Static testing

October 4th, 2023
Ana Paiva, José Campos

In this recitation class, we are going to explore the following tools that allow one to perform *static testing* on Java projects:

- Checkstyle
- PMD
- SpotBugs

Please make sure your machine is configured properly, i.e.:

- Java is installed on your machine and available through the command line. Disclaimer: this tutorial has been validated under Java-11. It may or may not work on other versions of Java. Let us know whether it does not work under Java-X, where X is a version higher than 11.
- Apache Maven is installed on your machine and available through the command line. In case Maven is not installed, please follow the following steps:
    - Download `apache-maven-3.9.4-bin.zip`
    - Extract `apache-maven-3.9.4-bin.zip`
    - On Windows, augment your environment variables with the full path to the `<extracted directory>/bin`. On Linux/MacOS, run `export PATH="<extracted directory>/bin:$PATH"`. (You might have to run the `export` everytime you restart the computer. For a more permanent solution, please consider adding that command to your bash profile.)

1. Get the `jpacman` game's source code

`jpacman` (available in here) is a Java version of the well-known game Pacman.

2. Run/Play `jpacman`

- Unzip the given `recitation-1-jpacman.zip` file.

2.1 Through the command line

```
Unset
# Navigate to the project's directory
```

```
cd <project's directory>

# Build the project (and skip the execution of tests for now)
mvn package -DskipTests

# Run it
java -cp target/jpacman-framework-8.1.1.jar
nl.tudelft.jpacman.Launcher

# Enjoy. :-)
```

2.2 Through an IDE

- Open your favourite IDE, e.g., Eclipse / IntelliJ IDEA / VSCode.
- Import the `jpacman` project.
- Compile the project, in case the IDE does not automatically compile the project's source code.
- Run the `nl.tudelft.jpacman.Launcher` class.
- Enjoy. :-)

3. Static testing

3.1 Checkstyle

"Checkstyle is a static code analysis tool used in software development for checking if Java source code is compliant with specified coding rules."

Open the `pom.xml` file (which is located in the root directory of the project under test) and configure Checkstyle. To achieve that, (1) add the following content to the `<build>` section:

```
Unset
<build>
 <plugins>
  ...

  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
```

```
   <artifactId>maven-checkstyle-plugin</artifactId>
   <version>3.2.0</version>
   <configuration>

<configLocation>${basedir}/rulesets/checkstyle-rules.xml</configL
ocation>
   </configuration>
   <dependencies>
    <dependency>
     <groupId>com.puppycrawl.tools</groupId>
     <artifactId>checkstyle</artifactId>
     <version>10.3.3</version>
    </dependency>
   </dependencies>
  </plugin>

  ...
 </plugins>
</build>
```

and (2) add the following lines to the `<reporting>` section:

Unset

```
<reporting>
 <plugins>
  ...

  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-checkstyle-plugin</artifactId>
   <version>3.2.0</version>
   <configuration>

<configLocation>${basedir}/rulesets/checkstyle-rules.xml</configL
ocation>
   </configuration>
```

```
  </plugin>

  ...
 </plugins>
</reporting>
```

In a nutshell, this will inform the build system (in this case maven) which plugin should be used, its version, and the location of the set of style rules to be checked. Note: the `rulesets/checkstyle-rules.xml` contains the styling rules suggested by Google.

Finally,

- Run `mvn site` to run Checkstyle and produce a report. (Note: in case the `mvn site` command does not run successfully, try `mvn checkstyle:checkstyle` instead.)
- Open `target/site/checkstyle.html` in your favourite browser and inspect the report, e.g., how many warnings and errors are reported overall and per java file? What family of warnings/errors?

So far, we have used the styling rules suggested by Google which might not be inline with our project's rules. For instance, according to the report produced by Checkstyle, most of the reported warnings are related to "indentation". Let's adapt those rules to the project under test.

- Open the `rulesets/checkstyle-rules.xml` file.
- Adapt the `<module name="Indentation">` group according to the project under test's indentation rules. Tip: you must first identify which indentation levels are used in the project under test and then adapt the rules.
- Re-run `mvn site`. (Note: in case the `mvn site` command does not run successfully, try `mvn checkstyle:checkstyle` instead.)
- Open `target/site/checkstyle.html` in your favourite browser and inspect the report. At this point, you should see fewer warnings. If not, re-do these four steps.

3.2 PMD

"PMD is an open source static source code analyzer that reports on issues found within application code. PMD includes built-in rule sets and supports the ability to write custom rules."

Open the `pom.xml` file (which is located in the root directory of the project under test) and configure PMD. To achieve that, (1) add the following content to the `<build>` section:

```
Unset
<build>
 <plugins>
  ...

  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-pmd-plugin</artifactId>
   <version>3.19.0</version>
   <configuration>
    <rulesets>
     <ruleset>${basedir}/rulesets/pmd-basic.xml</ruleset>
     <ruleset>${basedir}/rulesets/pmd-quickstart.xml</ruleset>
    </rulesets>
   </configuration>
  </plugin>

  ...
 </plugins>
</build>
```

and (2) add the following lines to the `<reporting>` section:

```
Unset
<reporting>
 <plugins>
  ...

  <plugin>
   <groupId>org.apache.maven.plugins</groupId>
   <artifactId>maven-pmd-plugin</artifactId>
   <version>3.19.0</version>
   <configuration>
    <rulesets>
     <ruleset>${basedir}/rulesets/pmd-basic.xml</ruleset>
     <ruleset>${basedir}/rulesets/pmd-quickstart.xml</ruleset>
```

```
    </rulesets>
   </configuration>
  </plugin>

  ...
 </plugins>
</reporting>
```

In a nutshell, this will inform the build system (in this case maven) which plugin should be used, its version, and the location of the set of rules to be checked.

Finally,

- Run `mvn site` to run PMD and produce a report. (Note: in case the `mvn site` command does not run successfully, try `mvn pmd:pmd` instead.)
- Open `target/site/pmd.html` in your favourite browser and inspect the report, e.g., how many violations are reported? What type of violations? Do you think those are true violations that need to be addressed, or are just false positives?

In the `<rulesets>` group you can define as many `<ruleset>` items as you like. Under the `rulesets/` directory (in the root directory of the project under test) there are many other rules you could instantiate PMD with. Lets augment the set of rules already defined in the `pom.xml` file with all the rules under the `rulesets/` directory. In other words,

- Open the `pom.xml` file (which is located in the root directory of the project under test).
- Replace the following lines

```
Unset
<rulesets>
 <ruleset>${basedir}/rulesets/pmd-basic.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-quickstart.xml</ruleset>
</rulesets>
```

with

Unset

```
<rulesets>
 <ruleset>${basedir}/rulesets/pmd-unusedcode.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-basic.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-braces.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-clone.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-codesize.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-comments.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-controversial.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-coupling.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-design.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-empty.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-finalizers.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-imports.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-j2ee.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-javabeans.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-junit.xml</ruleset>

<ruleset>${basedir}/rulesets/pmd-logging-jakarta-commons.xml</rul
eset>
 <ruleset>${basedir}/rulesets/pmd-logging-java.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-metrics.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-migrating_to_13.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-migrating_to_14.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-migrating_to_15.xml</ruleset>

<ruleset>${basedir}/rulesets/pmd-migrating_to_junit4.xml</ruleset
>
 <ruleset>${basedir}/rulesets/pmd-migrating.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-naming.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-optimizations.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-quickstart.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-strictexception.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-strings.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-sunsecure.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-typeresolution.xml</ruleset>
 <ruleset>${basedir}/rulesets/pmd-unnecessary.xml</ruleset>
```

```
</rulesets>
```

- Re-run `mvn site`. (Note: in case the `mvn site` command does not run successfully, try `mvn pmd:pmd` instead.)
- Open `target/site/pmd.html` in your favourite browser and inspect the report. At this point, you should see many more violations. If not, re-do the steps described above. Once more, how many violations are reported? What type of violations? Do you think those are true violations that need to be addressed, or are just false positives?

Note that PMD also reports duplicated code. The report regarding duplicated code is in `target/site/cpd.html`, which you could open in your favourite browser.

3.3 SpotBugs

"SpotBugs is a program which uses static analysis to look for *bugs* in Java code."

Open the `pom.xml` file (which is located in the root directory of the project under test) and configure SpotBugs. To achieve that, (1) add the following content to the `<build>` section:

```
Unset
<build>
 <plugins>
  ...

  <plugin>
   <groupId>com.github.spotbugs</groupId>
   <artifactId>spotbugs-maven-plugin</artifactId>
   <version>4.7.2.0</version>
  </plugin>

  ...
 </plugins>
</build>
```

and (2) add the following lines to the `<reporting>` section:

```
Unset
<reporting>
 <plugins>
  ...

  <plugin>
   <groupId>com.github.spotbugs</groupId>
   <artifactId>spotbugs-maven-plugin</artifactId>
   <version>4.7.2.0</version>
  </plugin>

  ...
 </plugins>
</reporting>
```

In a nutshell, this will inform the build system (in this case maven) which plugin should be used, its version, and where are the set of rules to be checked.

Finally,

- Run `mvn site` to run SpotBugs and produce a report. (Note: in case the `mvn site` command does not run successfully, try `mvn spotbugs:spotbugs` instead.)
- Open `target/site/spotbugs.html` in your favourite browser and inspect the report, e.g., how many *bugs* are reported? What type of categories are reported? What is the priority of each reported *bug*? Do you think those are true *bugs* that need to be addressed, or are just false positives?

SpotBugs can also be configured to the project under test. For instance, let's assume we only want SpotBugs to report *bugs* with *High* priority/confidence.

- Open the `pom.xml` file (which is located in the root directory of the project under test).
- Add the following lines to the two definitions of the plugin.

```
Unset
<configuration>

<includeFilterFile>rulesets/spotbugs-rules.xml</includeFilterFile
>
</configuration>
```

The `rulesets/spotbugs-rules.xml` defines the following rule

```
Unset
<Match>
 <Confidence value="1"/>
</Match>
```

which means that only *bugs* with a priority/confidence of 1 (*High*) would be reported by the tool.

- Re-run `mvn site`. (Note: in case the `mvn site` command does not run successfully, try `mvn spotbugs:spotbugs` instead.)
- Open `target/site/spotbugs.html` in your favourite browser and inspect the report. At this point, you should see fewer *bugs*. If not, re-do the steps described above. Once more, how many *bugs* are reported? What type of categories are reported? Do you think those are true *bugs* that need to be addressed, or are just false-positives?

Note that SpotBugs also has a GUI. To run it, execute the following command:

```
Unset
mvn spotbugs:gui
```

4. Exercise

1. Randomly select 5 of the many *bugs* reported by two of three static testing tools configured above. Note that we are interested in “correct positive” (i.e., a warning that is a true *bug*) and “false positive” (warning that is not a true *bug*) *bugs*.
2. Address/Fix those 5x2 *bugs*. Note that there are essential three ways to address/fix an issue reported by a static tool:
    - Disable specific rule(s). For example, if a rule X that is enabled by default does not apply to your project’s context, perhaps it could be disabled.
    - Adapt the rules of the tool to not report that issue. For example, if a project uses 4 spaces to indent code but the tool is, by default, configured to check whether 2 spaces are used, then, the rules should be adapted but not the source code. If a project uses a mix of 2 and 4 spaces, then you should first consistently use a number of spaces in the project and then configure the tool to check that specific number of spaces.
    - Modify the source code so that the issue / warning / violation is no longer reported.
3. (re-)Perform static testing and produce a new *test* report.

5. What should you submit/deliver?

Zip the project's directory and submit it here (M.EIC's moodle) or here (MESW's moodle).

Deadline: October 4, 2023, 11:59:00 pm.
Grades: available on October ~~11~~ 16, 2023.

Miscellaneous

- Guide to Configuring Maven Plug-ins
- Apache Maven Checkstyle Plugin
- Apache Maven PMD Plugin
- Using the SpotBugs Maven Plugin