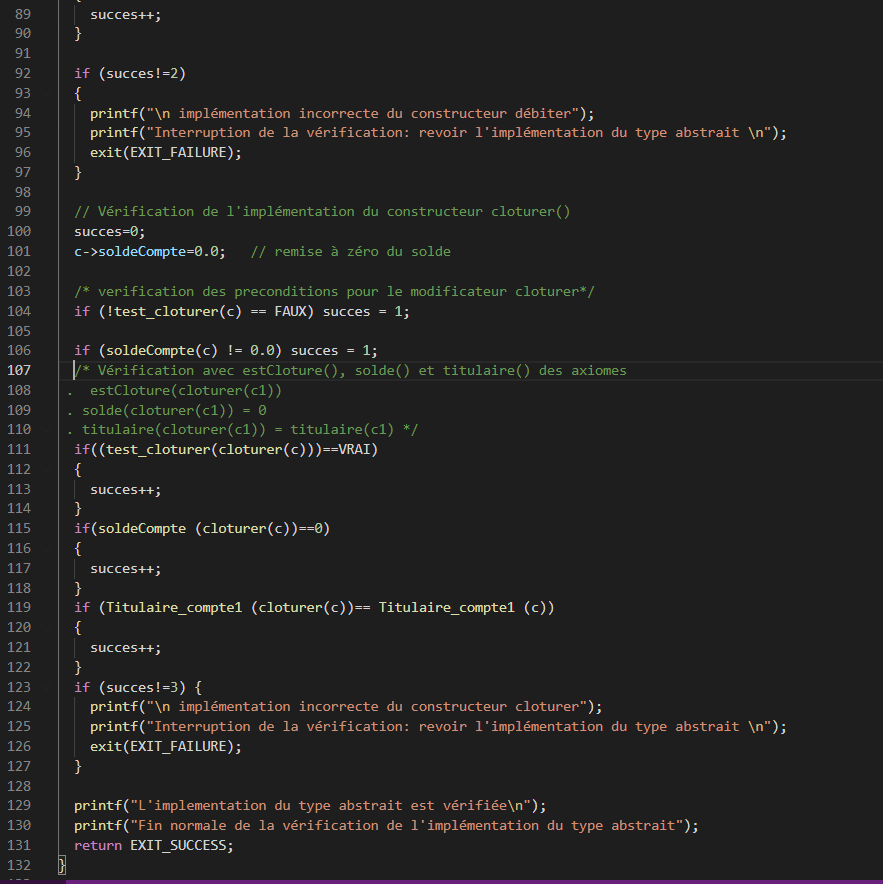
Sur l’image la fonction COMPTE\_Créer () permet d’initialiser le compte, c’est un constructeur. Ensuite les opérations sont décrites dans les fonctions qui suivent, ce sont les modificateurs. Pour finir, tous les accesseurs sont définis dans les 3 dernières fonctions, ils vont nous permettre d’observer l’état de l’objet compte.

ETAPE 4 : Vérification de l’implémentation

Ici il suffit de vérifier que le constructeur COMPTE\_Créer () et les modificateurs sont correctement implémentés. Tous les axiomes doivent être satisfaits pour que l’implémentation soit validée. Cette vérification se déroule dans un fichier nommé « PreuveCompte.c » en cas d’échec le programme s’arrête et renvoie une erreur.







**III – CONCLUSION :**

Sur le plan théorique, réaliser ce TP nous a permis de rentrer dans les détails du cycle de développement, nous pensons avoir mieux compris le rôle des 4 étapes. De plus la compréhension de la spécification est bien plus claire actuellement. On comprend mieux l’utilisation de la spécification qui s’avère très utile et indépendante de tout langage de programmation. La lecture d’une spécification est aussi un peu plus aisée maintenant même s’il reste tout de même du travail à fournir pour qu’elle soit optimale.

Sur le plan pratique, l’application des types abstraits permet de mettre en œuvre la spécification de type abstrait pour en faire un type concret. Elle présente plusieurs avantages, par exemple elle est indépendante de tout langage de programmation, la sémantique reste inchangée. Cette indépendance permet de palier facilement à un changement de langage ou à l’implémentation de nouvelles fonctions.