ODL - Exercice intégré

Organisation d'événements

François Roland,
Pierre Manneback

06-12-2018

L'exercice suivant sera réalisé au cours de trois séances de travaux pratiques. L'objectif est de réaliser une application complète, reprenant différentes facettes de la conception orientée objet, de la programmation Java et des Designs Patterns.

La première séance sera consacrée à la définition des classes métiers et de leurs méthodes (constructeurs, accesseurs, toString). Ces classes métiers seront choisies par rapport aux concepts clés de l'application tels que les ressources, les tâches, les utilisateurs, etc.

La deuxième séance permettra de se concentrer sur les interfaces graphiques de l'application.

La troisième séance sera consacrée à l'algorithme d'affectation des tâches aux ressources et aux travailleurs. Aucun algorithme d'affectation « évolué » ne doit être implémenté dans le logiciel ; cependant le logiciel doit être conçu pour pouvoir implanter d'autres algorithmes (cfr design pattern Strategy).

Une séparation claire entre ces trois aspects de l'application est demandée. L'utilisation de packages Java peut vous y aider.

Remise des résultats

Le projet réalisé sera sauvegardé régulièrement sur votre dépôt GitHub. La documentation produite sera enregistrée au format PDF à la racine de votre dépôt GitHub.

Seul le dernier commit effectué avant le vendredi 21/12/2018 sera pris en compte pour l'évaluation.

Cahier des charges

- Nous souhaitons un logiciel pour nous aider à organiser des événements de type " concert, spectacle ".
- La difficulté que nous rencontrons est de gérer l'ensemble des tâches et l'utilisation des ressources matérielles et humaines.
- Les personnes sont amenées à réaliser des tâches à différents endroits sur le terrain. Il peut s'agir d'installer une scène, de transporter du matériel d'un point à un autre, etc.
- Nous avons besoin de savoir à tout moment où se trouve le matériel et les personnes qui l'utilisent.
- L'idéal est que le logiciel nous planifie des tâches et des ressources. Chaque chef d'équipe peut avoir une manière différente de réaliser cette planification.
- Chacun doit à tout moment savoir ce qu'il a à faire et les chefs d'équipe doivent pouvoir suivre l'évolution du travail.
- Le manager souhaite avoir une vue d'ensemble de l'évolution des tâches.

Cet exercice intégré se concentrera sur les quatre aspects suivants du logiciel d'organisation d'événements :

1. la gestion des utilisateurs (Manager, Team, Worker) ;
2. la gestion des ressources (Resource) ;
3. la gestion des tâches (Task, Skill) ;
4. l'affectation des tâches aux ressources (Planning).

Pour les points 2 et 3, un écran de gestion doit être proposé afin de permettre la création, la modification et la suppression d'éléments. L'ensemble des employés (Manager, Team et Worker) est supposé connu avant le lancement du programme et donc instancié directement dans le code au début de la méthode main.

En plus de ces écrans de gestion, une vue de type «dashboard» (tableau de bord) doit être proposée afin d'indiquer à chacun ses informations d'intérêt :

- pour le Worker : son planning de travail pour les prochaines heures ;
- pour le Manager : un aperçu des tâches assignées aux équipes.

Lorsque le fonctionnement de base du programme est en place, les fonctionnalités suivantes peuvent être apportées :

1. permettre l'enregistrement et le chargement de l'état de l'application dans un fichier par la sérialisation d'objets ;

2. introduire un système de notifications pour permettre à un utilisateur d'être averti en cas de changement de certains objets (Resource, Task et Worker) (pensez au DP Observer);
3. offrir un écran de connexion avec login/motdepasse au démarrage de l'application. Avant cette étape, un simple menu déroulant du choix de l'utilisateur doit être proposé afin de prendre le rôle de l'un ou l'autre employé.

Le diagramme de classes ci-dessous peut être utilisé comme base de travail. Il doit être considéré à titre indicatif plutôt que comme une consigne à respecter. Certains changements et compléments devront être apportés au fil de l'implémentation du logiciel.

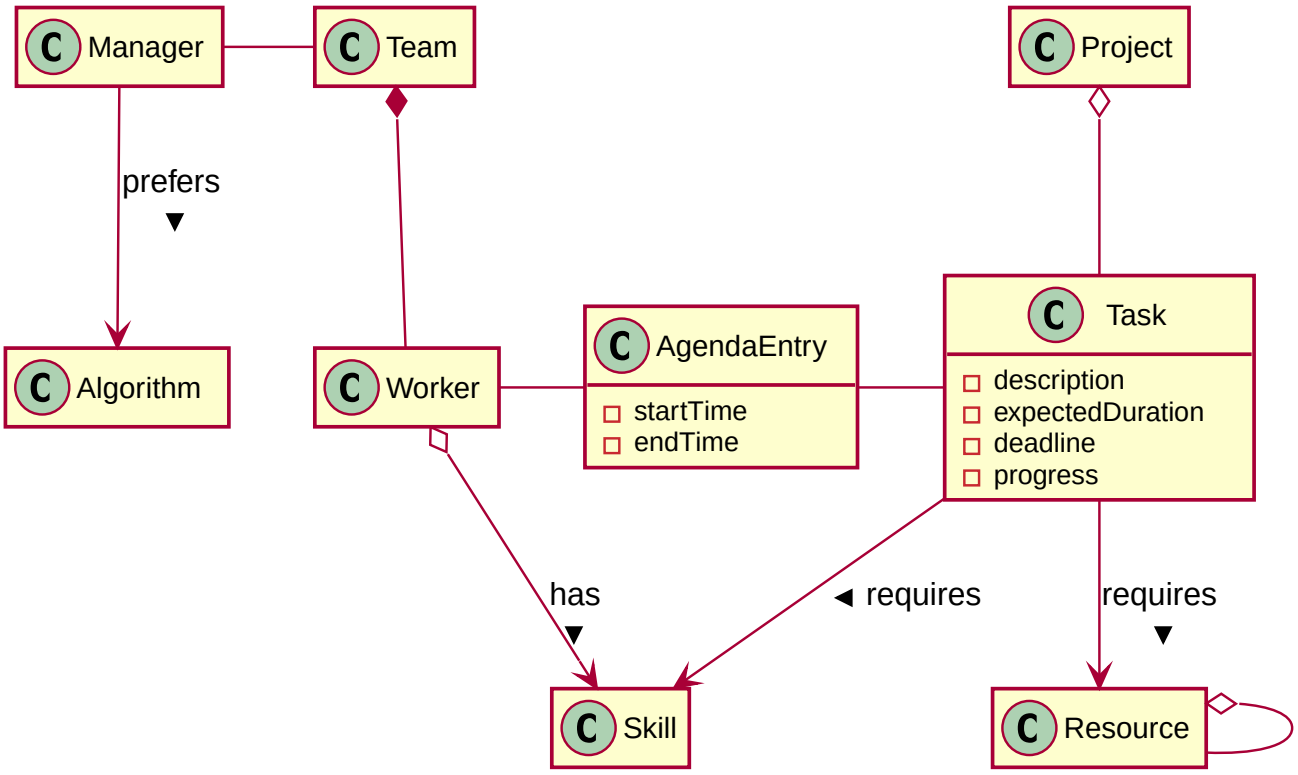

Figure 1. Ébauche du domain model

Comme cela a été exposé au cours, l'application peut tirer profit de certains Design Patterns. Nous vous demandons donc d'utiliser dans votre implémentation :

- le design pattern Strategy sera utilisé afin de gérer les différents algorithmes d'affectation des tâches aux ressources et aux travailleurs ;
- le design pattern Composite assurera la gestion des ressources pouvant contenir elles-mêmes des sous-ressources et ce, sans limite de « profondeur » ;
- le design pattern Observer pour permettre les notifications de changement dans les classes Resource, Task et Worker.

En plus du code créé, vous devrez également fournir une documentation. Cette documentation décrira le scope de l'application et mettra en évidence certaines parties de son design. Elle doit permettre à l'évaluateur de comprendre ce que vous avez fait et pourquoi vous l'avez fait de cette manière.

Références

- Deitel, H. M., & Deitel, P. J. (2002). *Java : comment programmer (4th ed.)*. Les éditions Reynald Goulet INC.
- Deitel, P. J., & Deitel, H. M. (2007). *Java: how to program (7th ed.)*. Les éditions Reynald Goulet INC.
- Evans, E. (2003). *Domain-driven design: tackling complexity in the heart of software (1st ed.)*. Addison-Wesley Professional.
- Manneback, P., & Frémal, S. (2016). *Travaux pratiques de Méthodologie et Langage de Programmation*. UMons.
- Manneback, P. (2005). *Méthodologie et Langages de Programmation*. UMons.
- *Java Platform Standard Edition 8 Documentation*. Récupéré de https://docs.oracle.com/javase/8/docs/