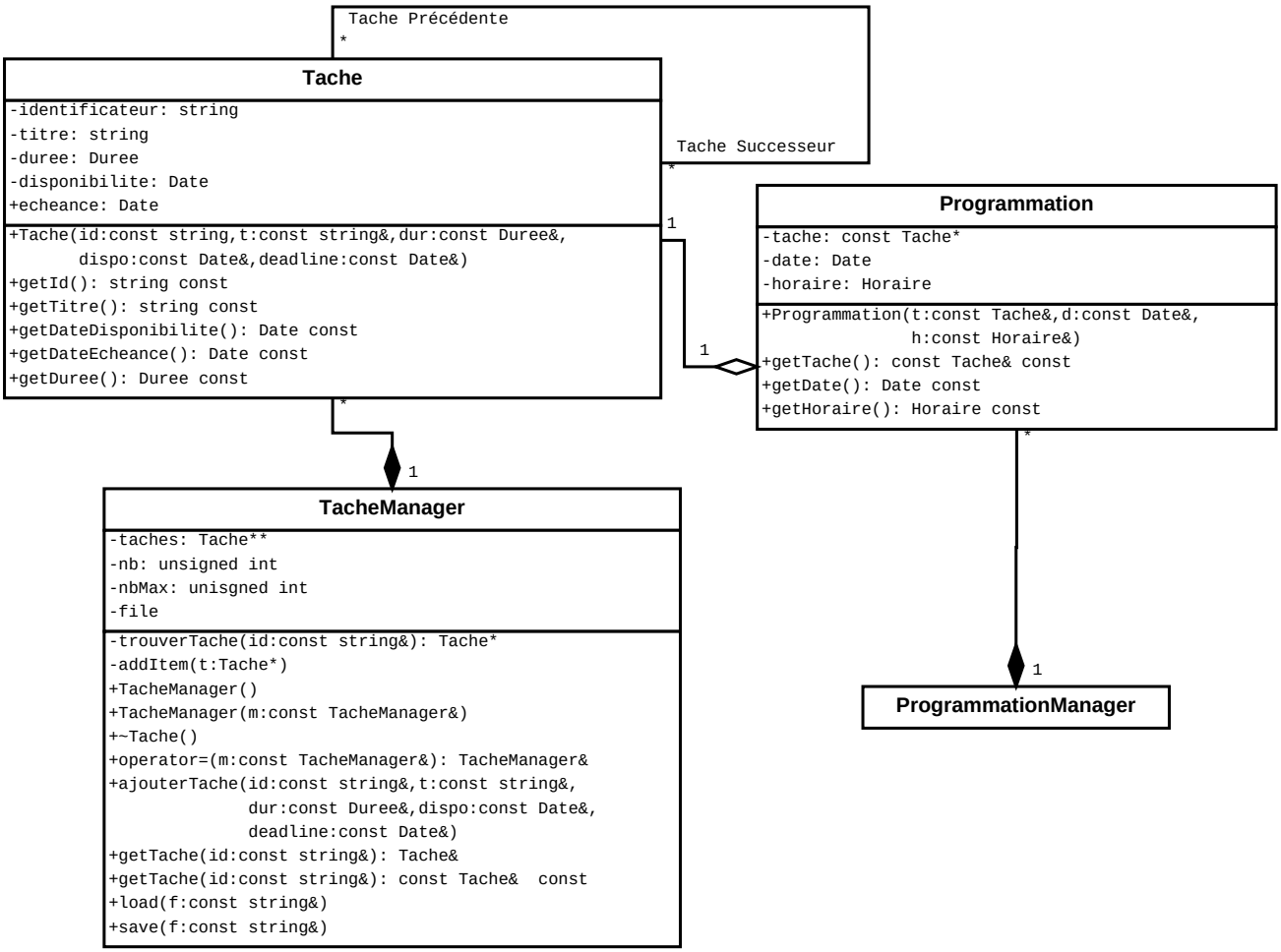
Exercice 24 - Design patterns

On a commencé à développer des classes pour l'application ProjectCalendar destinée à mixer des fonctionnalités d'un agenda électronique traditionnel et d'un outil de gestion de projet.

L'ensemble des classes déjà développées se trouvent dans une archive à votre disposition. Cette archive contient 3 fichiers. Les fichiers `Calendar.h` et `Calendar.cpp` contiennent l'ensemble des classes déjà existantes. Cet ensemble est résumé dans le diagramme de classe ci-dessous :



Remarque : Les classes fournies dans l'archive correspondent à celles développées dans le cadre des Exercices 22 et 23. Cependant, les exercices sont indépendants et il suffit de lire la description ci-dessous pour traiter cet exercice.

Une tâche est représentée par un objet de la classe `Tache` qui comporte un attribut `identificateur` de type `string`, un attribut `titre` de type `string`, un attribut `duree` de type `Duree`, un attribut `disponibilite` de type `Date` et un attribut `echeance` de type `Date`. **L'unique constructeur** de cette classe a 5 paramètres qui permettent d'initialiser les attributs d'un objet. Les accesseurs fournis permettent de connaitre les valeurs de ces attributs.

Les objets `Taches` utilisés pour l'application sont gérés par un module appelé `TacheManager` qui est responsable de leur création (et destruction). Un objet `TacheManager` peut créer un ensemble d'objets `Tache` déjà existant à partir d'un nom de fichier transmis en argument avec la méthode `load`. La classe possède aussi une méthode `ajouterTache` qui permet de créer une nouvelle tâche en transmettant les caractéristiques de cette nouvelle tâche à la méthode. La classe possède une méthode `getTache` qui permet d'obtenir une référence sur l'objet `Tache` dont le code est transmis en argument. Pour assurer la persistance de nouvelles informations, lorsqu'un objet `TacheManager` est détruit, la méthode `save` est utilisée pour mettre à jour le fichier de taches en cas d'éventuels ajouts de taches ou de mise à jour des taches déjà existantes.

Dans l'application, les tâches doivent être ordonnancées, *i.e.* une date et un horaire doivent leur être attribués. Pour cela on utilise des objets de la classe `Programmation`. Cette classe comporte un attribut `tache` représentant un objet de la classe `tache`, un attribut `date` de type `Date` et un attribut `horaire` de type `Horaire` renseignants sur l'ordonnancement de la tâche. Elle comporte aussi les accesseurs standards pour communiquer

avec les objets de la classe. Les objets `Programamtion` utilisés pour l'application sont gérés par un module appelé `ProgrammationManager` qui est responsable de leur création (et destruction).

Question 1

Expliciter des intérêts de mettre en place le Design Pattern *Singleton* pour la classe `TacheManager`. Etudier les différentes possibilités d'implémentation. Implémenter ce design pattern. Modifier votre code en conséquence. Mettre à jour le diagramme de classe.

Question 2

On remarque que la duplication malencontreuse d'un objet `Tache` pourrait poser des problèmes. Mettre en place les instructions qui permettent d'empêcher la duplication d'un objet `Tache`. De plus, faire en sorte que seule l'unique instance de la classe `TacheManager` puisse créer des objets `Tache`.

Question 3

Afin de pouvoir parcourir séquentiellement les taches stockées dans un objet `TacheManager`, appliquer le design pattern Iterator à cette classe en déduisant son implémentation du code suivant :

```
TacheManager& m=TacheManager::getInstance();
m.load("taches.dat");
for(TacheManager::Iterator it= m.getIterator();!it.isDone();it.next()){
  std::cout<<it.current()<<"\n";
}
```

Question 4

Refaire la question précédente en proposant une interface d'itérateur similaire à celle utilisée par les conteneurs standards du C++ (STL), *i.e.* qui permet de parcourir séquentiellement les différentes taches d'un objet `TacheManager` avec le code suivant :

```
for(TacheManager::iterator it=m.begin();it!=m.end();++it) std::cout<<*it<<"\n";
```

Question 5

Implémenter une classe d'iterateur qui permet de parcourir l'ensemble des objets `Tache` dont la date de disponibilité est avant une date donnée :

```
TacheManager& m=TacheManager::getInstance();
for(TacheManager::DisponibiliteFilterIterator
    it= m.getDisponibiliteFilterIterator(Date(3,2,2015))
    ;!it.isDone();it.next())
   std::cout<<it.current()<<"\n";
```