JEDIT

Code Smells Identified:

1) **Feature Envy:**

Refactoring type: Move Method
Source Entity: org.gjt.sp.jedit.gui.MakerViewer.java
Target Class: org.gjt.sp.jedit.View

Original method in org.gjt.sp.jedit.gui.MakerViewer.java:

```
MakerViewer.java
204
205     //{{{ goToSelectedMarker() method
206     private void goToSelectedMarker()
207     {
208         Object value = markerList.getSelectedValue();
209         if(!(value instanceof Marker))
210             return;
211
212         Marker mark = (Marker)value;
213         view.getTextArea().setCaretPosition(mark.getPosition());
214         view.toFront();
215         view.requestFocus();
216         view.getTextArea().requestFocus();
217     } //}}}
218
219     //{{{ updateSelection() method
```

- This method in **MarkerView** class seemed to be manipulating the data of **view** class more than its own.
- Using Marker object 'mark', this method sets the caret position and then calls various methods in 'view' class. This is an anti-pattern to object-orient design. This increases the coupling between two classes.
- This is a clear example of feature envy smell type. Therefore, I think the smell detected here is an actual smell.

Automatic Refactoring:

org.gjt.sp.jedit.gui.MakerViewer.java:

```
204
205     //{{{ goToSelectedMarker() method
206     private void goToSelectedMarker()
207     {
208         view.gotoSelectedMarker(markerList);
209     } //}}}
210
```

org.gjt.sp.jedit.View:

```
2351    }//}}}
2352     //}}}
2353
2354    public void gotoSelectedMarker(JList<Marker> markerList)
2355    {
2356        Object value = markerList.getSelectedValue();
2357        if(!(value instanceof Marker))
2358            return;
2359
2360        Marker mark = (Marker)value;
2361        getTextArea().setCaretPosition(mark.getPosition());
2362        toFront();
2363        requestFocus();
2364        getTextArea().requestFocus();
2365    }
2366 }
```

- The action performed by Eclipse IDE's Move Method is basically moving all the logic from `MarkerViewer` class into a new method in the `View` class.
- It makes sense for the new function `gotoSelectedMarker` to be in `View` class as it uses the other methods getTextArea, toFront and requestFocus which are also part of the same class.
- With this change the coupling between two classes reduced and JDeodorant no longer shows this smell on this class.

Testing:

As there are no tests implemented for these components, I used Randoop to automatically generate tests for the original code.

Generating tests on original code:

```
C:\Users\jyothsnk\git\cs515-801-s20-kodavati-jedit>java -Xmx3000m -classpath C:\Users\jyothsnk\Downloads\test\cs515-801-s20-kodavati-jedit\b
uild\classes\core;C:\Users\jyothsnk\Downloads\randoop-4.2.0\randoop-4.2.0\randoop-all-4.2.0.jar;C:\Users\jyothsnk\Downloads\classlist\classl
ist;C:\Users\jyothsnk\tests randoop.main.Main gentests --only-test-public-members=true --classlist=C:\Users\jyothsnk\Downloads\classlist\cla
sslist.txt --junit-output-dir=C:\Users\jyothsnk\tests
PUBLIC MEMBERS=623
Explorer = ForwardGenerator(allSequences:0, sideEffectFreeMethods:1119, subsumed_sequences:0, runtimePrimitivesSeen:38)

Progress update: steps=1, test inputs generated=0, failing inputs=0      (Wed Apr 08 23:42:00 MDT 2020     86MB used)
Progress update: steps=1000, test inputs generated=104, failing inputs=4      (Wed Apr 08 23:42:08 MDT 2020     17MB used)
Progress update: steps=2000, test inputs generated=149, failing inputs=4      (Wed Apr 08 23:42:11 MDT 2020     87MB used)
Progress update: steps=47000, test inputs generated=1723, failing inputs=4      (Wed Apr 08 23:43:40 MDT 2020     78MB used)
Progress update: steps=47253, test inputs generated=1738, failing inputs=4      (Wed Apr 08 23:43:40 MDT 2020     159MB used)
Normal method executions: 7118
Exceptional method executions: 10

Average method execution time (normal termination):      0.00252
Average method execution time (exceptional termination): 25.3
Approximate memory usage 159MB

Error-revealing test output:
Error-revealing test count: 4
Writing error-revealing JUnit tests...
Created file C:\Users\jyothsnk\tests\ErrorTest0.java
Created file C:\Users\jyothsnk\tests\ErrorTest.java
Wrote error-revealing JUnit tests.

About to look for failing assertions in 945 regression sequences.

Regression test output:
Regression test count: 945
Writing regression JUnit tests...
Created file C:\Users\jyothsnk\tests\RegressionTest0.java
Created file C:\Users\jyothsnk\tests\RegressionTest1.java
Created file C:\Users\jyothsnk\tests\RegressionTest.java
Wrote regression JUnit tests.
```

945 Junit tests were generated for the original code. These tests will be used to identify if any of the refactoring changes caused regression.

Running tests on the refactored code:

```
C:\Users\jyothsnk\git\cs515-801-s20-kodavati-jedit>java -classpath C:\Users\jyothsnk\Downloads\hamcrest-all-1.3.jar;C:\Users\jyothsnk\Downlo
ads\junit-4.13-beta-3.jar;C:\Users\jyothsnk\Downloads\test\cs515-801-s20-kodavati-jedit\bin;C:\Users\jyothsnk\tests org.junit.runner.JUnitCo
re RegressionTest
JUnit version 4.13-beta-3
.....................................................................................................................
.....................................................................................................................
.....................................................................................................................
.....................................................................................................................
.....................................................................................................................
.....................................................................................................................
.....................................................................................
Time: 0.612

OK (945 tests)
```

All passed.

2) Duplicated Code:

Source Entity: org.gjt.sp.jedit.textarea.TextArea

- It is possible for classes to have nearly identical methods or sequence of code with very minor difference. Although visually they appear to be different, they perform same kind of job.
- In `TextArea` class there are two methods 'toUpperCase' and 'toLowerCase'. The code looks similar in these two methods. The only difference is that they call two different internal functions.
- This clearly shows the duplicated code in this class. Therefore, I think this is an actual smell.

Original methods in org.gjt.sp.jedit.textarea.TextArea:

```
4415    public void toLowerCase()
4416    {
4417        if(!buffer.isEditable())
4418        {
4419            javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
4420            return;
4421        }
4422
4423        Selection[] selection = getSelection();
4424        int caret = -1;
4425        if (selection.length == 0)
4426        {
4427            caret = getCaretPosition();
4428            selectWord();
4429            selection = getSelection();
4430        }
4431        if (selection.length == 0)
4432        {
4433            if (caret != -1)
4434                setCaretPosition(caret);
4435            javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
4436            return;
4437        }
4438
4439        buffer.beginCompoundEdit();
4440
4441        for (Selection s : selection)
4442            setSelectedText(s, getSelectedText(s).toLowerCase());
4443
4444        buffer.endCompoundEdit();
4445        if (caret != -1)
4446            setCaretPosition(caret);
4447    } //}}}
```

```
4376    public void toUpperCase()
4377    {
4378        if(!buffer.isEditable())
4379        {
4380            javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
4381            return;
4382        }
4383
4384        Selection[] selection = getSelection();
4385        int caret = -1;
4386        if (selection.length == 0)
4387        {
4388            caret = getCaretPosition();
4389            selectWord();
4390            selection = getSelection();
4391        }
4392        if (selection.length == 0)
4393        {
4394            if (caret != -1)
4395                setCaretPosition(caret);
4396            javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
4397            return;
4398        }
4399
4400        buffer.beginCompoundEdit();
4401
4402        for (Selection s : selection)
4403            setSelectedText(s, getSelectedText(s).toUpperCase());
4404
4405        buffer.endCompoundEdit();
4406        if (caret != -1)
4407            setCaretPosition(caret);
4408    } //}}}
```

Manual Refactoring:

```java
public void toCase(String upperOrLower)
{
    if(!buffer.isEditable())
    {
        javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
        return;
    }

    Selection[] selection = getSelection();
    int caret = -1;
    if (selection.length == 0)
    {
        caret = getCaretPosition();
        selectWord();
        selection = getSelection();
    }
    if (selection.length == 0)
    {
        if (caret != -1)
            setCaretPosition(caret);
        javax.swing.UIManager.getLookAndFeel().provideErrorFeedback(null);
        return;
    }

    buffer.beginCompoundEdit();

    for (Selection s : selection)
    {
        if (upperOrLower == "UPPER")
            setSelectedText(s, getSelectedText(s).toUpperCase());
        else if (upperOrLower == "LOWER")
            setSelectedText(s, getSelectedText(s).toLowerCase());
    }

    buffer.endCompoundEdit();
    if (caret != -1)
        setCaretPosition(caret);
} //}}}
```

- Instead of having two different methods, we can combine the two methods in to one single method `toCase`. As there are only few callers to the original methods, this is an easier refactor to choose.
- The refactored method(combined) will take a String type argument that is either UPPER or LOWER.
- Based on this argument, a conditional statement can be used to select which internal function to call i.e, `toUpperCase` or `toLowerCase`