Projet refactoring

Mahmoud EL HAMLAOUI

<mahmoud.elhamlaoui@gmail.com>

Revision History

Revision 1.0	2017-01-03 -	MEH

1. Contexte

Vous êtes chargé de prendre la suite d'un développement pour laquelle la documentation est minimaliste et dont les sources sont disponibles ici :

https://github.com/codurance/task-list/tree/master/java

1. Récupérez les sources du projet :
   - soit en clonant le dépôt :

     ```
     git clone https://github.com/codurance/task-list.git
     ```

   - soit en téléchargeant directement le fichier .zip[1].

     ```
      Importez le projet en tant que projet {maven}:
     - menu:File[Import...>Maven>Existing Maven Project]
     - sélectionnez le fichier `pom.xml` du répertoire `java`
     ```

     NOTE: Si vous n'avez pas Maven :

   - Créez un projet java
   - Importez le répertoire `src` du répertoire `java` en cliquant droit sur votre paquetage `src` et en choisissant Import ‣ File System.
   - Fixez le problème des imports JUnit en ajoutant la librairie (par quickfix)

[1] https://github.com/codurance/task-list/archive/master.zip

- Changez la ligne dans `ApplicationTest` : `import org.hamcrest.Matchers.is` par `import static org.hamcrest.core.Is.*` (merci Hugues, Lino and others).

2. Lancez les tests pour vérifier que tout est OK (`ApplicationTest.java`)
3. À partir de ce code de test (cf. ci-dessous), déterminez ce que fait l'application et comment elle fonctionne.

ApplicationTest.java

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

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PipedInputStream;
import java.io.PipedOutputStream;
import java.io.PrintWriter;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static java.lang.System.lineSeparator;
import static org.hamcrest.MatcherAssert.assertThat;
import static org.hamcrest.Matchers.is;

public final class ApplicationTest {
    public static final String PROMPT = "> ";
    private final PipedOutputStream inStream = new PipedOutputStream();
    private final PrintWriter inWriter = new PrintWriter(inStream, true);

    private final PipedInputStream outStream = new PipedInputStream();
    private final BufferedReader outReader = new BufferedReader(new
 InputStreamReader(outStream));

    private Thread applicationThread;

    public ApplicationTest() throws IOException {
        BufferedReader in = new BufferedReader(new InputStreamReader(new
 PipedInputStream(inStream)));
        PrintWriter out = new PrintWriter(new PipedOutputStream(outStream), true);
        TaskList taskList = new TaskList(in, out);
        applicationThread = new Thread(taskList);
    }

    @Before public void
```

```
    start_the_application() {
        applicationThread.start();
    }

    @After public void
    kill_the_application() throws IOException, InterruptedException {
        if (!stillRunning()) {
            return;
        }

        Thread.sleep(1000);
        if (!stillRunning()) {
            return;
        }

        applicationThread.interrupt();
        throw new IllegalStateException("The application is still running.");
    }

    @Test(timeout = 1000) public void
    it_works() throws IOException {
        execute("show");

        execute("add project secrets");
        execute("add task secrets Eat more donuts.");
        execute("add task secrets Destroy all humans.");

        execute("show");
        readLines(
            "secrets",
            "    [ ] 1: Eat more donuts.",
            "    [ ] 2: Destroy all humans.",
            ""
        );

        execute("add project training");
        execute("add task training Four Elements of Simple Design");
        execute("add task training SOLID");
        execute("add task training Coupling and Cohesion");
        execute("add task training Primitive Obsession");
        execute("add task training Outside-In TDD");
        execute("add task training Interaction-Driven Design");

        execute("check 1");
        execute("check 3");
        execute("check 5");
        execute("check 6");
```

```
        execute("show");
        readLines(
                "secrets",
                "    [x] 1: Eat more donuts.",
                "    [ ] 2: Destroy all humans.",
                "",
                "training",
                "    [x] 3: Four Elements of Simple Design",
                "    [ ] 4: SOLID",
                "    [x] 5: Coupling and Cohesion",
                "    [x] 6: Primitive Obsession",
                "    [ ] 7: Outside-In TDD",
                "    [ ] 8: Interaction-Driven Design",
                ""
        );

        execute("quit");
    }

    private void execute(String command) throws IOException {
        read(PROMPT);
        write(command);
    }

    private void read(String expectedOutput) throws IOException {
        int length = expectedOutput.length();
        char[] buffer = new char[length];
        outReader.read(buffer, 0, length);
        assertThat(String.valueOf(buffer), is(expectedOutput));
    }

    private void readLines(String... expectedOutput) throws IOException {
        for (String line : expectedOutput) {
            read(line + lineSeparator());
        }
    }

    private void write(String input) {
        inWriter.println(input);
    }

    private boolean stillRunning() {
        return applicationThread != null && applicationThread.isAlive();
    }
}
```

Task.java

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

public final class Task {
    private final long id;
    private final String description;
    private boolean done;

    public Task(long id, String description, boolean done) {
        this.id = id;
        this.description = description;
        this.done = done;
    }

    public long getId() {
        return id;
    }

    public String getDescription() {
        return description;
    }

    public boolean isDone() {
        return done;
    }

    public void setDone(boolean done) {
        this.done = done;
    }
}
```

TaskList.java

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

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;

public final class TaskList implements Runnable {
    private static final String QUIT = "quit";
```

```
    private final Map<String, List<Task>> tasks = new LinkedHashMap<>();
    private final BufferedReader in;
    private final PrintWriter out;

    private long lastId = 0;

    public static void main(String[] args) throws Exception {
        BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
        PrintWriter out = new PrintWriter(System.out);
        new TaskList(in, out).run();
    }

    public TaskList(BufferedReader reader, PrintWriter writer) {
        this.in = reader;
        this.out = writer;
    }

    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;
            }
            execute(command);
        }
    }

    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;
        }
    }

    private void show() {
        for (Map.Entry<String, List<Task>> project : tasks.entrySet()) {
            out.println(project.getKey());
            for (Task task : project.getValue()) {
                out.printf("    [%c] %d: %s%n", (task.isDone() ? 'x' : ' '),
task.getId(), task.getDescription());
            }
            out.println();
        }
    }

    private void add(String commandLine) {
        String[] subcommandRest = commandLine.split(" ", 2);
        String subcommand = subcommandRest[0];
        if (subcommand.equals("project")) {
            addProject(subcommandRest[1]);
        } else if (subcommand.equals("task")) {
            String[] projectTask = subcommandRest[1].split(" ", 2);
            addTask(projectTask[0], projectTask[1]);
        }
    }

    private void addProject(String name) {
        tasks.put(name, new ArrayList<Task>());
    }

    private void addTask(String project, String description) {
        List<Task> projectTasks = tasks.get(project);
        if (projectTasks == null) {
            out.printf("Could not find a project with the name \"%s\".", project);
            out.println();
            return;
        }
        projectTasks.add(new Task(nextId(), description, false));
```

```
    }

    private void check(String idString) {
        setDone(idString, true);
    }

    private void uncheck(String idString) {
        setDone(idString, false);
    }

    private void setDone(String idString, boolean done) {
        int id = Integer.parseInt(idString);
        for (Map.Entry<String, List<Task>> project : tasks.entrySet()) {
            for (Task task : project.getValue()) {
                if (task.getId() == id) {
                    task.setDone(done);
                    return;
                }
            }
        }
        out.printf("Could not find a task with an ID of %d.", id);
        out.println();
    }

    private void help() {
        out.println("Commands:");
        out.println("  show");
        out.println("  add project <project name>");
        out.println("  add task <project name> <task description>");
        out.println("  check <task ID>");
        out.println("  uncheck <task ID>");
        out.println();
    }

    private void error(String command) {
        out.printf("I don't know what the command \"%s\" is.", command);
        out.println();
    }

    private long nextId() {
        return ++lastId;
    }
}
```

2. Refactoring

Vous devez améliorer le plus possible cette application en (cf. détails juste après cette liste) :

- y ajoutant de nouvelles fonctionnalités au passage (cf. ci-dessous).
- s'attaquant aux types primitifs massivement utilisés
- mettant en oeuvre les bonnes pratiques objet

2.1. Nouvelles fonctionnalités

Vous choisirez parmis les attentes clients suivantes (numérotées pour vos documentations, pas pour les prioriser) :

1. Deadlines
   a. Chaque tâche pourra disposer d'une "date limite" (exemple de commande : `deadline <ID> <date>`).
   b. La commande `today` permettra de voir toutes les tâches dont la deadline est aujourd'hui.
2. Suppression
   a. Autoriser l'utilisateur à supprimer des tâches (exemple de commande : `delete <ID>`).
3. Visualisations
   a. Visualiser les tâches par date (exemple de commande : `view by date`).
   b. Visualiser les tâches par deadline (exemple de commande : `view by deadline`).
   c. Sans changer la fonctionnalités qui permet à un utilisateur de voir les tâches par projet, changer la commande en `view by project`.
4. Permettre qu'une tâche puisse « appartenir » à plusieurs projets en même temps (si cochée dans l'une, cochée dans toutes par exemple)
5. Faire en sorte que l'application maintienne en permanence 2 listes de tâches par projet : les "cochées" et les "non cochées"
6. (Attention celle-là est coton) Faire en sorte qu'une tâche puisse être elle-même une liste de tâche

> On pourra faire des visualisations graphiques simples (genre post-its).

Attention, certaines fonctionnalités sont assez incompatibles entre elles, et certaines sont beaucoup plus complexes à mettre en oeuvre que d'autres.

2.2. Types primitifs

Si vous avez observé attentivement le code, vous vous êtes rendu compte qu'il utilise assez peu d'objets et par contre beaucoup de types primitifs (entiers, char, strings, collections, etc.). Les concepts métiers sont faiblement présent (`task` , `project` , etc.).

Essayez, en implémentant vos nouvelles fonctionnalités, de vous débarrasser le plus possible des types primitifs.

Un bon exemple pour voir si vous vous focalisez sur les concepts métiers est de vérifier que les types primitifs sont uniquement employés dans les paramètres des constructeurs, les variables locales ou les attributs provés. Ils ne devraient jamais être retournés ou être passés en paramètres à vos méthodes (exception faite des lectures clavier, etc.).

2.3. Bonnes pratiques

Pour n'en citer que quelques-unes (SOLID) :

- **S**ingle Responsibility Principle
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Essayez d'identifier clairement dans votre documentation les principes mis en oeuvre

3. À propos des tests

Vous remarquerez que le seul test fournit est un test fonctionnel. Il pourra vous être utile pour vérifier que votre application respecte toujours bien le cahier des charges initial. Vous pourrez même l'enrichir pour tester que vos nouvelles commandes (du type `view by …`) fonctionnent.

Par contre il ne vous dispense pas du tout des tests unitaires classiques pour les classes que vous allez produire.

4. Attendus du projet

4.1. Modèles à réaliser

Je ne vous embête pas ce coup-ci avec les modèles mais n'hésitez pas à en utiliser (des cohérents avec votre code) pour vos documentations.

Voici d'ailleurs le diagramme de classe minimaliste initial de l'application :

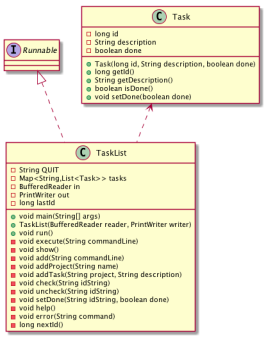

Figure 1. Diagramme de classe initial de l'application

4.2. *Delivrables* attendus

Le dépôt doit contenir les documents suivant :

- Un rapport au format `.pdf` contenant (outre les éléments habituels d'un rapport comme les noms et contact des binômes, une table des matières, …) une courte explication par chaque fonctionnalité nouvelle ou refactoring précis avec des extraits de code illustratifs et une justification.
- Un fichier `.zip` contenant un export de votre projet eclipse qui contiendra à minima un répertoire `src` avec les sources et un répertoire `doc` avec la javadoc.

> Pour générer ce .zip, chisissez bien l'option Export…□□□ General□□ Archive File après un click droit sur le projet en question.

4.3. Evaluation et critères

Vous pourrez travailler en groupe (2 max).

Les principaux critères qui guideront la notation seront :

- pertinence des choix
- qualité du code
- qualité du rapport (illustration, explications)
- extras (tests, modèles)

En cas de besoin, n'hésitez pas à me contacter (mahmoud.elhamlaoui@gmail.com[2]).

[2] mailto:mahmoud.elhamlaoui@gmail.com