Assignment 1 - G03P02

Group information

- Ana Inês Oliveira de Barros - `up201806593@fe.up.pt`;
- João de Jesus Costa - `up201806560@fe.up.pt`

Project description

The project is a time tracking tool that allows users to track their daily tasks and the time invested in each of them. With this app, users obtain a record of their activities and work time.

Project structure

- The project is organized into 4 packages;
- The main function resides in the `JTimeSchedApp.java` file (in the root package);
- Most of the code logic resides in the `Project` class (in the project package).

```
de.dominik_geyer.jtimesched            - root package
|-- JTimeSchedApp.java                 - location of the main function
|-- gui                                - GUI-related code
|   |-- ...
|    -- table
|       | ...
|-- misc
|    -- PlainTextFormatter.java
 -- project                           - task-related code
    |-- Project.java                   - where most code logic resides
    |   ...
```

Static testing

Static testing is one of the testing techniques for checking faults in software. The main advantage of static code analysis is the fact that it is performed without executing the code, thus being more lightweight than other types of testing. For this reason, it is usually employed before executing the rest of the software's test suite.

Static test tools can detect faults before the execution of other tests, and warn about *suspicious* aspects of the code. Another aspect that these tools help with is code maintainability/design by allowing enforcement of a certain code style.

Tool - Checkstyle

The Checkstyle tool makes it ideal for projects that want to enforce a coding standard. It can find both class and methods design problems, and it has also the ability to check code layout and formatting issues.

Configuration

The configuration is stored in `rulesets/checkstyle-rules.xml`. This configuration was based on the Google code style configuration. Some rules were adapted to reduce the number of irrelevant issues (**false-positives**).

- **Indentation** - The default configuration considers indentation using tabs as a warning. Since we are working with a fork of an open source project, we decided that the best option was to conform to the indentation style of the author of the code: tabs as indentation, *switch's case* statements not indented, etc...
- **Underscore in root package name** - There were multiple warnings related to the root package's name because it contains an underscore, `_`. In the default configuration package names are checked against a specific REGEX which does not allow the `_` character. Since we considered this error irrelevant, we changed the rule to accept the `de.dominik_geyer.jtimesched` package name.
- **Import declaration groups** - The tool was indicating that there was an issue regarding the groups of declaration of imports. For example, an import from a `java` package shouldn't be separated from a `javax` import. Since the author of the project organized the import declarations by grouping third-party packages separately from the java standard ones, we adapted this rule to accept this separation. Since there is no information about the author's preference regarding the grouping of static imports we left the default rule as it is (to separate the group of static import declarations).
- **If statement braces** - It is common for the author to skip braces on `if` statements with a single line, so we disabled this warning.

The Google ruleset was exaggerating the number of warnings because it was generating a warning for each line on the project. This was caused by using tabs as indentations.

Changing these configurations eliminates most warnings, but generates new ones that we consider **true-positives**.

Results

The default Google ruleset generated 4194 warnings. After the configurations described above, the number of warnings was reduced to 327. By fixing some of these problems (described in the next sections), we reduced the number of warnings to 263.

Bugs & Fixes - Method indentation level

Warning: *'method def lcurly' has incorrect indentation level 8, expected level should be 16* and *'method def rcurly' has incorrect indentation level 8, expected level should be 16*.

Location: file `JTimeSchedApp.java`, lines 100, 389, and 401.

```
// Before fix:
public static String getAppVersion()
{
// ---------------------------------------
// After fix:
public static String getAppVersion() {
```

Bugs & Fixes - Indentation tabs & spaces

Warning: *'method def' child has incorrect indentation level 8, expected level should be 16.*

Location: file `JTimeSchedFrame.java`, lines 136, 709, 712, 714, and 715 (and other locations).

Explanation: Some lines have a mix of spaces and tabs for indentation. We uniformized it to tabs since most of the project uses it.

Bugs & Fixes - Brace location

Warning: *'{' at column 3 should be on the previous line*.

Location: file `ProjectSerializer.java` line 81.

Explanation: the opening brace on this specific for loop is on the next line, which is inconsistent with the rest of the project.

```
// Before fix:
for (Project p:projects)
{
// ---------------------------------------
// After fix:
for (Project p:projects) {
```

Bugs & Fixes - Indentation level in for loop

Warning: *'for' child has incorrect indentation level 18, expected level should be one of the following: 24, 32.*

Location: file `ProjectSerializer.java` lines 82, 84, 85, and 86.

Explanation: the body of this for loop was over-indented using a mixture of tabs and spaces.

```
// Before fix:
for (Project p:projects) {
      startXmlElement(hd,"project",null);
// ---------------------------------------
// After fix:
for (Project p:projects) {
    startXmlElement(hd,"project",null);
```

Bugs & Fixes - Method call indentation level

Warning: *'.' has incorrect indentation level 40, expected level should be 48*.

Location: file `CustomCellRenderer.java` lines 71-76.

Explanation: line wrapping indentation here isn't consistent with the rest of the project.

```
// Before fix:
if (!prj.getNotes().isEmpty()) {
    String tooltip=prj.getNotes()
        .replaceAll("&","&amp;")
        .replaceAll("(\r\n|\n)","<br/>")
        .replaceAll(" "," ")
        .replaceAll("\t","    ");
```

```
// ---------------------------------------
// After fix:
if (!prj.getNotes().isEmpty()) {
    String tooltip=prj.getNotes()
            .replaceAll("&","&amp;")
            .replaceAll("(\r\n|\n)","<br/>")
            .replaceAll(" "," ")
            .replaceAll("\t","    ");
```

Tool - SpotBugs

SpotBugs is a program that uses heuristics to look for faults in Java code.

Configuration

The threshold was set to low, so we can see more warnings/bugs. We didn't install the extra web/mobile-related extensions since they don't fit the app context.

Results

The number of issues reported by the tool:

- **Before fixes**: 60
- **After fixes**: 54

There are numerous warnings related to the instantiation of Integers, Longs, and Booleans, e.g.: `new Integer(1)` (useless boxing). Switch cases are often missing their default case, which generates a warning.

Bugs & Fixes - Ignored return value

Warning: *Exceptional return value of java.io.File.mkdir() ignored in de.dominik_geyer.jtimesched.JTimeSchedApp.main(String[])*.

Location: file `JTimeSchedApp.java` line 57.

```
// Before fix:
if (!dirConf.isDirectory())
    dirConf.mkdir();
// ---------------------------------------
// After fix:
if (!dirConf.isDirectory()){
    if(!dirConf.mkdir()){
        System.err.println("Unable to create configuration directory:
"+JTimeSchedApp.CONF_PATH);
        System.exit(1);
    }
}
```

Bugs & Fixes - Resource Fail

Warnings:

- *de.dominik_geyer.jtimesched.gui.JTimeSchedFrame.backupProjects() may fail to clean up java.io.InputStream on checked exception*. File `JTimeSchedFrame.java`, line 709;
- *de.dominik_geyer.jtimesched.gui.JTimeSchedFrame.backupProjects() may fail to clean up java.io.OutputStream on checked exception*. File

`JTimeSchedFrame.java`, line 710.

```
// Before fix:
FileInputStream fis=null;
FileOutputStream fos=null;

fis=new FileInputStream(file);
fos=new FileOutputStream(new File(JTimeSchedApp.PRJ_FILE_BACKUP));

byte[]buf=new byte[1024];
int i=0;
while ((i=fis.read(buf))!=-1) {
    fos.write(buf,0,i);
}
fis.close();
fos.close();
// ----------------------------------------
// After fix:
try(FileInputStream fis=new FileInputStream(file);
        FileOutputStream fos=new FileOutputStream(new File(JTimeSchedApp.PRJ_FILE_BACKUP))){
    byte[]buf=new byte[1024];
    int i=0;
    while ((i=fis.read(buf))!=-1) {
        fos.write(buf,0,i);
    }
}
```

Although this issue was fixed, the tool continues to report it as an issue. As stated in the documentation, the heuristic used by the tool expects to find a *try/catch* block followed by a *finally* block cleaning up the resource. However, the Oracle recommended way to fix these types of errors is with a *try with resources*. Thus, we ignored further warnings about this bug since it is a **false positive**.

Bugs & Fixes - Inefficient Integer constructor

Warning: *inefficient new Integer(int) constructor; use Integer.valueOf(int) instead*.

Location: file `ProjectSerializer.java` line 66.

This bug might seem like an easy fix. However, next to this line of code, there is a comment with a link mentioning a bug. The link is broken, but we were able to find the bug report here. For this reason, we decided to leave this warning as it is since we are not sure of the consequences (which might be either a **false-positive** or a **correct-positive**).

Bugs & Fixes - Inefficient Boolean

Warning: *inefficient Boolean constructor; use Boolean.valueOf(...) instead*.

Location: file `ProjectTableModel.java` lines 75 and 81.

Explanation: This is a weird statement that is constructing a Boolean object from a boolean for no reason.

```
// Before fix:
o = (prj.isChecked()) ? new Boolean(true) : new Boolean(false);
// ---------------------------------------
// After fix:
o = prj.isChecked();
```

Bugs & Fixes - Inefficient integer

Warning: *inefficient new Integer(int) constructor; use Integer.valueOf(int) instead*.

Location: file `ProjectTableModel.java` lines 84 and 87.

Explanation: Another weird statement similar to the Boolean one before this one.

```
// Before fix:
o = new Integer(prj.getSecondsOverall());
// ---------------------------------------
// After fix:
o = prj.getSecondsOverall();
```

Bugs & Fixes - Missing default case

Warning: *default case is missing*.

Location: file `JTimeSchedFrame.java` line 971.

Explanation: It is nice to always include the default case in switch statements to handle errors/unexpected situations.

```
// Before fix:
switch (keyCode) {
// ...
case KeyEvent.VK_DELETE:
    handleDelete(ptm,p,row);
    e.consume();
    break;
}
// ---------------------------------------
// After fix:
switch (keyCode) {
// ...
case KeyEvent.VK_DELETE:
    handleDelete(ptm,p,row);
    e.consume();
    break;
default:
    // do nothing
    break;
}
```