**Département de génie informatique et logiciel**

POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

*INF1010 – Programmation orientée-objet*
TP3

Aut 2023

| | |
|---|---|
| **Remise du travail** | ● Date Lundi 30 Octobre **2023 avant 23h30**<br>● **Une note de 0 sera attribuée aux équipes qui remettent leur travail en retard.**<br>● Remettre **tous** les fichiers directement sous une archive **.zip** (sans dossier à l'intérieur et sans autres fichiers).<br>● ✂**Toute archive autre que zip ou remise ne respectant pas ces consignes sera refusée et se méritera une note de 0. Attention, une archive .7z ne compte pas comme une archive zip.** ✂<br>● Respectez le format de la remise! Nom de l'archive zip : **`matricule1_matricule2_groupe.zip`**<br>Exemple : 1234567_1234568_1.zip<br>● Vous devez uniquement remettre les fichiers .h et .cpp |
| **Références** | ● Notes de cours sur Moodle et le livre Big C++ |
| **Directives** | ● Les travaux s'effectuent obligatoirement en équipe de deux personnes faisant partie du même groupe.<br>● Les fonctions que vous décidez d'ajouter au programme doivent être documentées. |
| **Conseil** | ● Si vous avez des difficultés au cours du TP, rappelez-vous que de nombreux problèmes demandés sont assez couramment rencontrés en C++. Consulter la documentation sur cppreference, les notes de cours, et les forums sur Google peuvent vous donner de bonnes pistes de résolution!<br>● Veuillez utiliser la version v17 de C++. |

✂Tout cas de plagiat sera automatiquement reporté au Comité d'examen de fraude (CEF) ✂

## Spécifications générales

✂ Le non-respect des spécifications générales entraînera des pénalités. ✂

- Vous pouvez utiliser n'importe quel environnement autant que le compilateur soit C++17.
  - Des ressources sont disponibles pour vous aider à débuter avec votre environnement:
    - Visual studio: https://youtu.be/BqLw7RDnNew
    - Visual studio code: https://youtu.be/d5jy7YHLX0w
    - Xcode: https://youtu.be/oTeotPuDPbY

- **L'inclusion des fichiers d'en-tête (`.h`) doit être sensible aux majuscules et minuscules**. Malheureusement, sur Windows les fichiers ne sont pas sensibles aux majuscules et minuscule, donc des erreurs d'inclusions pourraient passer inaperçues. Relisez-vous bien.
- Toutes les méthodes doivent être définies dans le fichier d'implémentation (`.cpp`) **dans le même ordre** que leur déclaration dans le fichier d'en-tête (`.h`). Le non-respect de cette règle entraînera une pénalité au niveau du style.
- Utilisez le plus possible la liste d'initialisation des constructeurs. L'utilisation du corps des constructeurs à la place de la liste d'initialisation entraînera une pénalité de style. L'ordre des variables (attributs) dans la liste d'initialisation doit être la même que celle dans la liste des attributs de la classe dans le fichier d'en-tête.
- Suivez le guide de codage sur Moodle.
- Modifications/ajouts au guide de codage à respecter :
  - Vous pouvez utiliser le style d'accolades que vous désirez, **tant que vous êtes uniformes**. Sachez cependant qu'il est généralement attendu d'un(e) développeur(se) suivre la convention établie par le projet, ou la convention établie par la langue (si elle existe) en créant un nouveau projet. Dans le cas du TP fourni, les accolades ouvrantes sont sur leur propre ligne (style Allman).
  - Mettez un espace avant la parenthèse ouvrante des énoncés de contrôle comme if, for, while et switch, mais pas avant les parenthèses d'un appel de fonction. Ne mettez jamais d'espace tout juste après une parenthèse ouvrante ou tout juste avant une parenthèse fermante.
  - Le deux-points (`:`) et le signe égal (`=`) devraient être entourés d'espaces, sauf dans le cas des étiquettes de `case` et les spécificateurs d'accès (`private`, `protected` et `public`) qui ne devraient pas avoir d'espace avant le deux-points. Un espace devrait toujours suivre les éléments de ponctuation comme la virgule (`,`), le deux points (`:`) et le point-virgule (`;`).
  - N'utilisez pas `NULL` ou `0` pour les pointeurs, mais bien `nullptr`. Dans le cas de test de `nullptr` avec un pointeur, soyez explicite : préférez `if (p != nullptr)` à `if (p)`.
  - N'utilisez pas de `else` après un `return`.

## Mise en contexte

Maintenant que vous avez fait l'implémentation des différentes classes lors du dernier TP2, vous devez désormais vous familiariser avec les notions d'héritage et polymorphisme. Dans ce TP, vous aurez donc à faire l'implémentation de toutes ces notions de manière à faire passer tous les tests.

## Aperçu des classes du projet

- ***Utilisateur :*** Classe qui représente un utilisateur ayant vu certains medias;
- ***Cast :*** Classe de base pour les classes dérivées Acteur et Réalisateur;
- ***Acteur***: Classe qui représente un acteur d'un film ;
- ***Realisateur***: Classe qui représente le réalisateur d'un film ;
- ***Evaluation***: Interface ;
- ***Critique*** : Classe qui représente une critique d'un film;
- ***Media***: Classe de base pour les classes dérivées Film et Serie.
- ***Serie***: Classe qui représente une série avec son nombre d'épisodes et la durée de chaque épisode ;
- ***Film*** : Classe qui représente un film;
- ***Polyflix*** : Classe qui représente l'ensemble des médias et des utilisateurs.

## Travail à réaliser

Implémenter les classes des fichiers .cpp fournis.

## Classe *Cast*

*Cast* représente une classe de base pour les classes dérivées : Acteur et Réalisateur pour le polymorphisme. Cette classe contient les attributs suivants :

- Nom.
- Année de naissance.
- Biographie.
- Salaire.

Les constructeurs sont implémentés. Le destructeur est celui par défaut.
Les méthodes suivantes doivent être implémentées :

- Operator == compare le nom, l'année de naissance et la biographie.
- Operator ! = est la négation de l'opérateur ==.

- La méthode afficher() affiche le nom, l'année de naissance et la biographie.

La fonction globale operator << fait appel à la méthode afficher().

**Classe *Utilisateur***

*Utilisateur* représente un individu ayant visionné certains médias. Cette classe contient les attributs suivants :

- Nom.
- Mot de passe.
- Vector des médias utilisés

La méthode suivante doit être implémentée :

- ajouterMedia() qui ajoute le média utilisé dans le vector.

**Classe Acteur**

*Acteur* représente un acteur et dérive de la classe Cast.
Les attributs sont le nom de l'agence et le vector des rôles joués par l'acteur.

Les méthodes à implémenter sont :
Constructeur par défaut.
Le constructeur par paramètres qui fait appel au constructeur de la classe de base et initialise les autres attributs.
Le constructeur de copie.
La méthode accepterRole() retourne vraie si l'acteur accepte le rôle. L'acteur accepte le rôle

- si le premier élément de la paire du paramètre a un salaire supérieur ou égal au salaire de salaire de l'acteur et
- si le second élément de la paire du paramètre est un pointeur vers un objet Acteur.

La méthode calculerPopularite() retourne la valeur énumération ClasseCelebrite selon le nombre de rôles joués. Si le nombre de rôles est plus grand que le SEUIL_A, alors on retourne A_CELEBRITY et ensuite de suite pour les autres seuils.
L'opérator += ajouter un rôle à l'acteur et fait appel à la méthode ajouterRole().
La méthode afficherRoles() qui affiche la liste des rôles joués par l'acteur.
La méthode afficher() qui affiche les informations de la classe Cast, l'agence et les rôles.

**Classe Realisateur**

La classe *Realisateur* représente un réalisateur de films et dérive de la classe Cast.
Les attributs sont le nom du réalisateur et un vector de pairs qui représente les prix reçus, avec chaque pair contenant l'année et le nom du prix.
Les méthodes à implémenter sont :
Constructeur par défaut.
Le constructeur par paramètres qui fait appel au constructeur de la classe de base et initialise les autres attributs.
Le constructeur de copie.
La méthode ajouterPrix() ajoute l'année et le prix dans les deux vectors.
La méthode accepterRole() retourne vraie si le réalisateur accepte le rôle de réalisation . Le réalisateur accepte le rôle

- si le premier élément de la paire du paramètre a le même salaire que le salaire du réalisateur et
- si le second élément de la paire du paramètre est un pointeur vers un objet Realisateur.

La méthode calculerPopularite() retourne la valeur énumération ClasseCelebrite selon le nombre de prix obtenus. Si le nombre de prix est plus grand que le SEUIL_A, alors on retourne A_CELEBRITY et ensuite de suite pour les autres seuils.
L'opérator += ajoute une année et un prix aux 2 vectors.
La méthode afficherPrix() qui affiche la liste des prix du réalisateur.
La méthode afficher() qui affiche les informations de la classe Cast, et la liste des prix.

**Classe Evaluation**

Evalutation est une interface.

**Classe Critique**

*Critique* représente une critique de film et dérive de l'interface Evaluation.
Les méthodes à implémenter sont :
Constructeur de copie.
L'opérateur =.
L'opérateur ==.
L'opérateur !=.
La méthode obtenirEvaluation() retourne la note de l'évaluation.
La fonction globale operator <<.

**Classe Media**

La classe Media dérive de l'interface Evaluation et est une classe abstraite.
Les méthodes suivantes doivent être implémentées :
La méthode obtenirEvaluation() retourne la moyenne des évaluations du vector de critiques.
L'operator += ajoute un share cast au vector de Cast.
L'operator += ajoute une critique au vector de critiques.
La méthode afficher() affiche les informations de la classe Media : titre, année de sortie, la catégorie, le vector de Cast et de Critique.

La fonction globale operator << fait appel à la méthode afficher().

**Classe Film**

La classe *Film* dérive de la classe Media sert à représenter un film avec la durée du film et la durée du visionnement.
Les méthodes suivantes doivent être implémentées :
La méthode progression() retourne "Temps ecoule: " + to_string(dureeVisionne_) + "/" + to_string(duree_) + " minutes\n"

La méthode afficher() affiche les informations de la classe Media et la durée en minutes.

**Classe Serie**

La classe *Serie* dérive de la classe Media sert à représenter une série avec tous les épisodes et pour chaque épisode la durée de l'épisode.
Les constructeurs.
Les méthodes suivantes doivent être implémentées :
getNbEpisodes() : retourne le nombre d'épisodes
ajouterEpisode() : ajoute un épisode au vector d'épisodes
setEpisodeActuel() : initialise l'attribut episodeActuel_
getEpisodeActuel() : retourne la Valeur de l'attribut episodeActuel_
La méthode progression() retourne "Episode " + to_string(episodeActuel_.first) + ": Temps ecoule " +
to_string(episodeActuel_.second) + "/" + to_string(dureeTotal) + " minutes". La duréetotal est la durée totale de tous les épisodes.

La méthode afficher() affiche les informations de la classe Media et le nombre d'épisodes.

**Classe Polyflix**

La classe *Polyflix* est celle qui fait le lien entre toutes les classes précédentes. Cette classe est une agrégation de medias et d'utiliateurs. Cette classe contient les attributs suivants :

- Vector de share pointeurs à des medias.
- Vector de share Utilisateur.

Les méthodes suivantes doivent être implémentées :

- Le constructeur de copie.
- La méthode chercherUtilisateur() retourne le pointeur de l'utilisateur selon le nom reçu en paramètre.
- La méthode getNombreUtilisateurs() retourne le nombre d'utilisateurs.
- La méthode utilisateurExiste() vérifie si un utilisateur existe.
- La méthode connecterUtilisateur() si l'utilisateur existe, alors on le connecte.
- La méthode modifierMotDePasse() si l'utilisateur existe alors on modifie le mot de passe.
- La méthode visionnerMedia() si le media et l'utilisateur existent alors on ajoute le film visionné à l'utilisateur.
- La méthode getNombreTotalFilms() retourne le nombre de films.
- La méthode getNombreTotalSerie() retourne le nombre de series.
- La méthode getNombreTotatMedia retourne le nombre de medias.
- La méthode chercherMedia() retourne le média identifié par son titre.
- La méthode listerTousLesFilms() retourne un vecteur de films.
- La méthode listerTousLesFilms() retourne un vecteur de séries.
- La méthode listerMediaVisionnesParUtilisateur() retourne tous les medias visionnés par un utilisateur.
- La méthode listerFilmsParCategorie retourne tous les films de la catégorie.
- La méthode listerSeriesParCategorie retourne toutes les séries de la catégorie.

- L'operator += permet d'ajouter le media s'il n'existe pas dans le vector. La vérification de l'existence du media se fait selon le titre du film.
- L'operator += permet d'ajouter l'utilisateur s'il n'existe pas dans le vector.
- La fonction globale operator << () affiche tous les films et les séries.

**main.cpp**

Le programme principal appelle la fonction lancerTests() roule une série de tests pour l'option 1. Il faut écrire les instructions pour les options 4, 5, 6, 7 8.

**Correction**

La correction du TP se fait sur 20 points :

- [3 pt] Compilation du programme sans avertissements.
- [4 pt] Exécution du programme.
- [10 pt] Comportement exact de l'application
- [2 pt] Qualité et documentation du code.
- [1 pt] Absence de fuites de mémoire.

**Relisez bien les spécifications générales et les consignes de remise au début du TP avant de remettre votre travail!**