

*Avenue Paul Langevin*

*59655 Villeneuve d’Ascq cedex*

PROJET DE PPO

AGENDAS DE TACHES

*Projet réalisé par:**Tuteur:*

Yizhou LIN Walter RUDAMETKIN

Aimée UMUHOZA

Département Génie informatique et statistique 4A Année : 2015-2016

Sommaire

INTRODUCTION

1. Schéma UML
2. Analyse et Conception
3. Méthodes essentielles
4. Traitement d’une demande
5. Annulation d’une demande

Introduction

Au cours de ce projet, nous travaillons à la conception de logiciels d’agendas destinés à des services devant répondre à des demandes de travaux. L'objectif de ce projet est de mettre en place un logiciel résolvant les problèmes de gestion d'agendas de tâches. Pour résoudre ce projet, nous avons conçu une bibliothèque de classes générale ainsi que ses différentes applications dans un schéma UML. Dans ce rapport nous allons donc décrire les principales fonctionnalités de nos méthodes utilisées par rapport au classées concernées.

1. Schéma de conception UML
2. Analyse et conception
3. Section 1.1

Dans cette partie, il nous a été demandé de répartir des tâches aux employés suivant leur disponibilité sachant qu’une tâche doit être traitée pendant n créneaux consécutif.

Nous avons donc choisi de créer cinq classes « Service », « Tache », « Employé », « emploiTemps » et « Ressources » dans une bibliothèque nommé bibAgenda.

1. **Service**

Dans cette classe, nous avons huit méthodes :

* «***ajouterEmploye* »** prend en entrée le nom et le numéro de l’employé. Cette méthode nous permet d’ajouter les employés au service ensuite elle retourne “vrai“ si l’employé a été ajouté.
* «***listeEmploye*** » ne prend rien en paramètre et nous permet d’afficher tous les employés
* «***getStringEmployeTemps***» prend en entrée le nom de l’employé, ensuite elle affiche l’employé du temps de cet employé.
* «  ***getStringEmployeTemps***  » : ne prend rien en paramètre et nous permet d’afficher l’emploi du temps de tous les employés.
* « ***traiterDemande*** » cette méthode nous permet de traiter une demande de tâche quelconque car elle prend en entrée la nature de la tâche à traiter ainsi que le nombre d’heures nécessaire pour traiter cette demande et elle fait appel à la méthode ajouterTache, celle-ci permet de créer une tache ensuite l’ajouter dans un tableau de tâche. Elle retourne vrai si la demande a été traitée c'est-à-dire la tâche a été créée faux sinon.
* « ***ajouterTache*** » prend en entrée la nature de la tâche à ajouter dans un tableau de tâches à traiter et le nombre de créneaux nécessaire pour cette tâche. Et retourne la tache ajoutée.
* « ***traiterDemande***» cette méthode prend en entrée une tâche déjà créée et elle va chercher si on a un employé disponible pour faire cette tâche, si l’employé est trouvé on retourne vrai, sinon la tâche est annulée.
* « ***annulerDemande***» nous permet d’annuler une demande donnée, elle prend en paramètre la tâche a annulé ensuite elle fait appel à la fonction « annuler » définie dans la classe tache. Cette fonction retourne vrai si la demande a été annulée faux sinon.

1. **Tache**

* Pour qu’une tâche soit traitée, il faut que tous les ressources dont elle a besoin soient disponible, pour cela nous avons défini une méthode nommé « ***verifier*** », cette méthode vérifie si pour une journée et un créneau donnés tous les ressources dont la tâche a besoin sont disponible, elle retourne vrai sinon elle retourne faux.
* Pour chaque ressource, nous avons besoin de savoir celui-ci est disponible donc nous avons crée un emploi du temps, par exemple si notre ressource est une salle nous avons un emploi du temps permettant de répertorier des salles qui sont déjà prises pour tel date. Grâce à la méthode « ***addEmploiTempsRessource*** » qui prend en paramètre une journée et un créneau donnés, ensuite fait appel à « ajouterTache » qui va nous permet d’ajouter la tâche dans l’emploi du temps de chaque ressource correspondant.
* Nous avons ensuite une méthode ***« annuler »*** ne prend rien en paramètre et elle nous permet d’annuler n’importe quelle tâche soit tâche logiciel ou tâche salle. Dans la classe tâche nous avons une variable < état > spécifiant l’état d’une tâche, soit elle est en attente d’être traitée (par exemple tâche materiel), soit elle a été annulée ou soit elle est traitée.

Voici un tableau qui résume les valeurs pris par la variable état :

|  |  |
| --- | --- |
| **Valeur** | **Sens** |
| 0 | Tâche initialisée, en attente d’être traitée |
| 1 | Tâche traitée |
| -1 | Tâche annulée |

Cette méthode utilise le pseudo langage suivant :

|  |
| --- |
| Classe :Tache |
| annuler() :booléen |
| Local :resultat :booléen |
| boolean resultat = (etat==0);  //Pour vérifier si la tâche est annulée  //si le tâche est déjà annulée la méthode retourne faux  etat = -1;  return resultat; |

1. **EmploiTemps**

* Avant de traiter une tâche, il faut que cette dernière soit créée. Donc dans cette classe nous avons un constructeur «***EmploiTemps*** » permettant de créer une nouvelle tâche, cette tâche est créée par rapport à son type, soit une tâche d’entretien, soit une tâche de service informatique ou soit une tâche générique.

* Nous avons également une méthode « ***verifier***» cette méthode est paramétrée une journée, un créneau et le nombre de créneaux. Elle sera bien expliquée dans la partie de méthodes essentielles.

**d. Ressource**

La classe Ressource est une classe abstraite qui va être héritée par les classes « Employé », « Salle », « Matériel » et « Logiciel ». Cette classe n’est pas instanciable.

1. Méthodes essentielles
2. **Traitement de demande**

Nous allons d’abord créer une tâche par rapport à sont type, soit une tâche d’entretien, soit une tâche de service informatique, soit une tâche générique. Nous ajouterons dans le tableau de chaque tâche les ressources nécessaire pour traiter cette demande (exemple : Salle, matériels, logiciels …). Cette tâche sera ajoutée dans un tableau de tâche.

Ensuite, nous parcourons le tableau des employés pour chercher un employé disponible pour ce créneau. Pour ce faire, nous allons faire l’appel à la méthode « ajouterTache » dans l’emploi du temps de tous les objets de types Employé. Elle est retourne une variable booléenne pour confirmer si la tâche a été bien ajoutée avec succès.

Les tâches non traitées sont annulées sauf les tâches matérielles qui seront mise en attente en attendant qu’il y ait un créneau qui se libère.

Pseudo-code de la méthode **ajouterTache(Tache)** utilisée :

|  |
| --- |
| Classe :EmploiTemps |
| ajouterTache(Tache) :booléen |
| Donne de la classe :  nbCreneauJour :entier //nombre de créneaux possible dans une journée  nbJour :entier //nombre de jours dans une semaine  Donner : tache : Tache  Local :  nbLibre ptJour ptCre: entier initialisé à 0;  flagTrouve : booleen initialisé à false, vrai si la tâche est traitée |
| Tantque flagTrouve == false et ptJour < nbJour faire  nbLibre = 0; // nombre de creneaux libre  ptCre = 0;  Tantque flagTrouve == false et ptCre<nbCreneauJour faire  Si un créneaux est libre ou la tache a ete annulée alors  nbLibre = nbLibre+1  Si nbLibre >= nombre de créneau de tache et pour tous les  ressources dans la tableau de ressources de tache sont disponible alors  tableTache[ptJour][ptCre-tache.nbCreneaux+1] = tache;  tache.etat = 1;  flagTrouve = true;  Fin Si  ptCre = ptCre + 1  Sinon  nbLibre = 0;  ptCre = ptCre + tableTache[ptJour][ptCre].nbCreneaux;  Fin Si  Fait  ptJour = ptJour+1;  Fait  return flagTrouve; |
|  |

Nous avons également une méthode «***vérifier*** », cette méthode est appelée dans la méthode ci-dessus, elle permet de vérifier si tous les ressources dont on a besoin sont disponible, si oui la tâche sera ajoutée dans l’emploi du temps sinon elle est annulée.

Pour ce faire, nous avons deux étapes principales :

1. Chercher la tâche précédente pour savoir si le créneau donné est déjà occupé.
2. Compter le nombre de créneaux libre à partir du point donné.

Pseudo-code de la méthode **verifier(jour, creneaux, nbCreneaux)** utilisée :

|  |
| --- |
| Classe :EmploiTemps |
| verifier(int jour,int creneau,int nbCreneaux) :booléen |
| Donne de la classe :  nbCreneauJour :entier //nombre de créneaux dans un jour  nbJour :entier //nombre de jours dans une semaine  tableTache :Tache[][]  Donner : jour, creneau nbCreneaux : entier  Local :  TableTacheToday : Tache[]  ptCre : entier  lastTache : Tache  nbLibre ptJour ptCre: entier initialisé à 0;  flagTrouve : bool initialisé à false  i :entier //index de boucle |
| Tache[] tableTacheToday = tableTache[jour];  ptCre = creneau;  lastTache = null;  //chercher la tache précédente  Tq ptCre>=0&&lastTache==null faire  si tableTacheToday[ptCre]!=null  et  tableTacheToday[ptCre].etat!=-1  //donc il y a une tâche non annulé  faire  lastTache = tableTacheToday[ptCre];  fin si  ptCre--;  fait  si lastTache!=null faire  //il y a une tâche précédent ce jour la  si creneau<=ptCre+lastTache.nbCreneaux faire  //donc le créneau n'est pas disponible  return false;  fin si  fin si  //Compter nombre de créneaux libre à partie du point donné.  si creneau+nbCreneaux>nbCreneauJour faire  //il n'y a pas assez de créneaux apres  return false;  fin si  pour i de 0 à nbCreneaux-1 faire  //il existe une tache est il n'est pas annuele  Si tableTacheToday[creneau+i]!=null  et  tableTacheToday[creneau+i].etat!=-1  Faire  turn false;  fin si  fait  return true; |