Hiver 2019

Programmation avancée en C++

GIF-1003

Thierry EUDE, Ph.D

Travail individuel

à rendre avant
jeudi 11 avril 2019 14h
(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 #3

Hiérarchie de classes, contrat, test unitaire, gestion mémoire

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

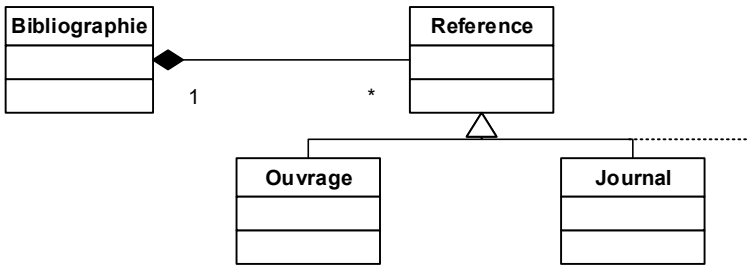
Notre objectif final est de construire un outil de gestion de références bibliographiques. La série de 4 travaux pratiques devrait tendre vers cet objectif. Chaque travail pratique constituera donc une des étapes de la construction de cet outil.

But du travail

- Apprivoiser le processus de développement de plusieurs classes dans un environnement d'implémentation structuré.
- Apprivoiser les concepts d'héritage et de polymorphisme.
- Utiliser un conteneur de STL.
- Respecter des normes de programmation et de documentation.
- Utiliser la théorie du contrat et mettre en place les tests unitaires.

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:



On se servira de cette hiérarchie pour implémenter le polymorphisme.

Classe Reference

La classe `Reference` représente ici tous les types de Références d'une bibliographie. Elle reprend ce qui a été développé dans la deuxième partie du projet de session (TP2). Elle contient les attributs :

`m_auteurs`
: Un objet string; le nom l'auteur ou des auteurs de la référence. Il doit être dans un format valide tel que déterminé par la fonction déjà développée (TP1):
`bool validerFormatNom (const std::string& p_nom)`

`m_titre`
: Un objet string; le titre doit être non vide.

`m_annee`
: Un entier; l'année d'édition de la référence. Elle doit être strictement plus grande que 0.

`m_identifiant`
: Un objet string; l'identifiant de la référence. C'est un code qui dépend du type de la référence. Il peut être un code ISBN, ISSN, … (voir classes dérivées).

Le classe inclut les méthodes suivantes:

- Un constructeur avec paramètres. On construit un objet `Reference` à partir de valeurs passées en paramètre du constructeur seulement si chacun de ces paramètres est considéré comme valide. Autrement, une erreur de contrat sera générée. Les données passées sont : le ou les auteurs, le

titre, l'année d'édition et l'identifiant. Pour déterminer les conditions préalables, voir la description des attributs.

- Des méthodes d'accès en lecture aux attributs de la classe (accesseurs), ceci pour tous les attributs, ainsi qu'une méthode (un mutateur) permettant de modifier les auteurs de la référence en assignant de nouveaux auteurs à la référence courante.

La classe inclut également les méthodes suivantes:
`virtual ~Reference(){} ;`

- Destructeur virtuel (voir manuel page 456 (attention à l'erreur typographique) ou Penser en C++ - Bruce Eckel)

- Un opérateur de comparaison d'égalité. La comparaison se fait sur tous les attributs.

- Une méthode virtuelle pure `reqReferenceFormate` qui retourne dans un objet `std::string` une partie des informations sur une référence formatées sous la forme suivante:

```
Homayoon Beigi. Fundamentals of Speaker Recognition.
```

Utilisez la classe ostringstream du standard pour formater les informations sur la référence.
La déclaration de cette méthode doit imposer son implémentation dans toute classe dérivée. Elle rend la classe `Reference` abstraite.

`virtual Reference* clone() const`
Cette méthode permet de faire une copie allouée sur le monceau de l'objet courant. Dans la classe de base `Reference`, cette méthode est virtuelle pure. Elle est à implémenter dans les classes dérivées.

Classe Ouvrage

La classe `Ouvrage` représente les références de la catégorie Ouvrage. Elle contient les attributs:
`m_editeur;`
: Un objet string; le nom de l'éditeur de l'ouvrage qui doit être dans un format valide tel que déterminé par la fonction : `bool validerFormatNom (const std::string& p_nom)`

`m_ville`
: Un objet string; la ville d'édition de l'ouvrage qui doit être dans un format valide tel que déterminé par la fonction : `bool validerFormatNom (const std::string& p_nom)`

En ce qui concerne les références de la catégorie Ouvrage, leur identifiant est un code ISBN. Le code ISBN doit être dans un format valide tel que déterminé par la fonction `util::validerCodeIsbn` déjà développée (TP1).

La classe inclut également les méthodes suivantes :

- Des accesseurs pour chacun des attributs spécifiques
- Un constructeur avec paramètres. Ce constructeur doit permettre de construire un objet `Ouvrage` valide à partir de données passées en paramètre. Si chacun de ces paramètres n'est pas considéré comme valide, une erreur de contrat sera générée.

Définissez la méthode `reqReferenceFormate :`
La méthode `reqReferenceFormate` qui augmente la méthode de la classe de base `Reference`, retourne dans un objet `std::string` les informations correspondant à un Ouvrage formatées sous le format suivant :

```
Homayoon Beigi. Fundamentals of Speaker Recognition. New York : Springer, 2011. ISBN 978-0-387-77591-3.
```

Ajoutez la méthode :
`virtual Reference* clone() const;`
Celle-ci permet de faire une copie allouée sur le monceau de l'objet courant. Pour faire une copie, il s'agit simplement de faire :

```
return new Ouvrage(*this);  // Appel du constructeur copie
```

Classe Journal

La classe `Journal` représente les références de la catégorie Journal. Elle contient les attributs :
`m_nom;`

Un objet string; le nom du journal dans laquelle a été publiée la référence. Comme il peut être composé de chiffres, la fonction `validerFormatNom` ne peut pas être utilisée ici. On se limitera donc à la contrainte qu'il ne doit pas être vide.

`m_volume;`
`m_numero;`
`m_page;`

Des entiers; respectivement, le volume, le numéro et la page où commence la référence publiée dans le journal.

En ce qui concerne les références de la catégorie `Journal`, leur identifiant est un code ISSN. Le code ISSN doit être dans un format valide tel que déterminé par la fonction `util::validerCodeIssn` déjà développée (TP1)

Il faut aussi prévoir un constructeur. Ce constructeur doit permettre de construire un objet `Journal` valide à partir de données passées en paramètre. Si chacun de ces paramètres n'est pas considéré comme valide, une erreur de contrat sera générée.

La classe inclut des méthodes d'accès qui retournent respectivement les attributs de la classe.

Ajouter les méthodes :
`virtual Reference* clone() const;`
Celle-ci permet de faire une copie allouée sur le monceau de l'objet courant. Pour faire une copie, il s'agit simplement de faire :

```
return new Journal(*this);  // Appel du constructeur copie
```

`reqReferenceFormate`
La méthode `reqReferenceFormate` qui augmente la méthode de la classe de base `Reference`, retourne dans un objet `std::string` les informations correspondant à une référence de journal formatées sous le format suivant (sur une seule ligne):

```
Hart. A survey of source code management tools for programming courses. Journal of Computing
Sciences in Colleges, vol. 24, no. 6, pp. 113, 2009. ISSN 1937-4771.
```

Classe Bibliographie

La classe `Bibliographie` permet de faire la gestion des Références.
Elle contient un constructeur avec un paramètre permettant d'initialiser le nom des bibliographies au moment de leur construction. Elle contient également un accesseur pour pouvoir lire cet attribut.

La classe `Bibliographie` doit contenir toutes les références dans un vecteur :
`std::vector<Reference*> m_vReferences;`
En fait, ce sont des pointeurs à `Reference`. Pour faire du polymorphisme en C++, il est nécessaire d'avoir un pointeur sur un objet.

La classe `Bibliographie` contient la méthode privée suivante:
`bool referenceEstDejaPresente(const std::string& p_identifiant) 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 l'identifiant de la référence (on ne veut pas avoir deux enregistrements avec le même identifiant).

La classe `Bibliographie` contient aussi les méthodes publiques suivantes :
`void ajouterReference (const Reference& p_nouvelleReference);`
Cette méthode permet d'ajouter une référence au vecteur des références. La classe `Bibliographie` conserve des pointeurs sur des références et est responsable de la gestion de la mémoire. Pour réaliser cet objectif, la `Reference` passée par référence constante est clonée et ajoutée dans le vecteur. Les références ont une méthode virtuelle `clone()` pour faire cela.
Exemple d'ajout d'une référence dans le vecteur :
`m_vReferences.push_back(nouvelleReference.clone());`

La méthode publique `reqBibliographieFormate` doit aussi être définie :
Cette méthode parcourt toutes les Références de la bibliographie et retourne dans un objet std::string les informations correspondant à chaque référence, ceci sous le format suivant :

```
Bibliographie
==============================
[1] Homayoon Beigi. Fundamentals of Speaker Recognition. New York : Springer, 2011.
ISBN 978-0-387-77591-3.
[2] Hart. A survey of source code management tools for programming courses. Journal
of Computing Sciences in Colleges, vol. 24, no. 6, pp. 113, 2009. ISSN 1937-4771.
```

Finalement, la classe `Bibliographie` contient aussi un destructeur qui est responsable de désallouer toutes les références d'une bibliographie dans le vecteur.
`~Bibliographie();`

La classe étant d'une forme de Coplien, il faut prévoir, un constructeur copie et un opérateur d'assignation. Bien que ce soit intéressant de les développer, on se satisfera de seulement empêcher leur utilisation par un code utilisateur, ceci pour assurer la fiabilité du code.

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 implémentée 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 implémenter 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 : `Reference_Testeur.cpp`, `Ouvrage_Testeur.cpp`, `Journal_Testeur.cpp`, `Bibliographie_Testeur.cpp`

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 du cours (semaine 5 dans activités) pour vous permettre de l'utiliser (syntaxe et balises à respecter, etc.).
Vous devez respecter les normes de programmation adoptée pour le cours. Ces normes de programmation sont à votre disposition dans la section "Notes de cours / Présentations utilisées en cours / Normes de programmation en C++ " du site Web du cours.
Aussi, comme indiqué dans ces normes, vous devez définir un espace de nom (namespace). Il devra porter le nom suivant : `biblio` (en minuscules). Les classes développées en feront partie.

Utilisation `(gestionBibliographie.cpp)`

Après avoir implanté et testé les classes `Reference`, `Ouvrage`, `Journal`, et `Bibliographie`, vous devez écrire un programme qui utilise la classe `Bibliographie` avec les différents types de références.
Le but est simple, il s'agit juste de créer 4 Références – 2 de chaque type – en demandant à l'utilisateur de saisir les informations nécessaires et de les associer à la bibliographie. Finalement, il faut afficher la bibliographie. Voir l'exemple d'exécution fourni. Cependant, respectez strictement l'ordre et les données saisies.
Rappelons que les références ne peuvent être construites qu'avec des valeurs valides. C'est la responsabilité du programme principal qui les utilise de s'assurer que ces valeurs sont valides. Les critères de validité ont été énoncés dans la description des attributs des classes `References`, `Ouvrage`, `Journal` et `Bibliographie`.

Modalités de remise, bien livrable

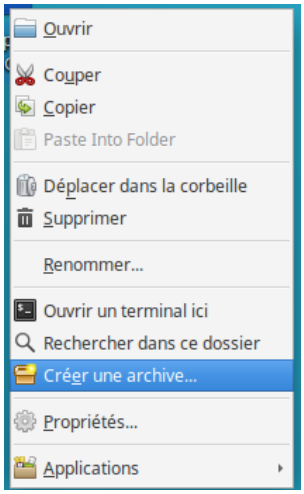
Le troisième travail pratique pour le cours GIF-1003 Programmation avancée en C++ est un travail individuel. Vous devez, en utilisant le dépôt de l'ENA (mon portail), remettre votre environnement de développement **complet**, dans un espace de travail (workspace) Eclipse mis dans une archive **.7z**.

Très important

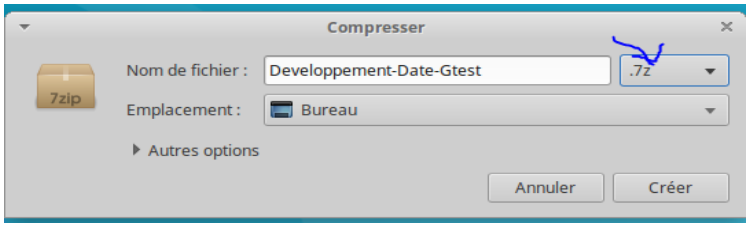
Pour faire votre archive, et pour éviter les problèmes dus à une archive trop volumineuse à la remise, auparavant dans le répertoire de votre workspace :

- .metadata\.plugins\org.eclipse.cdt.core , supprimer tous les fichiers d'extension .pdom.
- .metadata\.plugins\org.eclipse.core.resources supprimer le répertoire .history
- 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 fermez Eclipse avant de procéder**.
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 3. 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.**

Critères d'évaluation

1) Respect des biens livrables (très important!)
2) Respect des normes de programmation,
3) Documentation avec DOXYGEN
4) Structures, organisation du code, personnalisation de l'environnement de développement (template pour la complétion, préférences, etc.)
5) Exactitude du code
6) Théorie du contrat et tests unitaires
7) Utilisation des classes, c.-à-d. le programme principal

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 Eclipse sous peine de fortes pénalités*

Bon travail

Bibliographie

A survey of source code management tools for programming courses. **Hart, Delbert. 2009.** 6, 06 2009, Journal of Computing Sciences in Colleges, Vol. 24, pp. 113-114. ISSN 1937-4771.
Delannoy, Claude. 2017. *Programmer en langage C++.* 9. s.l. : Eyrolles, 2017. p. 886. ISBN 978-2-212-67386-9.