ODL – Exercice Intégré : Organisation d'événements

Documentation

Introduction

Ce document détaille toutes les classes que nous avons utilisées pour la réalisation d'un logiciel de gestion d'événements. Pour chaque classe, on peut trouver les attributs , les constructeurs et les méthodes qui la définissent.

Nous avons utilisé deux design patterns : Strategy et Composite. Strategy est mis en place pour le choix de l'algorithme d'affectation des tâches aux ressources et aux workers. Il est en effet très intéressant car nous pouvons définir plusieurs algorithmes d'affectation concrets sans difficulté. Composite est quant à lui utilisé pour la modélisation des ressources. Pour faire l'analogie avec la représentation théorique de ce design pattern, notre interface Ressource correspond au component et nos classes BasicRessource et ComposedRessource correspondent respectivement à Leaf et au Composite.

Package Classes Métiers

Class Manager

Classe définissant un manager.

Attributs	Description
String id	Id du manager
String name	Nom du manager

Constructeur(s)	Description
Public Manager(String nom)	Constructeur définissant l'id (auto incrémentation) et le nom d manager

Méthodes	Description
Public String getId()	Retourner l'id du manager
Public Sting getName()	Retourner le nom du manager
Public String toString()	Redéfinition de la méthode toString() pour afficher l'id et le nom du manager

Class Worker

Classe définissant un travailleur.

Attributs	Description

String id	Id du worker (auto incrémentation)
String name	Nom du worker
Skill skill	Skill du worker (voir classe Skill)

Constructeur(s)	Description
Public Worker(String name)	Constructeur définissant le nom et l'id du worker
Public Worker(String name, Skill skill)	Constructeur définissant le nom, l'id et le Skill du worker

Méthodes	Description
Public String getId()	Retourner l'id du worker
Public String getName()	Retourner le nom du worker
Public Skill getSkill()	Retourner le Skill du worker
Public void setSkill(Skill skill)	Définir le skill du worker
Public String toString()	Redéfinition de la méthode toString() pour afficher l'id, le nom et le skill du worker

Class Team

Classe définissant une équipe. Principalement caractérisée par une liste de Worker (voir Worker)

Attributs	Description
String teamName	Nom de l'équipe
Static int teamNumber	Numéro de l'équipe (auto incrémentation)
ArrayList<Worker> teamCompositon	Liste des workers constituant l'équipe

Constructeur(s)	Description
Public Team()	Constructeur par défaut, ne définit que le numéro de l'équipe
Public Team(ArrayList<Worker> teamComposition)	Constructeur définissant le numéro et la liste des Workers constituant l'équipe

Méthodes	Description
Public String getTeamName()	Retourner le nom de l'équipe
Public int getTeamNumber()	Retourner le numéro de l'équipe
Public ArrayList<Worker> getTeamComposition()	Retourner la liste des Workers constituant l'équipe
Public void setTeamName()	Définir le nom de l'équipe avec le numéro de l'équipe
Public void setTeamComposition(ArrayList<Worker> teamComposition	Définir la liste des Workers Constituant l'équipe
Public void addWorker(Worker w)	Ajouter un Worker à la liste des Workers constituant l'équipe

Public String toString()	Redéfinition de la méthode toString() pour afficher le nom de l'équipe et la liste des Workers constituant l'équipe

Class Skill

Classe définissant une capacité. Une capacité est attribuée à un worker (voirWorker).

Attributs	Description
String name	Nom de la capacité

Constructeur(s)	Description
Public Skill(String name)	Constructeur définissant le nom de la capacité

Méthodes	Description
Public String getName()	Retourner le nom de la capacité
Public String toString()	Redéfinition de la méthode toString() pour afficher le nom de la capacité

Class Task

Classe définissant une tâche. Dans le cas de la gestion d'événements, les tâches seront les éléments qui constitueront un projet (voir Project)

Attributs	Description
String name	Nom de la tâche
Int progress	Progrès, état d'avancement de la tâche
Worker worker	Worker assigné à la tâche
AgendaEntry agendaEntry	« plage horaire » de la tâche dans l'agenda

Constructeur(s)	Description
Public Task()	Constructeur par défaut initialisant le progrès à 0
Public Task(String name)	Constructeur définissant le nom de la tâche

Méthodes	Description
Public String getName()	Retourner le nom de la tâche
Public int getProgress()	Retourner le progrès de la tâche
Public Worker getWorker()	Retourner le Worker assigné à la tâche
Public AgendaEntry getAgendaEntry()	Retourner la « plage horaire » de la tâche
Public void setName(String name)	Définir le nom de la tâche
Public void setProgress(int progress)	Définir le progrès de la tâche
Public void setWorker(Worker worker)	Définir le worker devant réaliser la tâche
Public void setAgendaEntry(AgendaEntry agendaEntry)	Définir la « plage horaire » de la tâche

Public String toString()	Redéfinition de la méthode toString() pour afficher le nom, me progrès, le worker et la « plage horaire » de la tâche

Class Project

Classe définissant un projet

Attributs	Description
String projectName	Nom du projet
ArrayList<Task> tasks	Tâches à réaliser pour le projet

Constructeur(s)	Description
Public Project(String projectName)	Définition du nom du projet

Méthodes	Description
Public String getProjectName()	Retourner le nom du projet
Public ArrayList<Task> getTasks()	Retourner la liste des tâches du projet
Public void setProjectName(String projectName)	Définir le nom du projet
Public void setTasks(ArrayList<Tasks> tasks)	Définir la liste des tâches du projet
Public String toString()	Redéfinition de la méthode toString() pour afficher le nom et les tâches du projet

Class AgendaEntry

Classe définissant une plage horaire dans l'agenda ; une tâche (voir task) pourra avoir une une AgendaEntry

Attributs	Description
Int startTime	Début de la plage horaire
Int endTime	Fin de la plage horaire

Constructeur(s)	Description
Public AgendaEntry()	Constructeur par défaut ; définit les attributs à 0
Public AgendaEntry(int startTime, int endTime)	Constructeur définissant le début et la fin de la plage horaire

Méthodes	Description
Public in getStartTime()	Retourner le début de la plage horaire
Public int getEndTime()	Retourner la fin de la plage horaire
Public void setStartTime(int s)	Définir le début de la plage horaire
Public void setEndTime(in e)	Définir la fin de la plage horaire
Public String toString()	Redéfinition de la méthode toString() pour afficher le début et la fin de la plage horaire

Interface Ressource

Interface servant à la mise en œuvre du design pattern Composite sur les ressources matérielles. Elle comprend six méthodes suivantes qui devront être redéfinies dans les classes (BasicRessource et ComposedRessource) qui l'implémentent.

Méthodes	Description
Public String getName()	Retourner le nom de la ressource
Public float getState()	Retourner l'usure de la ressource
Public boolean getStatus()	Retourner le statut de la ressource (utilisé ou non)
Public void setName(String name)	Définir le nom de la ressource
Public void setState(float state)	Définir l'usure de la ressource
Public void setStatus(boolean status)	Définir de statut de la ressource

Class BasicRessource

Classe implémentant l'interface Ressource, les méthodes de celle-ci y étant définies de manière concrète.

Attributs	Description
String name	Nom de la ressource basique
Float state	Usure de la ressource basique ; 0 représente neuf et 1 très usé
Boolean status	Statut de la ressource ; true si utilisé et false si disponible

Constructeur(s)	Description
Public BasicRessource(String name)	Définition du nom de la ressource basique ; state et status par défaut 0 et false

Méthodes	Description
Public String getName()	Retourner le nom de la ressource basique
Public float getState()	Retourner l'usure de la ressource basique
Public boolean getStatus	Retourner le statut de la ressource basique
Public void setName(String name)	Définir le nom de la ressource
Public void setState(float state)	Définir l'usure de la ressource basique
Public void setStatus(boolean status)	Définir de statut de la ressource basique
Public String toString()	Redéfinition de toString() pour afficher name, state et status

Class ComposedRessource

Classe implémentant l'interface Ressource, on y redéfinit donc les méthodes de cette dernière. De plus, il y a une méthode qui permet d'ajouter des Ressources. Le design pattern Composite est mis en place via l'interface Ressource et les classes BasicRessource et ComposedRessource.

Attributs	Description
String name	Nom de la ressource basique
Float state	Usure de la ressource basique ; 0 représente neuf et 1 très usé
Boolean status	Statut de la ressource ; true si utilisé et false si disponible
ArrayList<Ressource> basicRessources	Liste des éléments de niveau inférieur qui constituent la ressource composée (Éventuellement une ressource composée peut être constituée de ressources composées, en accord avec le design pattern Composite)

Constructeur(s)	Description
Public ComposedRessource(String name)	Définition du nom de la ressource composée
Public ComposedRessource(String name, ArrayList<Ressource> basicRessource)	Définition du nom et des sous-ressources constituantes de la ressource composée

Méthodes	Description
Public String getName()	Retourner le nom de la ressource composée
Public float getState()	Retourner l'usure de la ressource composée
Public boolean getSatus()	Retourner le statut de la ressource composée
Public ArrayList<Ressource> getBasicRessource	Retourner la liste des éléments constituant la ressource composée
Public void setName(String name)	Définir le nom de la ressource composée
Public void setState(float state)	Définir l'usure de la ressource composée
Public void setStatus(boolean status)	Définir le statut de la ressource composée
Public void addBasicRessource(Ressource addedRessource)	Ajouter une sous-ressource à la liste des éléments constituant la ressource composée
Public void dropBasicRessource(Ressource droppedRessource)	Supprimer une sous-ressource de la liste des éléments constituant la ressource composée

Interface Algorithm

Interface servant à la mise en œuvre du design pattern Strategy pour le choix de l'algorithme d'affectation des tâches. Il suffit de créer une nouvelle classe implémentant cette interface et évidemment redéfinir les méthodes pour proposer un nouvel algorithme d'affection.

Méthodes	Description
void affectTaskToRessource()	Affecter les tâches aux ressources matérielles
void affectTaskToWorkers()	Affecter les tâches aux ressources humaines

Class Algo1

Classe implémentant l'interface Algorithm, on y définit les méthodes void affectTaskToRessource() et void affectTaskToWorkers() de manière concrète. Nous n'avons pas défini d'algorithme d'affection mais simplement mis en forme une classe pour illustrer le design pattern Strategy.

GUI :

Notre application dispose évidemment d'une interface graphique qui permettra aux utilisateurs d'effectuer les actions qu'on les a autorisés à faire. Ces actions sont énoncées dans le diagramme use case.

Diagramme de use case :

Il permet d'identifier les possibilités d'interaction entre le système et les acteurs (intervenants extérieurs au système), c'est-à-dire toutes les fonctionnalités que doit fournir le système. Il permet aussi de délimiter le système.

Ici, nos acteurs sont les Workers et le manager.

Le Worker n'a qu'une seule possibilité d'interaction, voir son propre planning.

Le Manager par contre, en a beaucoup plus. Il a accès à la gestion des utilisateurs où il va pouvoir voir tous les Workers, créer des équipes, voir des équipes et ajouter des Skills à ses Workers. La gestion des ressources lui permettra d'ajouter, consulter et modifier des Ressources. La gestion des tâches lui permettra d'ajouter, consulter et modifier des Tasks. Il pourra également affecter des tâches à des Workers et à des Ressources (cf. Algo1 et (I) Algorithm).

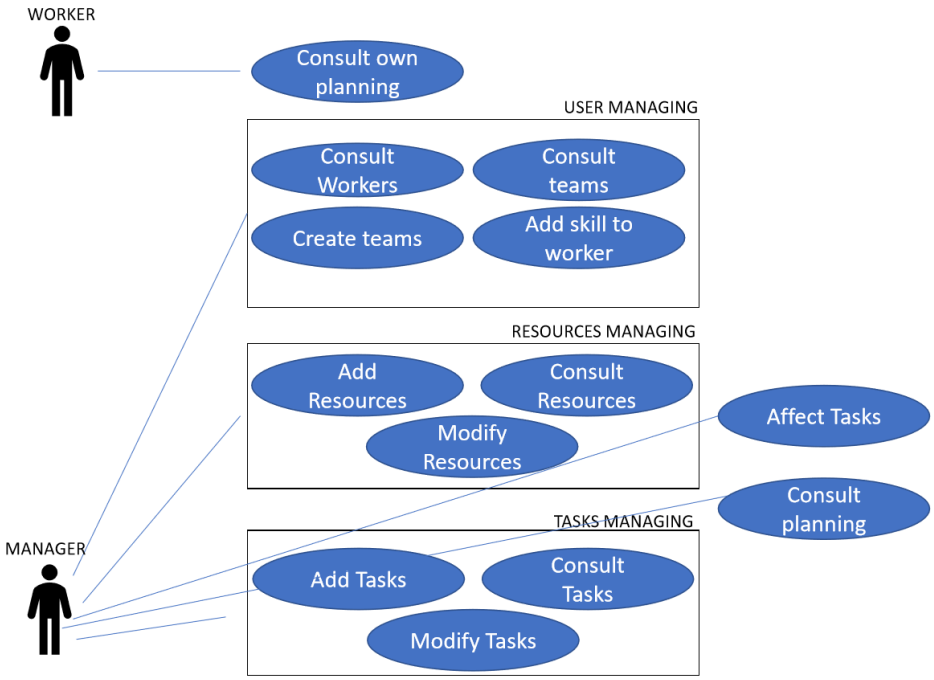
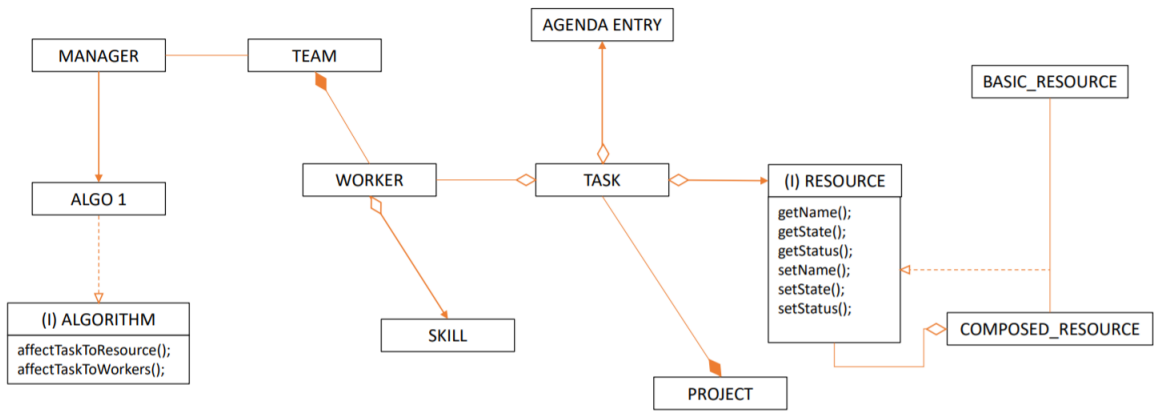


Diagramme de classe :

Il représente les classes intervenant dans le système. Le diagramme de classe est une représentation statique des éléments qui composent un système et de leurs relations.



Commentaire

(I) ALGORITHM est une interface que la classe ALGO1 implémente

MANAGER associe ALGO1

TEAM est composé de WORKER (-> composition)

WORKER contient SKILL (-> agrégation)

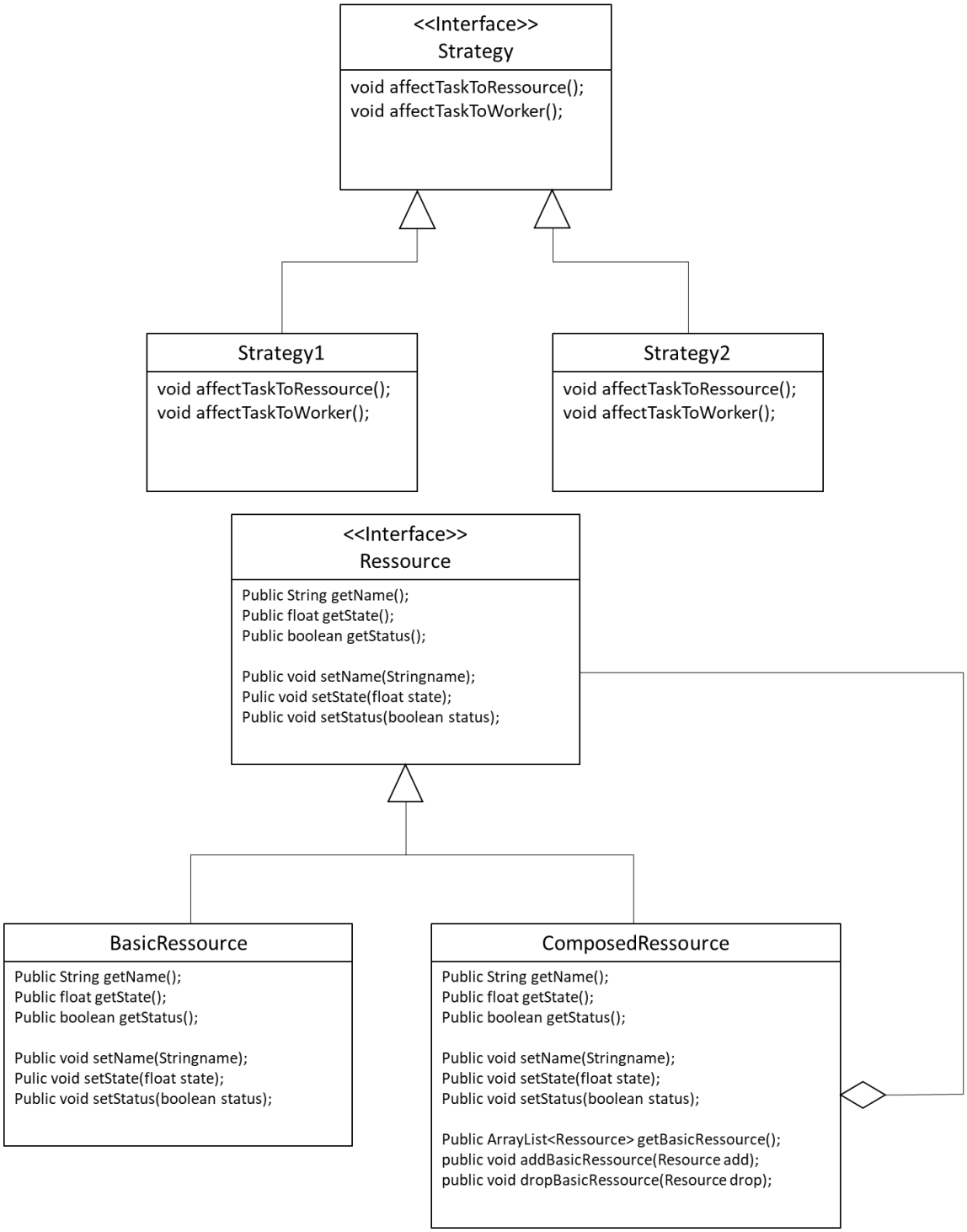
TASK contient WORKER, AGENDAENTRY et RESOURCE (-> agrégation)

(I) RESOURCE est une interface graphique implémenté par la classe BASICRESOURCE et COMPOSEDRESOURCE

COMPOSEDRESOURCE est composé de BASICRESOURCE (-> composition)

PROJECT est composé de TASK (-> composition)

Design patterns



Conclusion

En conclusion, nous avons réussi à implémenter un programme fonctionnel avec une interface graphique simple, permettant aux utilisateurs d'effectuer des actions. Il nous était demandé de nous concentrer sur la gestion des utilisateurs, des ressources, des tâches et l'affectation des tâches aux ressources ainsi que d'offrir pour ces quatre parties, un écran de gestion ce qui a bien été fait. Une représentation de l'agenda a également été implémentée bien que rudimentaire, cela fait partie des points à améliorer dans notre logiciel. Nous avons également eu l'occasion d'utiliser deux design pattern : un composite pour les ressources et un strategy pour gérer les algorithmes d'affectation des tâches.