Noah Blanchard – 2192826
Ina Laporte – 2155116

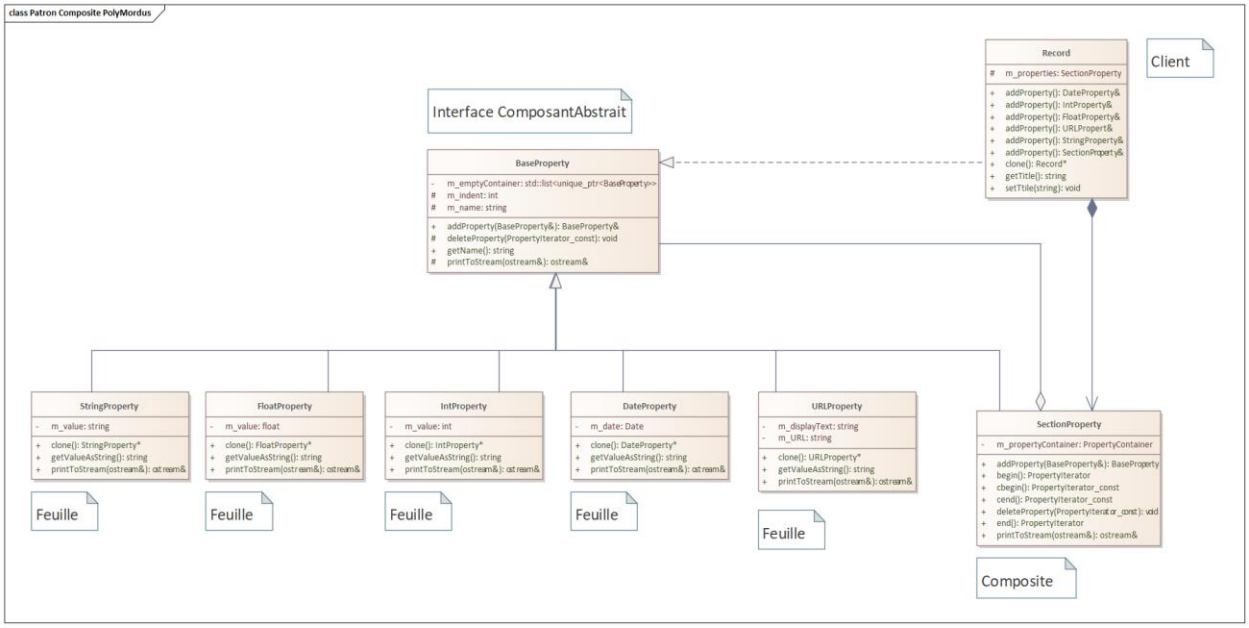
Rapport TP5 – LOG1410

Patron Composite

Question a)

L'intention première du patron composite est de traiter les différents objets, dans notre cas les différentes propriétés, de façon uniforme et sous le même nom. En effet, dans notre cas, toutes les propriétés de différents types (*StringProperty*, *IntProperty*), peuvent être traiter dans les code comme des *BaseProperty* qui est le composant abstrait du projet PolyMordus.

Question b)



Voici un diagramme de classe représentant le patron composite dans PolyMordus.

Question c)

L'attribut « m_indent » au sein de *BaseProperty* est l'attribut qui permet de gérer le niveau d'indentation au moment d'imprimer les informations dans la console. Cet attribut est « static » pour qu'il soit un attribut de classe et non un attribut d'instance, « m_indent » a donc la même valeur aux yeux de n'importe quelle instance de *BaseProperty* (ou des classes filles). Si l'attribut n'était pas « static », alors sa valeur serait unique pour chaque instance de propriété.

Question d)

La méthode « printToStream » est une méthode implémentée par toutes les classes filles (les feuilles) de *BaseProperty*. Elle permet de compléter l'appel de l'opérateur surchargé « << » et d'afficher les bonnes informations et le bon format en fonction de la propriété affichée. En effet, même si chaque propriété est traitée comme une *BaseProperty*, lorsque la méthode « printToStream » est utilisée, ce sera la bonne implémentation de celle-ci qui sera appelée.

Noah Blanchard – 2192826
Ina Laporte – 2155116

Patron Itérateur

Question a)

Le patron itérateur permet de fournir une manière d'accéder à une collection d'éléments de façon séquentielle. L'intérêt et d'éviter d'exposer la structure de la dite collection, tout en permettant l'accès aux éléments qu'elle contient.

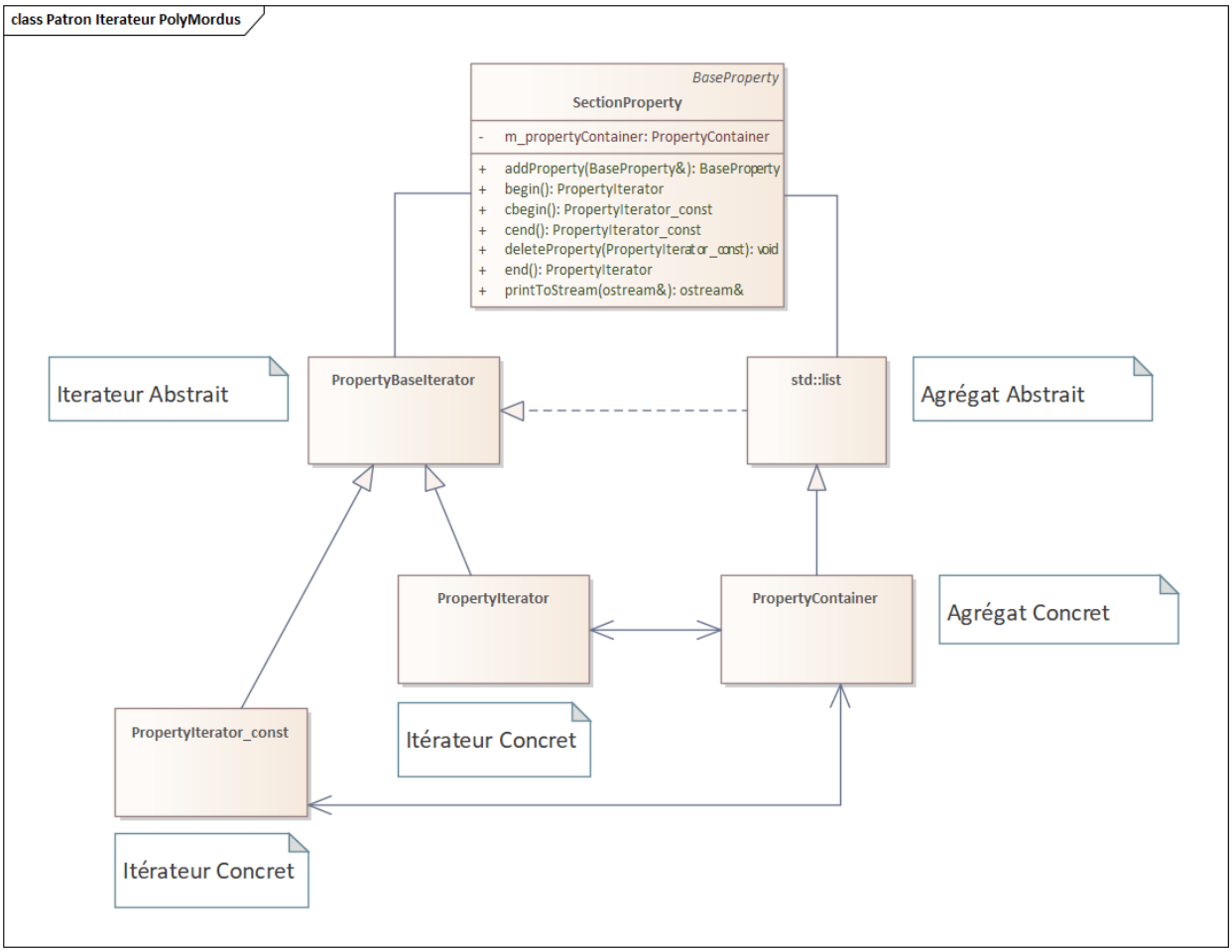
Question b)

Dans la conception fournie, on retrouve au sein de *SectionProperty* (le composite) un attribut « PropertyContainer m_propertyContainer; », et *PropertyContainer* et en réalité définie comme suit :

« using PropertyContainer = std::list<PropertyPtr>; »

La classe de la STL utilisée est donc la classe « list » qui est une classe de base pour les conteneurs de type liste qui sauvegarde et conserve les éléments dans une disposition linéaire.

Question c)



Voici un diagramme de classe représentant le patron itérateur dans PolyMordus.

Noah Blanchard – 2192826
Ina Laporte – 2155116

Question d)

L'attribut « m_emptyContainer » est « static » car cela permet de ne l'instancier qu'une seule fois pour toutes les instances de classes propriétés. Le même « m_emptyContainer » est partagé entre toutes les *BaseProperty*, et cet objet peut donc être renvoyé par *BaseProperty* sans avoir besoin d'être instancié de multiples fois. Il est aussi privé car il ne doit pas être accessible ou utilisable à l'extérieur, et ce même dans les classe de propriétés filles.

Question e)

Nous pouvons par exemple remplacer le conteneur par un « vector » de la STL :

```
private:
    /*PropertyContainer m_propertyContainer;*/
    std::vector<PropertyPtr> m_propertyContainer;
};
```

Faire ce changement implique aussi de changer le type de retour des méthodes « begin », « end », « cbegin », « cend ». En effet, pour l'instant ces méthodes renvoient des « PropertyIterator » ou des « PropertyIterator_const », mais comme nous avons modifié le type en un « std ::vector », il faut retourner plutôt « std::vector<PropertyPtr>::iterator » et « std::vector<PropertyPtr>::const_iterator ». Ces changements doivent aussi affecter *BaseProperty* qui est la classe mère de *SectionProperty* et qui dépend aussi du type d'itérateur.

Il faudrait donc changer les types de retour dans *SectionProperty* et *BaseProperty*, ou bien de façon plus générale, éditer les types dans *PropertyContainer.h*.

Noah Blanchard – 2192826
Ina Laporte – 2155116

Patron Proxy

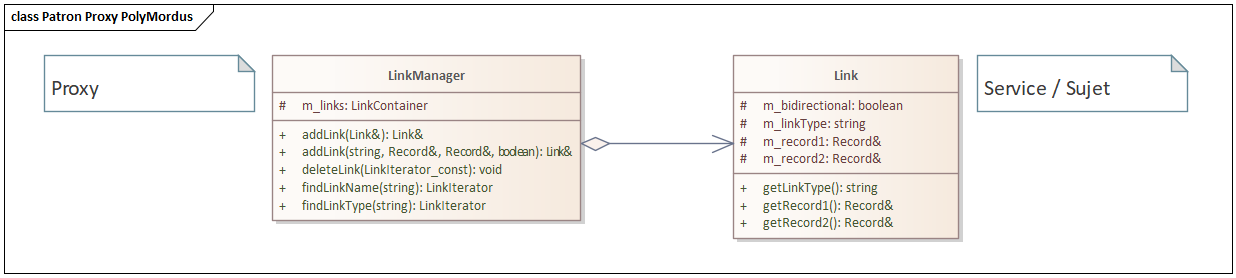
Question a)

Le patron proxy a pour objectif de proposer un substitut ou une alternative pour un objet dans le but de réguler son accès. En effet, ce patron propose un « proxy » qui contrôle l'accès à l'objet d'origine en permettant des traitements supplémentaires avant ou après avoir effectué une requête sur l'objet d'origine.

Question b)

Dans le cas de la conception proposée, le type de patron proxy utilisé est celui dit de « proxy virtuel ». En effet, le proxy virtuel est chargé de créer un ou des objets sur demande, et c'est le cas de *LinkManager* qui va créer sur demande des objets Link.

Question c)



Voici un diagramme de classe représentant le patron proxy dans PolyMordus.