Advance Topics In Software Engineering

Episode 3: Implementation presentation

```
while(true)
{
    …\\working methods
    if(contribution_finished)
    {
      contribution_patch_upload();
      System.out.print("my patch is already uploaded");
      break;
    }
}
```

Ioannis Sermetziadis 8020136

Department Of Management Science and Technology

Athens University Of Economics and Business

Intended & Finished changes(1)

- ***RedundantImportCheck.java* :** An 'if' statement was added to check if an import is imported by a wildcard import as well.

Code:

```
final String stringToCheck;

if(full.getText().indexOf(".*") != -1)
   stringToCheck = full.getText().substring(0, full.getText().lastIndexOf('.'));
else
   stringToCheck = full.getText();

if ((imp.getText().startsWith(stringToCheck) && !stringToCheck.equals(full.getText())) ||
        (imp.getText().equals(stringToCheck)) && stringToCheck.equals(full.getText())) {
   log(aAST.getLineNo(),
        aAST.getColumnNo(),
        "import.duplicate",
        new Integer(full.getLineNo()),
        imp.getText());
}
```

(Code added to ***ReduntantImportCheck.java***)

- ***UnusedImportsCheck.java* :** The intended change for that check in order to support wildcard imports was proven much more difficult. It could be easily implemented to check if any class of a wildcard import is used but was difficult to distinguish class's methods or variables. That was something that checkstyle's API doesn't support.

Intended & Finished changes(2)

- ***StrictDuplicateCode.java*** **:** Two setter methods where added, which give the choice to ignore or not the imports and the comments of the file that is checked for duplication code. An AutomaticBeans.class that exists in the application's API translates these methods to properties of the check, which are set by the user through the XML input file.

Code:

```
/**
 * Sets if imports will be ignored or not before the check starts
 *
 *@param mIgnoreImports must be set before
 * triggering a 'duplicate code' message. Default value = true
 */
public void setIgnoreImports(boolean aIgnoreImports)
{
   mIgnoreImports = aIgnoreImports;
}

/**
 * Sets if comments will be ignored or not before the check starts
 *
 *@param mIgnoreComments must be set before
 * triggering a 'duplicate code' message. Default value = false
 */
public void setIgnoreComments(boolean aIgnoreComments)
{
   mIgnoreComments = aIgnoreComments;
}
```

(Code added to ***StrictDuplicateCode.java***)

Intended & Finished changes(3)

- My check, **ClassBlankness.java :** It counts the blank lines and these that contain at most a '{' or '}' character. It also offers a setter method for the minimum blank lines that the block of the class should contain. If the blank lines are less than the minimum, there is a message that comes out to show that the class is not clear enough.

Code:

```java
/** Counts the blankness of a block of a class and gives about an integer
 * with the percentage of a class's block's lines that are blank.*/

private int blanknessCounter(String[] class_lines, int startLine, int finishLine)
{
    int blank_counter = 0;
    int block_length;
    Pattern regExpPatern = Pattern.compile("^\\s*(}|\\{)?\\s*$");
    for(block_length = startLine - 1; block_length < finishLine; block_length++) {

       Matcher regExpMatcher = regExpPatern.matcher(class_lines[block_length]);

       if(regExpMatcher.matches())
           blank_counter++;

    }
    return (int)(((double)blank_counter / (double)(finishLine - startLine)) * 100);
}
```

(Sample Code of **ClassBlankness.java**)

Intended & Finished changes(4)

- New Logger, ***JavaLogger.java*** **:** Also a .java file can be given as an output of the checks. This file will contain the source code of the checked file(s) and every check message will appear next to the line that is referred to, in comment form, like **\\ check_message**.

"-f java" should be given when calling the application from the command line, in order to use the JavaLogger.

Code:

```
    final String format =
        aLine.hasOption("f") ? aLine.getOptionValue("f") : "plain";

    AuditListener listener = null;
    if ("xml".equals(format)) {
        listener = new XMLLogger(aOut, aCloseOut);
    }
    else if ("plain".equals(format)) {
        listener = new DefaultLogger(aOut, aCloseOut);
    }
    else if ("java".equals(format)) {
        listener = new JavaLogger(aOut, aCloseOut);
    }
    else {
        System.out.println("Invalid format: (" + format
                           + "). Must be 'plain', 'xml' or 'java'.");
        usage();
    }
    return listener;
}
```

(Sample Code of **Main.java***)*

Changes made to checkstyle's GUI(5)

- Changes in Main.java and ParseTreeInfoPanel.java of com.puppycrawl.tools.checkstyle.gui package.

1) When a file is selected, its name appears on the frame's title.

2) A read-only text area was added under the tree, that depicts the file's structure according to the *"antlr.CommonAST"* package. The source code of the selected file is transferred there.

Due to difficulties faced with understanding the way that some methods of "javax.swing.tree" and "javax.swing.event" packages work, the EventListener that would highlight the corresponding line of a selected AST node, could not be implemented, which was intended. Also the documentation provided for this package is really poor.

How is checkstyle's GUI…

(after my changes)

Checkstyle : TreeWalker.java

Tree	Type	Line	Column	Text
ROOT[19x0]	EOF	19	0	ROOT
package[19x0]	PACKAGE_DEF	19	0	package
import[21x0]	IMPORT	21	0	import
import[22x0]	IMPORT	22	0	import
import[23x0]	IMPORT	23	0	import
import[24x0]	IMPORT	24	0	import
import[25x0]	IMPORT	25	0	import
import[26x0]	IMPORT	26	0	import
import[27x0]	IMPORT	27	0	import
import[28x0]	IMPORT	28	0	import
import[29x0]	IMPORT	29	0	import
import[30x0]	IMPORT	30	0	import
import[31x0]	IMPORT	31	0	import
import[32x0]	IMPORT	32	0	import
import[33x0]	IMPORT	33	0	import
import[34x0]	IMPORT	34	0	import
import[37x0]	IMPORT	37	0	import
import[38x0]	IMPORT	38	0	import
import[39x0]	IMPORT	39	0	import
import[40x0]	IMPORT	40	0	import
import[42x0]	IMPORT	42	0	import
import[43x0]	IMPORT	43	0	import
import[44x0]	IMPORT	44	0	import
import[45x0]	IMPORT	45	0	import
import[46x0]	IMPORT	46	0	import

```
/**
 * Creates a new <code>TreeWalker</code> instance.
 */
public TreeWalker()
{
    setFileExtensions(new String[]{"java"});
    // Tree walker can use two possible algorithms for
    // tree processing (iterative and recursive.
    // Recursive is default for now.
    String recursive =
        System.getProperty("checkstyle.use.recursive.algorithm", "true");
    mRecursive = "true".equals(recursive);
    if (mRecursive) {
        sLog.debug("TreeWalker uses recursive algorithm");
    }
```

Select Java File | Reload Java File

What had to be done later…?

➢ To write or modify the classes for the tests of the changed or inserted files, so that the test reports that the application generates when running the tests, are at good level – percentage.

1)

```
public void testWithChecker()
    throws Exception
{
    final DefaultConfiguration checkConfig =
        createCheckConfig(RedundantImportCheck.class);
    final String[] expected = {
        "7:1: Redundant import from the same package - com.puppycrawl.tools.checkstyle.*.",
        "8:38: Duplicate import to line 7 - com.puppycrawl.tools.checkstyle.GlobalProperties.",
        "8:38: Redundant import from the same package - com.puppycrawl.tools.checkstyle.GlobalProperties.",
        "10:1: Redundant import from the java.lang package - java.lang.*.",
        "11:1: Duplicate import to line 10 - java.lang.String.",
        "11:1: Redundant import from the java.lang package - java.lang.String.",
        "14:1: Duplicate import to line 13 - java.util.List.",
        "26:1: Duplicate import to line 25 - javax.swing.WindowConstants.*."
    };
    verify(checkConfig, getPath("InputImport.java"), expected);
}
```

(Sample Code of **RedundantImportCheckTest.java**)

After testing the output of the check on the changed classes, the *String[] expected* had to be modified.

Go on testing…

2) To add testing methods to StrictDuplicateCodeCheckTest.java for the methods inserted in StrictDuplicateCode.java

```
public void testNotIgnoreImports() throws Exception
        {
            final DefaultConfiguration checkConfig = createCheckConfig(StrictDuplicateCodeCheck.class);
            checkConfig.addAttribute("min", "3");
            checkConfig.addAttribute("ignoreImports", "false");
            final String aPath = getPath("duplicates/A.java");
            final String bPath = getPath("duplicates/B.java");
            final String[] expected = {
                    "duplicates/B.java:1: Found duplicate of 7 lines in duplicates/A.java, starting from line 1"
                    };
            final File[] checkedFiles = new File[] {
                    new File(aPath),
                    new File(bPath),
                    };
            verify(createChecker(checkConfig), checkedFiles, aPath, expected);
}

public void testIgnoreComments() throws Exception
                {
                    final DefaultConfiguration checkConfig = createCheckConfig(StrictDuplicateCodeCheck.class);
                    checkConfig.addAttribute("min", "3");
                    checkConfig.addAttribute("ignoreComments", "true");
                    final String aPath = getPath("duplicates/A.java");
                    final String bPath = getPath("duplicates/B.java");
                    final String[] expected = {
                            //no comments in A.java and B.java so nothing is expected
                            };
                    final File[] checkedFiles = new File[] {
                            new File(aPath),
                            new File(bPath),
                            };
                    verify(createChecker(checkConfig), checkedFiles, aPath, expected);
}
```

(Sample Code of StrictDuplicateCodeCheckTest.java)

Go on testing…

3) To write a simple test class for the new check, ***ClassBlanknessCheckTest.java***

```java
package com.puppycrawl.tools.checkstyle.checks.whitespace;

import com.puppycrawl.tools.checkstyle.BaseCheckTestCase;
import com.puppycrawl.tools.checkstyle.DefaultConfiguration;

public class ClassBlanknessCheckTest
    extends BaseCheckTestCase
{
    DefaultConfiguration checkConfig;

    public void setUp()
    {
        checkConfig = createCheckConfig(ClassBlanknessCheck.class);
    }

    public void test()
        throws Exception
    {
        final String[] expected = {
            //the checks comes out without errors
        };
        verify(checkConfig, getPath("InputWhitespace.java"), expected);
    }
}
```

(Sample Code of ClassBlanknessCheckTest.java)

Ready to fade out…

- After all, a patch file with all the changes was extracted with the command “cvs diff -u > mychanges.patch” .

- A contribution package is already uploaded to the project’s patch repositoty and can be accessed by:

http://sourceforge.net/tracker/index.php?func=detail&aid=1499180&group_id=29721&atid=397080

Thank you for your attention and your time:-)

Questions?