# Projet CPOA

Lopez Clément
Robert Guillaume
1A

BLAGNAC
iut Midi-Pyrénées
UNIVERSITÉ TOULOUSE
Jean Jaurès

# Table des matières

# Introduction

Ce projet permet de mettre en place les bonnes pratiques et méthodes de programmation apprises lors des cours de CPOA. Pour cela nous avons à notre disposition une application qui doit être améliorée. Cependant cette application n'est pas optimisée pour l'extension. Notre but ici est d'y ajouter des fonctionnalités et de permettre d'avoir un code qui soit expliqué et ouvert aux futures extensions. Nous devons permettre aux prochains utilisateurs de comprendre très rapidement comment marche l'application.

## Existant

En début de projet nous avons dû nous familiariser avec l'existant. Le code permettant de gérer les différentes commandes est un switch :

```
private void execute(String commandLine) {
    String[] commandRest = commandLine.split(" ", 2);
    String command = commandRest[0];
    switch (command) {
        case "show":
            show();
            break;
        case "add":
            add(commandRest[1]);
            break;
        case "check":
            check(commandRest[1]);
            break;
        case "uncheck":
            uncheck(commandRest[1]);
            break;
        case "help":
            help();
            break;
        default:
            error(command);
            break;
    }
}
```

Par conséquent à chaque ajout d'une nouvelle fonctionnalité il nous faut agrandir ce switch. Cela peut vite devenir très long et compliqué. Aussi, toutes les méthodes utilisées sont dans cette classe.

```
public TaskLists(BufferedReader reader, PrintWriter writer) {
public void run() {
private void execute(String commandLine) {
private void show() {
private void add(String commandLine) {
private void addProject(String name) {
private void addTask(String project, String description) {
private void check(String idString) {
private void uncheck(String idString) {
private void setDone(String idString, boolean done) {
private void help() {
private void error(String command) {
private long nextId() {
}
```

Pour finir, ces méthodes de sont pas commentées c'est pourquoi il a fallu du temps avant de comprendre comment elles fonctionnaient.

# Solutions techniques

## Patron

Afin d'optimiser l'extension et l'utilisation de cette application nous avons utilisé le patron **Stratégie** :

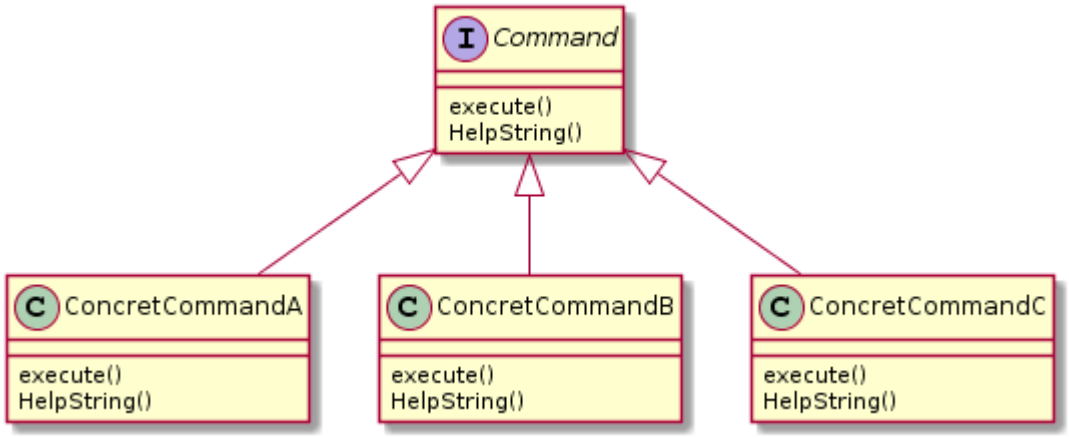


*Figure 1 : Diagramme de classe (patron stratégie)*

En effet nous voulions permettre à l'application de pouvoir effectuer le traitement de manière dynamique. Grâce à ce patron nous avons pu réaliser une application qui respectée les principes SOLID. En particulier Open-Closed Principle sur les commandes.

## Re-factoring

Nous avons donc revu l'organisation de l'application afin d'appliquer les bonnes pratiques (SOLID). Tout d'abord nous avons fait une interface *Command* avec une méthode execute() qui permet d'exécuter le code de chaque commande.

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

import java.util.List;

public interface Command {

    public static String HelpString(){
        return "help method";
    }

    List<Project> execute(String commandLine, List<Project> projects);

}
```

Cela nous a permis d'avoir une méthode execute() beacoup plus légère dans la classe principale. Aussi nous n'avons plus à utiliser de switch. Grâce à cette réorganisation, on pourra ajouter autant de commande que l'on veut, sans modifier une seule ligne de cette méthode.

```java
    /**
     * This method calls the different commands and executes them
     * @param commandLine
     * @throws ArrayIndexOutOfBoundsException
     */
    private void execute(String commandLine) throws ArrayIndexOutOfBoundsException {
        String[] commandRest = commandLine.split(" ", 2);
        String command = commandRest[0];
        String arguments;
        if (command.length()==commandLine.length()){
            arguments="chaine pour passer le try/catch";
        }
        else{
            arguments = commandRest[1];
        }
        try{
            tasks = ListCommands.get(command).execute(arguments, tasks);
        }
        catch(Exception e){
            error(command);
        }
    }
```

## Fonctionnalités

Grâce à cette réorganisation expliquée précédemment, lorsque l'on veut ajouter une nouvelle commande on procède en 2 étapes :

- Création de la classe qui implémente Command,
- Ajout de cette commande dans « InitiateListCommands » afin de pouvoir y accéder grâce au nom de la commande en String (voir capture d'écran ci-dessous).

```
//List of commands
private static Map<String, Command> ListCommands = new HashMap<String, Command>();

//Add of the commands in the Map
public static void InitiateListCommands() {
    ListCommands.put("delete", new Delete());
    ListCommands.put("add", new Add());
    ListCommands.put("check", new Check());
    ListCommands.put("today", new Today());
    ListCommands.put("help", new Help());
    ListCommands.put("deadline", new Deadline());
    ListCommands.put("view", new view());
    ListCommands.put("attach", new Attach());
}
```

1. **Deadline**

   Pour ajouter une date limite à une tâche on a créé une nouvelle commande « deadline ». Celle-ci permet de modifier la valeur de deadline d'une tâche grâce à la méthode setDeadline().

2. **Suppression**

   On cherche la tâche qui a l'id correspondant à l'id recherché en parcourant les différents projets.

3. **Visualisations**

   Lors de la création de cette fonctionnalité nous avons réservé différents traitements en fonction des arguments, par projet ou par deadline. Cependant dans les deux cas on parcourt les projets.

4. **Une tâche, plusieurs projets**

   Concernant la réalisation de cette tâche, nous avons créé une nouvelle commande nommée « Attach ». Cette commande permet à l'utilisateur d'attacher une tâche déjà existante à un autre projet.

### 5. Deux listes de tâches

### 6. Tâche, liste de tâche

Pour cela nous avons notre classe Project extends de Task. Comme les projets contiennent une liste de tâches alors on peut ajouter une tâche ou un projet à un sous-projet.

## Diagramme de classe

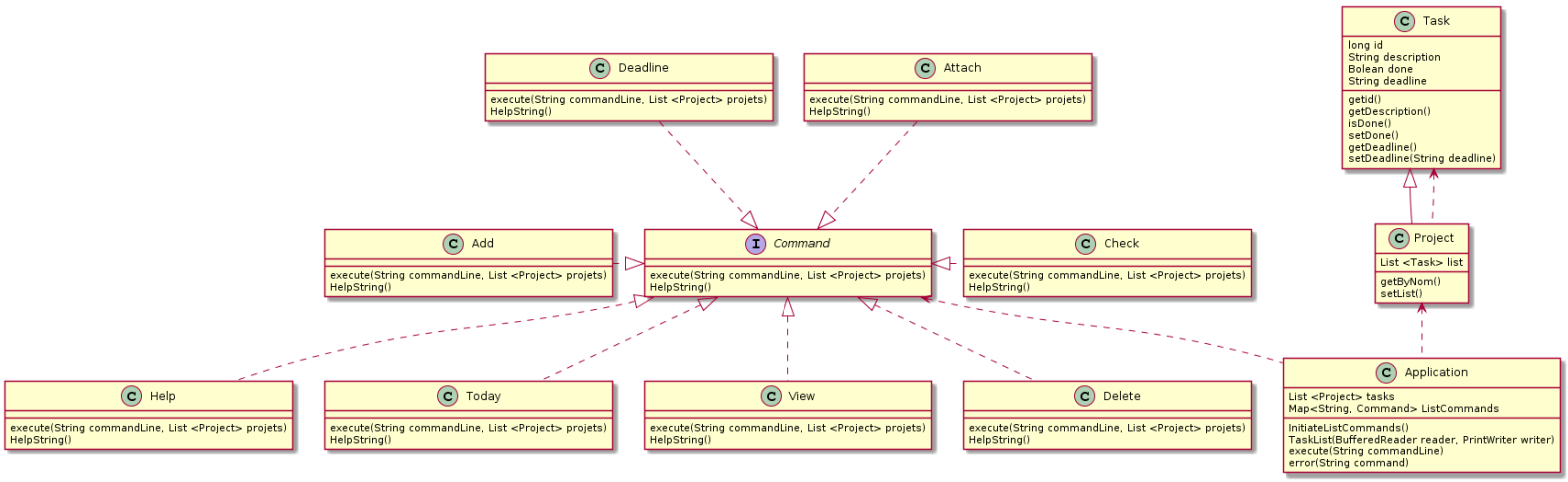


*Figure 2 : Diagramme des classes*

# Conclusion

Pour conclure on peut dire que l'application est fonctionnelle, cependant elle peut être améliorée. Grâce à la réorganisation effectuée, les extensions de cette application seront facilitées. D'autres fonctionnalités pourrait y être ajoutées par exemple en enregistrant les données à chaque fois que l'on quitte l'application ce qui permettrait de ne pas recréer tous les projets. Nous avons vu ici, l'importance de la documentation et d'un code clair et lisible.