

# LO21 P15 - ProjectCalendar

## Rapport

Méwen MICHEL & Loïc DERYCKERE

### Table des matières

- Introduction
- Description de l'architecture
  - Gestion de projets
    - ProjetManager
    - Projet
    - Tâche
    - Composite
    - Unitaire
  - Programmation d'événements
    - ProgrammationManager
    - Programmation
    - Evénement
    - Activité
- Argumentation
  - Protection de l'architecture
  - Evolution de l'architecture
- Annexes
  - Utilisation de l'application
    - Edition
    - Programmation
    - Export

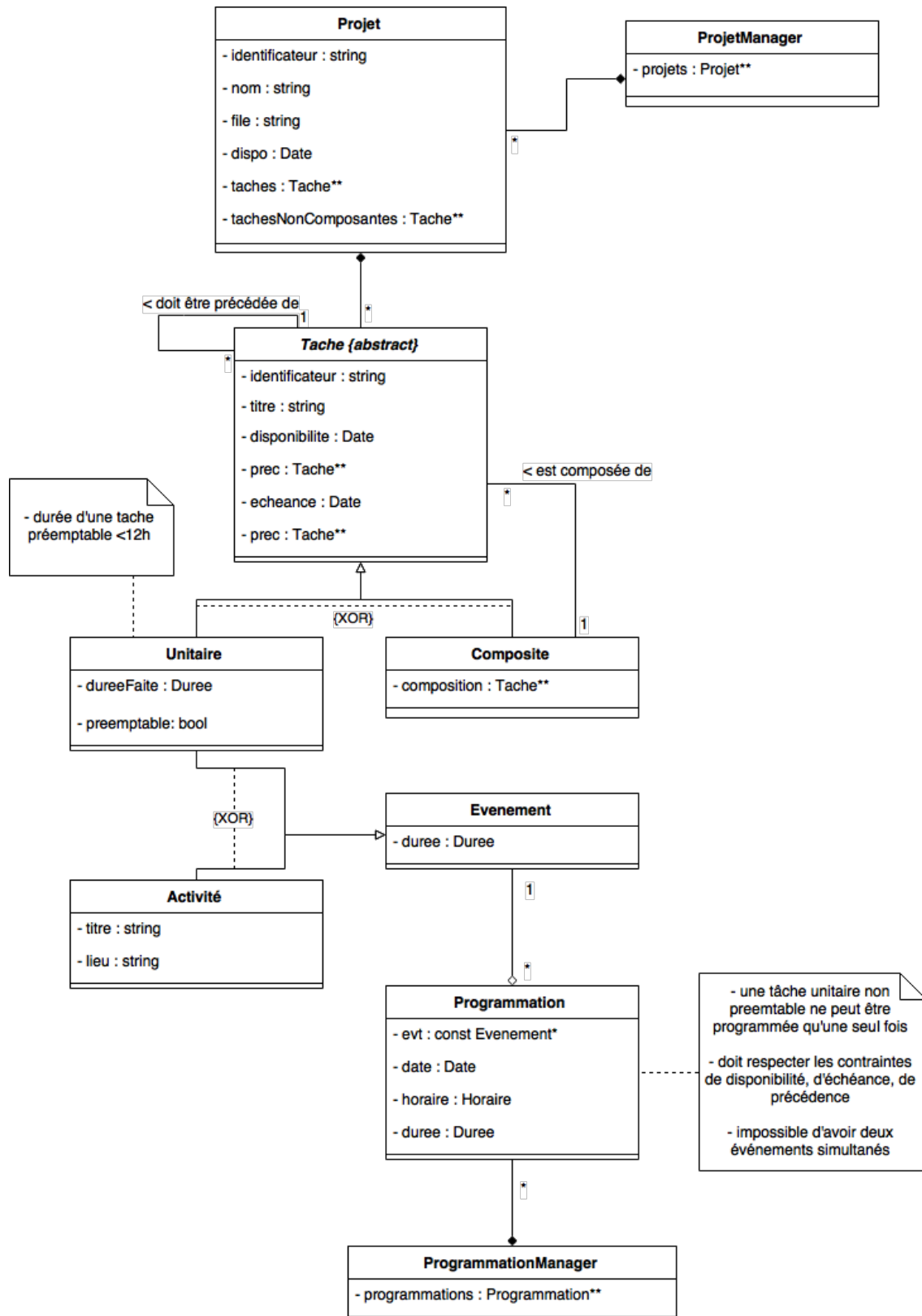
### Introduction

Ce projet de LO21 consiste à créer à l'aide de la Programmation Orientée Objet en C++, une application destinée à mixer des fonctionnalités d'un agenda électronique traditionnel et d'un outil de gestion de projet.

L'agenda en question doit afficher des vues des éléments programmés par semaines, et l'outil de gestion de projet doit pouvoir séparer les tâches des projets en deux catégories : unitaire et composite, ayant toutes deux des contraintes de précédence sur d'autres tâches.

Nous avons utilisé les pages "timing.h" et "timing.cpp" fournies en TD pour manipuler plus facilement les horaires, dates et durées des différents éléments de notre architecture.

## Description de l'architecture



Notre architecture s'oriente en deux parties :

- la gestion de projets ;
- la programmation d'événements.

## **Gestion de projets**

Les Tâches composent des projets et les Projets composent le ProjetManager. Ainsi une tâche ne peut pas se trouver dans deux projets à la fois.

### **ProjetManager**

ProjetManager est un singleton, géré par un handler qui est l'unique élément capable d'instancier et de détruire le ProjetManager. La classe possède un tableau de pointeurs de Projets qui crée les instances de Projet, et un itérateur qui peut parcourir ces projets.

### **Projet**

Projet est une classe comportant un id généré automatiquement par l'application, un nom et une date de disponibilité donnés par l'utilisateur et un tableau de pointeurs de Tâches qui crée les instances de Tâche, et un itérateur qui peut parcourir ces tâches. Il peut aussi renvoyer une date d'échéance qui dépendra uniquement des dates d'échéance des Tâches qui composent le Projet.

Un Projet pourra être exporté en XML, il représentera alors tous ces attributs et les attributs des Tâches qui le composent.

### **Tâche**

Tâche est une classe abstraite, Unitaire et Composite héritent d'elle. Elle a un tableau de pointeurs de Tâches qui représentent les contraintes de précédences et un itérateur qui parcourt ce tableau. Ce tableau ne créera pas de nouvelle Tâche, il faudra ajouter des pointeurs de Tâches déjà existantes.

Une Tâche possède aussi un id généré automatiquement, un nom, une date de disponibilité et une date d'échéance qui sont donnés par l'utilisateur, et un pointeur vers le projet parent. La date de disponibilité ne doit pas être inférieure à celle du projet parent.

### **Composite**

Composite hérite de Tâche, elle a un second tableau de pointeurs de Tâches qui montre des contraintes de composition. Ce tableau ne créera pas de nouvelles Tâches, il faudra ajouter des pointeurs de Tâches déjà existantes.

### **Unitaire**

Unitaire hérite de Tâche. Elle possède une durée (durée totale) donnée par l'utilisateur, un booléen préemptif qui permettra de savoir si la Tâche peut être programmée en plusieurs fois

(et qui sera forcément égal à TRUE si la durée de la Tâche dépasse 12h), mais aussi une durée faite (initialisée à 0) et une durée restante (qui sera la soustraction de la durée totale avec la durée faite) qui permettront de savoir le temps restant à programmer pour la Tâche.

Unitaire hérite aussi d'Événement décrit plus bas.

## **Programmation d'événements**

### **ProgrammationManager**

ProgrammationManager est un singleton dont l'instance est retournée à l'aide d'une structure handler.

Elle possède un tableau de Programmmations qui la composent, elle est donc responsable de leur création et destruction par les fonctions d'ajoutProgrammation. Dans le cas d'une programmation d'un événement Activité, c'est cette même fonction qui créera l'activité.

### **Programmation**

Programmation compose ProgrammationManager. Une Programmation possède une date, un horaire, une durée et un pointeur vers un Événement. Le principe de la programmation est d'associer date, horaire et durée à un événement.

### **Événement**

Événement est une classe abstraite dont hérite Activité et Unitaire. Elle est composée uniquement d'une durée. Elle sert à pouvoir faire des Programmmations à la fois d'Activités et de Tâches Unitaires.

### **Activité**

Activité hérite d'Événement. Elle possède un nom et un lieu.

## Argumentation

### Protection de l'architecture

On observera d'abord que notre architecture est protégée.

En effet, ProjetManager et ProgrammationManager sont des singletons, ce qui empêche d'avoir plusieurs entités de ces classes, ce qui serait absurde.

Ensuite, la plupart des attributs des classes sont privés et ne peuvent être modifiés que par des méthodes de la classe.

Les constructeurs par copie ont souvent été interdits afin d'empêcher des situations où deux éléments ont le même identificateur, ou deux programmations sont identiques par exemple.

Les classes Tâche et Événement sont abstraites, empêchant ainsi de les instancier, ce qui conduirait à la programmation d'un élément n'ayant qu'une durée par exemple.

### Evolution de l'architecture

Notre architecture peut facilement évoluer.

En effet, les classes Tâche et Événement étant abstraites, on peut aisément créer de nouvelles classes héritant de l'une, de l'autre ou des deux afin de complexifier les structures de projets avec un nouveau type de Tâche ou afin d'introduire un nouveau type d'Événement ayant une liste de personnes présentes par exemple.

## Annexes

### Utilisation de l'application

#### Edition

Sur l'onglet d'édition on peut observer une TreeView des projets de l'application. Pour éditer un projet ou une tâche il suffit de cliquer sur son nom, de modifier les informations sur la fenêtre à droite de l'arborescence et de sauvegarder les changements avec le bouton prévu à cet effet.

Un bouton d'annulation permet de remettre les paramètres aux dernières valeurs sauvegardées.

Pour ajouter un projet un bouton Projet + est situé en dessous de l'arborescence.

Pour ajouter une tâche unitaire ou composite il faut cliquer sur l'élément de l'arborescence dans laquelle on souhaite la placer (i.e. un projet ou une tâche composite). Si une tâche unitaire est sélectionnée au moment d'un ajout, le nouvel élément est placé en parallèle de celle-ci.




Pour ajouter une précédence à une tâche il faut cliquer sur le bouton d'ajout dans la fenêtre d'édition de la tâche, sélectionner une tâche et valider l'ajout.

#### Programmation

Pour programmer une activité, il faut aller dans l'onglet d'édition et cliquer sur le bouton sous l'arborescence, il faudra alors préciser les détails de l'activité et les détails de sa programmation.

Pour programmer une tâche unitaire, il faut la sélectionner dans l'arborescence et cliquer sur le bouton sous l'arborescence. Il faut alors préciser les détails de la programmation.

Pour une meilleure lisibilité l'arborescence affiche l'état des éléments de projets :

-  Un rond vide signifie que l'élément n'a pas été commencé ;
-  A moitié rempli signifie qu'une partie des éléments contenus sont programmés ;
-  Un rond plein signifie que la totalité de l'élément est programmée ou que sa date d'échéance a été dépassée.

Pour voir le détail des programmations il faut aller dans l'onglet agenda et sélectionner la semaine à voir depuis le calendrier.

#### Export

Pour exporter un projet il faut aller dans l'onglet d'édition, le sélectionner et cliquer sur le bouton d'export dans la fenêtre à droite de l'arborescence.

Pour exporter une semaine de l'agenda il faut aller à une date correspondante dans le calendrier de l'onglet agenda et cliquer sur le bouton d'export en haut à gauche.