Assignment #1

Group Information

- Degree: M.EIC
- Group id: 06
- Project id: 03
- Students:
  - João Matos, up201703884
  - Tiago Gomes, up201806658

Project description

What is it?

The jdotxt project is an implementation of a todo application. It contains a GUI where it is possible to insert new tasks and marked them as completed. The information is saved in a text file in a location chosen by the user.

How is the source-code organized?

The project contains two packages at the root of the source code:

com.chschmid.jdotxt

This package contains the entrypoint of the application along with 2 other packages. One with utilities and the other with the code responsible for building the GUI. The latter has its own subpackages with more utilities and code for the controls of the UI.

com.todotxt.todotxttouch

This package contains some constants and exception related code at its root. Furthermore, it contains 2 subpackages, one for utilities and another for task related code. This code is responsible for parsing, filtering and sorting (in its own package) the tasks created by the user.

What static testing is and why it is import and relevant?

Static testing is a technique which allows to check for faults in a software application and associated documents without requiring it to be executed, enabling faster feedback. This technique is usually used to find faults in the code which could make it less maintainable.

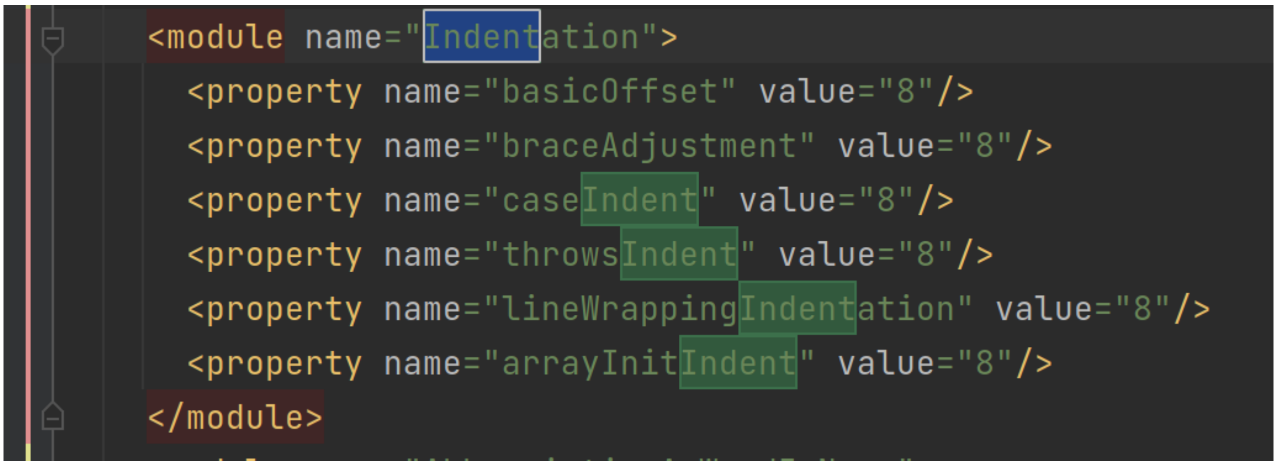
Static testing tools used

CheckStyle

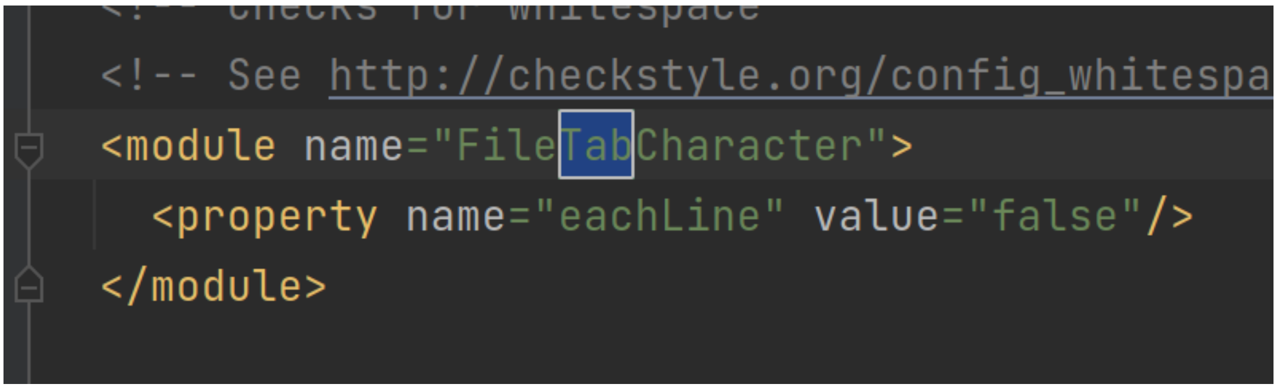
This tool is used to make sure the Java code in a project follows the styling rules that have been defined.

Configuration

To use CheckStyle in the project, we executed the following steps:

- Add the CheckStyle configuration to the pom.xml file, as explained in the Recitation #1;
- Create the file `checkstyle-rules.xml` on the `rulesets` folder;
- Change the following configurations on the `checkstyle-rules.xml` file:
  - 

    ```
    <module name="Indentation">
      <property name="basicOffset" value="8"/>
      <property name="braceAdjustment" value="8"/>
      <property name="caseIndent" value="8"/>
      <property name="throwsIndent" value="8"/>
      <property name="lineWrappingIndentation" value="8"/>
      <property name="arrayInitIndent" value="8"/>
    </module>
    ```

    - The indentation values were changed to better match those used by the project;
  - 

    ```
    <!-- Checks for Whitespace
    <!-- See http://checkstyle.org/config_whitespa
    <module name="FileTabCharacter">
      <property name="eachLine" value="false"/>
    </module>
    ```

    - By default, CheckStyle does not allow the use of tabs, this was changed since the project does make use of tabs.

Report summary

The report emitted by Checkstyle is composed by the summary, files, rules and details sections.

In the summary section, it is possible to see the number of files, number of warnings and number of errors. In the case of the jdotxt project, there are 68 files, 12481 warnings and 0 errors.

The files section exposes a list with more information about each file, namely the number of warnings and number of errors.

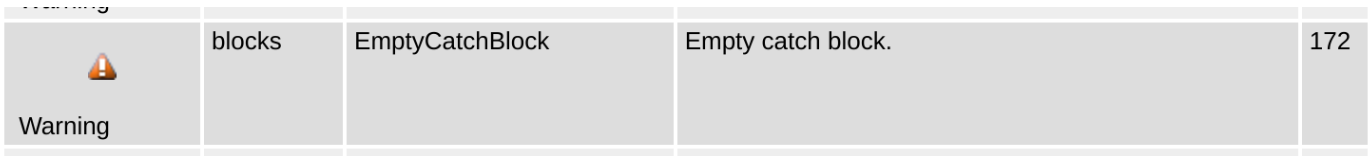
The section about the rules has information about the rules used by Checkstyle, showing the number of violations and the severity of each one. For the jdotxt project, the rules with the highest number of violations are the *Indentation* and the *FileTabCharacter*.

Lastly, the details section shows every warning and error, with a message and its location.

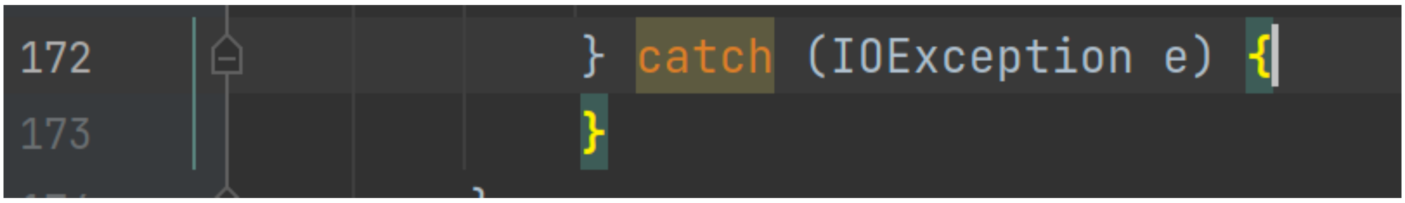
Bugs solved

Warning *EmptyCatchBlock*

CheckStyle was throwing a warning because of the empty catch blocks present on the source code.

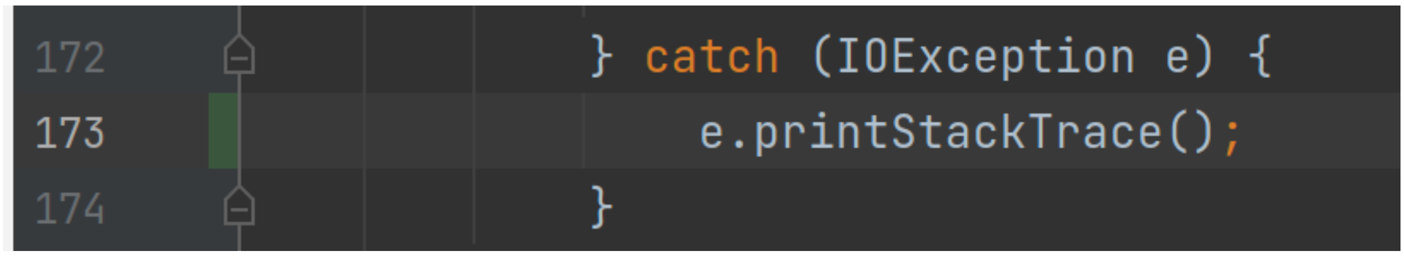


⚠ Warning	blocks	EmptyCatchBlock	Empty catch block.	172



```
        } catch (IOException e) {
        }
```

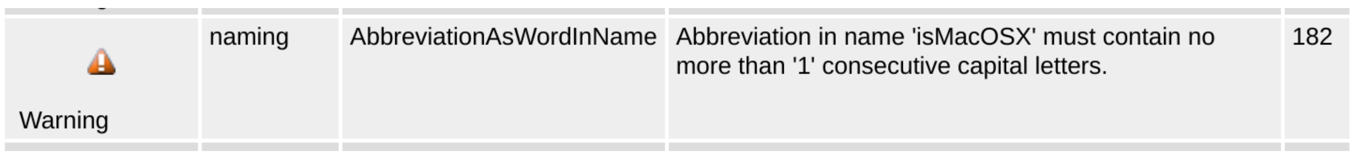
To fix this warning, we decided to add a line which prints the stack trace. This way, when an exception is catched, the programmer can better understand the source of the problem.



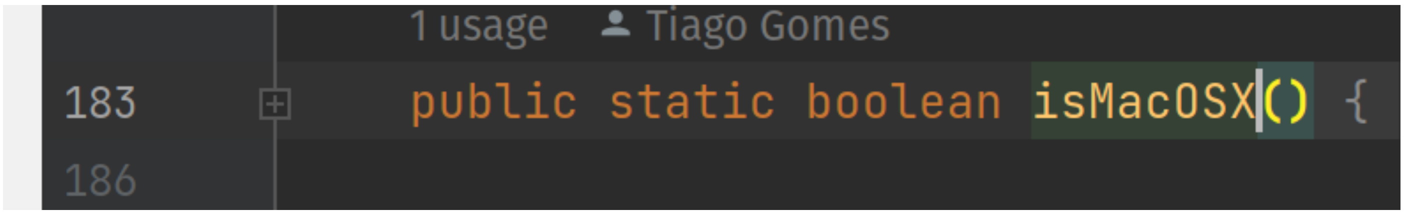
```
        } catch (IOException e) {
            e.printStackTrace();
        }
```

Warning *AbbreviationAsWordInName*

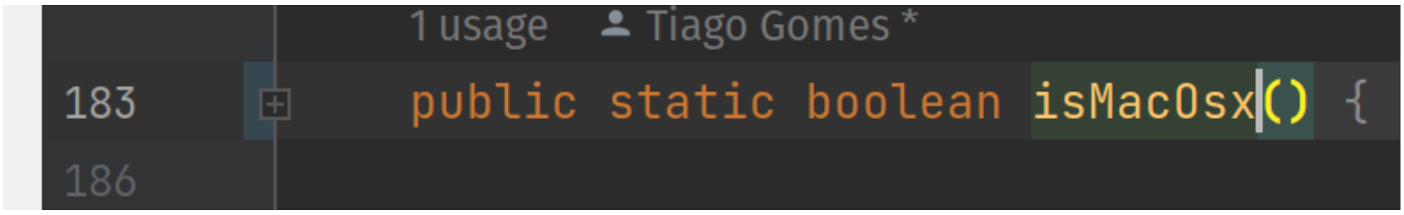
CheckStyle alerted us to the existence of a function with a name that didn't follow the naming convention properly and had 2 consecutive capital letters.



⚠ Warning	naming	AbbreviationAsWordInName	Abbreviation in name 'isMacOSX' must contain no more than '1' consecutive capital letters.	182



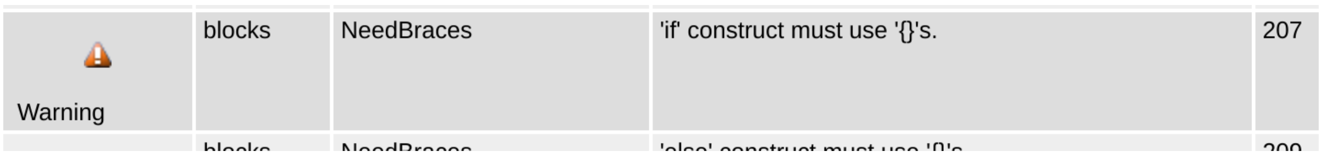
```
1 usage  Tiago Gomes
public static boolean isMacOSX() {
```



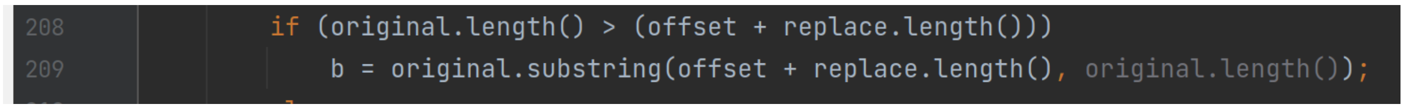
```
1 usage  Tiago Gomes *
public static boolean isMacOsx() {
```

Warning *NeedBraces*

CheckStyle presented a warning about the use of braces ({}) on the if constructs.

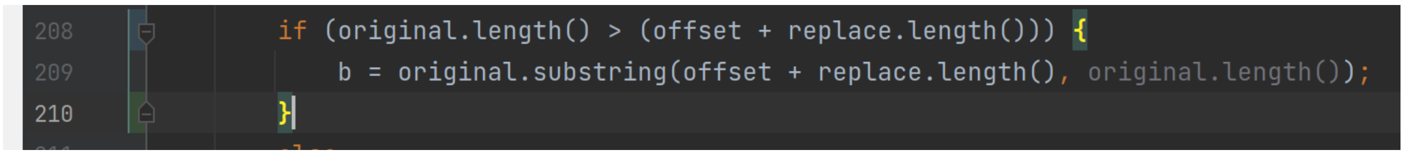


⚠ Warning	blocks	NeedBraces	'if' construct must use '{}'s.	207



```
    if (original.length() > (offset + replace.length()))
        b = original.substring(offset + replace.length(), original.length());
```

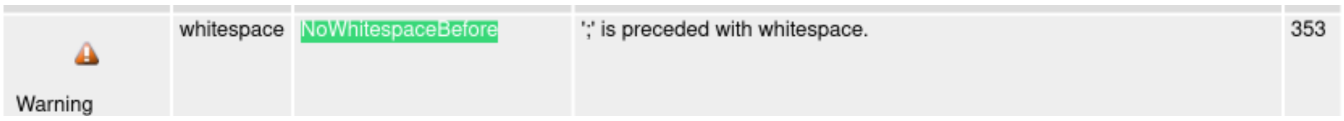
To fix this warning, we added braces on the corresponding if constructs.



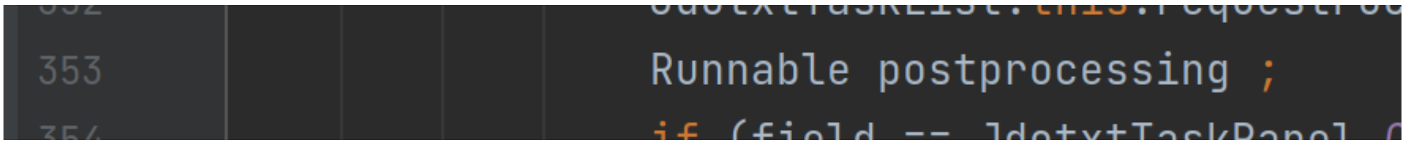
```
    if (original.length() > (offset + replace.length())) {
        b = original.substring(offset + replace.length(), original.length());
    }
```

Warning *NoWhitespaceBefore*

CheckStyle was throwing a warning because of the whitespace before the ";".

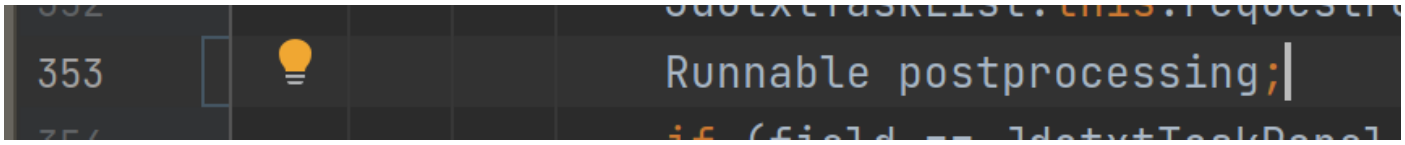


⚠ Warning	whitespace	NoWhitespaceBefore	';' is preceded with whitespace.	353



```
Runnable postprocessing ;
```

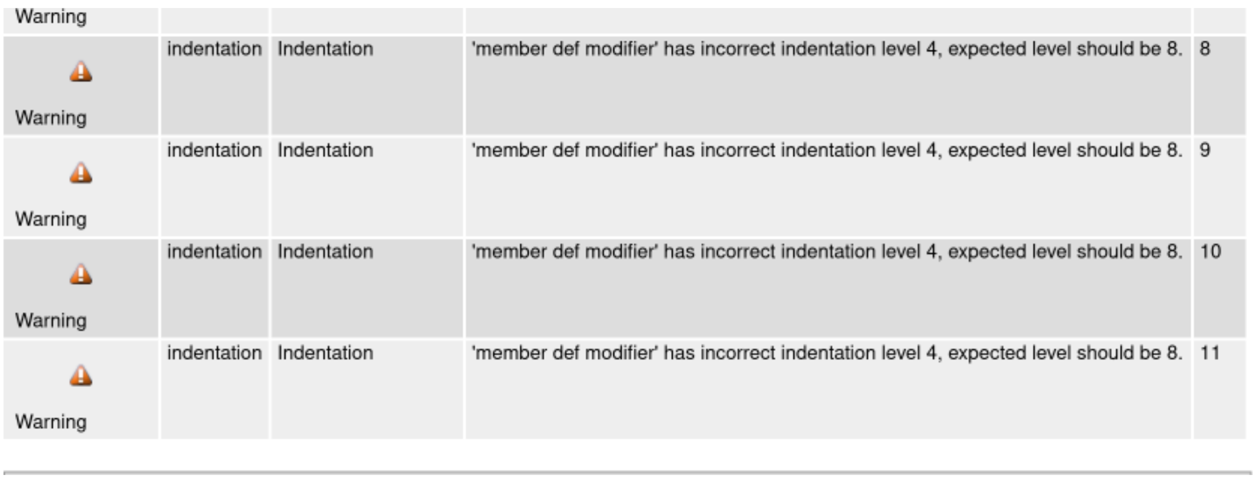
To fix this issue, we removed the whitespace.



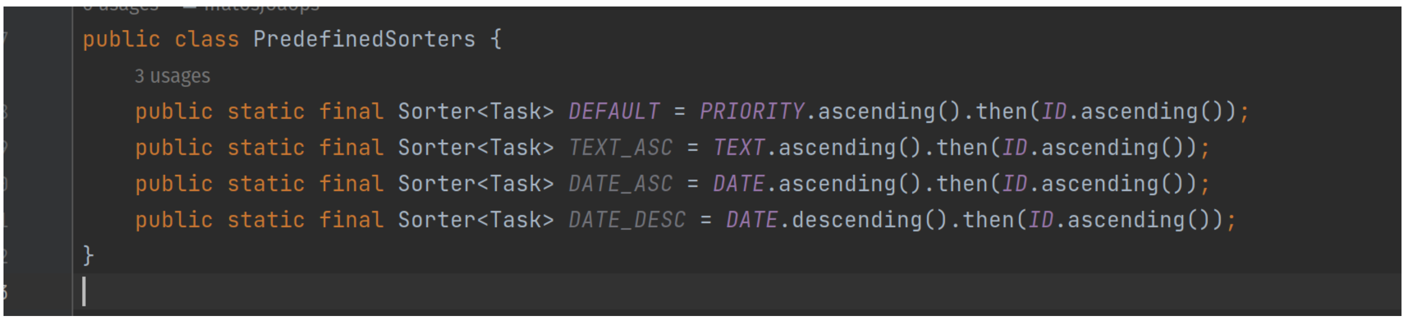
```
Runnable postprocessing;
```

Warning *Indentation*

CheckStyle alerted us about the value of the indentation: it should be 8 instead of 4. We could change the ruleset to fix this problem, but as the majority of the source code was using an indentation level of 8, we decided to maintain it.

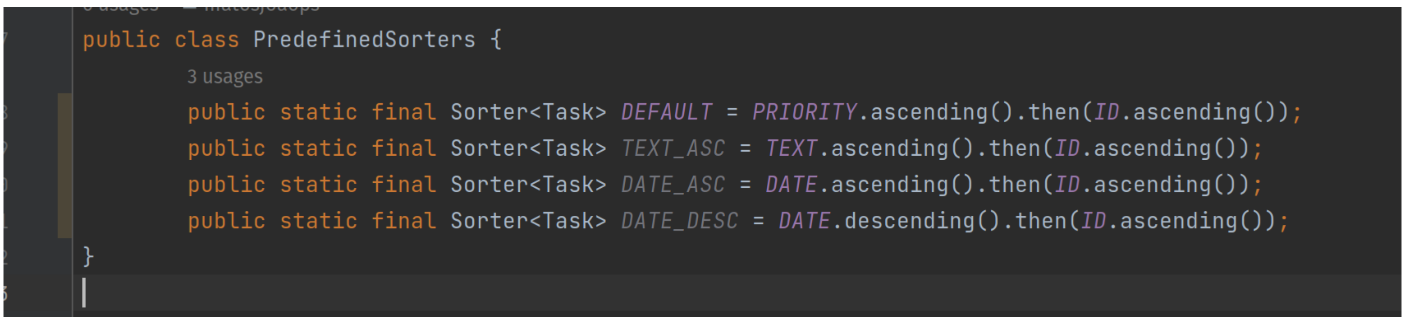


Warning				
⚠ Warning	indentation	Indentation	'member def modifier' has incorrect indentation level 4, expected level should be 8.	8
⚠ Warning	indentation	Indentation	'member def modifier' has incorrect indentation level 4, expected level should be 8.	9
⚠ Warning	indentation	Indentation	'member def modifier' has incorrect indentation level 4, expected level should be 8.	10
⚠ Warning	indentation	Indentation	'member def modifier' has incorrect indentation level 4, expected level should be 8.	11



```
public class PredefinedSorters {
    3 usages
    public static final Sorter<Task> DEFAULT = PRIORITY.ascending().then(ID.ascending());
    public static final Sorter<Task> TEXT_ASC = TEXT.ascending().then(ID.ascending());
    public static final Sorter<Task> DATE_ASC = DATE.ascending().then(ID.ascending());
    public static final Sorter<Task> DATE_DESC = DATE.descending().then(ID.ascending());
}
```

To fix this issue, we changed the indentation level from 4 to 8.



```
public class PredefinedSorters {
        3 usages
        public static final Sorter<Task> DEFAULT = PRIORITY.ascending().then(ID.ascending());
        public static final Sorter<Task> TEXT_ASC = TEXT.ascending().then(ID.ascending());
        public static final Sorter<Task> DATE_ASC = DATE.ascending().then(ID.ascending());
        public static final Sorter<Task> DATE_DESC = DATE.descending().then(ID.ascending());
}
```

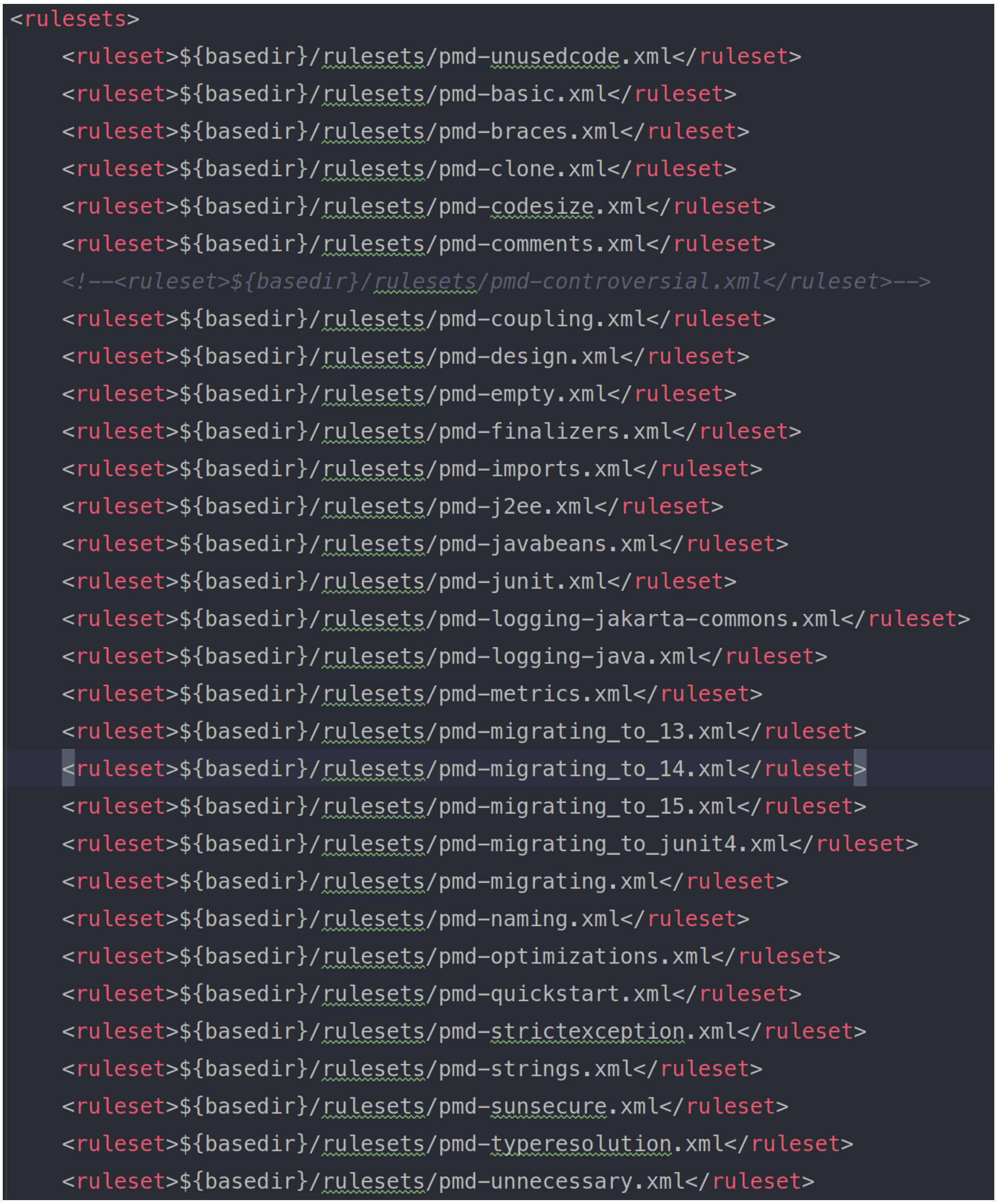
PMD

PMD is a tool used to report issues on the source code of an application. It has predefined rules and gives the possibility to costumize the ruleset.

Configuration

To use PMD in the project, we executed the following steps

- Add the PMD configuration to the pom.xml file, as explained in the Recitation #1;

- Copy the PMD ruleset files used in the class to the `rulesets` folder;
- Change the plugin configuration in order to enable and disable some rulesets:
    - 

```xml
<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</ruleset>
    <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>
```

        - Controversial rules were disabled since some of them did not make sense and were even marked as deprecated by PMD's own documentation.

Report summary

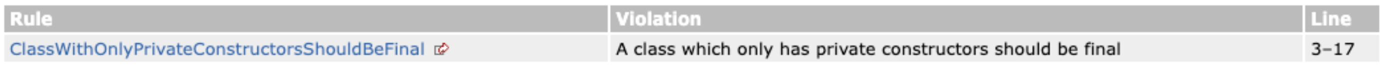
Before applying our custom configuration, the original report contained multiple violations with priority 1. The ones with the most violations were *AvoidUsingShortType* and *VariableNamingConventions*. For priority 2, we mostly see violations related to *SystemPrintln* and

AvoidReassigningParameters but we didn't see a large amount of violations with this priority level. For priority levels 3-5, there is a large variety of violations being reported across the entire project. The report also contains the file, description and line where each violation is detected.

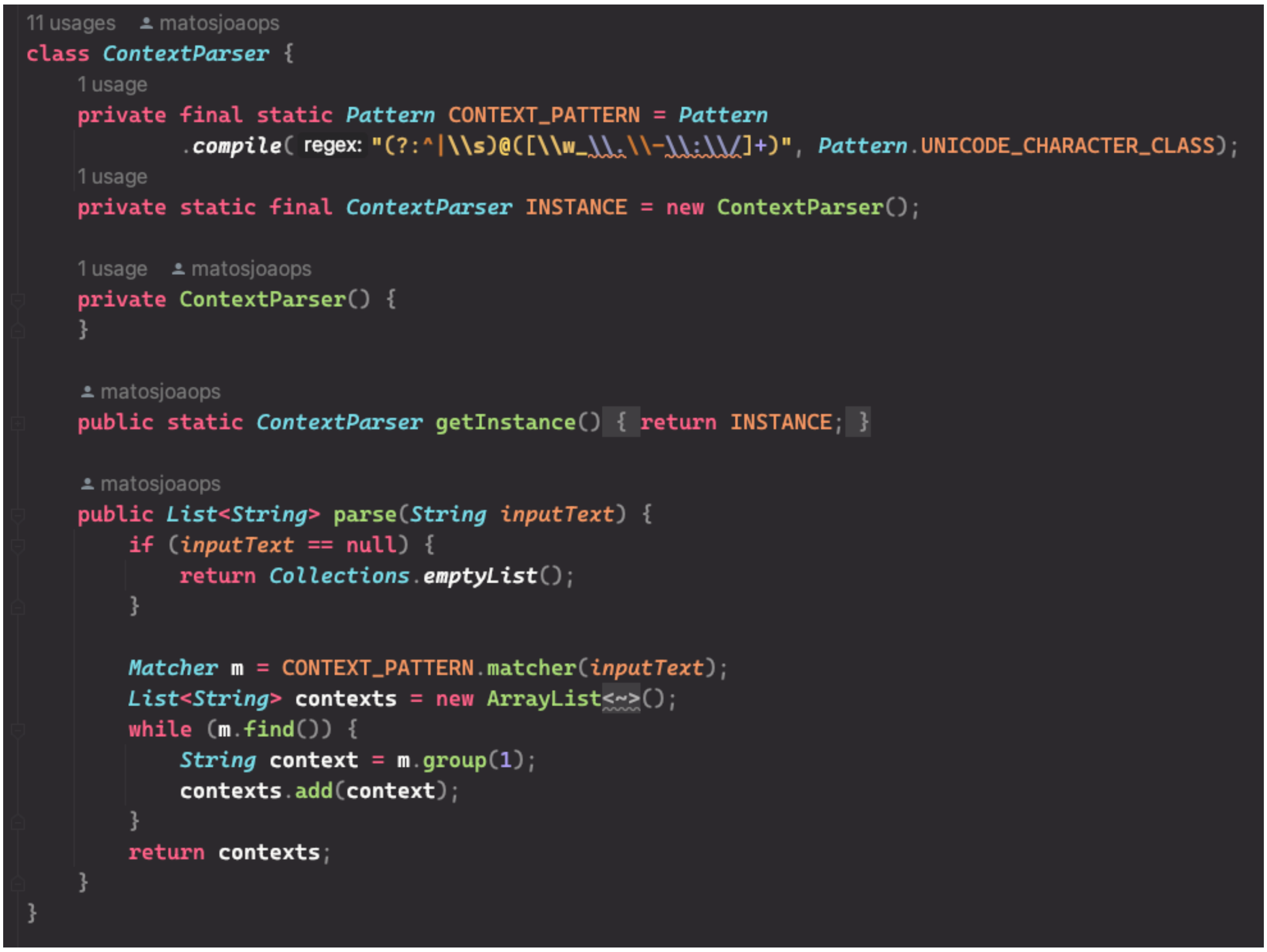
Bugs solved

Warning *ClassWithOnlyPrivateConstructorsShouldBeFinal*

PMD was throwing a warning because of a class which only has private constructors. In this case, the class should be marked as final.

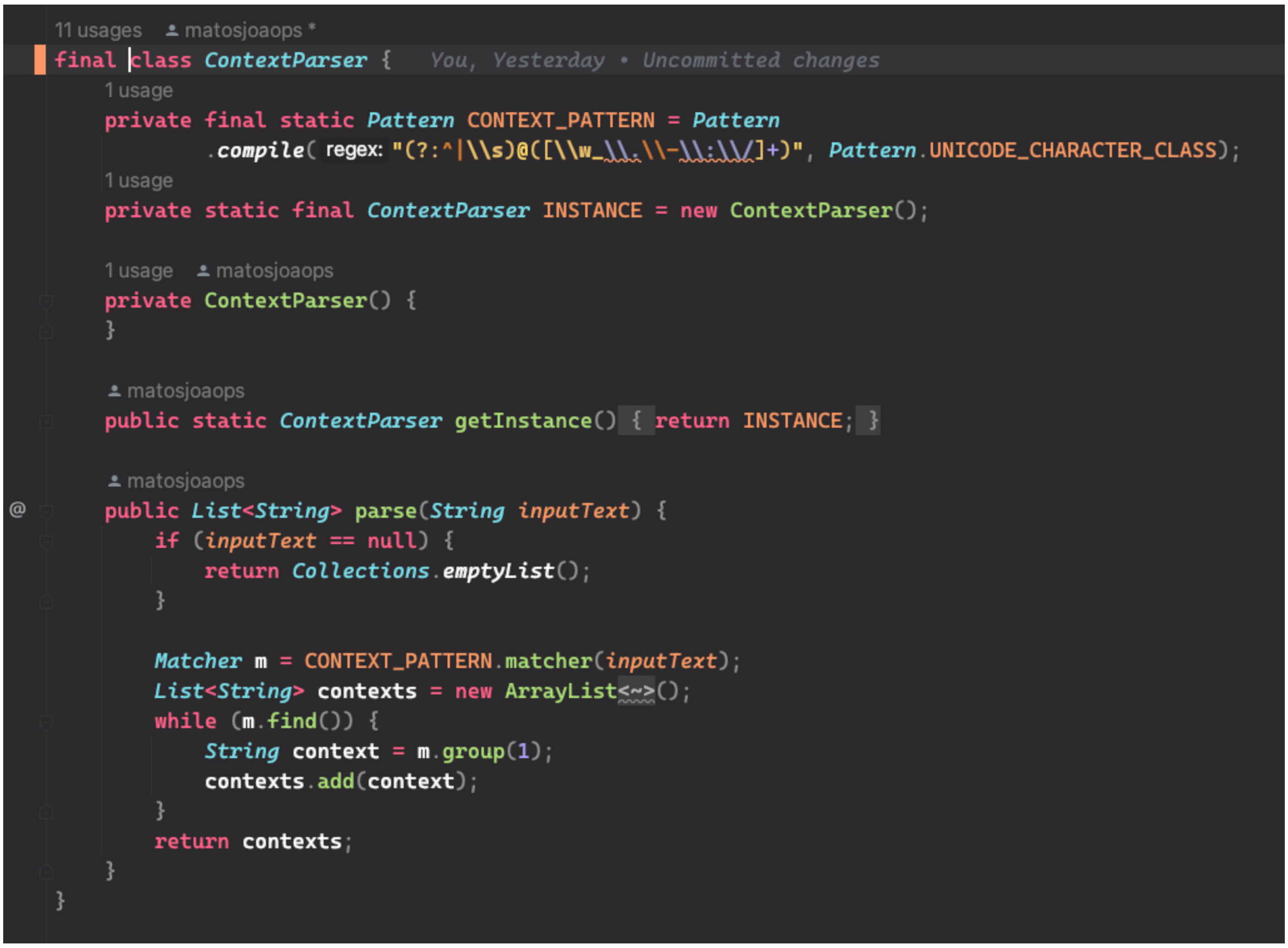


Rule	Violation	Line
ClassWithOnlyPrivateConstructorsShouldBeFinal	A class which only has private constructors should be final	3–17



```
11 usages   matosjoaops
class ContextParser {
    1 usage
    private final static Pattern CONTEXT_PATTERN = Pattern
            .compile( regex: "(?:^|\\s)@([\\w_\\.\\-\\:\\/]+)", Pattern.UNICODE_CHARACTER_CLASS);
    1 usage
    private static final ContextParser INSTANCE = new ContextParser();

    1 usage   matosjoaops
    private ContextParser() {
    }

    matosjoaops
    public static ContextParser getInstance() { return INSTANCE; }

    matosjoaops
    public List<String> parse(String inputText) {
        if (inputText == null) {
            return Collections.emptyList();
        }

        Matcher m = CONTEXT_PATTERN.matcher(inputText);
        List<String> contexts = new ArrayList<~>();
        while (m.find()) {
            String context = m.group(1);
            contexts.add(context);
        }
        return contexts;
    }
}
```

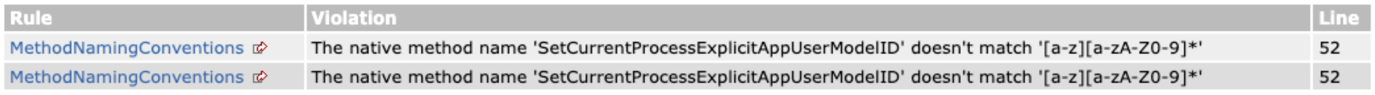
To fix this warning, we added the final keyword to the class.



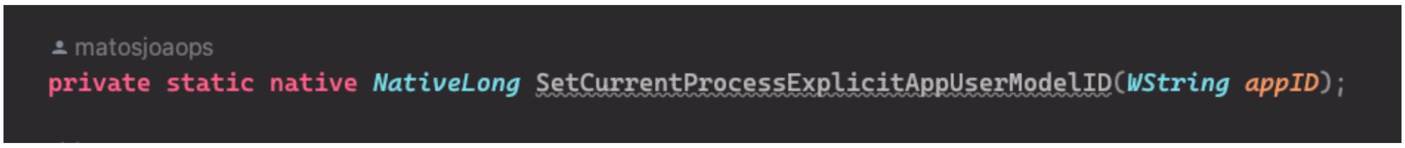
```
11 usages  matosjoaops *
final class ContextParser {    You, Yesterday • Uncommitted changes
    1 usage
    private final static Pattern CONTEXT_PATTERN = Pattern
            .compile( regex: "(?:^|\\s)@([\\w_\\.\\-\\:\\/]+)", Pattern.UNICODE_CHARACTER_CLASS);
    1 usage
    private static final ContextParser INSTANCE = new ContextParser();

    1 usage  matosjoaops
    private ContextParser() {
    }

    matosjoaops
    public static ContextParser getInstance() { return INSTANCE; }

    matosjoaops
@   public List<String> parse(String inputText) {
        if (inputText == null) {
            return Collections.emptyList();
        }

        Matcher m = CONTEXT_PATTERN.matcher(inputText);
        List<String> contexts = new ArrayList<~>();
        while (m.find()) {
            String context = m.group(1);
            contexts.add(context);
        }
        return contexts;
    }
}
```

Warning *MethodNamingConventions*

PMD was throwing a warning because a method was not following camel case convention.

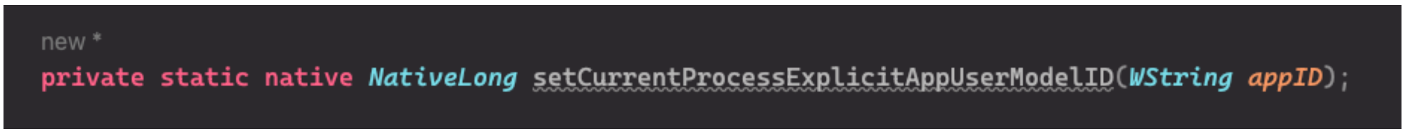


Rule	Violation	Line
MethodNamingConventions	The native method name 'SetCurrentProcessExplicitAppUserModelID' doesn't match '[a-z][a-zA-Z0-9]*'	52
MethodNamingConventions	The native method name 'SetCurrentProcessExplicitAppUserModelID' doesn't match '[a-z][a-zA-Z0-9]*'	52



```
matosjoaops
private static native NativeLong SetCurrentProcessExplicitAppUserModelID(WString appID);
```

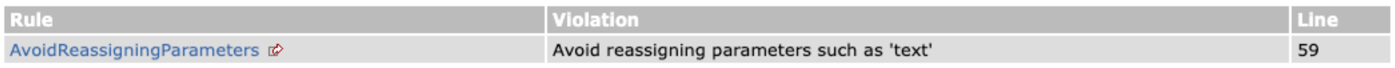
To fix this warning, we added the changed the first character of the method name from upper to lower case.



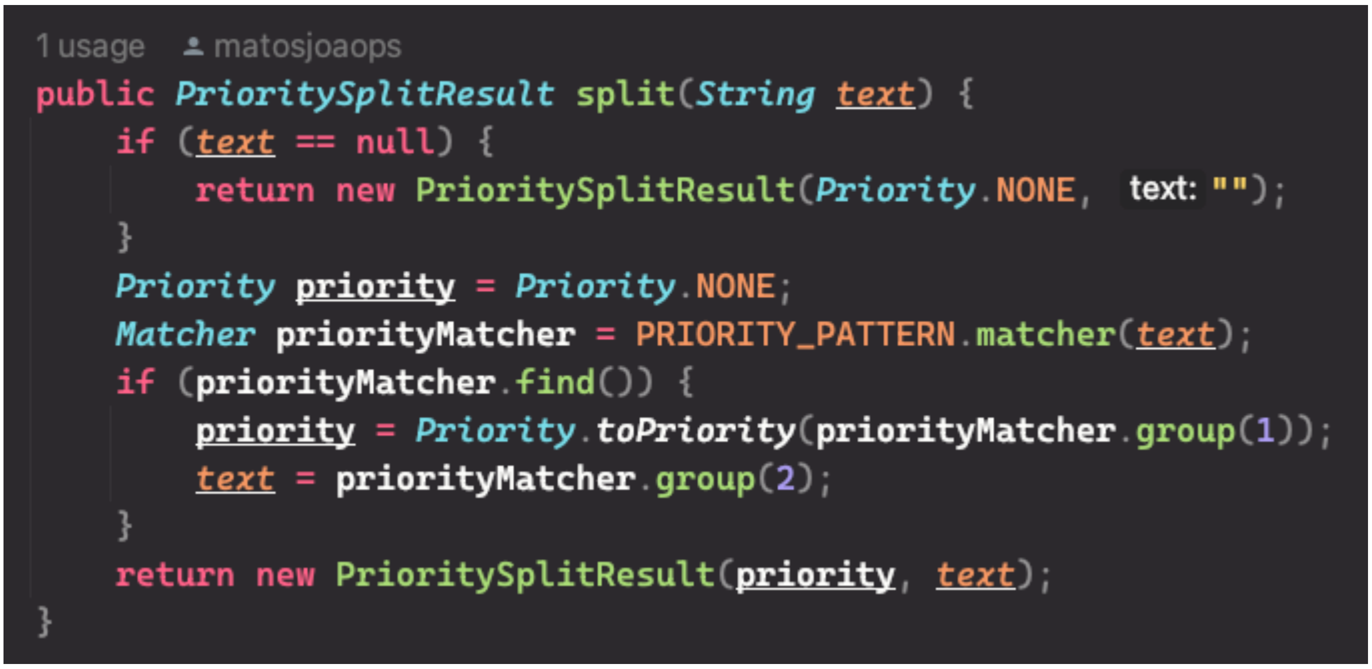
```
new *
private static native NativeLong setCurrentProcessExplicitAppUserModelID(WString appID);
```

Warning *AvoidReassigningParameters*

PMD was throwing a warning saying there was a parameter reassigment.

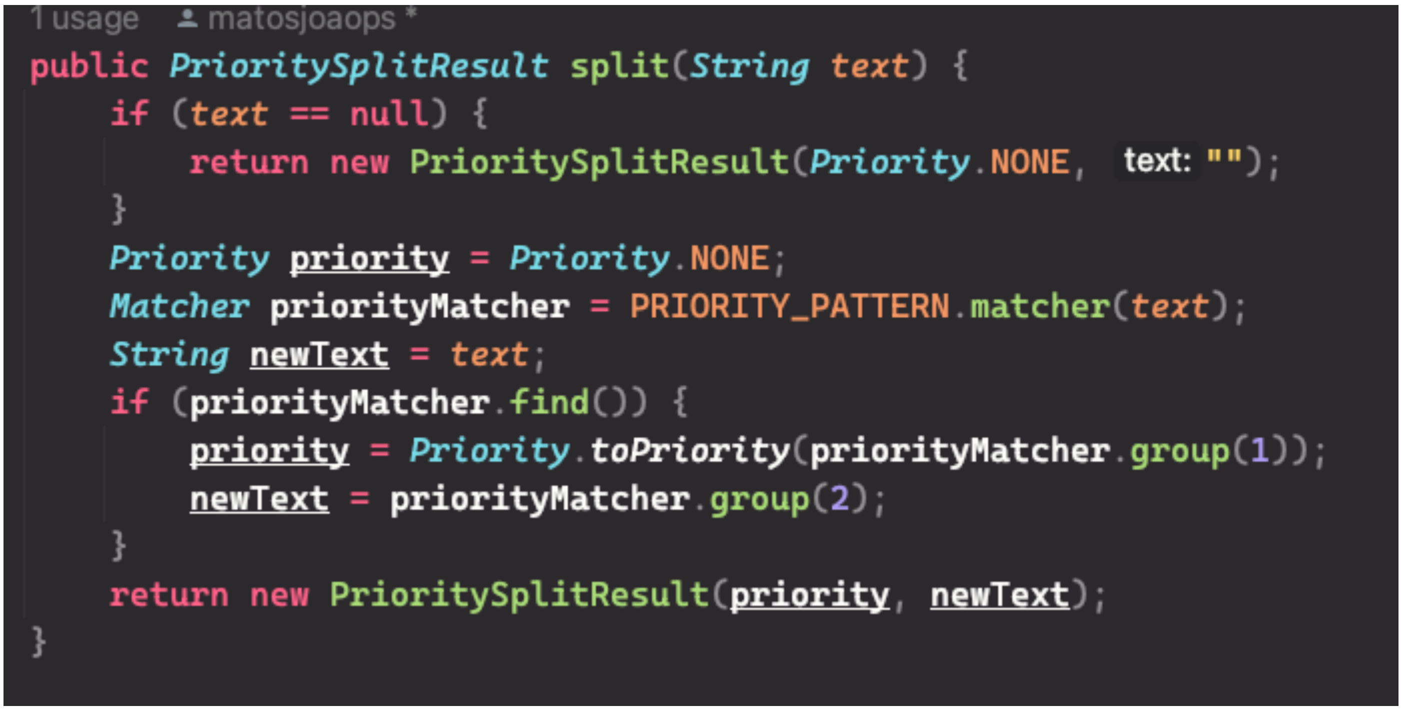


Rule	Violation	Line
AvoidReassigningParameters	Avoid reassigning parameters such as 'text'	59



```
1 usage   matosjoaops
public PrioritySplitResult split(String text) {
    if (text == null) {
        return new PrioritySplitResult(Priority.NONE,  text: "");
    }
    Priority priority = Priority.NONE;
    Matcher priorityMatcher = PRIORITY_PATTERN.matcher(text);
    if (priorityMatcher.find()) {
        priority = Priority.toPriority(priorityMatcher.group(1));
        text = priorityMatcher.group(2);
    }
    return new PrioritySplitResult(priority, text);
}
```

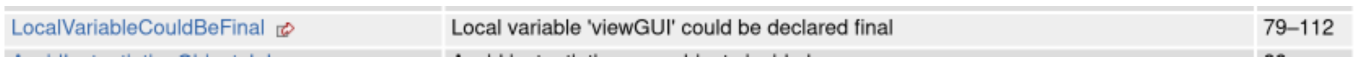
To fix this warning, we created a new variable to be assigned instead of the parameter.



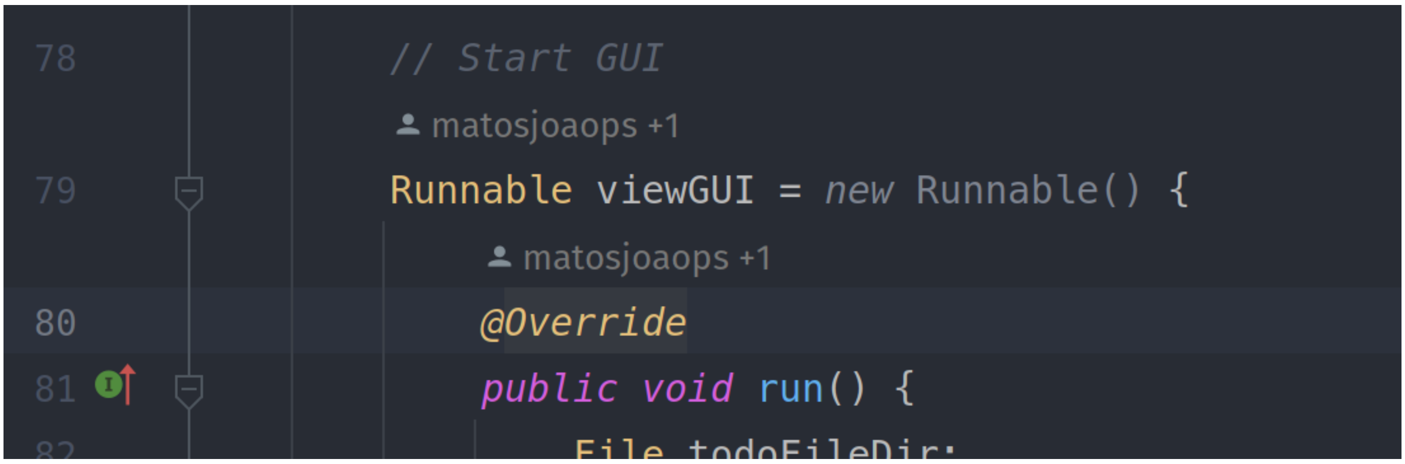
```
1 usage   matosjoaops *
public PrioritySplitResult split(String text) {
    if (text == null) {
        return new PrioritySplitResult(Priority.NONE,  text: "");
    }
    Priority priority = Priority.NONE;
    Matcher priorityMatcher = PRIORITY_PATTERN.matcher(text);
    String newText = text;
    if (priorityMatcher.find()) {
        priority = Priority.toPriority(priorityMatcher.group(1));
        newText = priorityMatcher.group(2);
    }
    return new PrioritySplitResult(priority, newText);
}
```

Warning *LocalVariableCouldBeFinal*

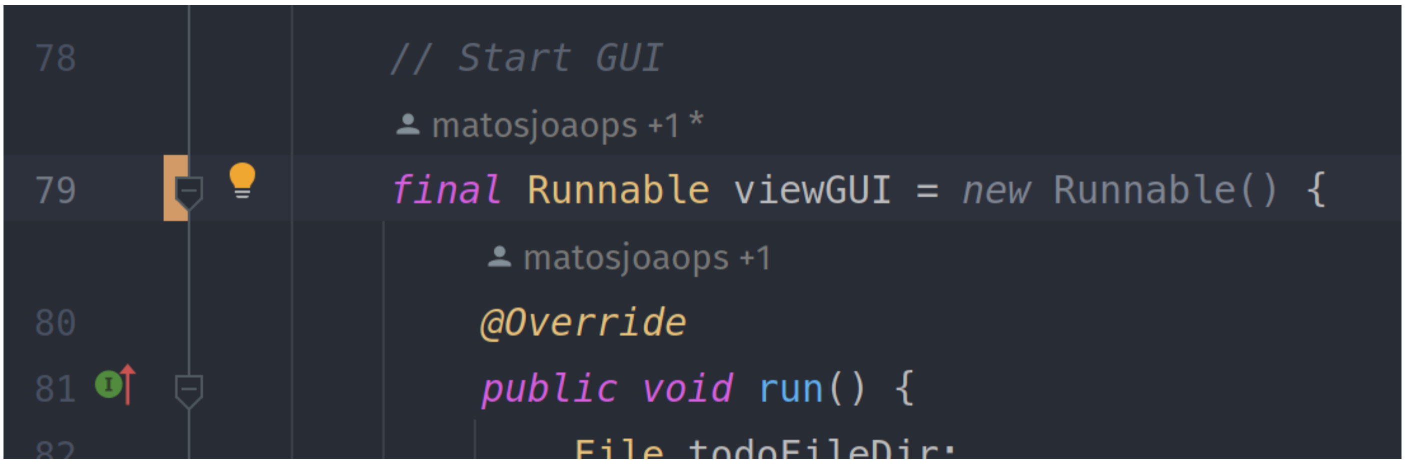
PMD showed us a warning regarding a local variable that could be declared as final.



LocalVariableCouldBeFinal	Local variable 'viewGUI' could be declared final	79–112

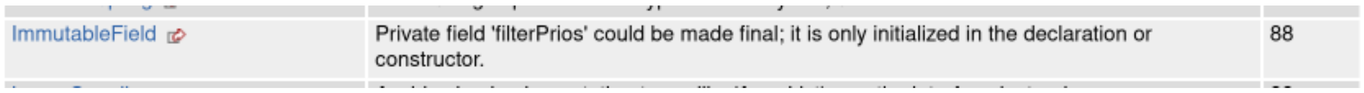

```
// Start GUI
matosjoaops +1
Runnable viewGUI = new Runnable() {
    matosjoaops +1
    @Override
    public void run() {
        File todoFileDir;
```

To fix the warning, the "final" keyword was added in the declaration.

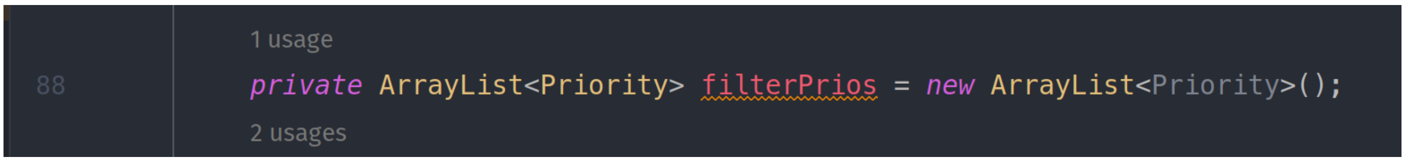

```
// Start GUI
matosjoaops +1 *
final Runnable viewGUI = new Runnable() {
    matosjoaops +1
    @Override
    public void run() {
        File todoFileDir;
```

ImmutableField

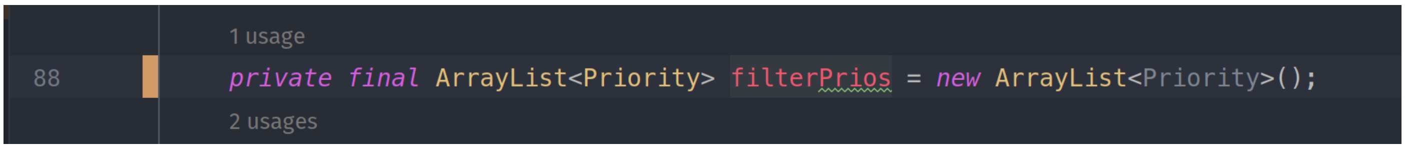
PMD showed us a warning regarding a field whose value was never reassigned and could, therefore, be made final.



ImmutableField	Private field 'filterPrios' could be made final; it is only initialized in the declaration or constructor.	88


```
1 usage
private ArrayList<Priority> filterPrios = new ArrayList<Priority>();
2 usages
```

The "final" keyword was added, which made the warning go away.


```
1 usage
private final ArrayList<Priority> filterPrios = new ArrayList<Priority>();
2 usages
```