Hiver 2018

Programmation avancée en C++

GIF-1003

Thierry EUDE, Ph.D

Travail à faire exclusivement en équipe de 2 personnes

à rendre avant
jeudi 26 avril 2018 14h00
(Voir modalités de remise à la fin de l'énoncé)

Tout travail remis constitue une contribution originale et distincte de travaux remis par d'autres. Le plagiat est strictement défendu. Tout travail plagié obtiendra la note 0. De plus, les étudiants ou étudiantes ayant collaboré au plagiat seront soumis aux sanctions normalement prévues à cet effet par les règlements de l'Université.

Par ailleurs, vu l'importance des communications écrites dans le domaine de la programmation, il sera tenu compte autant de la présentation que de la qualité du français et ce, dans une limite de 10% des points accordés.

Travail Pratique #4

Intégration, travail en équipe

UNIVERSITÉ LAVAL
Faculté des sciences et de génie
Département d'informatique
et de génie logiciel

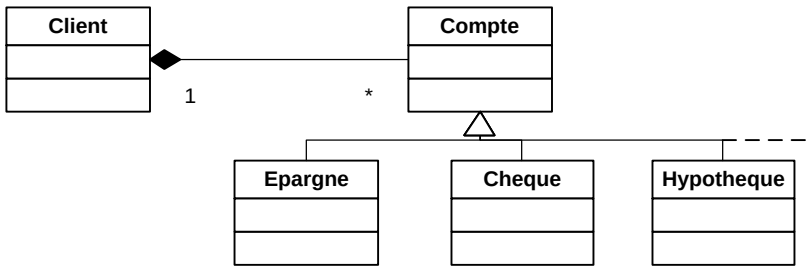
Notre objectif final est de construire un outil de gestion de comptes des clients d'une banque. La série de 4 travaux pratiques devrait tendre vers cet objectif. Chaque travail pratique constituera donc une des étapes de la construction de cet outil.

But du travail

- Utiliser les exceptions
- Respecter des normes de programmation et de documentation.
- Utiliser la théorie du contrat et mettre en place les tests unitaires.
- Utiliser des itérateurs pour les accès aux conteneurs de la STL
- Approfondir la gestion de l'utilisation de la mémoire
- Intégrer des classes préalablement testées pour le développement d'une application
- Développer une application avec interface graphique
- Utiliser un système de gestion de versions (Git)

Utilisation de la hiérarchie des comptes

Au sommet de la hiérarchie, la classe `Compte` est utilisée. Cette classe est spécialisée par des types de compte différents. Le premier est Epargne, le second est Cheque, on pourra en imaginer d'autres dans le futur :



On se servira de cette hiérarchie pour implémenter le polymorphisme.

Ce que vous avez développé pour le TP3 va donc être réutilisé et complété.

Classe Compte

La classe Compte programmée au troisième TP, représente tous les types de comptes d'un client. Elle reste inchangée.

Classe Epargne

La classe Epargne programmée au troisième TP représente les comptes dont les intérêts sont calculés en fonction du taux d'intérêt et du solde. Elle reste inchangée.

Classe Cheque

La classe Cheque représente les comptes dont les intérêts sont calculés en fonction du nombre de transactions et du taux d'intérêt. Elle reste inchangée.

Classe Client

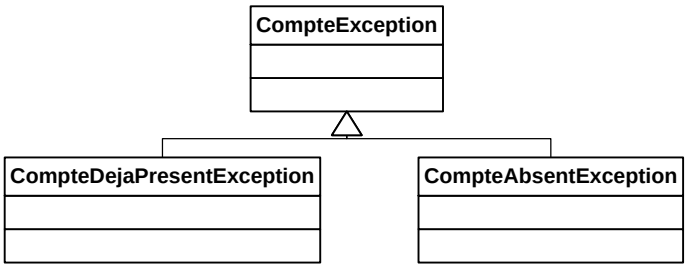
La classe Client programmée au troisième tp, doit être modifiée (**tous les « parcours » de `m_vComptes` doivent être faits à l'aide d'itérateur**) et complétée. Elle permet de faire la gestion des comptes.

La méthode
```
void ajouterCompte (const Compte& p_nouveauCompte);
```
doit également être modifiée. Cette méthode permet d'ajouter un compte à un client **seulement si le compte n'est pas déjà présent dans la liste**.
Dans le cas où le Client possède déjà ce compte (p_noCompte), une exception du type `CompteDejaPresentException` sera lancée. On construit l'exception avec un objet string décrivant la situation. Par exemple :
```
throw CompteDejaPresentException(p_nouveauCompte.reqCompteFormate());
```
pourrait faire l'affaire.

La méthode
```
void supprimerCompte (int p_noCompte);
```
doit être ajoutée à la classe. Cette méthode supprime un compte de la liste dont le numéro est reçu en paramètre. S'il n'y a pas de Client qui possède ce numéro de compte dans la liste des comptes du client, une `CompteAbsentException` est lancée (voir plus bas). Sinon, la méthode doit trouver la position du compte dans le vecteur, avec un itérateur, et appeler la méthode `erase` :
```
m_vComptes.erase (iter);
```
où `iter` pointe sur la position du compte à supprimer. N'oubliez pas de libérer la mémoire avant de supprimer le pointeur dans le vecteur.

Hiérarchie d'exception

Pour ce travail, il est demandé d'élaborer les classes d'une nouvelle hiérarchie d'exception. Toutes ces classes doivent être définies dans les mêmes fichiers nommés `CompteException (.h et .cpp)`, comme dans l'outil fourni pour appliquer la théorie du contrat.



Classe CompteException

Cette classe permet de gérer l'exception liée aux comptes. Cette classe hérite de `std::runtime_error`. Elle contient seulement une méthode qui est le constructeur suivant :
```
CompteException(const std::string& p_raison);
```
Ce constructeur ne fait qu'appeler le constructeur de la classe parent en lui passant la raison comme paramètre.

Classe CompteDejaPresentException

Cette classe permet de gérer l'exception de l'ajout d'un doublon de compte dans le client. Cette classe hérite de `CompteException` parce que cette exception est une erreur qui pourra toujours se produire, ce n'est pas une erreur de programmation.
Elle contient seulement une méthode qui est le constructeur suivant :

```
CompteDejaPresentException(const std::string& p_raison);
```

Ce constructeur ne fait qu'appeler le constructeur de la classe parent en lui passant la raison comme paramètre.
La méthode `what()` de la classe `std::exception` au haut de la hiérarchie permet d'obtenir l'objet string qu'on a passé.

Classe CompteAbsentException

Cette classe permet de gérer l'exception de la tentative d'effacement d'un compte absent dans le Client. Cette classe hérite de `CompteException` parce que cette exception est une erreur qui pourra toujours se produire, ce n'est pas une erreur de programmation.
Elle contient seulement une méthode qui est le constructeur suivant :

```
CompteAbsentException(const std::string& p_raison);
```

Ce constructeur ne fait qu'appeler le constructeur de la classe parent en lui passant la raison comme paramètre.
La méthode `what()` de la classe `std::exception` en haut de la hiérarchie permet d'obtenir l'objet string qu'on a passé.

Théorie du contrat

Les classes doivent implanter la théorie du contrat en mettant les PRECONDITIONs, POSTCONDITIONs et INVARIANTs aux endroits appropriés. Bien sûr, la méthode `void verifieInvariant() const` doit être implantée pour vérifier les invariants. Voir l'exemple avec la présentation de la théorie du contrat. Vous devez aussi déterminer les endroits où tester l'invariant de la classe. Ajouter dans votre projet les fichiers `ContratException.h` et `ContratException.cpp` pour implanter la théorie du contrat.

Test unitaire

Pour chaque classe, vous devez construire un test unitaire selon la méthode prescrite dans le cours en vous appuyant sur la théorie du contrat. Les fichiers de test doivent s'appeler, pour respecter les conventions :

```
ClientTesteur.cpp, CompteTesteur.cpp, CompteEpargneTesteur.cpp, ...
```

Cependant, comme les classes `Compte`, `Cheque` et `Epargne` n'ont pas changé par rapport au TP3, les testeurs pour ces classes ne seront pas évalués, seuls les testeurs des classes modifiées le seront.
Les testeurs des classes d'exception ne sont pas attendus.

Documentation

Toutes les classes ainsi que toutes les méthodes devront être correctement commentées pour pouvoir générer une documentation complète à l'aide de l'extracteur DOXYGEN. Des précisions sont fournies sur le site Web pour vous permettre de l'installer correctement si vous travaillez sur votre propre machine (tout est déjà configuré dans la machine virtuelle mise à votre disposition), et de l'utiliser (syntaxe et balises à respecter, etc.).
Remarque importante :
La documentation du code de la partie intégration (programme « principal ») n'est pas attendue.

Utilisation

Après avoir implanté et testé les classes demandées, vous devez écrire le début d'un programme qui permet de faire la gestion de comptes de clients.
Ce programme devra être « construit » en utilisant le framework Qt.
On aura au départ un seul client (l'évolution future de l'application pourra en prévoir plusieurs).
Les possibilités proposées par l'application sont les suivantes :

- Ajouter un compte au client (deux types de compte possibles)
- Supprimer un compte à un client
- Quitter (pour terminer l'application)

Vous devez développer une interface graphique conviviale, présentant des menus et/ou barres d'outils et/ou boites de dialogue adéquates pour réaliser les opérations précédemment citées.
Par exemple :
Pour l'ajout, le choix d'un type de compte fera apparaitre une fenêtre de dialogue permettant de saisir les informations sur le compte, et ce selon le type de compte choisi précédemment (boites de dialogue spécifiques au type de compte à ajouter). Une fois les informations saisies, l'action sur un bouton ok provoquera un retour à la fenêtre principale. Au centre de celle-ci, on retrouvera l'affichage des informations formatées sur les comptes du client mises à jour.
Pensez à gérer les erreurs telles que par exemple, la tentative d'ajouter un compte existant…
Exemple : Dans le cas d'un compte déjà présent (ajout) ou absent (suppression) un message sera affiché.

Le choix de la présentation (ergonomie, esthétique, …) est laissé à votre discrétion.

Travail d'équipe, utilisation du système de gestion de version

Vous devez vous organiser avec votre partenaire pour la répartition des tâches. Attention, ne vous « spécialisez » pas. Faites en sorte que chacun puisse mettre en pratique toutes les notions abordées. Utilisez un dépôt Git (sur le serveur de la FSG) pour pouvoir partager et suivre l'évolution du code.
Pour votre dépôt Git, vous devez faire une demande de dépôt en utilisant Pixel, menu Applications/Logiciels :

Des tutoriels présentant le fonctionnement de Git, son utilisation à l'aide de différents outils, des exercices, etc. sont à votre disposition sur le site Web du cours; voir la 2ème partie du laboratoire de la semaine 4 Git - Principes et utilisations.
Revoir à l'occasion, l'enregistrement du laboratoire 4.

Modalités de remise, bien livrable

Le quatrième travail pratique pour le cours GIF-1003 Programmation avancée en C++ doit être réalisé **exclusivement en équipe de 2 personnes**. Vous devez remettre votre environnement de développement complet, dans un espace de travail (workspace) Eclipse mis dans **une archive .7z** en utilisant le dépôt de l'ENA. Attention, vérifiez qu'une fois déplacé et décompressé, votre workspace est toujours fonctionnel (il doit donc être « portable »). Pensez au correcteur ! Sachez qu'il utilisera la machine virtuelle fournie pour le cours.

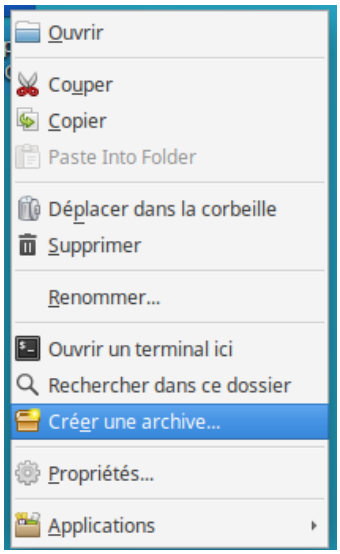
⚠ Très important ⚠

Pour faire votre archive, et pour éviter les problèmes dus à une archive trop volumineuse à la remise, auparavant dans le répertoire de votre workspace :

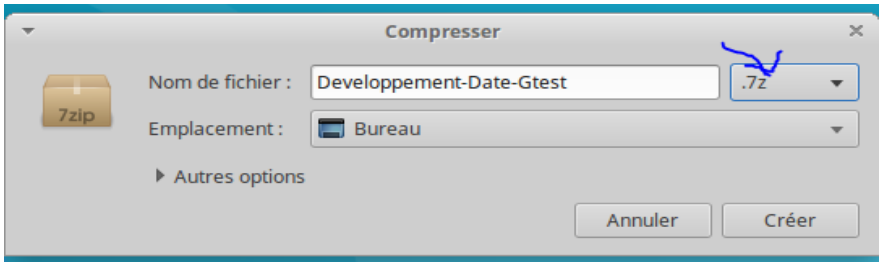
- .metadata/.plugins/org.eclipse.cdt.core/, supprimer tous les fichiers d'extension .pdom.
- .metadata/.plugins/org.eclipse.epp.logging.aeri.ide/ supprimer le répertoire org.eclipse.epp.logging.aeri.ide.server
- La documentation générée avec Doxygen. Le correcteur devrait pouvoir la régénérer par un simple clic dans Eclipse.

Rappel: un tutoriel est disponible sur la page des travaux pratiques pour vous guider, mais attention, bien choisir le format **7z** et non zip comme dans le tutoriel.
Utilisez l'outil natif de la machine virtuelle Linux du cours:

Clic droit sur le répertoire



choisir créer une archive, sélectionner **.7z** (premier de la liste)



Ce travail est intitulé TP 4. Aucune remise par courriel n'est acceptée.

Vous pouvez remettre autant de versions que vous le désirez.
Pensez à supprimer vos anciennes versions sur l'ENA pour ne laisser que celle qui sera corrigée. **Il est de votre responsabilité de vous assurer de ce que vous avez déposé sur le serveur.**

Date de remise

Ce travail doit être rendu avant le jeudi 26 avril 2018 14h. Pour tout retard non motivé (voir plan de cours; motifs acceptables pour s'absenter à un examen), la note 0 sera attribuée.

Critères d'évaluation

1) *Respect des normes de programmation,*
2) *Documentation avec DOXYGEN (uniquement pour les classes développées)*
3) *3) Structures, organisation du code, personnalisation de l'environnement de développement (création de template pour la complétion, changement de préférences par défaut, ...) (très important!)*
4) *Exactitude du code*
5) *Théorie du contrat et tests unitaires*
6) *Le programme principal (fonctionnalité, convivialité de l'interface)*
7) *Utilisation du système de gestion de versions (Git)*

Particularités du barème

- *Si des pénalités sont appliquées, elles le sont sur l'ensemble des points.*
- *Si un travail comporte ne serait-ce qu'une erreur de compilation, il sera fortement pénalisé, et peut même se voir attribuer la note zéro systématiquement.*
- *Il est très important que votre travail respecte strictement les consignes indiquées dans l'énoncé, en particulier les prototypes des méthodes, les noms des fichiers et la structure de développement sous Eclipse sous peine de fortes pénalités*

Bon travail!