Assignment # 2

**SE 6356 Software Maintenance, Evolution and Re-Engineering**

Mahabubul Alam

Abdullah Moyeen

Table of Contents

[1 Class cohesion in OO software 3](#_Toc436347649)

[1.1 System: jEdit 3](#_Toc436347650)

[1.1.1 The top 2 most cohesive classes, based on SourceMeter results: 3](#_Toc436347651)

[1.1.2 The top 2 least cohesive classes, based on SourceMeter results: 3](#_Toc436347652)

[1.2 System: aTunes 4](#_Toc436347653)

[1.2.1 The top 2 most Cohesive Classes, based on SourceMeter results: 4](#_Toc436347654)

[1.2.2 The top 2 least Cohesive Classes, based on SourceMeter results: 4](#_Toc436347655)

[2 Code smells using JDeodorant and InCode 5](#_Toc436347656)

[2.1 aTunes code smells 5](#_Toc436347657)

[2.1.1 Smell: Feature Envy - net.sourceforge.atunes.kernel.controllers.playListControls 5](#_Toc436347658)

[2.1.2 Smell: God Class - net.sourceforge.atunes.kernel.handlers.PlayListHandler 6](#_Toc436347659)

[2.1.3 Smell: Data Clumps - net.sourceforge.atunes.kernel.controllers.stats 8](#_Toc436347660)

[2.2 jEdit code smells 8](#_Toc436347661)

[2.2.1 Smell: Type checking - org.gjt.sp.jedit.textarea 8](#_Toc436347662)

[2.2.2 Smell: Internal duplication - org.gjt.sp.jedit.textarea 9](#_Toc436347663)

[2.2.3 Smell: Message Chains - org.gjt.sp.jedit.textarea 10](#_Toc436347664)

[3 Refactoring using tool support 11](#_Toc436347665)

[3.1 aTunes – removing “God Class” smell 11](#_Toc436347666)

[3.1.1 Justification of refactoring 11](#_Toc436347667)

[3.1.2 Description and Rationale 11](#_Toc436347668)

[3.1.3 Code Smell visualization 11](#_Toc436347669)

[3.1.4 Changes to be performed 12](#_Toc436347670)

[3.1.5 Changed code 14](#_Toc436347671)

[3.1.6 Tests 15](#_Toc436347672)

[3.1.7 Tool result before refactoring 15](#_Toc436347673)

[3.1.8 Tool results after refactoring 16](#_Toc436347674)

[3.2 jEdit – removing “Type Checking” smell 17](#_Toc436347675)

[3.2.1 Justification for refactoring 17](#_Toc436347676)

[3.2.2 Description and Rationale 17](#_Toc436347677)

[3.2.3 Code smell visualization 17](#_Toc436347678)

[3.2.4 Changes to be performed 17](#_Toc436347679)

[3.2.5 Changed code 18](#_Toc436347680)

[3.2.6 Tests 18](#_Toc436347681)

[3.2.7 Tool result before refactoring 19](#_Toc436347682)

[3.2.8 Tool result after refactoring 19](#_Toc436347683)

[4 Manual Refactoring 20](#_Toc436347684)

[4.1 aTunes – removing “Feature Envy” smell 20](#_Toc436347685)

[4.1.1 Justification for refactoring 20](#_Toc436347686)

[4.1.2 Description and rationale 20](#_Toc436347687)

[4.1.3 Code before refactoring 20](#_Toc436347688)

[4.1.4 Refactored code 20](#_Toc436347689)

[4.1.5 Tests 21](#_Toc436347690)

[4.1.6 Tool results 22](#_Toc436347691)

[4.2 jEdit – removing “Internal duplication” smell 23](#_Toc436347692)

[4.2.1 Justification for refactoring 23](#_Toc436347693)

[4.2.2 Description and Rationale 23](#_Toc436347694)

[4.2.3 Code before refactoring 23](#_Toc436347695)

[4.2.4 Refactored code 24](#_Toc436347696)

[4.2.5 Tests 24](#_Toc436347697)

[4.2.6 Tool results 25](#_Toc436347698)

[5 References 26](#_Toc436347699)

# Class cohesion in OO software

## System: jEdit

### The top 2 most cohesive classes, based on SourceMeter results:

|  |  |
| --- | --- |
| **Class Name** | **Lack of Cohesion in Methods 5 (LCOM5)** |
| org.gjt.sp.jedit.textarea.TextAreaException | 0 |
| org.gjt.sp.jedit.syntax.SyntaxStyle | 0 |

* The **TextAreaException** class implements a single functionality - Exception that the TextArea can throw when an error occurs. It can be catched and an error dialog can be displayed.
* The **SyntaxStyle** class is a simple text style class. It implements the functionality of specifying the color, italic flag, and bold flag of a run of text and is self-contained.

### The top 2 least cohesive classes, based on SourceMeter results:

|  |  |
| --- | --- |
| **Class Name** | **Lack of Cohesion in Methods 5 (LCOM5)** |
| org.gjt.sp.jedit.MiscUtilities | 38 |
| org.gjt.sp.jedit.GUIUtilities | 22 |

* The **MiscUtilities** class can be split into 38 coherent classes which is also apparent by examining the code which shows a lot of methods busy, implementing functionalities such as path manipulation, string manipulation, URL name manipulation and more. Also, the class is coupled with 19 other classes and there are 135 incoming invocations along with 52 outgoing invocations.
* According to SourceMeter analysis, The **GUIUtilities** class can be split into 22 coherent classes. The class is coupled with 25 other classes and has 228 incoming and 43 outgoing invocations. The class implements various GUI utility functions related to icons, menus, toolbars, keyboard shortcuts, etc. The most frequently used members of this class are:



## System: aTunes

### The top 2 most Cohesive Classes, based on SourceMeter results:

|  |  |
| --- | --- |
| **Class Name** | **Lack of Cohesion in Methods 5 (LCOM5)** |
| net.sourceforge.atunes.kernel.modules.repository.tags.reader.TagDetector | 0 |
| net.sourceforge.atunes.kernel.modules.amazon.AmazonAlbum | 0 |

* The **TagDetector** class is limited to a single functionality of getting MP3 tags from the audio file. The class has 3 private attributes (ID3v2TagReader, ID3v1\_1TagReader, ID3v1TagReader) and the single method, getTags() inside this class is only using these 3 attributes.
* The **AmazonAlbum** class has 4 attributes that it uses to implement the responsibility of getting the album’s artist, album and url information and is self-contained in that.

### The top 2 least Cohesive Classes, based on SourceMeter results:

|  |  |
| --- | --- |
| **Class Name** | **Lack of Cohesion in Methods 5 (LCOM5)** |
| net.sourceforge.atunes.kernel.modules.state.ApplicationState | 19 |
| net.sourceforge.atunes.kernel.handlers.PlayListHandler | 8 |

* The **ApplicationState** class, according to SourceMeter could be divided into 19 distinct cohesive classes, which is also apparent by looking at the code. We can easily see that it has methods to do a lot things which are not necessarily related to each other. For example,



* The **PlayListHandler** class is implementing quite a few functionalities and some of which are not related. Such as, sorting and reordering PlayList and editing tags.

The main difference between the classes with the highest and lowest cohesion lies in the number of functionalities of the class, i.e. how well or badly the class adheres to the single responsibility principle. To identify the most and least cohesive classes, we depended on the LCOM5 metric analyzed by source meter which measures the lack of cohesion and computes into how many coherent classes the class could be split. We also looked at the coupling metrics to see how many incoming and outgoing invocations the classes had. A high value indicated a low cohesive class and a lower value indicated a class with higher cohesion.

# Code smells using JDeodorant and InCode

## aTunes code smells

### Smell: Feature Envy - net.sourceforge.atunes.kernel.controllers.playListControls

Protected method **addBindings()** in **PlayListControlsController.java** is heavily using data from external class **PlayListControlsPanel.java** to add bindings to **PlaylistControlsListener.java class**.

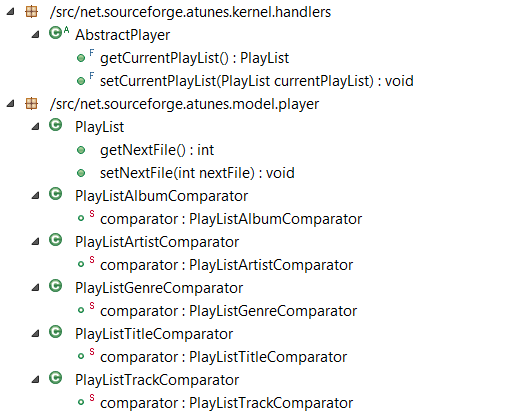


It is a smell because a method is accessing the data of another object more than its own data.

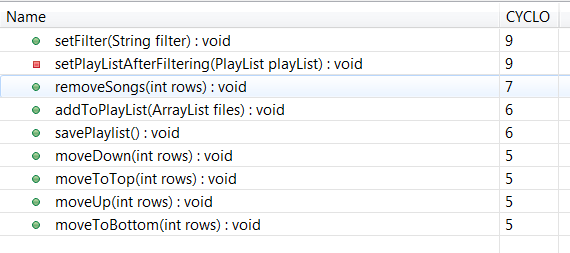


### Smell: God Class - net.sourceforge.atunes.kernel.handlers.PlayListHandler

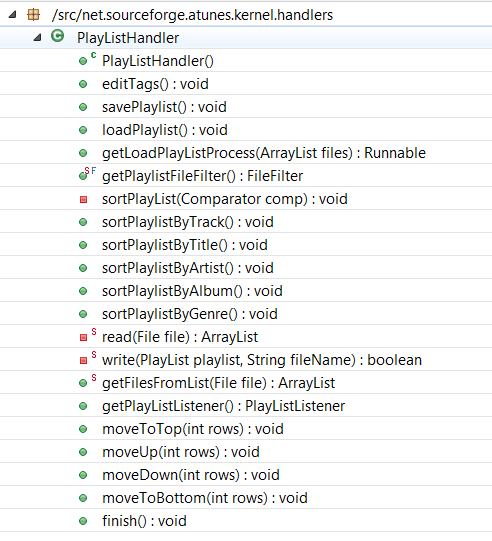
The **PlayListHandler** class uses many attributes from external classes –



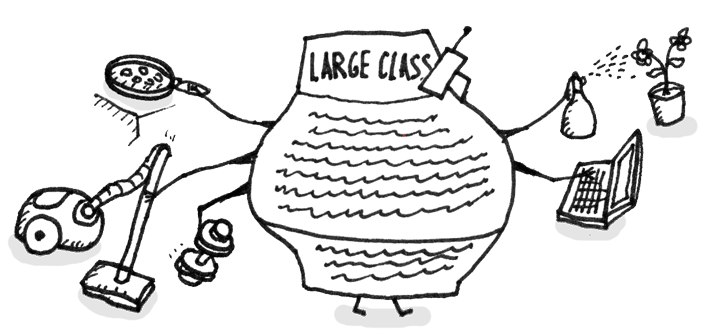
The **PlayListHandler** class is excessively large and complex, due to its methods having a high cyclomatic complexity and nesting level



This **PlayListHandler** class is very non-cohesive, in terms of how class attributes are used by its methods.

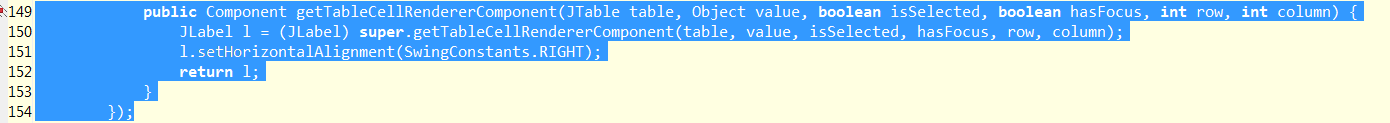


The **PlayListHandler** class is affected by the “God Class” smell because it contains many fields/methods/lines of code and can be considered a violation of the single responsibility principle of Object Oriented design.

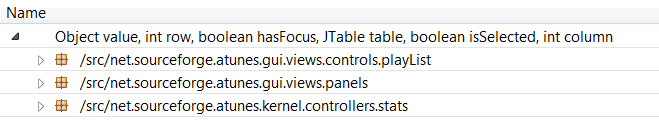


### Smell: Data Clumps - net.sourceforge.atunes.kernel.controllers.stats

getTableCellRendererComponent in the **StatsDialogController** class is affected by Data Clumps because the method has a long parameter list, and its signature or a significant fragment thereof is duplicated by other methods. This is a sign that the group of parameters, being passed around collectively to multiple methods in the system, could form a new abstraction that could be extracted to a new class.



The detected parameter clusters show individual parameter list fragments that are repeatedly used all around the system.



## jEdit code smells

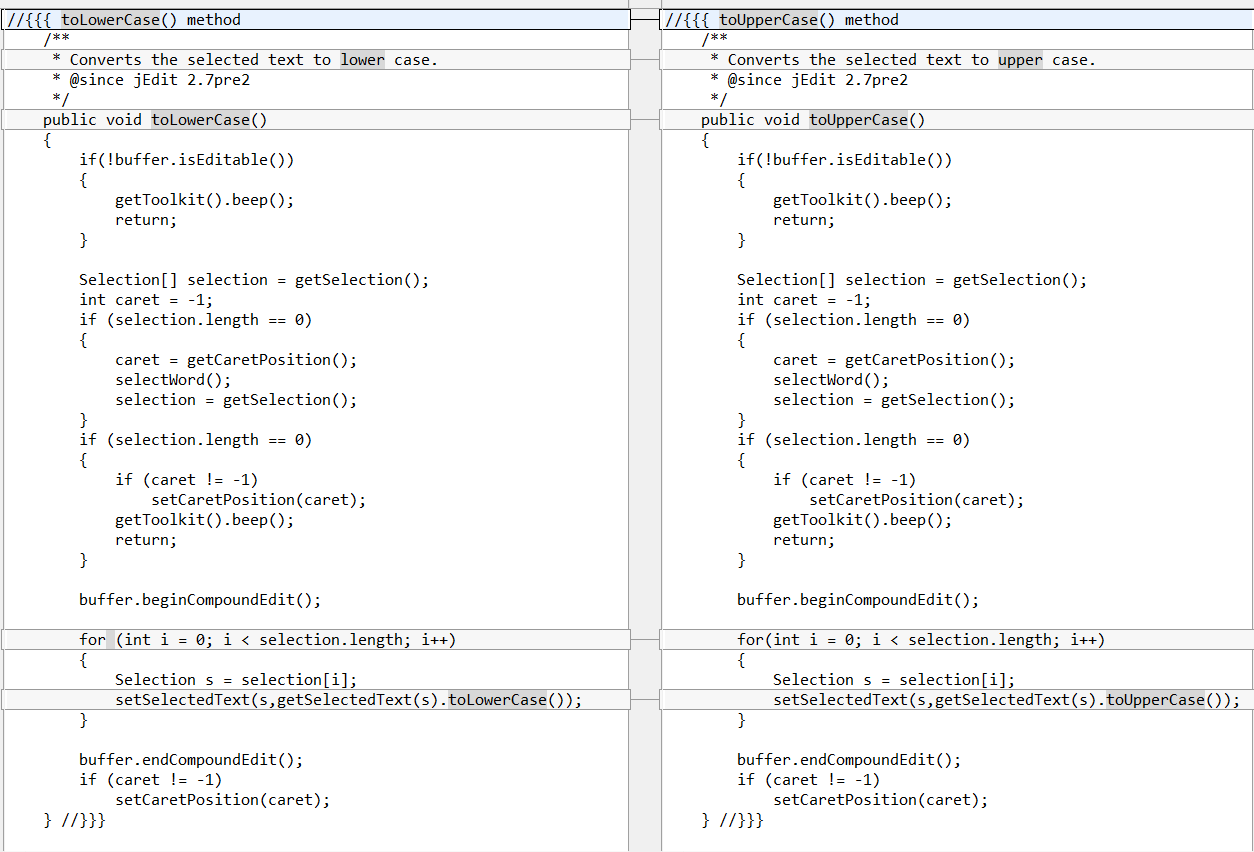
### Smell: Type checking - org.gjt.sp.jedit.textarea

A sequence of if statements in delete(boolean forward) method of TextArea.java class, where the delete operation is being performed based on different type of text selections,

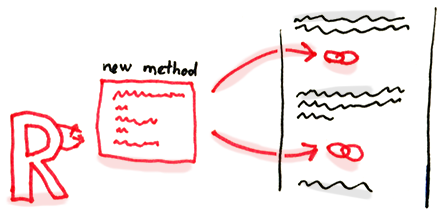


### Smell: Internal duplication - org.gjt.sp.jedit.textarea

Public methods toUpperCase() and toLowerCase() in the **TextArea** class are exactly identical except just in one line where they are calling different String extensions of toUpperCase() and toLowerCase() respectively.

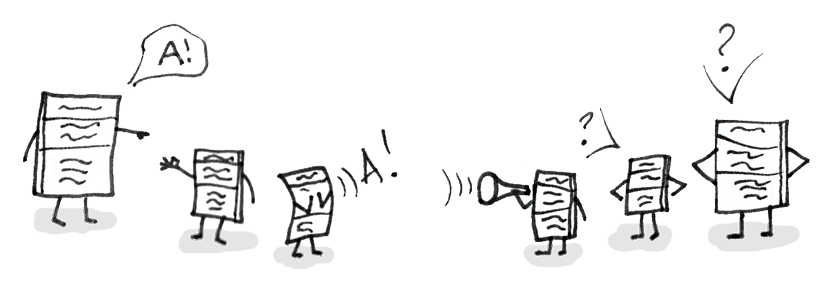


This is internal duplication smell, because the same code is found in two or more methods in the same class and the suggestion is to use Extract Method and place calls for the new method in both places.

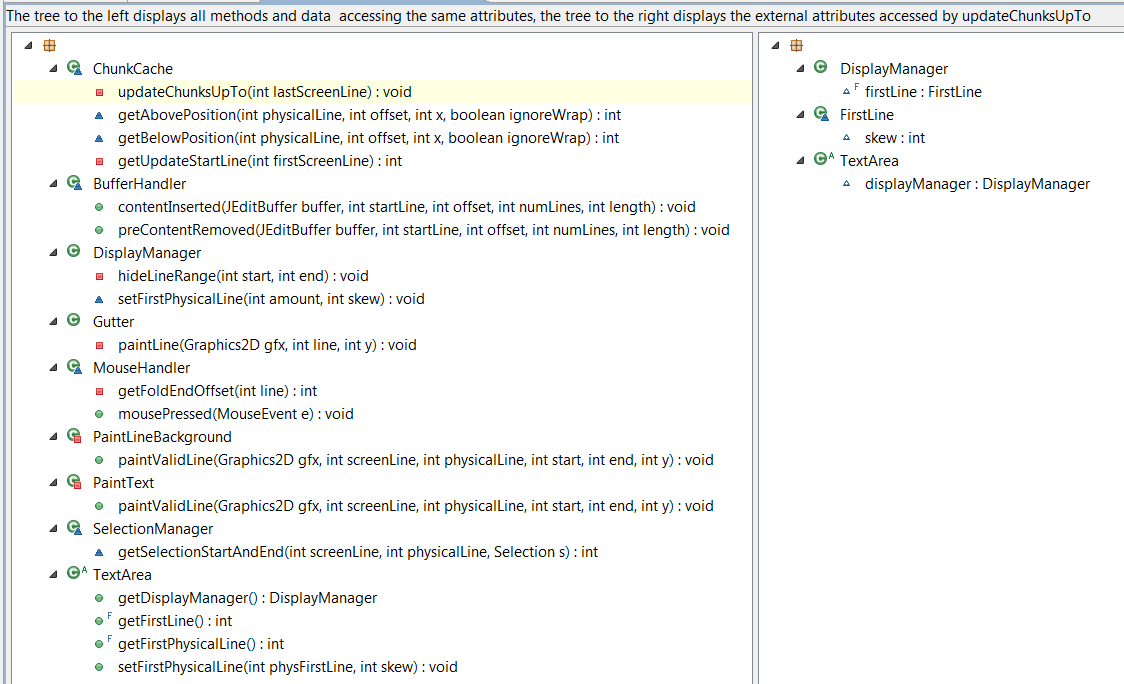


### Smell: Message Chains - org.gjt.sp.jedit.textarea

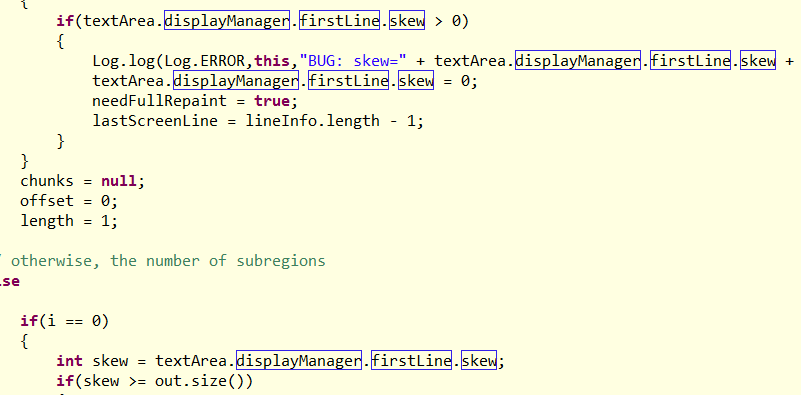
The updateChunksUpTo method in the **ChunkCache** class is affected by Message Chains because the method uses one object to access another object, then uses the obtained object to access another object, and so on, all objects having different types.



The screengrab below from InCode shows the used external Attributes, and it helps better understand what external data is used, and who else is using it.



The highlighted attributes below are involved in Message Chains and is a good indication of where the message chain originates:



# Refactoring using tool support

## aTunes – removing “God Class” smell

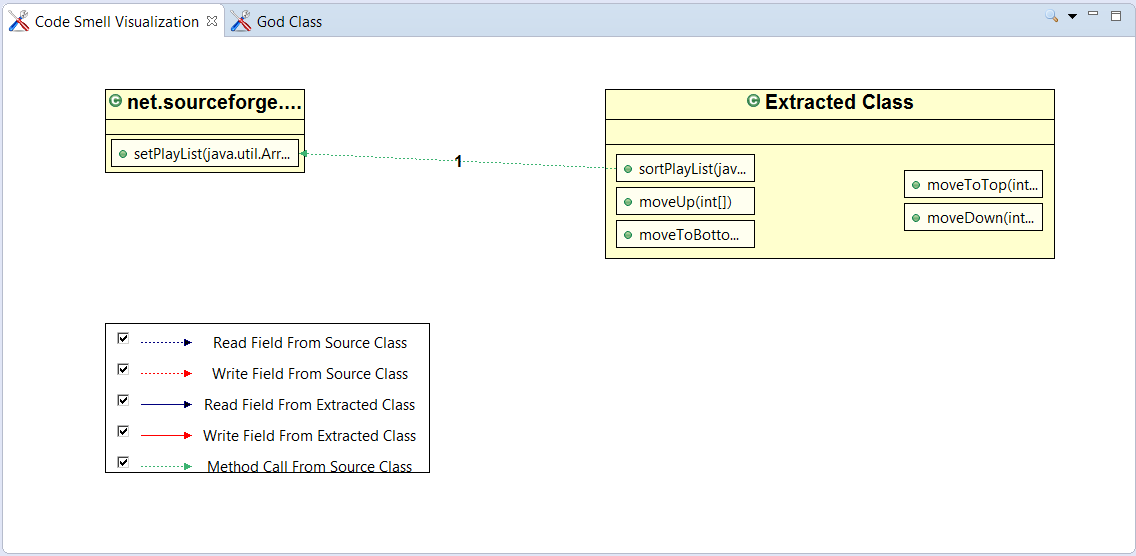
### Justification of refactoring

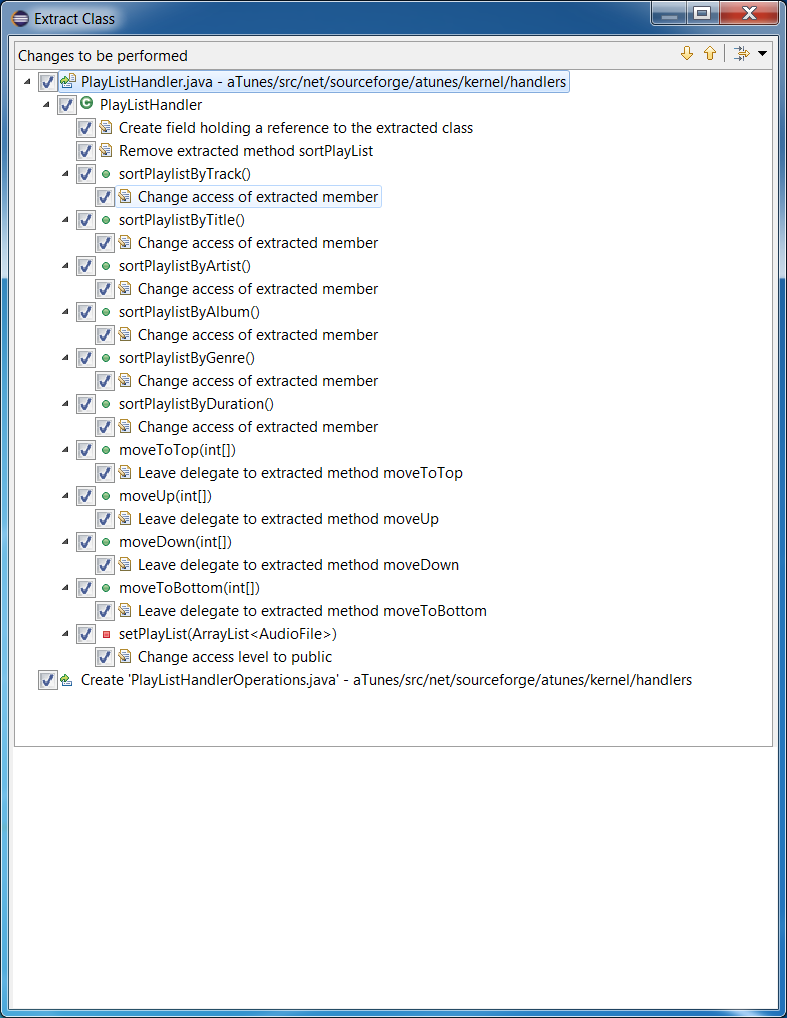
Refactoring of these classes spares developers from needing to remember a large number of attributes for a class.  
In many cases, splitting large classes into parts avoids duplication of code and functionality.

### Description and Rationale

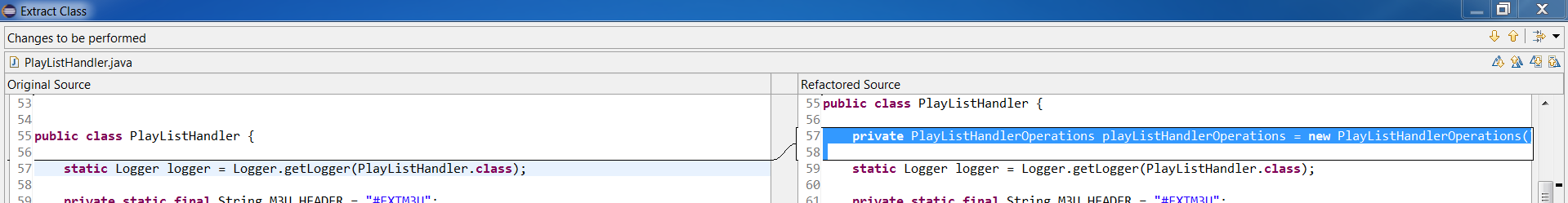
We have extracted some of the operations on playlist from God Class PlayListHandler.java (namely - sortPlaylistByTrack(), sortPlaylistByTitle(), sortPlaylistByArtist(), sortPlaylistByAlbum(), sortPlaylistByGenre(), sortPlaylistByDuration(), moveToTop(), moveUp(), moveDown(), moveToBottom()) to a new class PlayListHandlerOperations.java to simplify the former.

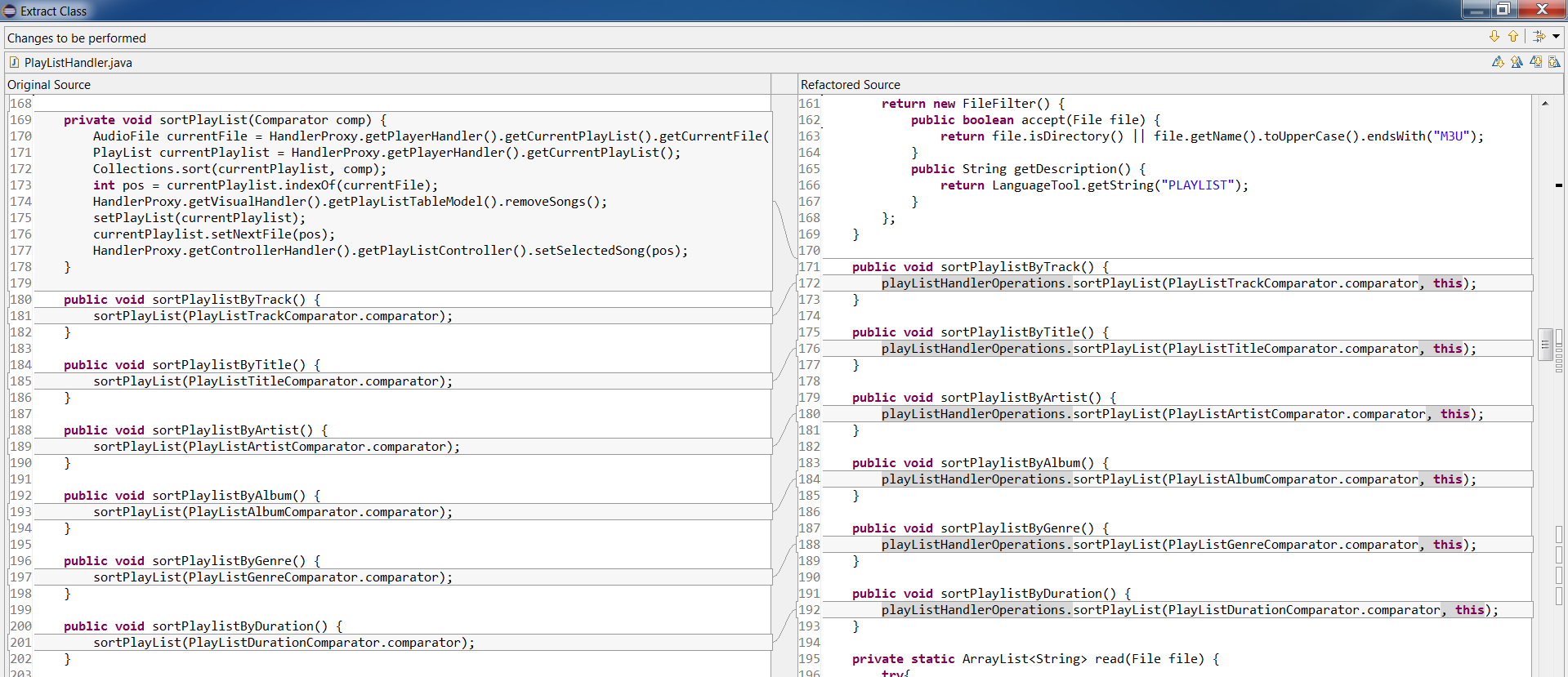
### Code Smell visualization

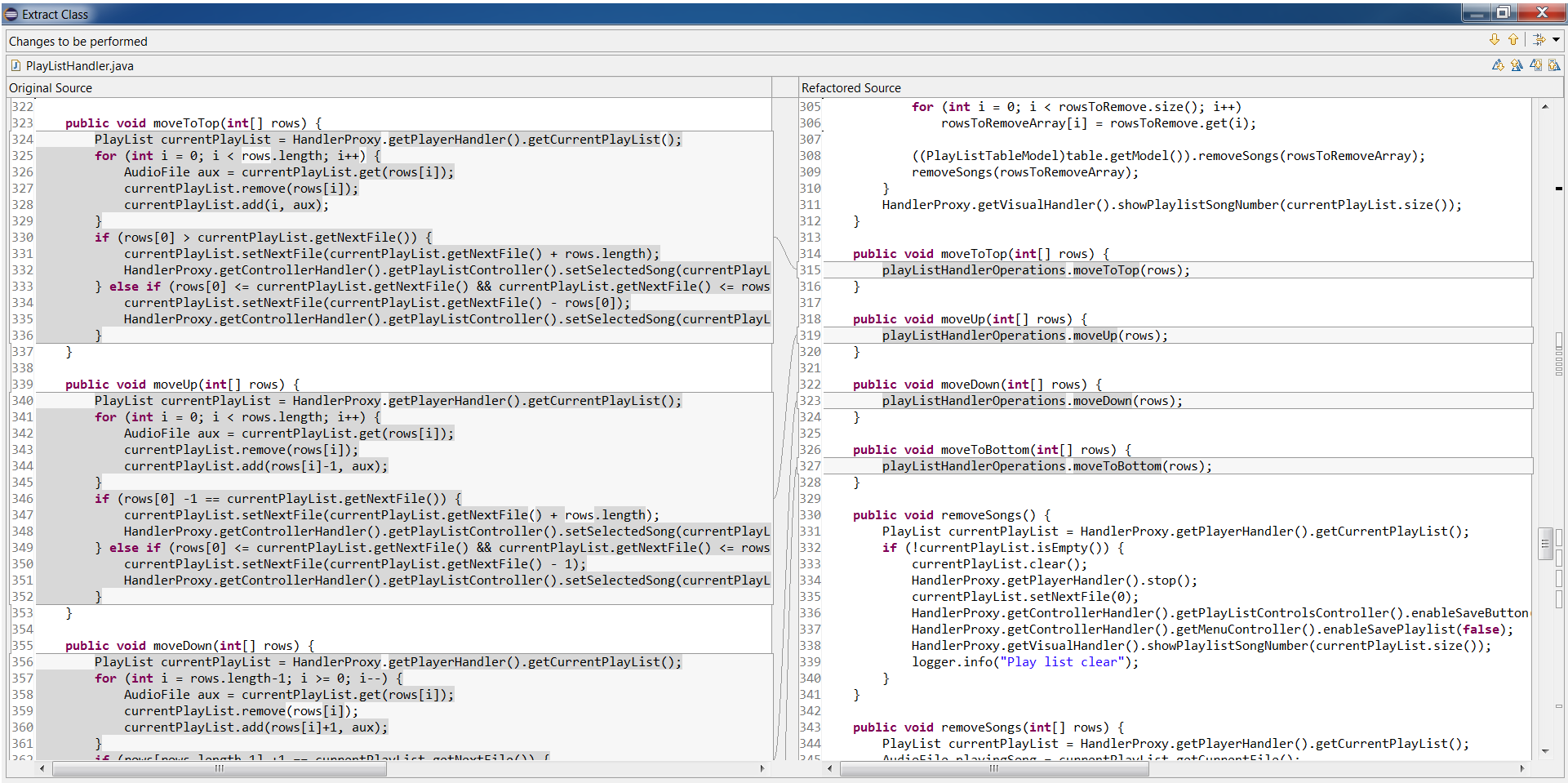


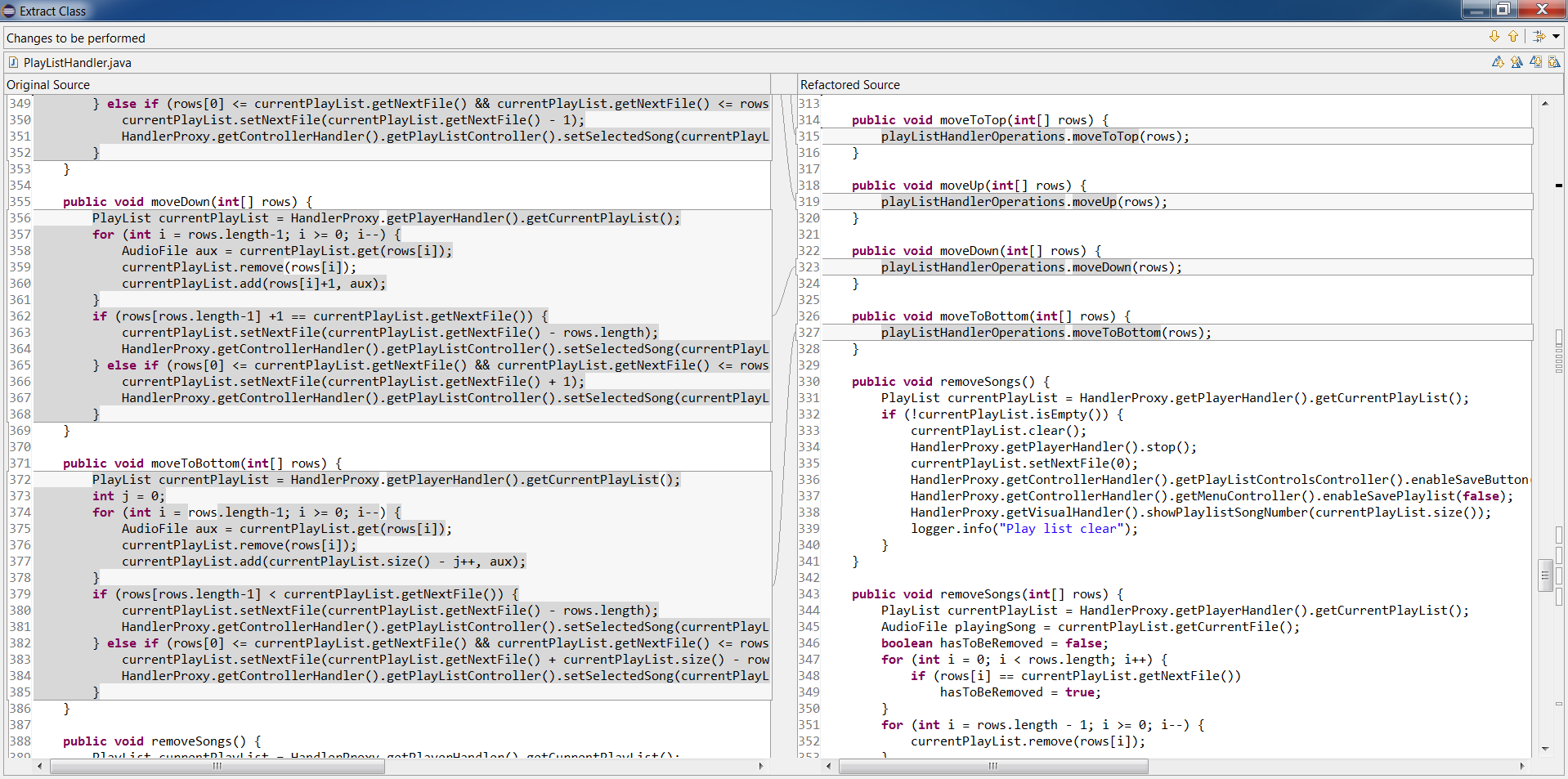


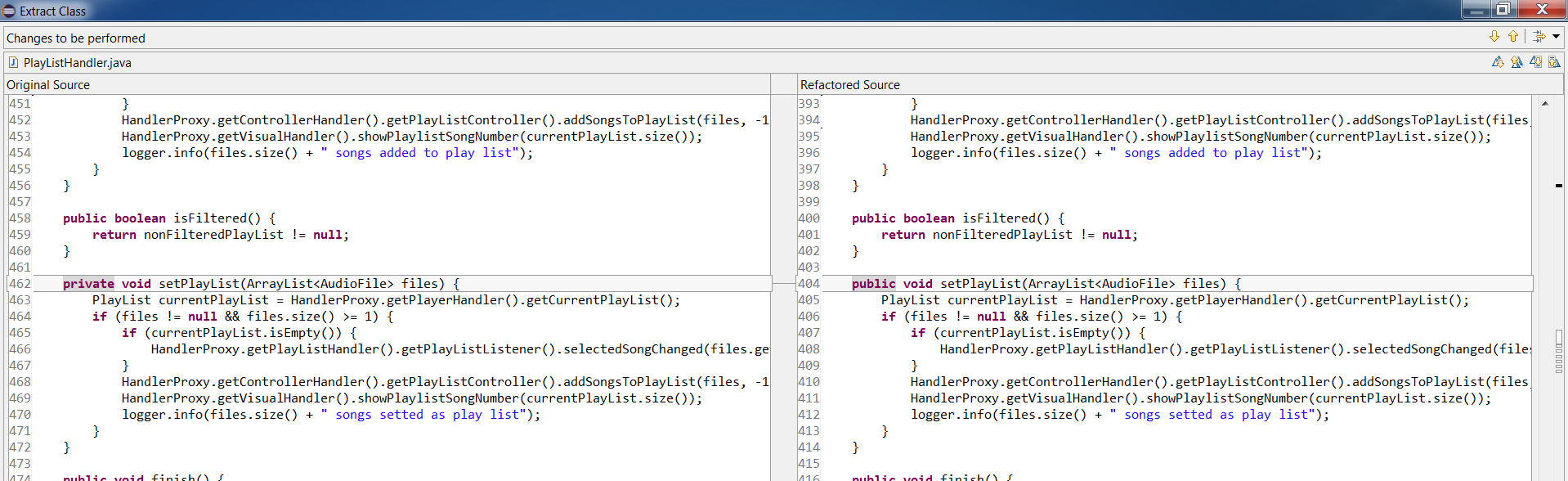
### Changes to be performed



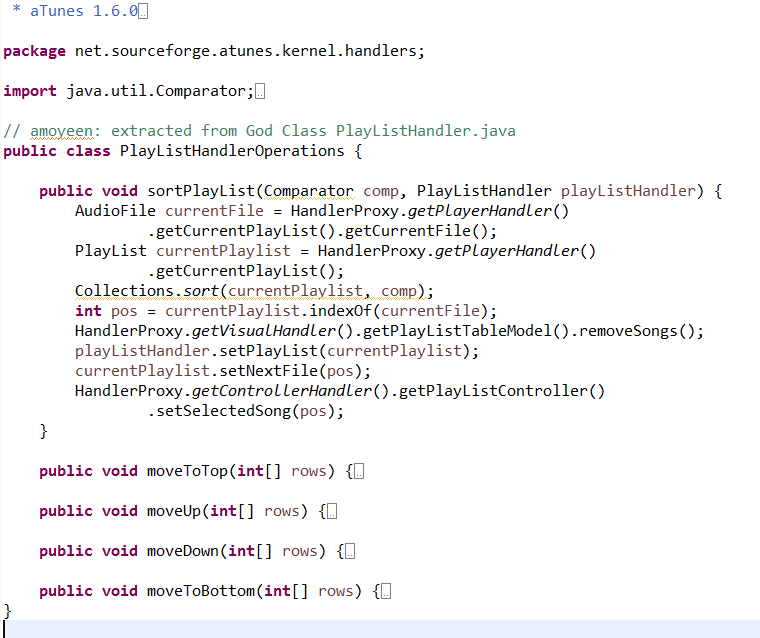








### Changed code

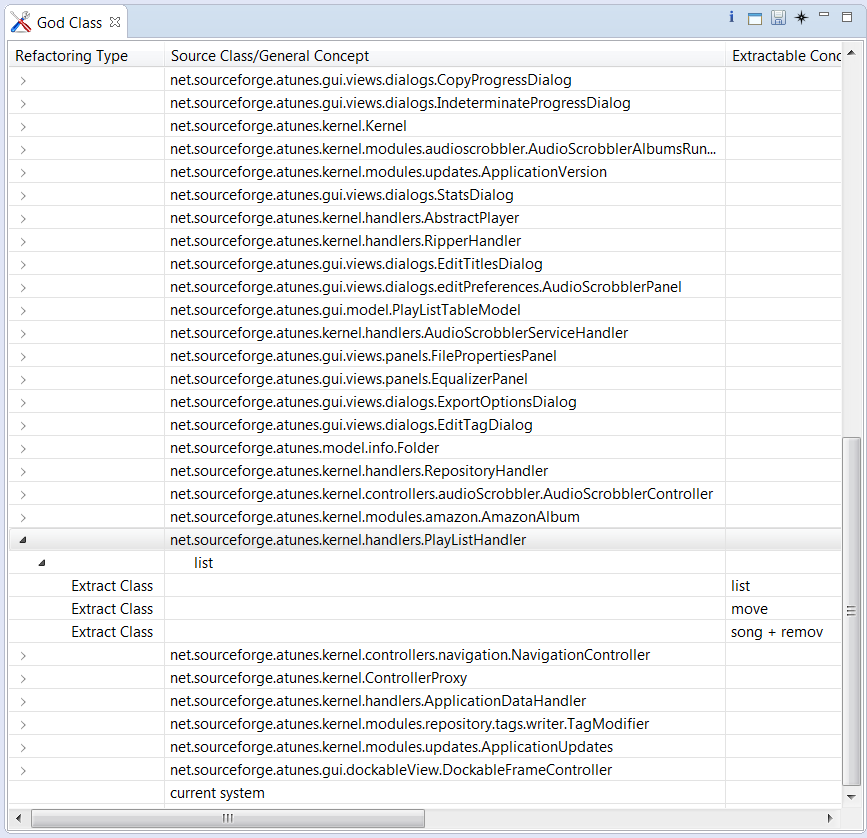




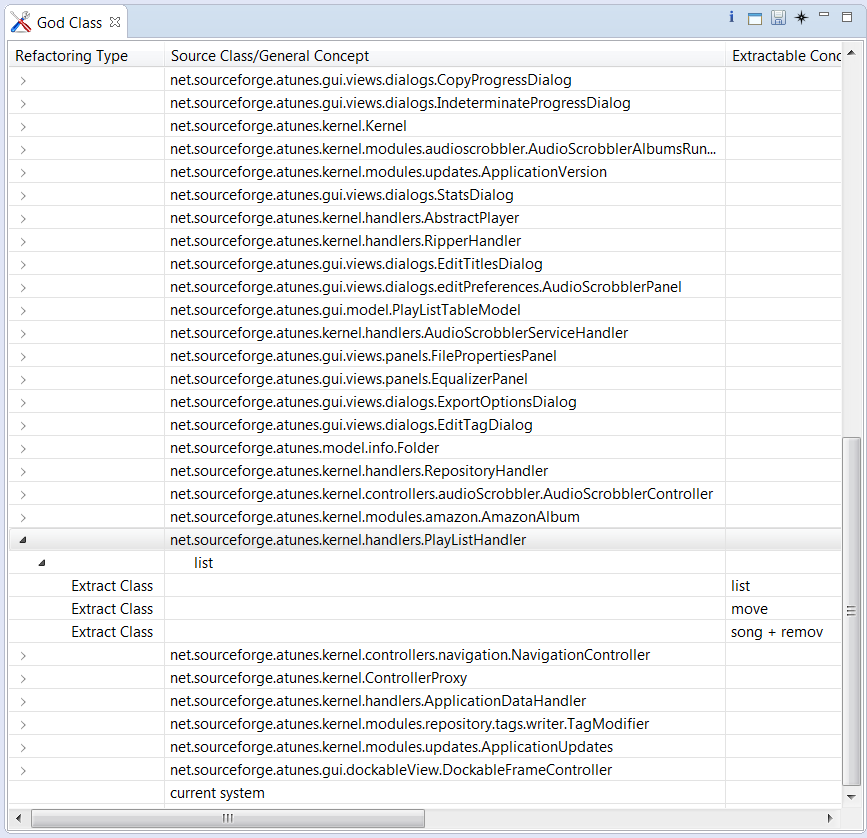
### Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case #** | **Description** | **Execution before Refactoring** | **Execution before Refactoring** |
| T1 | Demonstrate that program can be invoked successfully | Pass | Pass |
| T2 | Demonstrate that songs can be added to playlist | Pass | Pass |
| T3 | Demonstrate that playlist can be sorted by track | Pass | Pass |
| T4 | Demonstrate that playlist can be sorted by title | Pass | Pass |
| T5 | Demonstrate that playlist can be sorted by artist | Pass | Pass |
| T6 | Demonstrate that playlist can be sorted by album | Pass | Pass |
| T7 | Demonstrate that playlist can be sorted by genere | Pass | Pass |
| T8 | Demonstrate that Move to Top moves the selected song to the top of playlist | Pass | Pass |
| T9 | Demonstrate that Move Up moves the selected song up one place | Pass | Pass |
| T10 | Demonstrate that Move Down moves the selected song down one place | Pass | Pass |
| T11 | Demonstrate that Move to Bottom moves the selected song to the bottom of playlist | Pass | Pass |

### Tool result before refactoring



### Tool results after refactoring



## jEdit – removing “Type Checking” smell

### Justification for refactoring

• Improved code organization.

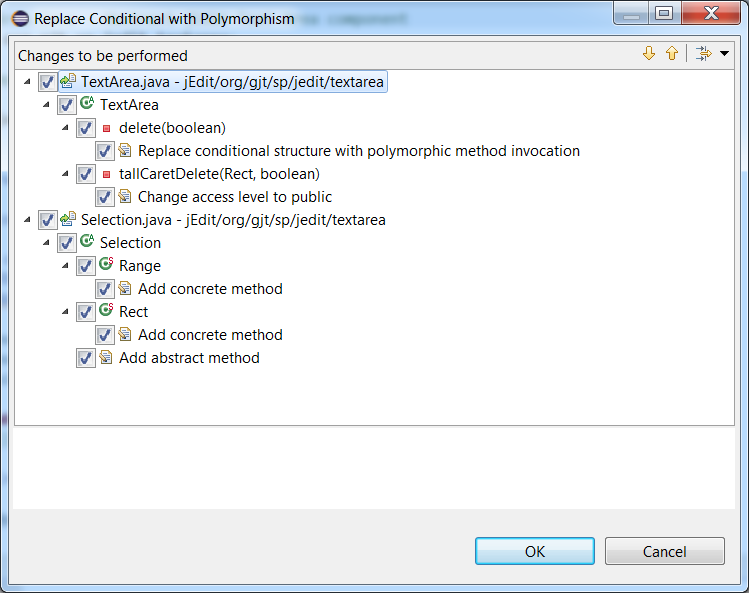
• Utilization of Object Orientation by removing Procedural elements.

• Reduction of complicated branching - simplifying the code base, making it easier to read and test.

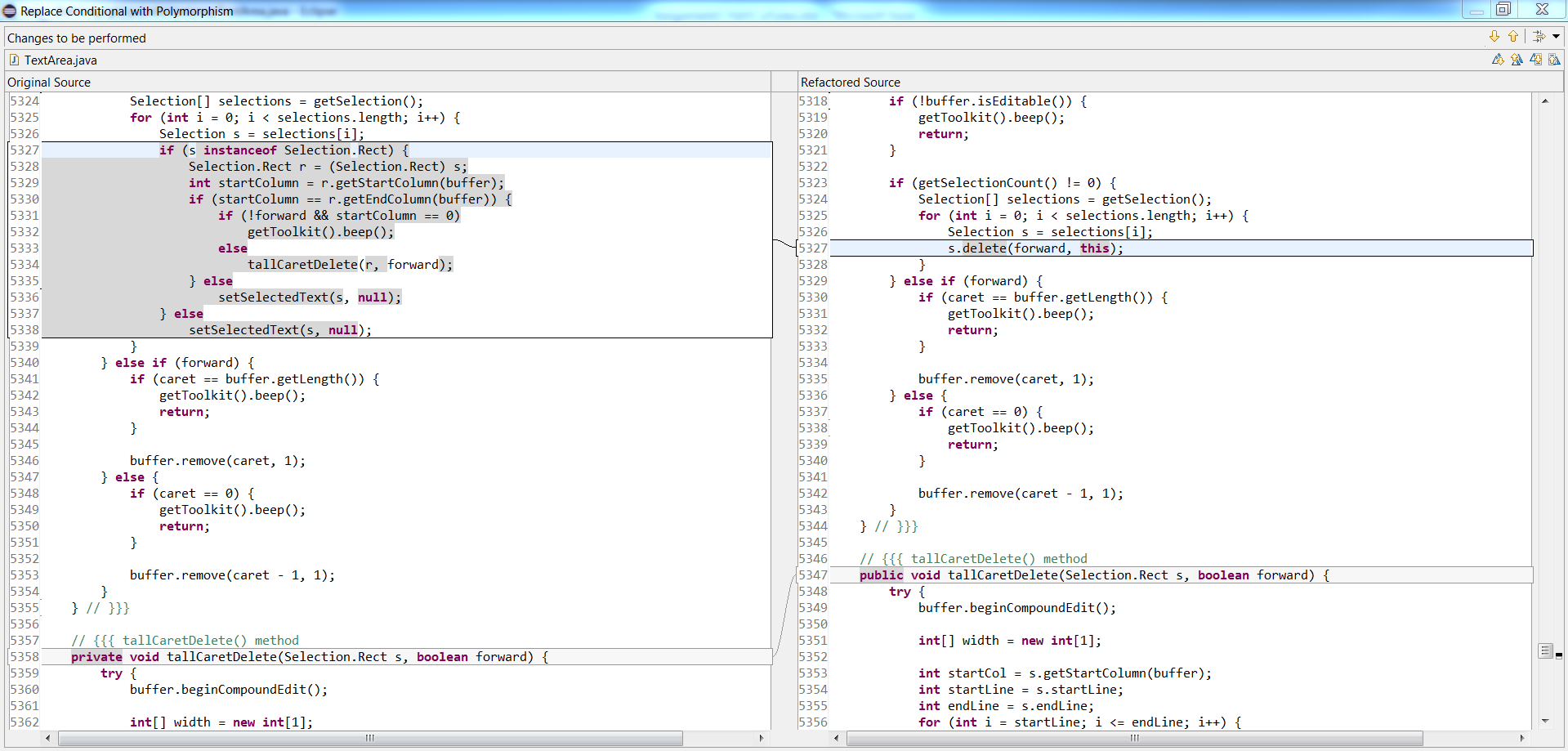
### Description and Rationale

A sequence of if statements in delete(boolean forward) method of TextArea.java class, where the delete operation was being performed based on different type of text selections, were replaced and removed the Type Checking/Switch Statements smell using polymorphism by adding an abstract delete() method in Selection.java abstract class and adding concrete delete() methods in both Range and Rect concrete classes that extend Selection class.

### Code smell visualization



### Changes to be performed



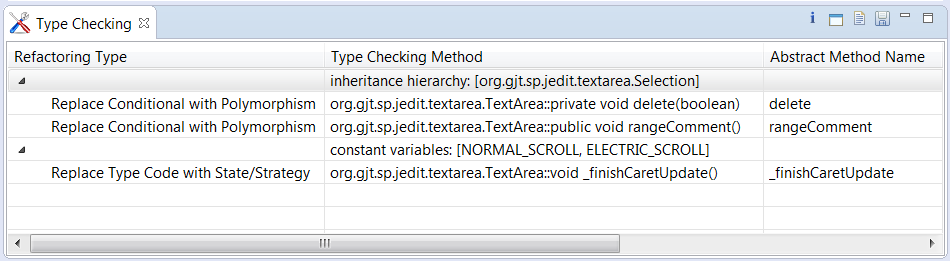
### Changed code



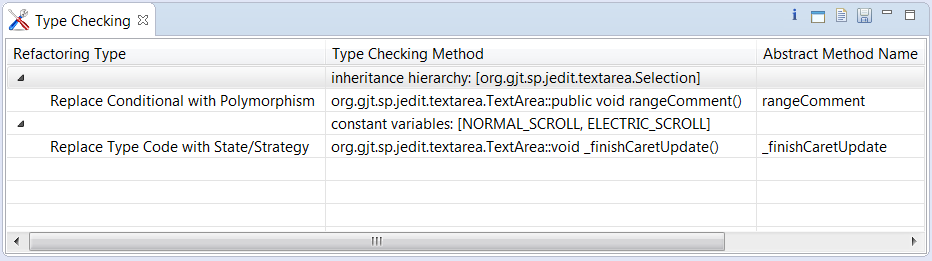
### Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case #** | **Description** | **Execution before Refactoring** | **Execution before Refactoring** |
| T1 | Demonstrate that program can be invoked successfully | Pass | Pass |
| T2 | Demonstrate that an existing test file can be opened | Pass | Pass |
| T3 | Demonstrate that part of text from open text file can be selected | Pass | Pass |
| T4 | Demonstrate that selected text can be deleted using [Delete] key on the keyboard | Pass | Pass |
| T5 | Demonstrate that selected text can be deleted using toolbar icon | Pass | Pass |
| T6 | Demonstrate that selected lines can be deleted using Edit > Text > Delete Lines | Pass | Pass |
| T7 | Demonstrate that text from current cursor position to start of line can be deleted using Edit > Text > Delete to Start of Line | Pass | Pass |
| T8 | Demonstrate that text from current cursor position to end of line can be deleted using Edit > Text > Delete to End of Line | Pass | Pass |
| T9 | Demonstrate that whole paragraph where cursor currently is can be deleted using Edit > Text > Delete Paragraph | Pass | Pass |

### Tool result before refactoring



### Tool result after refactoring



# Manual Refactoring

## aTunes – removing “Feature Envy” smell

### Justification for refactoring

• Less code duplication (if the data handling code is put in a central place).

• Better code organization (methods for handling data are next to the actual data).

### Description and rationale

Protected method addBindings() in PlayListControlsController.java was heavily using data from external class PlayListControlsPanel.java to add bindings to PlaylistControlsListener.java class. So, we have created a public method addBindings(PlaylistControlsListener listener) in PlayListControlsPanel.java class that takes an instance of PlaylistControlsListener as an input and adds the bindings to its properties accordingly - which then is called from PlayListControlsController.java.

### Code before refactoring

/aTunes/src/net/sourceforge/atunes/kernel/controllers/playListControls/PlayListControlsController.java



### Refactored code

/aTunes/src/net/sourceforge/atunes/kernel/controllers/playListControls/PlayListControlsController.java



/aTunes/src/net/sourceforge/atunes/gui/views/panels/PlayListControlsPanel.java



### Tests

| **Test Case #** | **Description** | **Execution before Refactoring** | **Execution before Refactoring** |
| --- | --- | --- | --- |
| T1 | Demonstrate that program can be invoked successfully | Pass | Pass |
| T2 | Demonstrate that songs can be added to playlist | Pass | Pass |
| T3 | Demonstrate that playlist can be sorted by track | Pass | Pass |
| T4 | Demonstrate that playlist can be sorted by title | Pass | Pass |
| T5 | Demonstrate that playlist can be sorted by artist | Pass | Pass |
| T6 | Demonstrate that playlist can be sorted by album | Pass | Pass |
| T7 | Demonstrate that playlist can be sorted by genere | Pass | Pass |
| T8 | Demonstrate that Save Playlist invokes save dialog box | Pass | Pass |
| T9 | Demonstrate that Load Playlist invokes load dialog box | Pass | Pass |
| T10 | Demonstrate that Move to Top moves the selected song to the top of playlist | Pass | Pass |
| T11 | Demonstrate that Move Up moves the selected song up one place | Pass | Pass |
| T12 | Demonstrate that Remove deletes the selected song from playlist | Pass | Pass |
| T13 | Demonstrate that Move Down moves the selected song down one place | Pass | Pass |
| T14 | Demonstrate that Move to Bottom moves the selected song to the bottom of playlist | Pass | Pass |
| T15 | Demonstrate that Info invokes info dialog box | Pass | Pass |
| T16 | Demonstrate that Clear Playlist deletes all songs from playlist | Pass | Pass |
| T17 | Demonstrate that Set as Favorite Song sets selected song as favorite song | Pass | Pass |
| T18 | Demonstrate that Set as Favorite Album sets album of selected song as favorite album | Pass | Pass |
| T19 | Demonstrate that Set as Favorite Artist sets artist of selected song as favorite artist | Pass | Pass |
| T20 | Demonstrate that Options > Show Track Number checkbox can show/hide track number column in playlist | Pass | Pass |
| T21 | Demonstrate that Options > Show Artist checkbox can show/hide artist column in playlist | Pass | Pass |
| T22 | Demonstrate that Options > Show Genre checkbox can show/hide genre column in playlist | Pass | Pass |
| T23 | Demonstrate that Options > Show Duration checkbox can show/hide duration column in playlist | Pass | Pass |
| T24 | Demonstrate that Options > Show Album checkbox can show/hide album column in playlist | Pass | Pass |
| T25 | Demonstrate that Set Artist as Playlist sets only songs from artist of selected song as playlist | Pass | Pass |
| T26 | Demonstrate that Set Album as Playlist sets only album of selected song as playlist | Pass | Pass |

### Tool results

|  |  |
| --- | --- |
| **Tool result before refactoring** | **Tool result after refactoring** |
|  |  |

## jEdit – removing “Internal duplication” smell

### Justification for refactoring

• Merging duplicate code simplifies the structure of the code and makes it shorter.

• Simplification + Shortness = code that is easier to simplify and cheaper to support.

### Description and Rationale

Public methods toUpperCase() and toLowerCase() in TextArea.java were exactly identical except just in one line where they were calling different String extensions of toUpperCase() and toLowerCase() respectively. So, we created a private method changeCase(String toCase) with an input parameter that tells which case to change to - and called that private method from the formers with appropriate inputs.

### Code before refactoring

/jEdit/org/gjt/sp/jedit/textarea/TextArea.java

|  |  |
| --- | --- |
|  |  |
| Refactored code |  |



### Tests

|  |  |  |  |
| --- | --- | --- | --- |
| **Test Case #** | **Description** | **Execution before Refactoring** | **Execution before Refactoring** |
| T1 | Demonstrate that program can be invoked successfully | Pass | Pass |
| T2 | Demonstrate that an existing test file can be opened | Pass | Pass |
| T3 | Demonstrate that part of text from open text file can be selected | Pass | Pass |
| T4 | Demonstrate that selected text can be converted to Upper Case using Edit > Text > To Upper Case menu item | Pass | Pass |
| T5 | Demonstrate that selected text can be converted to Lower Case using Edit > Text > To Lower Case menu item | Pass | Pass |

### Tool results

|  |  |
| --- | --- |
| **Tool result before refactoring** | **Tool result after refactoring** |
|  |  |

# References

*Bad Smells detailed descriptions and solutions*. (n.d.). Retrieved from https://sourcemaking.com/refactoring/smells

*InCode User Documentation*. (n.d.). Retrieved from https://www.intooitus.com/products/incode

*Source Meter for Java User Guide*. (n.d.). Retrieved from https://www.sourcemeter.com/resources/java/