22/01/2016

Projet de Refactoring de CPOA

Baptiste CRESPIN - Nicolas CELLIER

baptiste-crespin@univ-tlse2.fr | nicolas-cellier@univ-tlse2.fr

Sommaire

Table des matières

I – Refactoring

1 - Mise en place des bonnes pratiques

Nous avons fait le choix de débuter le projet par améliorer le plus possible le code déjà existant, afin d'avoir de bonnes bases avant de démarrer, et faciliter le travail ultérieur, et l'ajout de fonctionnalités futures.

2 - Projet.java

Nous avons commencé par modifier le code existant pour utiliser un objet Projet, plutôt qu'une liste de taches. Pour cela nous avons créé la class Projet, qui a pour attribut un nom, ainsi qu'une liste de projets. Il y a un getter pour chacun de ses attributs, et on peut ajouter ou supprimer des taches du projet.

```
package com.codurance.training.tasks;

import java.util.ArrayList;

public final class Projet {
    private final String name ;
    private ArrayList<Task> tasks;

    public Projet(String name){
        this.name = name;
        tasks = new ArrayList<Task>();
    }

    public String getName(){
        return this.name ;
    }

    public ArrayList<Task> getTasks(){
        return tasks;
    }

    public void addTask(Task tache){
        tasks.add(tache);
    }

    public void deleteTask(Task task){
        tasks.remove(task);
    }

}
```

3 – Application.java

Nous avons ensuite modifié application.java, afin que celle-ci soit en adéquation avec la classe Projet.

II – Ajout de fonctionnalités

1 – Deadlines

a) Date limite

Pour l'ajout de la fonctionnalité deadline, nous avons commencé par ajouter dans Task.java un attribut de type private Date deadline dans la classe Task, ainsi qu'un getter et un setter.

```
public Date getDeadline() {
    return deadline;
}

public void setDeadline(Date date) {
    try {
        this.deadline = sdf.parse(sdf.format(date));
    } catch (ParseException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
```

Puis nous avons initialisé celui-ci à null dans le constructeur par défaut.

Ensuite, après avoir ajouté les choix correspondant dans le menu, ainsi que l'aide, nous avons ajouté la méthode deadline, permettant de modifier la deadline d'une tache en donnant en paramètre l'ID de la tache ainsi que la date de la nouvelle deadline. Pour cela, elle parcourt toutes les taches de tous les projets jusqu'à rencontrer une tache à l'ID correspondant au paramètre. Puis une fois cette tâche trouvée, elle modifie sa deadline.

```
private void deadline(String commandLine) {
    String[] subCommandRest = commandLine.split(" ", 2);
    int id = Integer.parseInt(subCommandRest[0]);
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (task.getId() == id) {
                Date temp = null;
                try {
                    temp = sdf.parse(subCommandRest[1]);
                } catch (ParseException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
                }
                task.setDeadline(temp);
                // out.println(sdf.format(task.getDeadline()));
                return;
            }
        }
    }
    out.printf("Could not find a task with an ID of %d.", id);
    out.println();
}
```

b) Today

La commande today permet d'afficher toutes les taches dont la deadline est située aujourd'hui. Pour cela, elle va parcourir toutes les taches de tous les projets en affichant la tache au passage si la deadline de celle-ci est égale à celle d'aujourd'hui.

```
private void today() {
    Date today = new Date();
    String temp = sdf.format(today);
    String temp2;
    for (Projet projet : projects) {
        out.println(projet.getName());
        int cpt = 0;
        for (Task task : projet.getTasks()) {
            if (task.getDeadline() != null) {
                temp2 = sdf.format(task.getDeadline());
                if (temp.equals(temp2)) {
                    out.printf("    [%c] %d: %s%n", (task.isDone() ? 'x' : ' '), task.getId(),
                            task.getDescription());
                    cpt++;
                }
            }
        }
        if (cpt == 0) {
            out.print("    Pas de tâche à finir aujourd'hui pour ce projet.");
        }
        out.println();
    }
}
```

2 – Suppression

Pour implémenter la suppression d'une tâche, nous avons, après avoir ajouté les choix correspondant dans le menu, ainsi que l'aide, ajouter la méthode deleteTask, qui parcours toutes les taches de tous les projets jusqu'à trouver une tache ayant un ID correspondant. Si l'ID correspond, la tâche est supprimée du projet à l'aide de la fonction deleteTask de projet :

deleteTask de projet :

```
public void deleteTask(Task task){
    tasks.remove(task);
}
```

deleteTask de l'application :

```
private void deleteTask(int id) {
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (task.getId() == id) {
                projet.deleteTask(task);
                return;
            }
        }
    }
}
```

3 – Visualisation

a) View by date

Pour l'ajout de la fonctionnalité view by date, nous avons commencé par ajouter dans Task.java un attribut de type private Date date dans la classe Task, ainsi qu'un getter. Cette date est automatiquement initialisée à la date d'aujourd'hui au moment de la création d'une tâche.

Ensuite, après avoir ajouté les choix correspondant dans le menu, ainsi que l'aide, nous avons ajouté la méthode viewByDate, permettant d'afficher toutes les taches, en fonction de leur date de création. Pour cela, la méthode parcourt toutes les taches de tous les projets en ajoutant leur date dans une liste, dans leur ordre de création. Puis, on parcourt la liste des dates, en affichant chacune des taches correspondantes.

```
private void viewByDate() {
    ArrayList<Date> dateTrouve = new ArrayList<Date>();
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (!dateTrouve.contains(task.getDate())) {
                dateTrouve.add(task.getDate());
            }
        }
    }
    for (Date date : dateTrouve) {
        out.println(sdf.format(date));
        for (Projet projet : projects) {
            for (Task task : projet.getTasks()) {
                if (date.compareTo(task.getDate()) == 0) {
                    out.printf("    [%c] %d: %s%n", (task.isDone() ? 'x' : ' '), task.getId(),
                            task.getDescription());
                }
            }
        }
    }
}
```

b) View by deadline

View by deadline reprends le même principe que view by date, mais en utilisant cette fois ci les deadlines des projets. Cependant, toutes les tâches n'ont pas forcement de deadline, il a donc fallu ajouter quelques conditions afin de vérifier que deadline soit différent de null.

```
private void viewByDeadline() {
    ArrayList<Date> dateTrouve = new ArrayList<Date>();
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (task.getDeadline() != null) {
                if (!dateTrouve.contains(task.getDeadline())) {
                    dateTrouve.add(task.getDeadline());
                }
            }
        }
    }
    for (Date date : dateTrouve) {
        out.println(sdf.format(date));
        for (Projet projet : projects) {
            for (Task task : projet.getTasks()) {
                if (task.getDeadline() != null) {
                    if (date.compareTo(task.getDeadline()) == 0) {
                        out.printf("    [%c] %d: %s%n", (task.isDone() ? 'x' : ' '), task.getId(),
                                task.getDescription());
                    }
                }
            }
        }
    }
}
```

c) View by project

La méthode view by project existait déjà sous le nom show. Il n'y a donc pas eu de modification à apporter hormis dans le menu et l'aide.

```
private void viewByProject() {
    for (Projet projet : projects) {
        out.println(projet.getName());
        for (Task task : projet.getTasks()) {
            out.printf("    [%c] %d: %s%n", (task.isDone() ? 'x' : ' '), task.getId(), task.getDescription());
        }
        out.println();
    }
}
```

4 - Permettre qu'une tâche puisse « appartenir » à plusieurs projets en même temps

Pour implémenter cette fonctionnalité, nous avons modifié la méthode addTask, afin de permettre au programme d'ajouter des taches dans différents projets. Pour cela avons ajouté que la méthode crée une nouvelle tâche, et si celle-là a le même nom qu'une tache déjà existante, elle change son ID afin que les taches correspondent.

Nous avons dû également modifié Task.java, afin de faciliter l'accès à l'ID et le rendre modifiable, nous avons enlevé le final de l'attribut ID, et ajouté un setter.

```
private void addTask(String project, String description) {
    Projet temp = null;
    for (Projet projet : projects) {
        if (projet.getName().equals(project)) {
            temp = projet;
        }
    }
    if (temp == null) {
        out.printf("Could not find a project with the name \"%s\".", project);
        out.println();
        return;
    }
    Task tempTask = new Task(nextId(), description, false);
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (task.getDescription().equals(tempTask.getDescription())) {
                tempTask.setId(task.getId());
                if (task.isDone()) {
                    tempTask.setDone(true);
                }
            }
        }
    }
    temp.addTask(tempTask);
}
```

Afin de faire en sorte que lorsqu'une tache est cochée dans un projet, elle le soit aussi dans tous les autres projets, nous avons dû également modifier la méthode check, afin qu'elle coche toutes les taches de même ID :

```
private void check(String idString) {
    for (Projet projet : projects) {
        for (Task task : projet.getTasks()) {
            if (task.getId() == Long.parseLong(idString)) {
                task.setDone(true);
            }
        }
    }
}
```

5 - Faire en sorte que l'application maintienne en permanence 2 listes de tâches par projet

Nous avons créé deux listes de tâches au lieu d'une seule dans la classe Projet.

```
private ArrayList<Task> tasksco;
private ArrayList<Task> tasksde;
```

Puis nous avons doublé le code là où il y avait l'ancienne liste comme par exemple sur la méthode viewByProject() ;

```
private void viewByProject() {
    for (Projet projet : projects) {
        out.println(projet.getName());
        for (Task task : projet.getTasksco()) {
            out.printf("    [%c] %d: %s%n", 'x', task.getId(), task.getDescription());
        }
        for (Task task : projet.getTasksde()) {
            out.printf("    [%c] %d: %s%n", ' ', task.getId(), task.getDescription());
        }
        out.println();
    }
}
```

Et enfin nous avons refait les méthodes check et uncheck un peu plus compliquées que les autres.

```
private void check(String idString) {
    int temp = -1;
    ArrayList<Task> toRemove = new ArrayList<Task>();
    ArrayList<Task> toAdd = new ArrayList<Task>();
    for (Projet projet : projects) {
        for (Task task : projet.getTasksde()) {
            if (task.getId() == Long.parseLong(idString)) {
                task.setDone(true);
                toAdd.add(task);
                toRemove.add(task);
                temp = 1;
            }
        }
        if (!toAdd.isEmpty() && !toRemove.isEmpty()) {
            projet.getTasksco().addAll(toAdd);
            projet.getTasksde().removeAll(toRemove);
            toAdd.clear();
            toRemove.clear();
        }
    }
    if (temp == -1) {
        out.printf("Could not find a task check with an ID of %s.", idString);
        out.println();
    }

}
```

```
private void uncheck(String idString) {
    int temp = -1;
    ArrayList<Task> toRemove = new ArrayList<Task>();
    ArrayList<Task> toAdd = new ArrayList<Task>();
    for (Projet projet : projects) {
        for (Task task : projet.getTasksco()) {
            if (task.getId() == Long.parseLong(idString)) {
                task.setDone(false);
                toAdd.add(task);
                toRemove.add(task);
                temp = 1;
            }
        }
        if (!toAdd.isEmpty() && !toRemove.isEmpty()) {
            projet.getTasksde().addAll(toAdd);
            projet.getTasksco().removeAll(toRemove);
            toAdd.clear();
            toRemove.clear();
        }
    }
    if (temp == -1) {
        out.printf("Could not find a task uncheck with an ID of %s.", idString);
        out.println();
    }
}
```

Ne pouvant pas ajouter et supprimer des tâches directement depuis le parcours de ces tâches on a donc crée deux ArrayList pour y ajouter les tâches à enlever et à mettre dans les listes de la classe Projet cochée/décochée et appliquer ces changements en sortant de la boucle.

6- Faire en sorte qu'une tâche puisse être elle-même une liste de tâche

Nous n'avons malheureusement pas eu le temps d'implémenter cette fonctionnalité.

III – Tests

Nous avons rencontré un problème lors de l'exécution des tests, ils ne marchent pas.