Development Frameworks - Task 2

Thibault Neveu
Master Computer Intelligence
Student id number: 17903371
tdgn2@kent.ac.uk

10/11/2017

Contents

1 Introduction **3**

2 Unit Test **3**
2.1 Configure the project for Unit Test: 3
2.2 Create a first Unit Test: 3

3 Test coverage **3**
3.0.1 Advanced settings 4

1 Introduction

For this assessment, I choose to used IntelliJ IDEA to run my test coverage, a test coverage give the possibility to figure out which part of the code is cover by my unit test. IntelliJ is pre-installed with it's on test/code coverage tool. The plugin is available by default/ Moreover, to launch our first coverage, we just need to click on the appropriate button, which is just under the button to compile the project.

2 Unit Test

I choose to handle my Unit test with JUnit. The latter is not directly installed, but this can be done in a few simple actions by navigating into the menus. The JUnit library is available within the Maven repository.

2.1 Configure the project for Unit Test:

It is not a mandatory part, however, as it is a great practice to create a folder dedicated to Unit Test, I started with this step. To do it we need to precise to Iteliji the location of our test folder. We can do it by going to the "Project Structure Setting" and select a folder of our choice.

2.2 Create a first Unit Test:

By pressing Alt+Enter on any class, we can create a first JUnit test and automatically select the methods we are interested in. Then, a new pre-filled script is built in the Test folder. Once I am in my file, I can create any analysis I want, method per method. One of the good things here is the possibility to test any of them one by one. Without running all the class. Good point, however, this impel us to declare a new instance of the class to test in each method. What is not very convenient.

From a design point of view, the user experience is very nice, a green bar appears to the bottom to directly indicate that 100% tests are confirmed. It is further what we are looking for. Moreover, If somehow we forgot to add methods to the test class, we can quickly add it by pressing alt+insert. This, open a new menu where we can select the desired method to insert.

Once we are happy with our first test methods, we can run a test using the test coverage tool. This one can be directly accessible by clicking slightly below the run method.

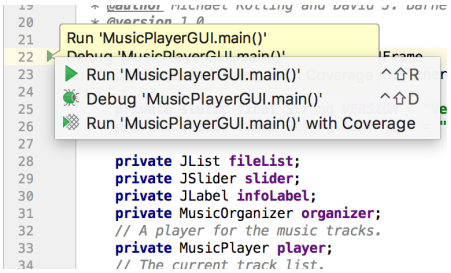


Figure 1: Run a test coverage

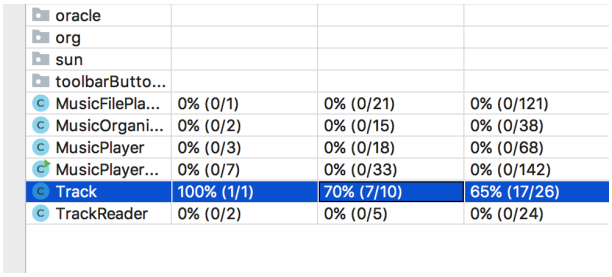
3 Test coverage

To check the result of the test coverage we got a direct access on the right of the window to a summary table showing some useful pieces of information. First, a list indicates whether a class is covered by some test or not. Then, the percentage of methods analyze and the portion of lines cover. In order to get more details about which method and which line are tested, all we need to dot is to click on the corresponding class. Then, above each line, a color label (red or green) is added in order to distinguish both class (tested or not).

Note about the color: Is possible to edit the coverage tools to changes colors, which a perfect features, showing once again the ambition from IntelliJ to propose good technical services as well as a good user experience.

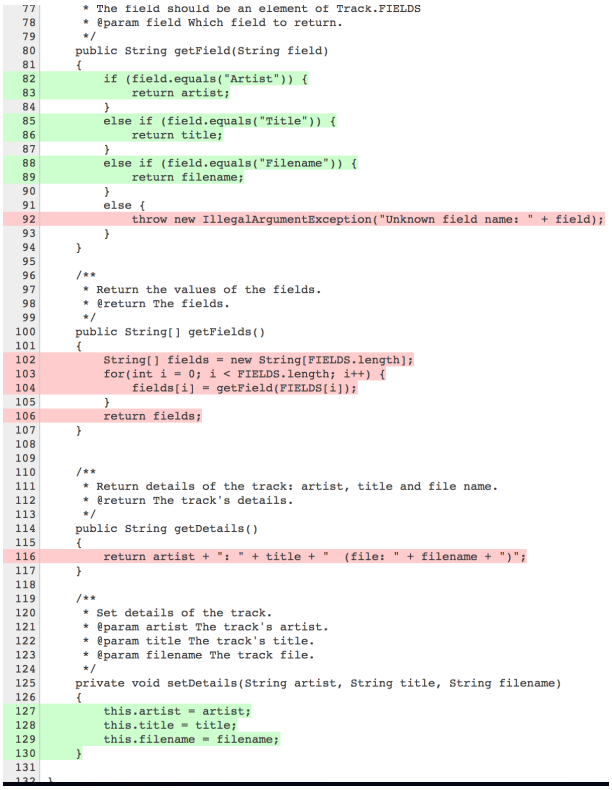
3.0.1 Advanced settings

If we want to go further, it is possible to customise the coverage tool. Indeed, instead of running a Test class, we can do the same thing for a Pattern or a Category. Moreover, we can choose the way the test code is executed by running each task in a separate process. One of my favorites features is the "Test Cover export". Indeed its give the possibility to export the result of the coverage in a separate document. In this case, in HTML. This report is a breakdown of all method and classes. We can use it is to work with a team and to share/save the state of a project at a given time.



oracle			
org			
sun			
toolbarButto...			
MusicFilePla...	0% (0/1)	0% (0/21)	0% (0/121)
MusicOrgani...	0% (0/2)	0% (0/15)	0% (0/38)
MusicPlayer	0% (0/3)	0% (0/18)	0% (0/68)
MusicPlayer...	0% (0/7)	0% (0/33)	0% (0/142)
Track	100% (1/1)	70% (7/10)	65% (17/26)
TrackReader	0% (0/2)	0% (0/5)	0% (0/24)

Figure 2: Result of test coverage - Percentages



```
77      * The field should be an element of Track.FIELDS
78      * @param field Which field to return.
79      */
80     public String getField(String field)
81     {
82         if (field.equals("Artist")) {
83             return artist;
84         }
85         else if (field.equals("Title")) {
86             return title;
87         }
88         else if (field.equals("Filename")) {
89             return filename;
90         }
91         else {
92             throw new IllegalArgumentException("Unknown field name: " + field);
93         }
94     }
95
96     /**
97      * Return the values of the fields.
98      * @return The fields.
99      */
100    public String[] getFields()
101    {
102        String[] fields = new String[FIELDS.length];
103        for(int i = 0; i < FIELDS.length; i++) {
104            fields[i] = getField(FIELDS[i]);
105        }
106        return fields;
107    }
108
109
110    /**
111     * Return details of the track: artist, title and file name.
112     * @return The track's details.
113     */
114    public String getDetails()
115    {
116        return artist + ": " + title + "  (file: " + filename + ")";
117    }
118
119    /**
120     * Set details of the track.
121     * @param artist The track's artist.
122     * @param title The track's title.
123     * @param filename The track file.
124     */
125    private void setDetails(String artist, String title, String filename)
126    {
127        this.artist = artist;
128        this.title = title;
129        this.filename = filename;
130    }
131
132 }
```

Figure 3: Result of test coverage - Class