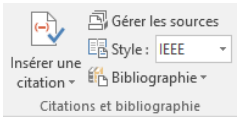
Automne 2021

Programmation avancée en C++

GIF-1003

Thierry EUDE, Ph.D

Travail à faire exclusivement en équipe de 2 personnes



à rendre avant
jeudi 9 décembre 14h
(Voir modalités de remise à la fin de l'énoncé)

Tout étudiant qui commet une infraction au Règlement disciplinaire à l'intention des étudiants de l'Université Laval dans le cadre du présent cours, notamment en matière de plagiat, est passible des sanctions qui sont prévues dans ce règlement. Il est très important pour tout étudiant de prendre connaissance des articles 23 à 46 du Règlement disciplinaire. Celui-ci peut être consulté à l'adresse suivante:
http://ulaval.ca/reglement-disciplinaire
Tout étudiant est tenu de respecter les règles relatives au plagiat. Constitue notamment du plagiat le fait de:

- remettre un travail copié d'un autre étudiant (avec ou sans l'accord de cet autre étudiant);
- remettre un travail téléchargé d'un site d'achat ou d'échange de travaux scolaires.

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.
Tous les travaux sont soumis à un logiciel de détection de plagiat.

Travail Pratique #4

Intégration, travail en équipe

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

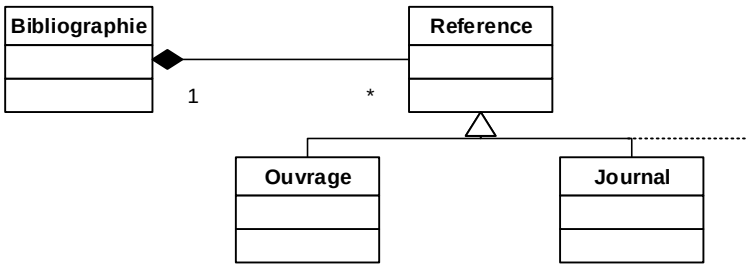
Notre objectif final est de construire un outil de gestion de références bibliographiques. La série de 4 travaux pratiques devrait tendre vers cet objectif. Chaque travail pratique constituera donc une des étapes de la construction de cet outil, les évaluations intermédiaires étant prévues pour valider régulièrement votre développement pour bien avancer.

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)

Mise en place de la hiérarchie de références

Au sommet de la hiérarchie, la classe Référence sera utilisée. Cette classe est spécialisée par deux types différents. Le premier est Ouvrage et le second est Journal.



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

Classe Reference

La classe `Reference` programmée au troisième TP, représente ici tous les types de Références d'une bibliographie. Elle reste inchangée.

Classe Ouvrage

La classe `Ouvrage` programmée au TP3 reste inchangée.

Classe Journal

La classe `Journal` programmée au TP3 reste inchangée.

Classe Bibliographie

La classe `Bibliographie` permet de faire la gestion des Références. Cette classe développée au TP3 doit être modifiée.
Toutes les méthodes qui impliquent un parcours du vector doivent le faire en utilisant un itérateur plutôt que d'utiliser l'indexage pour les tableaux (notation [i]).

Vous devez modifier les méthodes suivantes :
```
void ajouterReference (const Reference& p_reference);
```
Cette méthode permet d'ajouter une référence au vecteur des **références seulement si la référence n'est pas déjà présente dans cette liste**.
Dans le cas où la bibliographie possède déjà cette référence (on pourra faire simple en se basant seulement sur l'identifiant), une exception du type `ReferenceDejaPresenteException` sera lancée. On construit l'exception avec un objet string décrivant la situation. Par exemple :
```
throw ReferenceDejaPresenteException(p_reference.reqReferenceFormate());
```
pourrait faire l'affaire.

```
bool referenceEstDejaPresente(const Reference& p_reference) const;
```
Cette méthode permet de vérifier si la bibliographie a déjà une référence. Si oui, elle retourne `true` et `false` sinon. La vérification se fait sur les attributs de la référence (classe de base). On pourra se servir de l'opérateur d'égalité déjà développé au tp2.

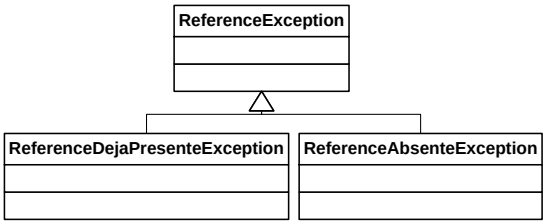
Vous devez également ajouter la méthode suivante
```
void supprimerReference (const string& p_identifiant)
```

Cette méthode supprime la référence dont l'identifiant est reçue en paramètre. S'il n'y a pas de référence qui possède cet identifiant dans la bibliographie, une `ReferenceAbsenteException` est lancée (voir plus bas). Sinon, la méthode doit trouver la position de la référence dans le vecteur, avec un itérateur, et appeler la méthode `erase` :
```
m_vReference.erase (iter);
```
où `iter` pointe sur la position de la référence à supprimer. N'oubliez pas de libérer la mémoire avant de supprimer le pointeur dans le vecteur.

ou (au choix)
```
void supprimerReference (const Reference& p_reference)
```
plus compet mais demande un peu plus de développement au niveau de l'interface gui.

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 le même module nommé `ReferenceException`, comme pour la théorie du contrat.



Classe ReferenceException

Cette classe permet de gérer l'exception liée aux références. Cette classe hérite de `std::runtime_error`. Elle contient seulement une méthode qui est le constructeur suivant :

```
ReferenceException (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 ReferenceDejaPresenteException

Cette classe permet de gérer l'exception de l'ajout d'un doublon de référence pour la bibliographie. Cette classe hérite de `ReferenceException` 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 :

```
ReferenceDejaPresenteException (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 que l'on a passé.

Classe ReferenceAbsenteException

Cette classe permet de gérer l'exception de la tentative d'effacement d'une référence absente dans la bibliographie. Cette classe hérite de `ReferenceException` 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 :

```
ReferenceAbsenteException (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 que l'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 normes : `ReferenceTesteur.cpp`, `OuvrageTesteur.cpp`, `JournalTesteur.cpp`, `BibliographieTesteur.cpp`
Cependant, comme les classes `Reference`, `Ouvrage` et `Journal` 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 (il ne doit pas y avoir d'erreur de compilation de signalée). Des précisions sont fournies sur le site Web (dans activité de la semaine 6, voir également dans les normes de programmation) pour vous permettre de l'utiliser (syntaxe et balises à respecter etc.).

Remarque importante :

Pour limiter la taille de l'archive de votre remise, ne remettez que le ficher de configuration doxyfile, la documentation devant pouvoir être produite par un simple clic pour la générer. Supprimez le répertoire qui contient cette documentation que vous avez produit, ceci avant la remise (voir le tutoriel genererDocumentationDoxygen.pdf).
La documentation du code de la partie intégration (programme « principal ») n'est pas attendue, mais cela ne vous empêche pas de prévoir d'inclure des commentaires pour aider sa compréhension et faciliter le développement.

Utilisation

Après avoir implanté, modifié et testé les classes comme demandé, vous devez écrire un programme proposant une interface graphique (GUI) qui utilise la classe `Bibliographie` avec les différents types de références.
Ce programme devra être « construit » en utilisant le framework Qt. On aura au départ une seule bibliographie dont le nom est « fixé » dans le code (l'évolution future de l'application pourra en prévoir plusieurs).
La fenêtre principale proposera un menu Operations. On y retrouvera les choix correspondants aux actions :

- Ajouter une référence (deux types possibles)
- Supprimer une référence
- 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 fera apparaitre une fenêtre de dialogue permettant de saisir les informations sur la référence, et ce selon le type choisi précédemment (boites de dialogue spécifiques au type de référence). Une fois les informations saisies, l'action sur un bouton ok provoquera un retour à la fenêtre principale.
Le choix de supprimer fera apparaitre une fenêtre de dialogue permettant de saisir l'identifiant de la référence à supprimer de la bibliographie. La confirmation de suppression sera alors demandée et provoquera un retour à la fenêtre principale.
Au centre de celle-ci on retrouvera l'affichage des informations formatées sur la bibliographie mise à jour.
Dans le cas d'une référence déjà présente (ajout) ou absente (suppression) le traitement de ces erreurs devra être prévu en conséquence (exemple : l'ajout n'est pas complété et un message doit prévenir l'usager).
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 que cela puisse être validé) pour pouvoir partager et suivre l'évolution du code (gestion de versions).
Pour votre dépôt Git, si vous ne l'avez pas fait à l'occasion du laboratoire 4, vous devez faire une demande de dépôt en utilisant Pixel (https://pixel.fsg.ulaval.ca/), 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, en utilisant le dépôt de l'ENA (mon portail), remettre votre environnement de développement complet, dans un espace de travail (répertoire maitre contenant des projets) mis dans une archive **.7z**.

Rappel: un tutoriel est disponible sur la page des travaux pratiques pour vous guider. 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 9 décembre 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 biens livrables (très important!)
2) Respect des normes de programmation,
3) Documentation avec DOXYGEN (voir plus haut)
4) Structures, organisation du code (très important!)
   En particulier :
   - Organisation de l'espace de travail
   - Configuration de l'espace de travail (propriétés)
   - Utilisation des outils recommandés (contrat, test unitaire)
   - Adaptation de l'espace de travail (personnalisation)
5) Exactitude du code
6) Théorie du contrat et tests unitaires
7) Le programme principal (fonctionnalité, convivialité de l'interface)
8) 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 noms des méthodes, les noms des fichiers et la structure de développement sous Netbeans sous peine de fortes pénalités*

Bon travail