Module R3.04 – TP04
2 séances de 1h30

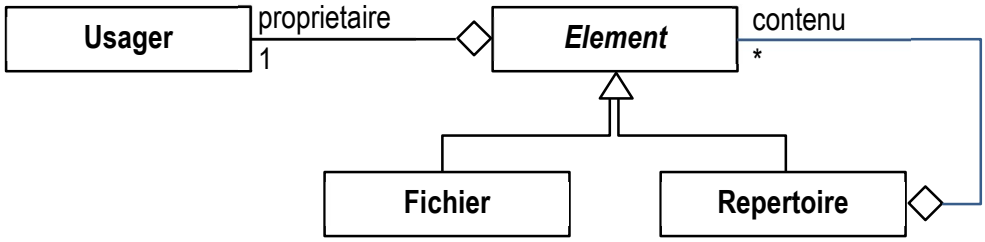
- *Téléchargez l'archive TP04.zip et décompressez-la dans votre répertoire R3-04*
- *Ouvrez dans CLion le projet TP04*

Exercice 1. Pattern Composite (Cours 4 - Chapitre 9)

On veut modéliser le contenu d'un système de stockage. Il y a deux types **d'éléments** à représenter :

- Des fichiers qui sont caractérisés par leur nom, leur taille (en Ko) et leur date de modification.
- Des répertoires qui peuvent contenir des fichiers et/ou d'autres répertoires. Chaque répertoire a également un **nom**. La **taille** et la **date de modification** d'un répertoire dépendent de son contenu :
  - La taille d'un répertoire est égale à 4 Ko **plus** la somme des tailles de tout ce qu'il contient.
  - La date de modification d'un répertoire est ici définie **comme la date de modification du plus récent élément qu'il contient**. Si un répertoire est vide, sa date de modification sera « 1970-01-01 » (date de référence de l'heure Unix).
- Chaque élément (fichier ou répertoire) a également un propriétaire qui est un usager caractérisé par son login (par exemple « martinp ») et son groupe (par exemple « profs »).

Pour modéliser le contenu du disque, on a rapidement identifié qu'il fallait mettre en œuvre le **Pattern Composite** de la façon suivante :



Les Opérations

Sur chaque élément du « Composite », on veut disposer des **opérations** suivantes :

- **`const string & getNom()`** : renvoie le nom
- **`void setNom(const string & nom)`** : change le nom
- **`const Usager & getProprietaire()`**: renvoie le propriétaire
- **`void setProprietaire(const Usager & proprietaire)`**: change le propriétaire
- **`unsigned int getTaille()`** : renvoie la taille
- **`const string & getDateModification()`** : renvoie la date
- **`void setDateModification(const string & date)`** : change la date
- **`void afficher()`** : affiche le nom, le propriétaire (login et groupe), la taille et la date
- **`void ajouter(Element* element)`** : ajoute un élément dans le cas d'un répertoire, lève une exception dans le cas d'un fichier

Les types :

- Nom d'élément, login, groupe sont des chaînes de caractères (**`string`**).
- Taille est un entier non signé (**`unsigned int`**)
- La date de modification est une chaîne de caractères (**`string`**) au format "AAAA-MM-JJ" (exemple : "2020-10-12"). Ainsi l'ordre alphabétique des chaînes correspond à l'ordre chronologique : la chaîne la plus grande correspond à la date la plus récente.

Question 1.1. Mettre en œuvre le Pattern Composite

- Ecrivez la spécification puis l'implémentation des classes **`Usager`**, **`Element`**, **`Fichier`** et **`Repertoire`**
- Lors de la spécification, réfléchissez bien pour factoriser le plus de choses possibles dans la classe **`Element`** afin de mettre en œuvre correctement le Pattern Composite
- Vous devez bien sûr continuer à appliquer les bonnes pratiques vues jusqu'à maintenant
- Le fichier **`exercice1.cpp`** contient un exemple d'utilisation du « Composite », exemple qui doit fonctionner sans que vous ayez besoin de le modifier. La trace attendue est précisée en commentaire dans le fichier.

Question 1.2. Rechercher un élément (facultatif)

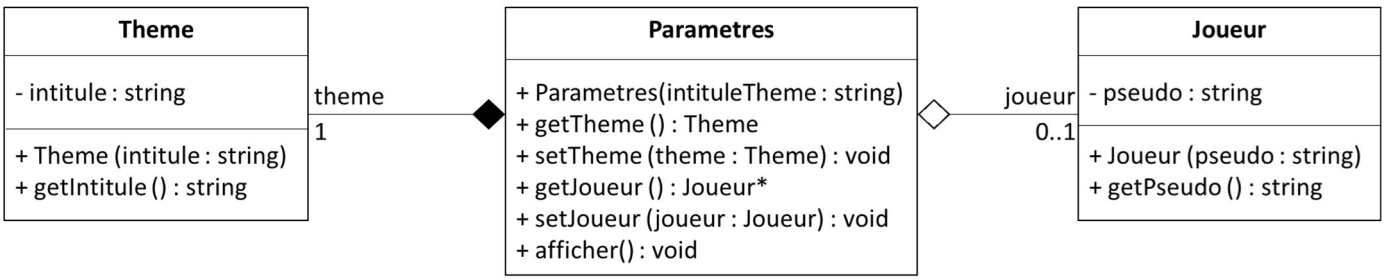
- Ajoutez une opération : **`string rechercher(const string & nom)`**
- L'opération, appliquée à un répertoire, doit renvoyer le chemin complet, depuis ce répertoire, jusqu'à l'élément recherché, s'il existe. Sinon renvoyer une chaîne vide. La recherche doit être récursive : l'élément recherché peut être dans un sous-répertoire. L'opération, si elle est invoquée sur un fichier, renvoie le nom de ce fichier s'il correspond, une chaîne vide sinon.

Exercice 2. Pattern Singleton (Cours 4 - Chapitre 9)

Dans le cadre du développement d'un jeu, on souhaite faire en sorte qu'un seul et unique objet « **Parametres** » soit accessible partout dans le code du jeu. Les paramètres du jeu sont les suivants :

- Un thème (**obligatoire**), modélisé par un objet de la classe **Theme** (fournie)
- Un joueur authentifié, modélisé par un objet de la classe **Joueur** (fournie). Le joueur est **facultatif** car il est possible de jouer en « anonyme »

Le diagramme UML de ces classes est le suivant :



Pour simplifier on a mis un minimum d'attributs et de méthodes dans les classes **`Theme`** et **`Joueur`**

Problème : On ne veut pas devoir passer explicitement un objet **`Parametres`** à toutes les méthodes qui en auront besoin (c'est pénible). On ne veut pas non plus déclarer un objet **`Parametres`** en variable globale (c'est moche). On va donc appliquer le **Pattern Singleton** à la classe **`Parametres`**.

Question 2.1. Mettre en œuvre le Pattern Singleton

- Spécifiez et implémentez la classe **`Parametres`**
- Tant qu'un thème n'a pas été précisé, le singleton **`Parametres`** doit comporter un thème intitulé « Par Défaut »
- La méthode **`afficher()`** devra afficher les paramètres à l'écran, c'est-à-dire l'intitulé du thème et le pseudo du joueur s'il existe, « Anonyme » sinon.
- Testez le singleton **`Parametres`** en complétant le fichier **`exercice2.cpp`** selon les indications fournies en commentaires. Vous devrez obtenir la trace également indiquée en commentaire.