Institut Universitaire De Technologie
De Blagnac-Toulouse II
Département Informatique
Conception et programmation objet avancées

CPOA
Projet refactoring

Réalisé par :

Dimitri CALMELS - dimitri.calmels@etu.univ-tlse2.fr

Xavier FERNANDEZ - xavier.fernandez@etu.univ-tlse2.fr

Table des matières

Conception

Contexte :

Une application de gestion de projets doit être améliorée en repensant son architecture fragile pour la rendre solide et optimisée et y ajouter de nouvelles fonctionnalités. Tout cela devra être mis en place mettant en œuvre les bonnes pratiques objet.

Modèles choisis :

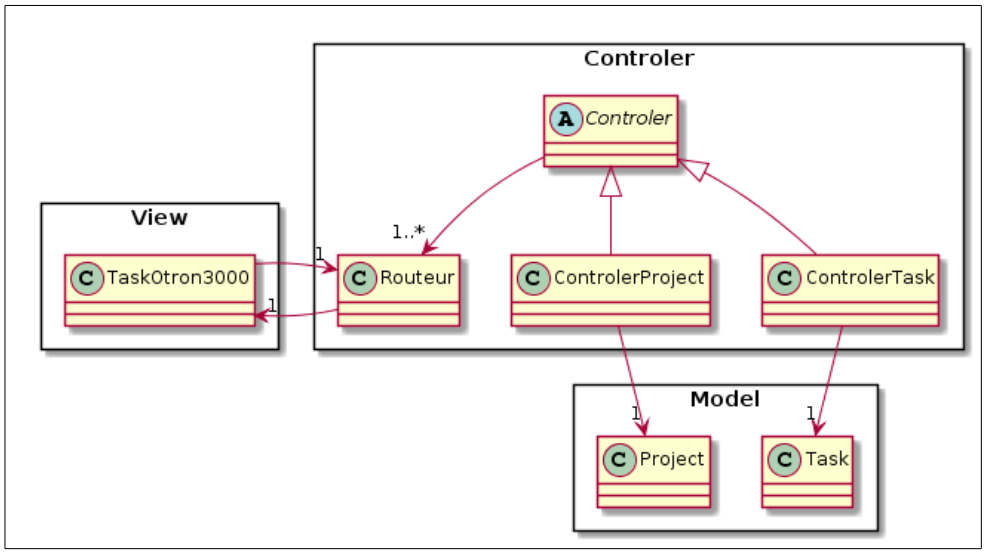
Nous avons décidé à remodeler l'application en suivant deux modèles : le Modèle Vue Contrôleur et le patron Observer.

Modèle Vue Contrôleur :

Nous avons tout d'abord pensé à revoir l'architecture de l'application en lui appliquant le MVC. Voici les principaux arguments qui nous ont poussé à choisir ce modèle :

- la clarté du code
- le traitement des commandes entrées simplifié et plus clair grâce à un routeur
- l'ajout/modification/suppression simplifié de nouvelles fonctionnalités
- la possibilité de développer en même temps pour plusieurs personnes, chaque parties étant relativement indépendantes les unes des autres

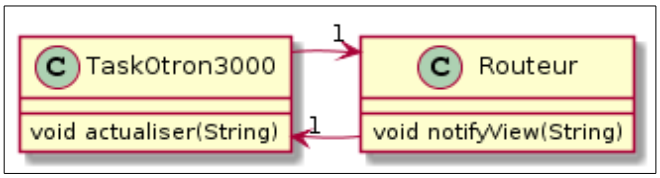
L'architecture pensée devrait ainsi ressembler à ceci :



Nous avons ajouté une classe Project pour séparer le Contrôleur des Modèles cela a donc induit que nous fassions deux classes de Contrôleurs héritant d'une classe abstraite pour que le Routeur puisse utiliser soit l'une, soit l'autre.

Observer :

A la suite de la conception de l'architecture principale de notre application, nous avons choisi d'appliquer le patron Observer sur la Vue et le routeur. Il nous a semblé en effet logique que la Vue soit constamment informée et mise à jour selon les actions effectuées par le routeur.

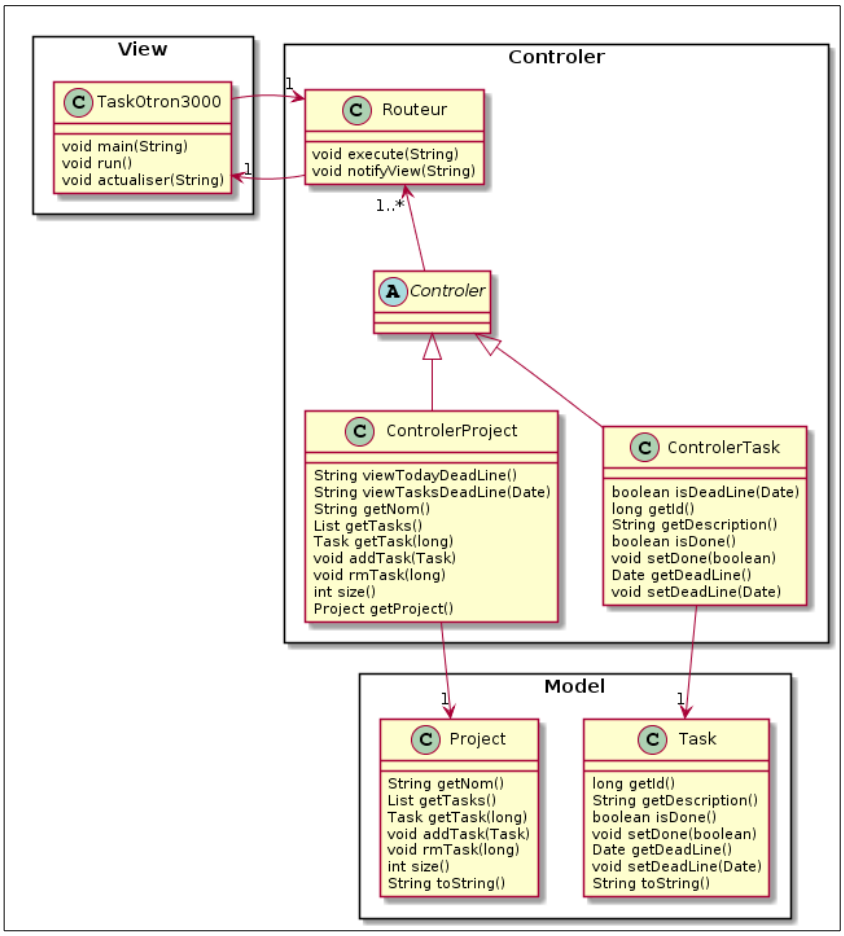


Ici, l'Observateur est le Vue qui possédera donc une fonction pour s'actualiser et le Routeur devra pouvoir notifier la Vue qu'il contient des changements.

Implémentation

Modèle Vue Contrôleur :

L'architecture finale implémentée ressemble donc à ceci :



La Vue renvoie vers le Routeur les commandes entrées par l'utilisateur pour qu'elles soient traitées :

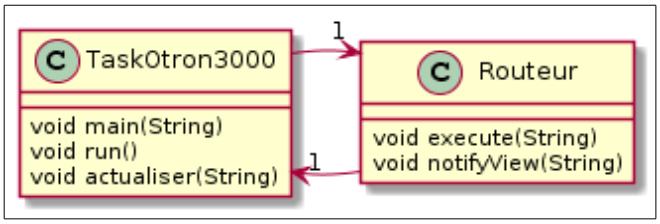
```
public void run() {
    while (true) {
        out.print("> ");
        out.flush();
        String command;
        try {
            command = in.readLine();
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
        if (command.equals(QUIT)) {
            break;
        }
        routeur.execute(command);
    }
}
```

Voici un extrait des traitements :

```
switch (command) {
case "show":
    String msg = "";
    for (ControlerProject p : this.lProject) {
        msg += "Project " + p.getNom() + " :\n";
        for (Task task : p.getTasks()) {
            ControlerTask CT = new ControlerTask(task);
            if(CT.isDone())
                msg += "    [x] " + CT.getId() + " " + CT.getDescription() + " " + CT.getDeadLine() + "\n";
            else
                msg += "    [ ] " + CT.getId() + " " + CT.getDescription() + " " + CT.getDeadLine() + "\n";
        }
        notifyView(msg);
    }
    break;
case "add":
    try {
        if (commandRest[1].equals("project")) {
            try {
                lProject.add(new ControlerProject(new Project(commandRest[2])));
            }catch (Exception e) {
                notifyView("add <project> <Project Name>");
            }
        } else if (commandRest[1].equals("task")) {
            try {
                for (ControlerProject p : this.lProject) {
                    if(p.getNom().equals(commandRest[2])) {
                        String[] date = commandRest[4].split("/");
                        p.addTask(new Task(lastID, commandRest[3], false, new Date(Integer.parseInt(date[2])-1900, Integer.parseInt(d
                        lastID++;
                    }
                }
            } catch (Exception e) {
                notifyView("add <task> <Project Name> <Task Description> <dd/mm/YYYY>");
            }
        }else {
            notifyView("add <project|task> <Project Name> <Task Description>");
        }
    } catch (Exception e) {
        notifyView("add <project|task> <Project Name> <Task Description>");
    }
    break;
case "check":
```

Observer :

Le patron Observer implémenté ressemble donc à ceci :



Le Routeur notifie la Vue au travers de messages selon les actions qu'il exécute :

```
public void notifyView(String msg) {
    view.actualiser(msg);
}
```

```
    case "help":
        notifyView(help());
        break;
    case "today":
        for (ControlerProject p : lProject) {
            notifyView(p.viewTodayDeadLine());
        }
        break;
    case "byday":
        for (ControlerProject p : lProject) {
            notifyView(p.viewTasksDeadLine(new Date()));
        }
        break;
    default:
        notifyView(error(commandLine));
        break;
    }
}
```

La Vue pourra ainsi afficher les modifications à chaque action exécutée.

```
public void actualiser(String msg) {
    out.print(msg + "\n");
    out.flush();
}
```

Application finale

Aperçu :

```
> add project Projet
> add task Projet Tache 01/01/2018
> show
Project Projet :
    [ ] 0 Tache Mon Jan 01 00:00:00 CET 2018

> check 0
> show
Project Projet :
    [x] 0 Tache Mon Jan 01 00:00:00 CET 2018

> uncheck 0
> show
Project Projet :
    [ ] 0 Tache Mon Jan 01 00:00:00 CET 2018
```

Avec une gestion des erreurs :

```
> add bob
add <project|task> <Project Name> <Task Description>
> add task Projet Tache2
add <task> <Project Name> <Task Description> <dd/mm/YYYY>
> check 10
Erreur la tache est inexistante !
```

Nouvelles fonctionnalités :

Ajout de deadlines (et des possibilités de visualisation) :

Chaque tâche a maintenant une deadline qui est affichée et l'utilisateur peut ainsi voir toutes les tâches selon leur deadline (avec les commandes byday ou today).

```
> show
Project Projeeet :
    [ ] 0 TacheAjd Thu Jan 25 00:00:00 CET 2018
    [ ] 1 TacheDemain Fri Jan 26 00:00:00 CET 2018

> today
Projeeet
0 : TacheAjd 25/1/118

> byday 26/01/2018
Projeeet
1 : TacheDemain 26/1/118
```

Avec gestion d'erreur :

```
> byday
byday <dd/mm/YYYY>
> byday 12-12-2012
byday <dd/mm/YYYY>
```

Ajout de la suppression :

L'utilisateur peut maintenant supprimer des tâches grâce à son ID.

```
> show
Project Projet1 :
    [ ] 0 Tache Wed Dec 12 00:00:00 CET 2012
    [ ] 1 TacheASuppr Tue Dec 13 00:00:00 CET 2011
Project Projet2 :
    [ ] 2 Tache2 Sun Oct 10 00:00:00 CEST 2010

> remove 1
> show
Project Projet1 :
    [ ] 0 Tache Wed Dec 12 00:00:00 CET 2012
Project Projet2 :
    [ ] 2 Tache2 Sun Oct 10 00:00:00 CEST 2010
```

Avec gestion d'erreur :

```
> remove
remove <ID>
> remove 10
Erreur la tache est inexistante !
```