Student: Intrevado Michele 284789

Report Advanced Verification and Validation

In this project has been possible to verify to inject some profiling probes in the code of already existing Java unit tests. It has been possible by combining different tools:

- Junit
- Java Flight Recorder (JFR)
- JfrUnit
- OpenRewrite

Junit: Junit is a testing framework for developing unit tests in java. The JUnit tests present in open-source repositories will be the starting point for the extension with JfrUnit.

Java Flight Recorder (JFR): JFR is a profiling framework integrated in the Java Virtual Machine (JVM) used to collect and profile all the events generated during the execution of a Java Application by maintaining a low level of overhead (about 2%).

JfrUnit: JfrUnit is a JUnit extension that allows to start a JFR recording while executing a JUnit test and eventually to assert JFR events.

OpenRewrite: OpenRewrite is an open-source library used for large-scale refactoring. This library works by making changes to Lossless Semantic Trees (LST), that are a tree representation of the source code. The changes on the LST are performed by Visitors, that are aggregated into Recipes.

OpenRewrites provides a wide set of recipes, it is also possible to develop a custom refactoring recipe. In particular, has been to inject JfrUnit in already existing JUnit test by developing and combining two custom recipes:

- CompleteJfrUnitInjection (https://github.com/micheleintrevado/JfrUnit-injection-openrewrite-recipe/blob/main/completeJfrUniWorkflow/src/main/resources/META-INF/rewrite/rewrite.yml): this custom recipe is used to perform all the necessary workflow to inject JfrUnit into the JUnit test. In this way has been possible to combine a set of pre-made recipe with my custom recipe (JfrRecipe).
- JfrRecipe (https://github.com/micheleintrevado/JfrUnit-injection-openrewrite-recipe/blob/main/michele-JfrUnit/src/main/java/org/michele/recipe/JfrRecipe.java): this custom recipe has been developed to add the JfrUnit code needed to extend a JUnit test like annotations, statements, etc…

Usage guide

To execute the whole workflow you need to run the following steps:

1) Clone the recipe repository (https://github.com/micheleintrevado/JfrUnit-injection-openrewrite-recipe)
2) Run on the terminal the command “gradlew publishToMavenLocal” in the directory “JfrUnit-injection-openrewrite-recipe/michele-JfrUnit”
3) Run on the terminal the command “gradlew publishToMavenLocal” in the directory “JfrUnit-injection-openrewrite-recipe/completeJfrUniWorkflow”
4) Add the following maven plugin in the in the pom.xml file of the project to refactor:

```
<plugin>
```

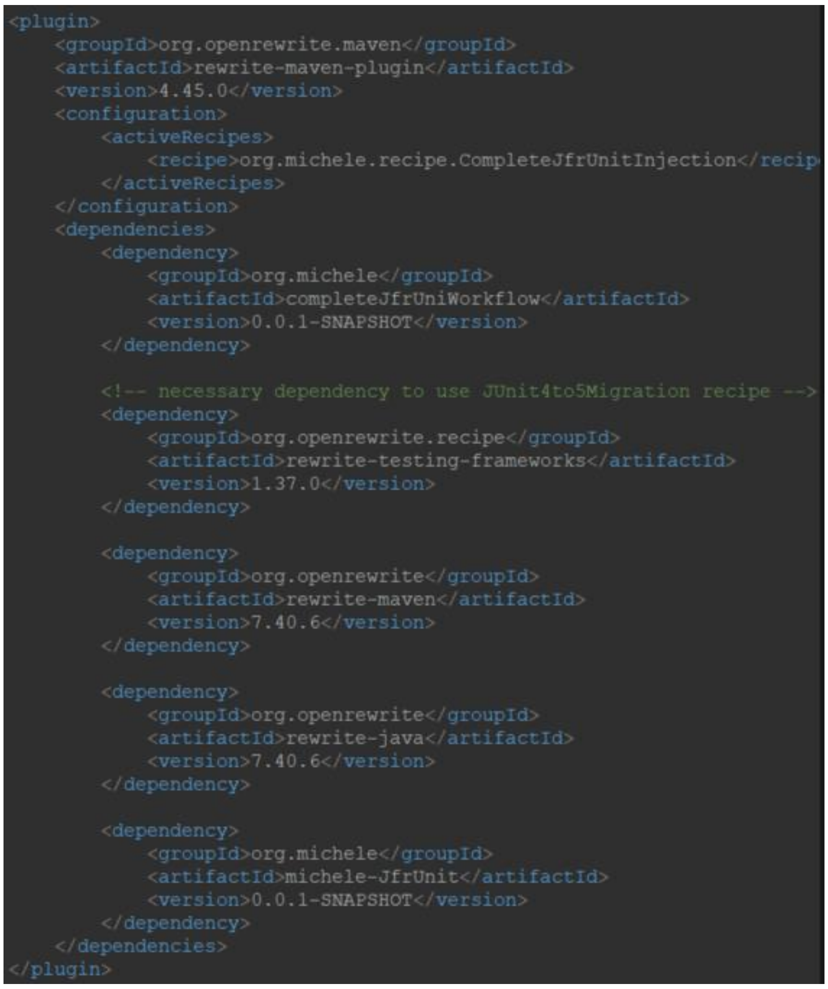
```
<groupId>org.openrewrite.maven</groupId>
<artifactId>rewrite-maven-plugin</artifactId>
<version>4.45.0</version>
<configuration>
<activeRecipes>
    <recipe>org.michele.recipe.CompleteJfrUnitInjection</recipe>
</activeRecipes>
</configuration>
<dependencies>
    <dependency>
        <groupId>org.michele</groupId>
        <artifactId>completeJfrUniWorkflow</artifactId>
        <version>0.0.1-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>org.openrewrite.recipe</groupId>
        <artifactId>rewrite-testing-frameworks</artifactId>
        <version>1.37.0</version>
    </dependency>
    <dependency>
        <groupId>org.openrewrite</groupId>
        <artifactId>rewrite-maven</artifactId>
        <version>7.40.6</version>
    </dependency>
    <dependency>
        <groupId>org.openrewrite</groupId>
        <artifactId>rewrite-java</artifactId>
        <version>7.40.6</version>
    </dependency>
    <dependency>
        <groupId>org.michele</groupId>
```

```
      <artifactId>michele-JfrUnit</artifactId>

      <version>0.0.1-SNAPSHOT</version>

    </dependency>

</dependencies>

</plugin>
```

5) Run on the terminal the command "mvn rewrite:run" in the directory of the project to refactor
6) Run on the terminal the command "mvn clean test" in the directory of the project that I've just refactored

Tested repositories

The refactoring recipe has been tested on different repositories (also with JUnit 4) by simply adding the recipe as plugin in the pom file of the repository that will be tested. E.g.:

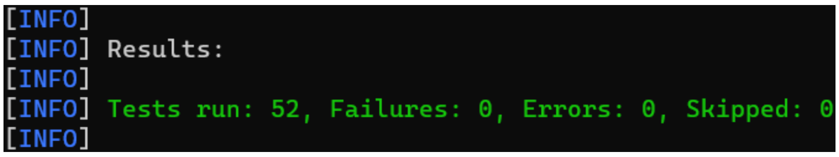


```
<plugin>
    <groupId>org.openrewrite.maven</groupId>
    <artifactId>rewrite-maven-plugin</artifactId>
    <version>4.45.0</version>
    <configuration>
        <activeRecipes>
            <recipe>org.michele.recipe.CompleteJfrUnitInjection</recip
        </activeRecipes>
    </configuration>
    <dependencies>
        <dependency>
            <groupId>org.michele</groupId>
            <artifactId>completeJfrUniWorkflow</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>

        <!-- necessary dependency to use JUnit4to5Migration recipe -->
        <dependency>
            <groupId>org.openrewrite.recipe</groupId>
            <artifactId>rewrite-testing-frameworks</artifactId>
            <version>1.37.0</version>
        </dependency>

        <dependency>
            <groupId>org.openrewrite</groupId>
            <artifactId>rewrite-maven</artifactId>
            <version>7.40.6</version>
        </dependency>

        <dependency>
            <groupId>org.openrewrite</groupId>
            <artifactId>rewrite-java</artifactId>
            <version>7.40.6</version>
        </dependency>

        <dependency>
            <groupId>org.michele</groupId>
            <artifactId>michele-JfrUnit</artifactId>
            <version>0.0.1-SNAPSHOT</version>
        </dependency>
    </dependencies>
</plugin>
```

I obtained the following results:

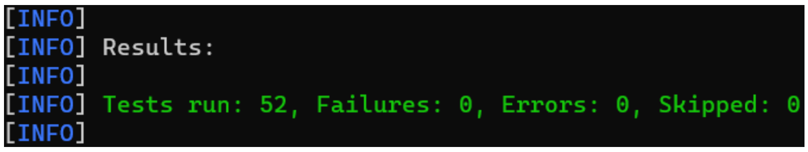
- Repositories that makes use of JUnit 4:
    - **AmazingER** (https://github.com/MigadaTang/AmazingER):

        - Original:



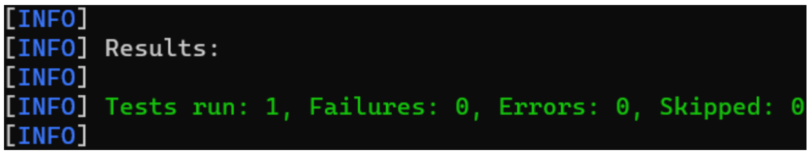
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 52, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

        - Refactored:



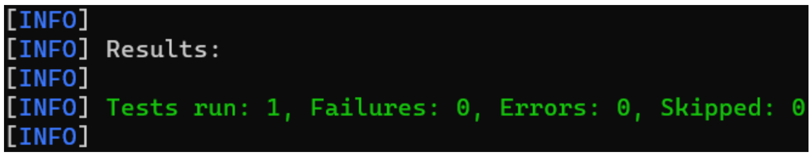
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 52, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

    - **craftColours** (https://github.com/jewlexx/CraftColours):
        - Original:



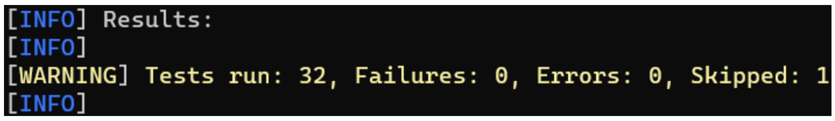
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

        - Refactored:



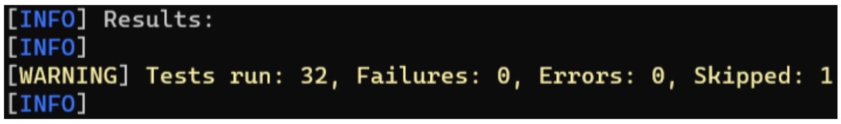
```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

    - **junit-quickcheck** (https://github.com/pholser/junit-quickcheck):
        - Original:



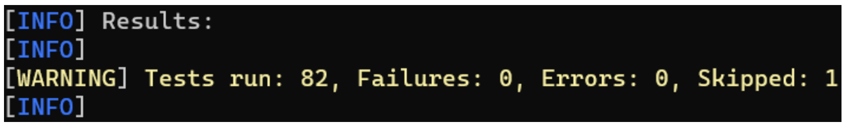
```
[INFO] Results:
[INFO]
[WARNING] Tests run: 32, Failures: 0, Errors: 0, Skipped: 1
[INFO]
```

        - Refactored:



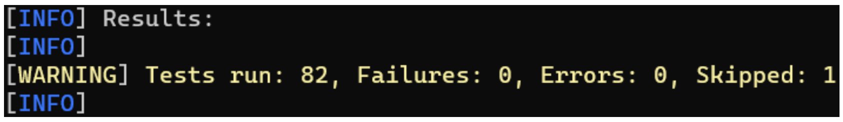
```
[INFO] Results:
[INFO]
[WARNING] Tests run: 32, Failures: 0, Errors: 0, Skipped: 1
[INFO]
```

    - **sonarScannerCli** (https://github.com/SonarSource/sonar-scanner-cli):
        - Original:



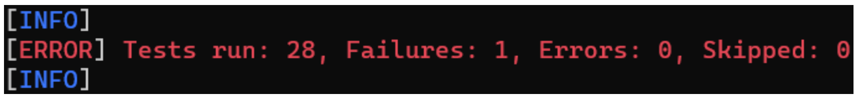
```
[INFO] Results:
[INFO]
[WARNING] Tests run: 82, Failures: 0, Errors: 0, Skipped: 1
[INFO]
```

        - Refactored:



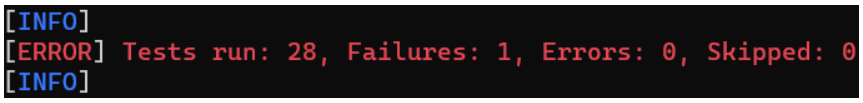
```
[INFO] Results:
[INFO]
[WARNING] Tests run: 82, Failures: 0, Errors: 0, Skipped: 1
[INFO]
```

- Repositories that makes use of JUnit 5:
    - **calculator-java** (https://github.com/lidonis/calculator-java):
        - Original:



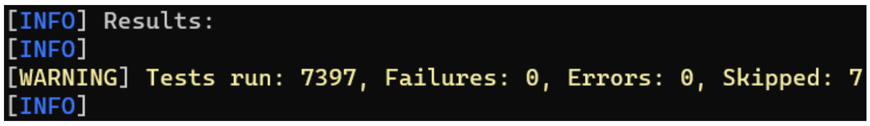
```
[INFO]
[ERROR] Tests run: 28, Failures: 1, Errors: 0, Skipped: 0
[INFO]
```

- Refactored:



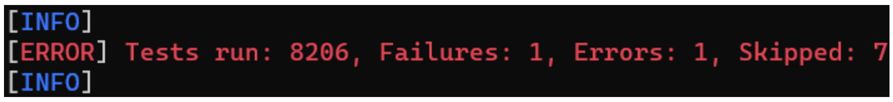
```
[INFO]
[ERROR] Tests run: 28, Failures: 1, Errors: 0, Skipped: 0
[INFO]
```

- **commons-lang** (https://github.com/apache/commons-lang):
  - Original:



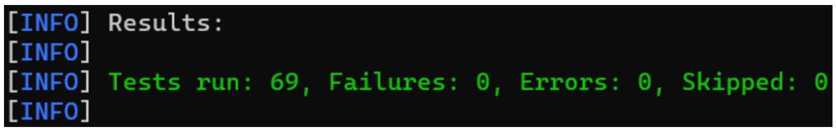
```
[INFO] Results:
[INFO]
[WARNING] Tests run: 7397, Failures: 0, Errors: 0, Skipped: 7
[INFO]
```

  - Refactored:



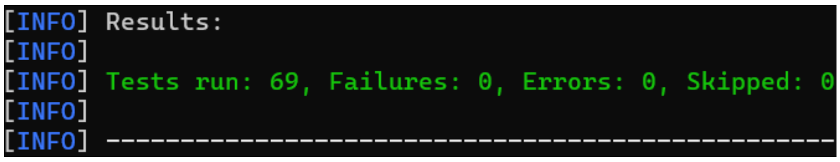
```
[INFO]
[ERROR] Tests run: 8206, Failures: 1, Errors: 1, Skipped: 7
[INFO]
```

- **lambda** (https://github.com/alexengrig/lambda):
  - Original:



```
[INFO] Results:
[INFO]
[INFO] Tests run: 69, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```

  - Refactored:



```
[INFO] Results:
[INFO]
[INFO] Tests run: 69, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------
```

Finally, it's possible to say that the initial objective of the project has been achieved, in the different study cases previously reported, despite the limitations of the various tools (e.g.: JfrUnit needs of the explicit access modifier to be "public" in the test method).