DUT-Info/S3/M3105 (CPOA) : Projet refactoring

Table des Matières

1. Refactoring d'un code existant

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

- ☐ Replacez et utilisez ce README.adoc comme rapport de votre refactoring.
- ☐ Expliquez comment deployer votre application (e.g., `mvn install` ou `gradle install`)

2. Initialisation

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.

     

     Attention de bien copier/cloner dans votre workspace, ou en tout cas sur un répertoire pérenne (conseil valables si vous utilisez les PC des salles machines).

2. Importez le projet en tant que projet Maven:
   - **File › Import… › Maven › Existing Maven Project**
   - sélectionnez le fichier `pom.xml` du répertoire `java`

Si vous n'avez pas Maven :

1. Créez un projet java
2. Importez le répertoire `src` du répertoire `java` en cliquant droit sur votre paquetage `src` et en choisissant **Import › File System**.
3. Fixez le problème des imports JUnit en ajoutant la librairie (par quickfix)
4. Changez la ligne dans `ApplicationTest` : `import org.hamcrest.Matchers.is` par `import static org.hamcrest.core.Is.*` (merci Hugues, Lino and others).

Seul le répertoire `java` nous intéresse. Mais n'hésitez pas à regarder les autres langages si ça vous aide.

3. Lancez les tests pour vérifier que tout est OK (`ApplicationTest.java`)
4. 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;
    }
}
```

3. Travail à effectuer

Vous avez seulement 3 séances de 1,5h en semaines 1 pour améliorer le plus possible le code (sans en changer les fonctionalités ni en ajouter nécessairement) de cette application, en y intégrant vos acquis de l'IUT :

- du module CPOA : intégration de patrons de conception. Cela peut être ceux vus en cours, ou d'autres (il y en a plein sur Internet !),
- du module COO : bonnes pratiques de la conceptions orientée objet. Pensez à `SOLID`, l'encapsulation, votre expérience en développement Java !

Commencez d'abord par établir un objectif et vous répartir les tâches ! Vous perdrez énormément de temps si vos changements s'avèrent non adaptés à l'application ! Validez-le ensuite avec votre intervenant de TP, il est là pour vous aider.

Commencez par le plus simple. Le patron le plus complexe n'est pas toujours le plus adapté !

Dans votre étude, anticipez de possibles futures évolutions de l'application. Ex: affichage déporté, configuration de plusieurs algorithmes, types de stockages des données… (l'objectif de ce projet est de refactorer le code, pas d'ajouter de nouvelles fonctionnalités).

4. Attendus du projet

4.1. Dépôt GitHub

Vous travaillerez sur le projet GitHub créé via le lien fourni (classroom, comme en TP).

Vous penserez à ajouter `jmbruel` ainsi que votre prof de TD/TP comme contributeur.

La branche `master` sera celle où nous évaluerons votre `README` (en markdown ou asciidoc et contenant votre "rapport" avec entre autre le nom des 2 binômes), vos codes (répértoire `src`), vos documentations éventuelles (répétoire `doc`).

4.2. Modèles à réaliser

On 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.

4.3. *Delivrables* attendus

Votre projet sera constitué du contenu de la branche master de votre dépôt à la date du **vendredi 14/01/2021** à minuit.

Votre rapport sera votre `README`, 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.

5. Evaluation et critères

Vous pourrez travailler en groupe de 2 max.

Les principaux critères qui guideront la notation seront :

- pertinence des choix
- pertinence des tests

- qualité du code
- qualité du rapport (illustration, explications)
- nombre et difficulté des fonctionalités ajoutées (pensez à utiliser les numéros de fonctionalités)
- extras (modèles)

En cas de besoin, n'hésitez pas à me contacter (jean-michel.bruel@univ-tlse2.fr) ou à poser des questions sur le channel `cpoa` du Discord de l'IUT.