Assignment # 2, Fall 2015

SE 6356 Software Maintenance, Evolution and Re-Engineering

Mahabubul Alam

Abdullah Moyeen

TABLE OF CONTENTS

1 CLASS COHESION IN OO SOFTWARE

1.1 SYSTEM: JEDIT

1.1.1 The top 2 most cohesive classes, based on source-meter 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.

1.1.2 The top 2 least cohesive classes, based on source-meter 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 source-meter 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:

```
loadIcon(String)
confirm(Component,String,Object[],int,int)
error(Component,String,Object[])
message(Component,String,Object[])
showVFSFileDialog(View,String,int,boolean)
loadGeometry(Window,String)
saveGeometry(Window,String)
showPopupMenu(JPopupMenu,Component,int,int)
```

1.2 SYSTEM: ATUNES

1.2.1 The top 2 most Cohesive Classes, based on source-meter 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.

1.2.2 The top 2 least Cohesive Classes, based on source-meter 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 source-meter 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,

```
setSongProperties()
setWindowLocation()
setShowAlbumInPlayList()
```

- 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.

2 CODE SMELLS USING JDEODORANT AND INCODE

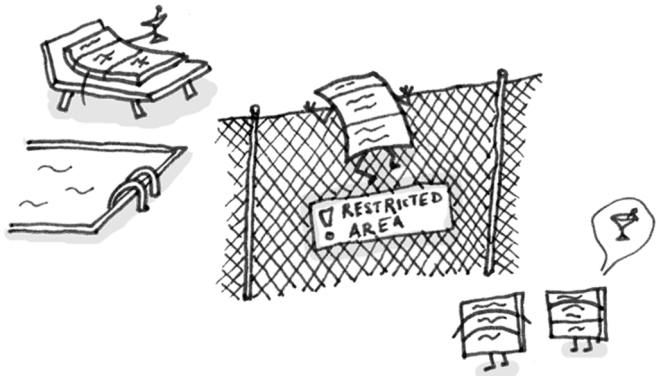
2.1 ATUNES CODE SMELLS

2.1.1 Smell: Feature Envy - package 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**.

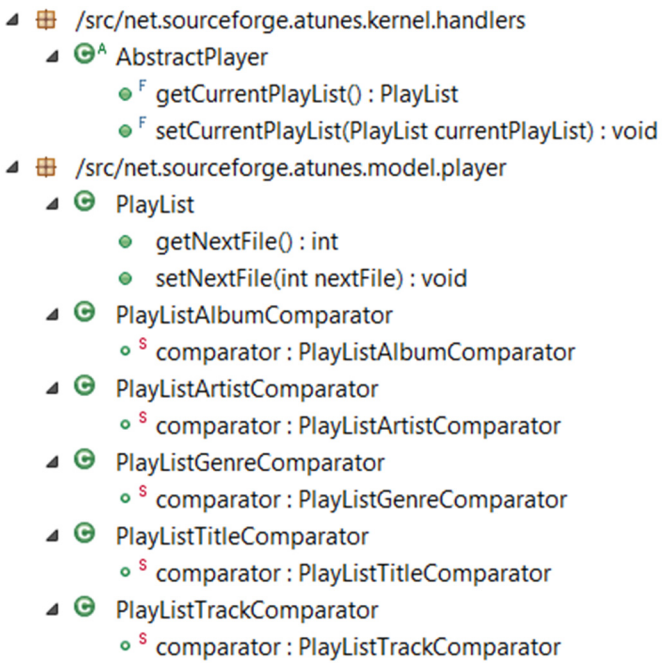
```
	protected void addBindings() {
		final PlayListControlsPanel panel = (PlayListControlsPanel)
panelControlled;

		PlayListControlsListener listener = new PlayListControlsListener(panel);

		panel.getSortByTrack().addActionListener(listener);
		panel.getSortByTitle().addActionListener(listener);
		panel.getSortByArtist().addActionListener(listener);
		panel.getSortByAlbum().addActionListener(listener);
		panel.getSortByGenre().addActionListener(listener);
		panel.getSavePlaylistButton().addActionListener(listener);
		panel.getLoadPlaylistButton().addActionListener(listener);
		panel.getTopButton().addActionListener(listener);
		panel.getUpButton().addActionListener(listener);
		panel.getDeleteButton().addActionListener(listener);
		panel.getDownButton().addActionListener(listener);
		panel.getBottomButton().addActionListener(listener);
		panel.getInfoButton().addActionListener(listener);
		panel.getClearButton().addActionListener(listener);
		panel.getFavoriteSong().addActionListener(listener);
		panel.getFavoriteAlbum().addActionListener(listener);
		panel.getFavoriteArtist().addActionListener(listener);
		panel.getShowTrack().addActionListener(listener);
		panel.getShowArtist().addActionListener(listener);
		panel.getShowGenre().addActionListener(listener);
		panel.getShowAlbum().addActionListener(listener);
		panel.getArtistButton().addActionListener(listener);
		panel.getAlbumButton().addActionListener(listener);
	}
```

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



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

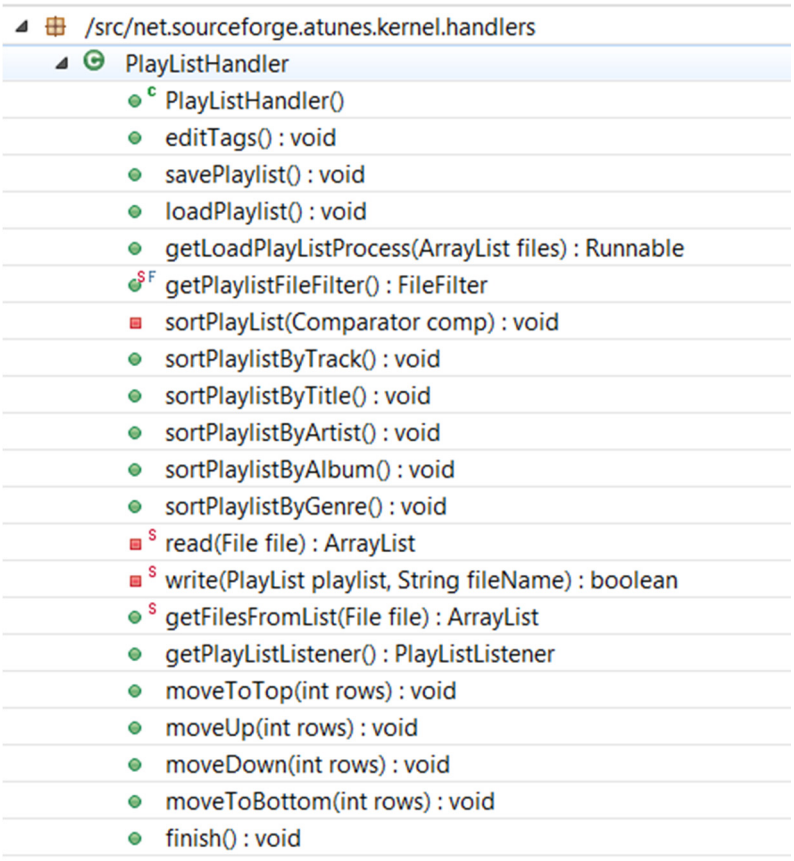
The **PlayListHandler** class uses many attributes from external classes –



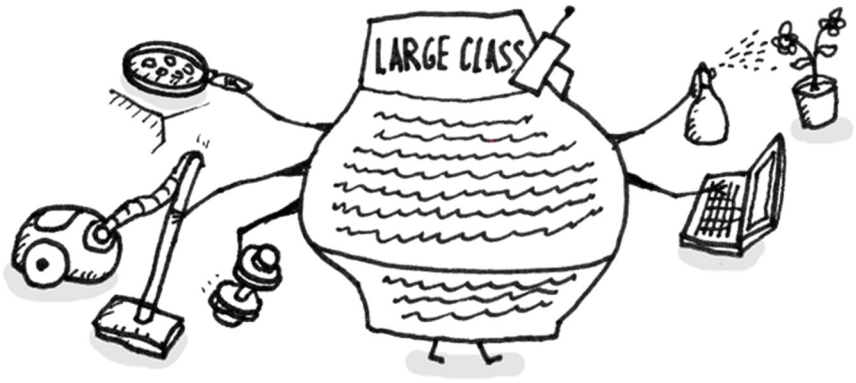
The **PlayListHandler** class is excessively large and complex, due to its methods having a high cyclomatic complexity and nesting level

Name	CYCLO
setFilter(String filter) : void	9
setPlayListAfterFiltering(PlayList playList) : void	9
removeSongs(int rows) : void	7
addToPlayList(ArrayList files) : void	6
savePlaylist() : void	6
moveDown(int rows) : void	5
moveToTop(int rows) : void	5
moveUp(int rows) : void	5
moveToBottom(int rows) : void	5

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.

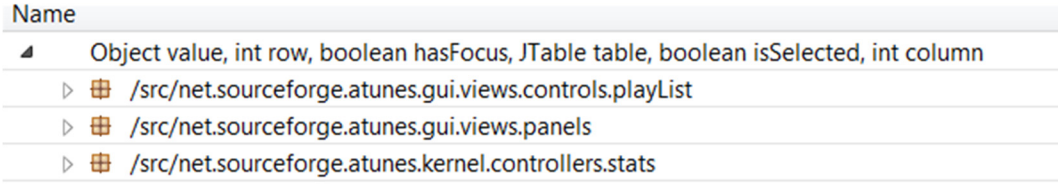


2.1.3 Smell: Data Clumps - net.sourceforge.atunes.kernel.controllers.stats

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.

```
149             public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
150                 JLabel l = (JLabel) super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
151                 l.setHorizontalAlignment(SwingConstants.RIGHT);
152                 return l;
153             }
154         });
```

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



2.2 JEDIT CODE SMELLS

2.2.1 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,

```
private void delete(boolean forward)
{
    if(!buffer.isEditable())
    {
        getToolkit().beep();
        return;
    }

    if(getSelectionCount() != 0)
    {
        Selection[] selections = getSelection();
        for(int i = 0; i < selections.length; i++)
        {
            Selection s = selections[i];
            if(s instanceof Selection.Rect)
            {
                Selection.Rect r = (Selection.Rect)s;
                int startColumn = r.getStartColumn(buffer);
                if(startColumn == r.getEndColumn(buffer))
                {
                    if(!forward && startColumn == 0)
                        getToolkit().beep();
                    else
                        tallCaretDelete(r,forward);
                }
                else
                    setSelectedText(s,null);
            }
            else
                setSelectedText(s,null);
```

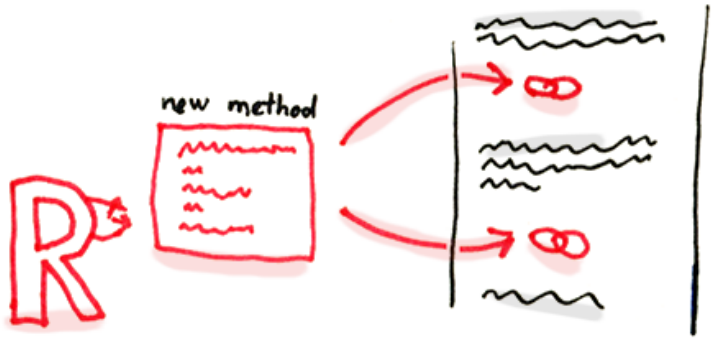
2.2.2 Smell: Internal duplication - org.gjt.sp.jedit.textarea

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.

```
//{{{ toLowerCase() method
    /**
     * Converts the selected text to lower case.
     * @since jEdit 2.7pre2
     */
    public void toLowerCase()
    {
        if(!buffer.isEditable())
        {
            getToolkit().beep();
            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);
            getToolkit().beep();
            return;
        }

        buffer.beginCompoundEdit();

        for (int i = 0; i < selection.length; i++)
        {
            Selection s = selection[i];
            setSelectedText(s,getSelectedText(s).toLowerCase());
        }

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

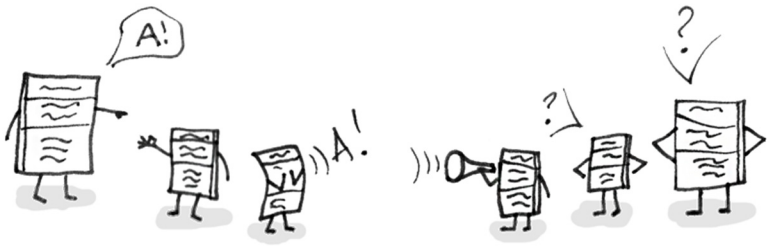
```
//{{{ toUpperCase() method
    /**
     * Converts the selected text to upper case.
     * @since jEdit 2.7pre2
     */
    public void toUpperCase()
    {
        if(!buffer.isEditable())
        {
            getToolkit().beep();
            return;
        }

        Selection[] selection = getSelection();
        int caret = -1;
        if (selection.length == 0)
        {
            caret = getCaretPosition();
            selectWord();
            selection = getSelection();
        }
        if (selection.length == 0)
        {
            if (caret != -1)
                setCaretPosition(caret);
            getToolkit().beep();
            return;
        }

        buffer.beginCompoundEdit();

        for(int i = 0; i < selection.length; i++)
        {
            Selection s = selection[i];
            setSelectedText(s,getSelectedText(s).toUpperCase());
        }

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

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.

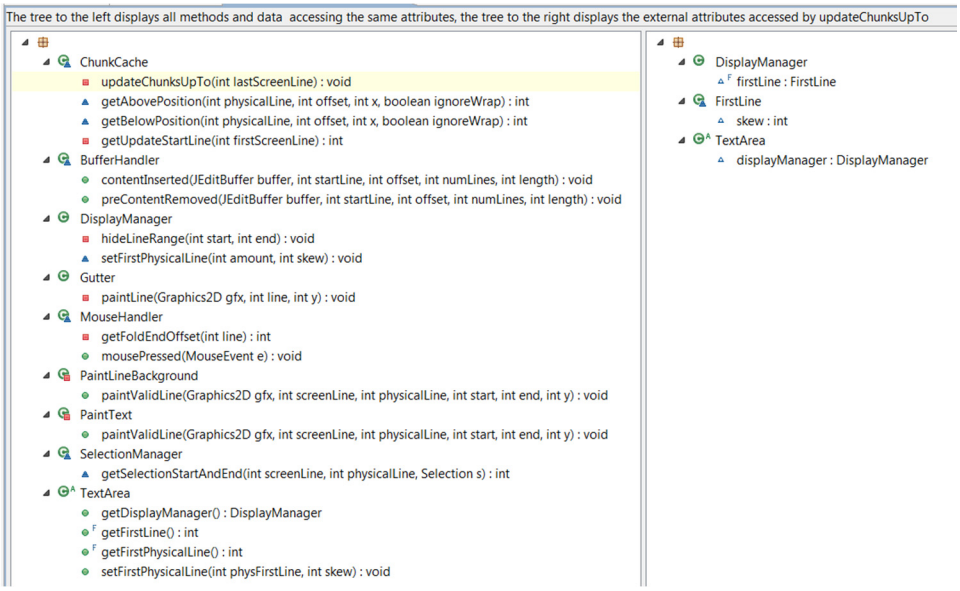


2.2.3 Smell: Message Chains - org.gjt.sp.jedit.textarea

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:

```
    if(textArea.displayManager.firstLine.skew > 0)
    {
        Log.log(Log.ERROR,this,"BUG: skew=" + textArea.displayManager.firstLine.skew +
        textArea.displayManager.firstLine.skew = 0;
        needFullRepaint = true;
        lastScreenLine = lineInfo.length - 1;
    }
}
chunks = null;
offset = 0;
length = 1;

otherwise, the number of subregions
se

if(i == 0)
{
    int skew = textArea.displayManager.firstLine.skew;
    if(skew >= out.size())
```

3 REFACTORING USING TOOL SUPPORT

3.1 ATUNES – REMOVING "GOD CLASS" SMELL

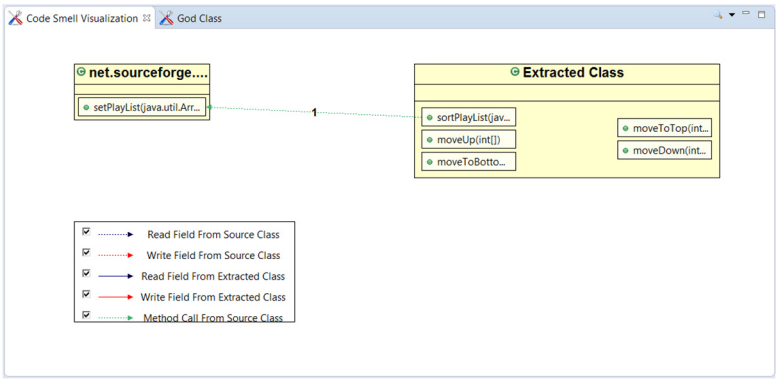
3.1.1 Justification of refactoring

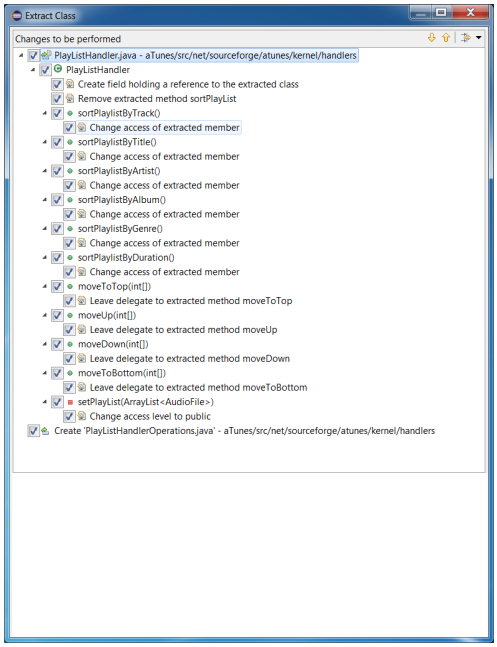
Refactoring of these classes spares developers from needing to remember a large number of attributes for a class.
In many cases, splitting large classes into parts avoids duplication of code and functionality.

3.1.2 Description and Rationale

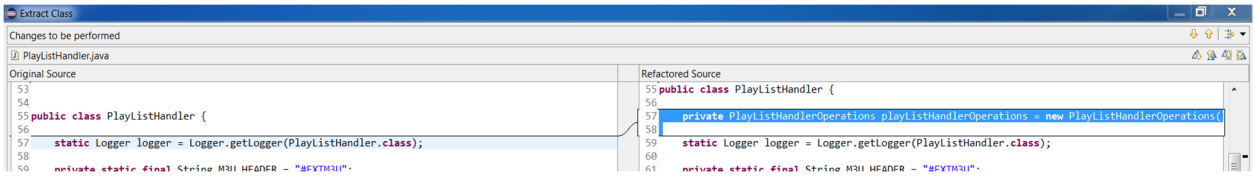
We have extracted some of the operations on playlist from God Class PlayListHandler.java (namely - sortPlaylistByTrack(), sortPlaylistByTitle(), sortPlaylistByArtist(), sortPlaylistByAlbum(), sortPlaylistByGenre(), sortPlaylistByDuration(), moveToTop(), moveUp(), moveDown(), moveToBottom()) to a new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

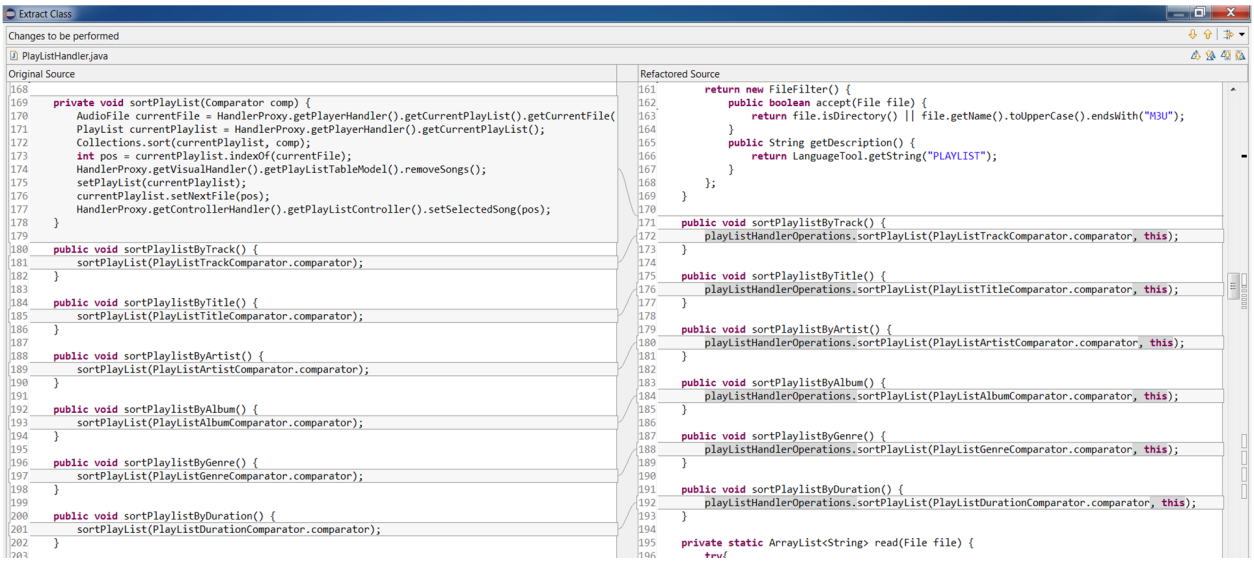
3.1.3 Code Smell visualization

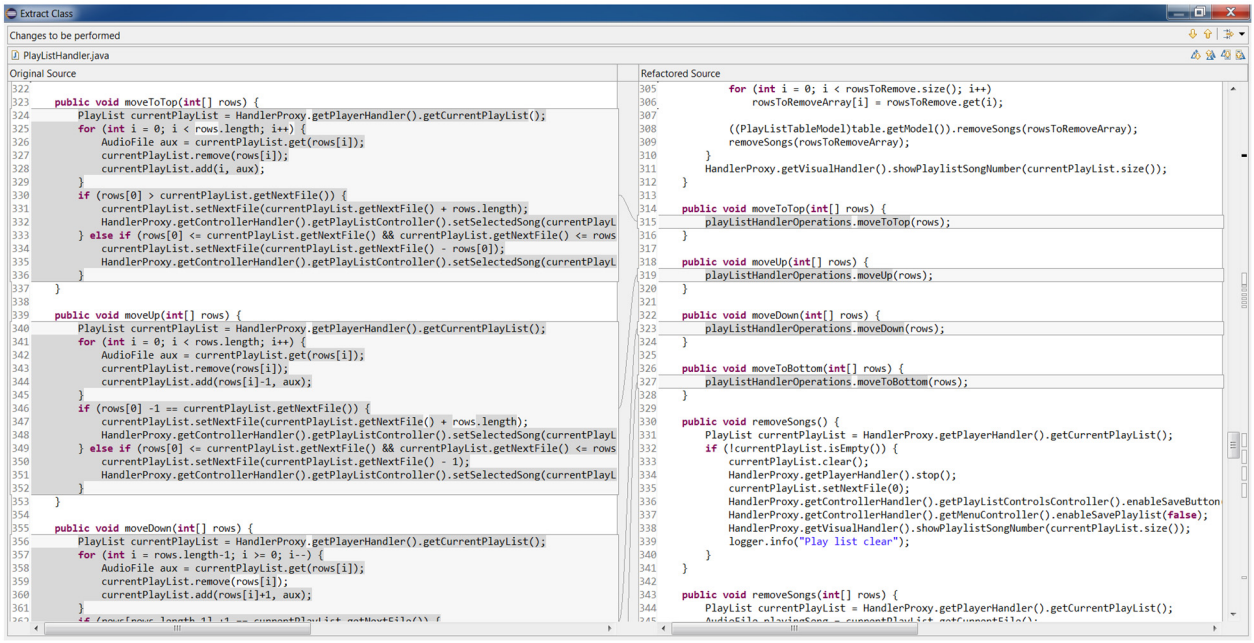


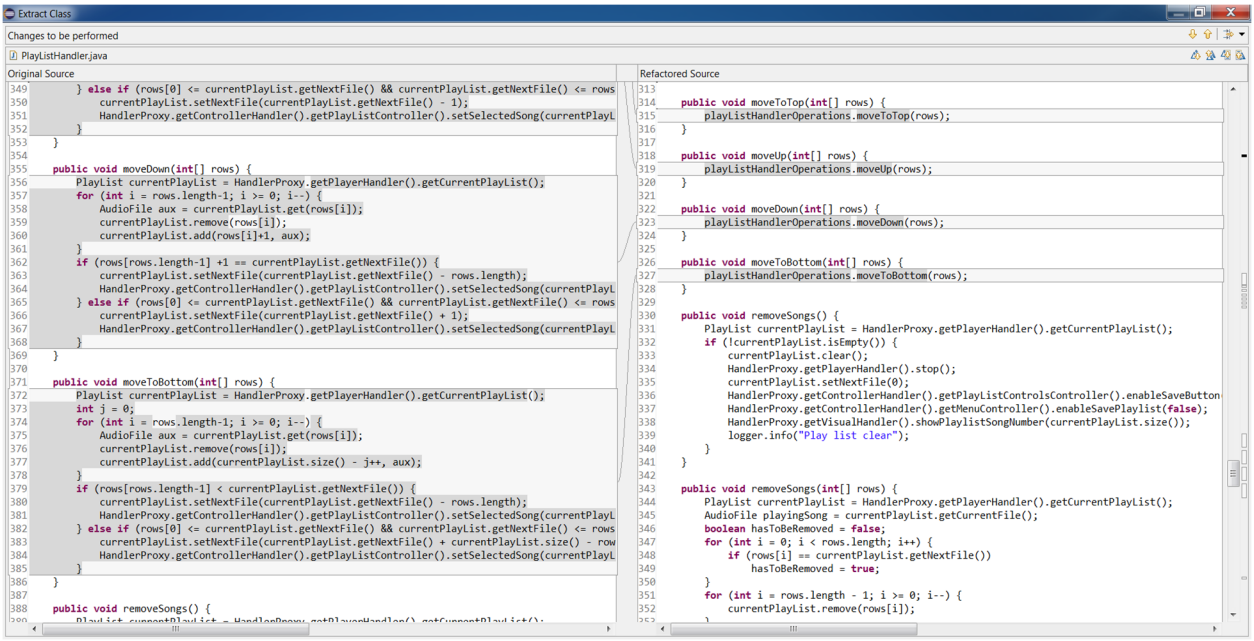


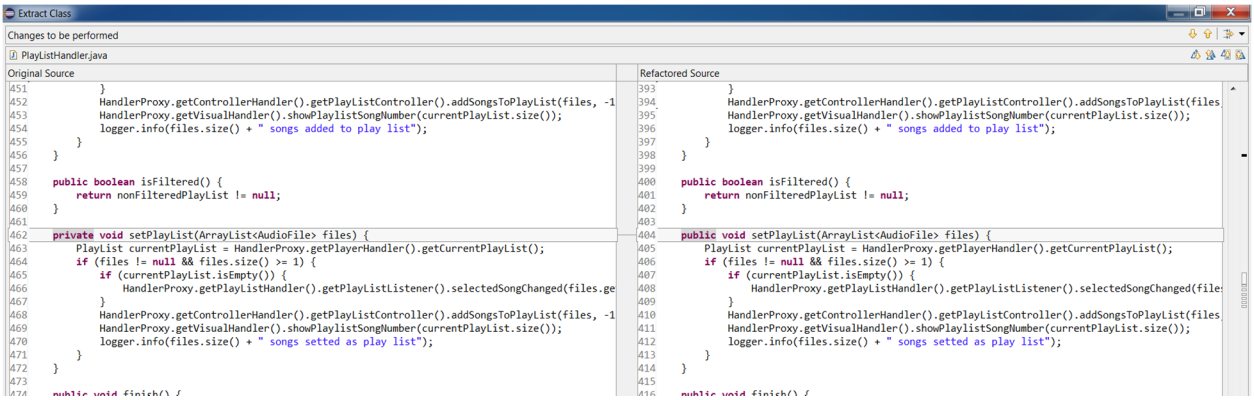
3.1.4 Changes to be performed








Extract Class
Changes to be performed
PlayListHandler.java
Original Source
Refactored Source


Extract Class
Changes to be performed
PlayListHandler.java
Original Source
Refactored Source

3.1.5 Changed code

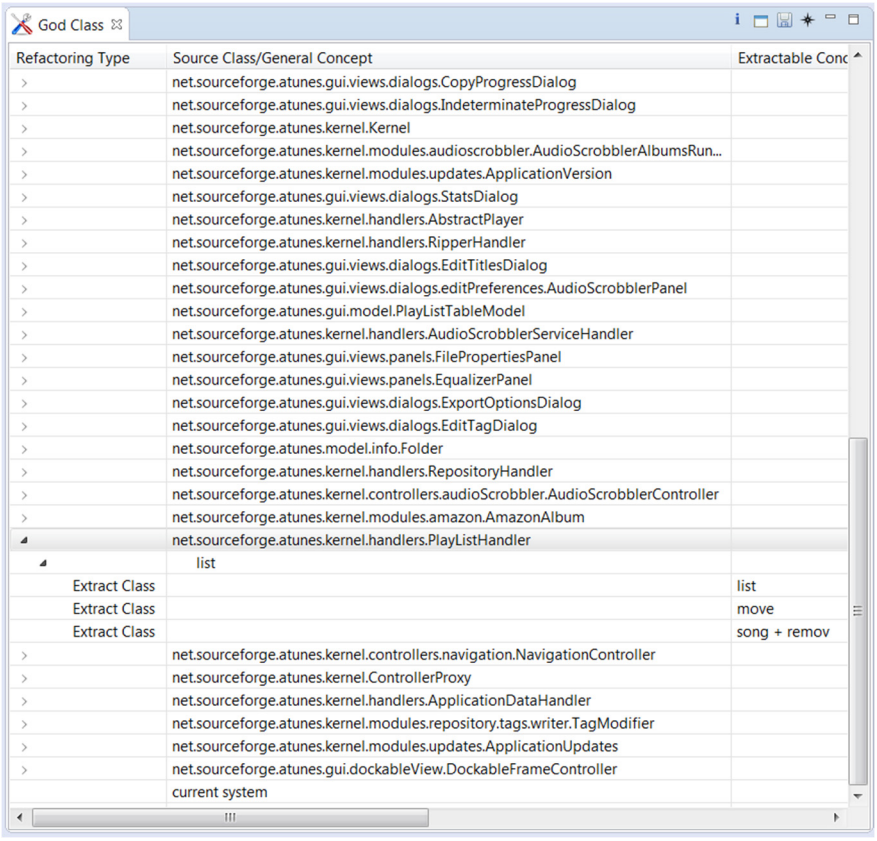
```
 * aTunes 1.6.0

package net.sourceforge.atunes.kernel.handlers;

import java.util.Comparator;

// amoyeen: extracted from God Class PlayListHandler.java
public class PlayListHandlerOperations {

    public void sortPlayList(Comparator comp, PlayListHandler playListHandler) {
        AudioFile currentFile = HandlerProxy.getPlayerHandler()
                .getCurrentPlayList().getCurrentFile();
        PlayList currentPlaylist = HandlerProxy.getPlayerHandler()
                .getCurrentPlayList();
        Collections.sort(currentPlaylist, comp);
        int pos = currentPlaylist.indexOf(currentFile);
        HandlerProxy.getVisualHandler().getPlayListTableModel().removeSongs();
        playListHandler.setPlayList(currentPlaylist);
        currentPlaylist.setNextFile(pos);
        HandlerProxy.getControllerHandler().getPlayListController()
                .setSelectedSong(pos);
    }

    public void moveToTop(int[] rows) {

    public void moveUp(int[] rows) {

    public void moveDown(int[] rows) {

    public void moveToBottom(int[] rows) {
}
```

```
    public void sortPlaylistByTrack() {
        playListHandlerOperations.sortPlayList(PlayListTrackComparator.comparator, this);
    }

    public void sortPlaylistByTitle() {
        playListHandlerOperations.sortPlayList(PlayListTitleComparator.comparator, this);
    }

    public void sortPlaylistByArtist() {
        playListHandlerOperations.sortPlayList(PlayListArtistComparator.comparator, this);
    }

    public void sortPlaylistByAlbum() {
        playListHandlerOperations.sortPlayList(PlayListAlbumComparator.comparator, this);
    }

    public void sortPlaylistByGenre() {
        playListHandlerOperations.sortPlayList(PlayListGenreComparator.comparator, this);
    }

    public void sortPlaylistByDuration() {
        playListHandlerOperations.sortPlayList(PlayListDurationComparator.comparator, this);
    }

    public void moveToTop(int[] rows) {
        playListHandlerOperations.moveToTop(rows);
    }

    public void moveUp(int[] rows) {
        playListHandlerOperations.moveUp(rows);
    }

    public void moveDown(int[] rows) {
        playListHandlerOperations.moveDown(rows);
    }

    public void moveToBottom(int[] rows) {
        playListHandlerOperations.moveToBottom(rows);
    }
```

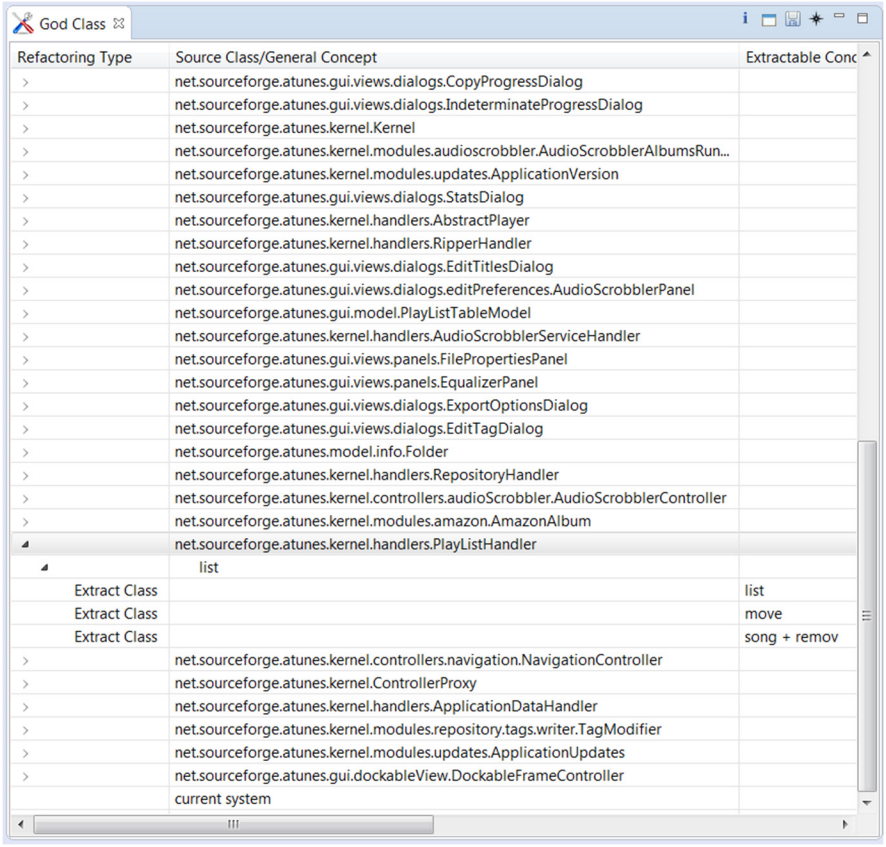
3.1.6 Tests

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

3.1.7 Tool result before refactoring



3.1.8 Tool results after refactoring



3.2 JEDIT – REMOVING "TYPE CHECKING" SMELL

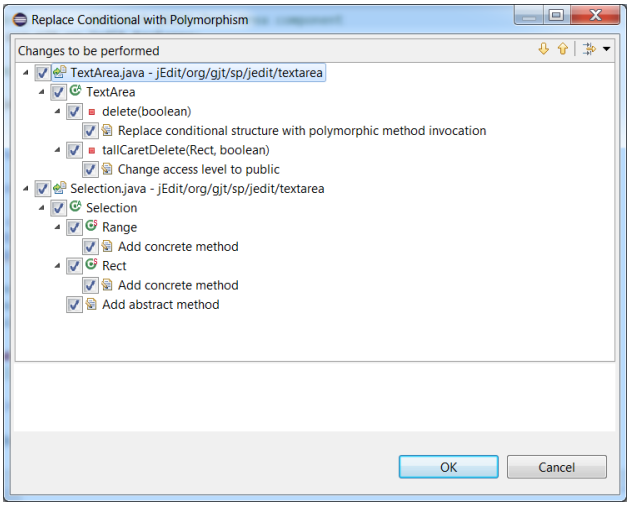
3.2.1 Justification for refactoring

- 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.

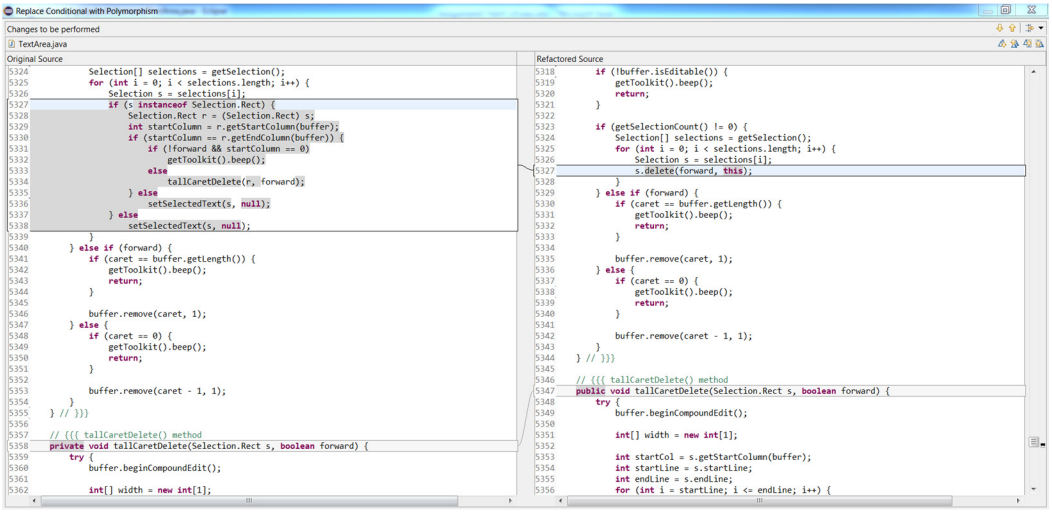
3.2.2 Description and Rationale

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.

3.2.3 Code smell visualization



3.2.4 Changes to be performed



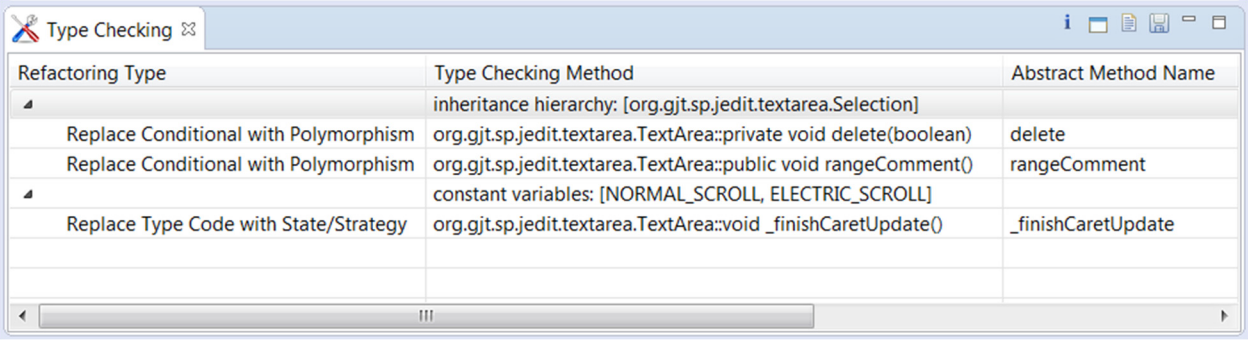
3.2.5 Changed code

```
// {{{ delete() method
private void delete(boolean forward) {
    if (!buffer.isEditable()) {
        getToolkit().beep();
        return;
    }

    if (getSelectionCount() != 0) {
        Selection[] selections = getSelection();
        for (int i = 0; i < selections.length; i++) {
            Selection s = selections[i];
            s.delete(forward, this);
        }
    } else if (forward) {
        if (caret == buffer.getLength()) {
            getToolkit().beep();
            return;
        }

        buffer.remove(caret, 1);
    } else {
        if (caret == 0) {
            getToolkit().beep();
            return;
        }

        buffer.remove(caret - 1, 1);
    }
} // }}}
```

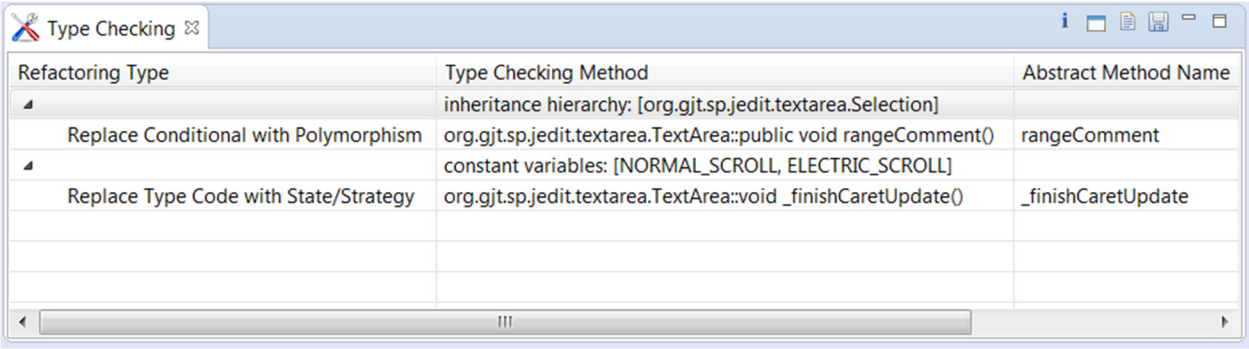
3.2.6 Tests

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

3.2.7 Tool result before refactoring



3.2.8 Tool result after refactoring



4 MANUAL REFACTORING

4.1 ATUNES – REMOVING "FEATURE ENVY" SMELL

4.1.1 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).

4.1.2 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.

4.1.3 Code before refactoring

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

```
protected void addBindings() {
    final PlayListControlsPanel panel = (PlayListControlsPanel) panelControlled;

    PlayListControlsListener listener = new PlayListControlsListener(panel);

    panel.getSortByTrack().addActionListener(listener);
    panel.getSortByTitle().addActionListener(listener);
    panel.getSortByArtist().addActionListener(listener);
    panel.getSortByAlbum().addActionListener(listener);
    panel.getSortByGenre().addActionListener(listener);
    panel.getSavePlaylistButton().addActionListener(listener);
    panel.getLoadPlaylistButton().addActionListener(listener);
    panel.getTopButton().addActionListener(listener);
    panel.getUpButton().addActionListener(listener);
    panel.getDeleteButton().addActionListener(listener);
    panel.getDownButton().addActionListener(listener);
    panel.getBottomButton().addActionListener(listener);
    panel.getInfoButton().addActionListener(listener);
    panel.getClearButton().addActionListener(listener);
    panel.getFavoriteSong().addActionListener(listener);
    panel.getFavoriteAlbum().addActionListener(listener);
    panel.getFavoriteArtist().addActionListener(listener);
    panel.getShowTrack().addActionListener(listener);
    panel.getShowArtist().addActionListener(listener);
    panel.getShowGenre().addActionListener(listener);
    panel.getShowDuration().addActionListener(listener);
    panel.getShowAlbum().addActionListener(listener);
    panel.getArtistButton().addActionListener(listener);
    panel.getAlbumButton().addActionListener(listener);
}
```

4.1.4 Refactored code

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

```
protected void addBindings() {

    final PlayListControlsPanel panel = (PlayListControlsPanel) panelControlled;

    PlayListControlsListener listener = new PlayListControlsListener(panel);

    // amoyeen: modified to remove feature envy
    panel.addBindings(listener);
}
```

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

```
protected void addBindings() {

    final PlayListControlsPanel panel = (PlayListControlsPanel) panelControlled;

    PlayListControlsListener listener = new PlayListControlsListener(panel);

    // amoyeen: modified to remove feature envy
    panel.addBindings(listener);
}
```

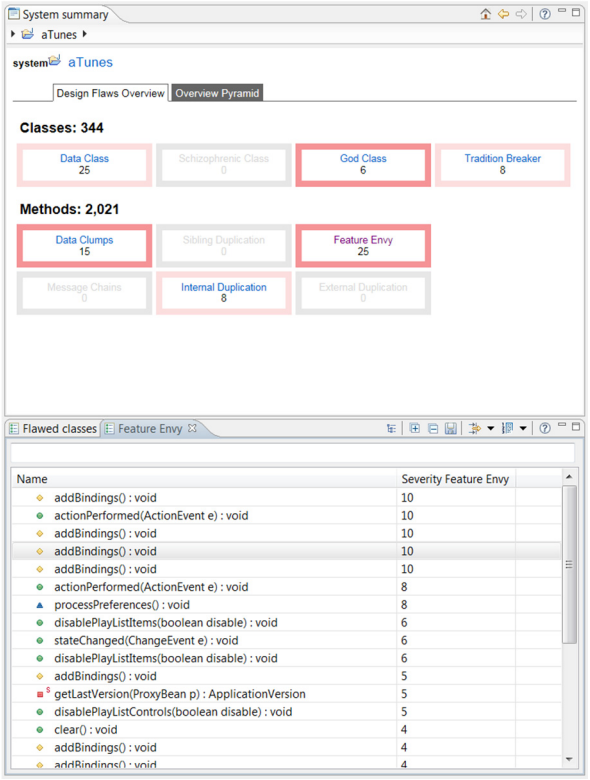
4.1.5 Tests

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

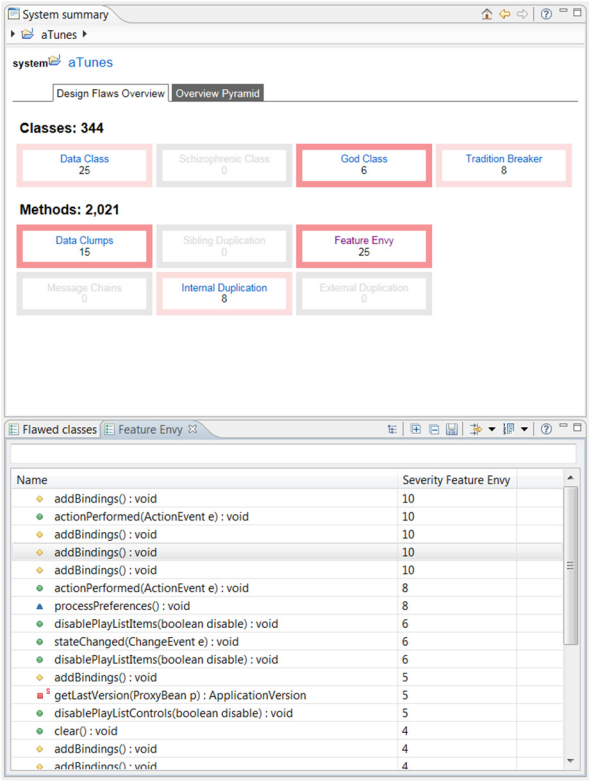
Test Case #	Description	Execution before Refactoring	Execution before Refactoring
T22	Demonstrate that Options > Show Genre checkbox can show/hide genre column in playlist	Pass	Pass
T23	Demonstrate that Options > Show Duration checkbox can show/hide duration column in playlist	Pass	Pass
T24	Demonstrate 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

4.1.6 Tool results

Tool result before refactoring



Tool result after refactoring



4.2 JEDIT – REMOVING "INTERNAL DUPLICATION" SMELL

4.2.1 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.

4.2.2 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.

4.2.3 Code before refactoring

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

```
// {{{ toUpperCase() method
/**
 * Converts the selected text to upper case.
 *
 * @since jEdit 2.7pre2
 */
public void toUpperCase() {
    if (!buffer.isEditable()) {
        getToolkit().beep();
        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);
        getToolkit().beep();
        return;
    }

    buffer.beginCompoundEdit();

    for (int i = 0; i < selection.length; i++) {
        Selection s = selection[i];
        setSelectedText(s, getSelectedText(s).toUpperCase());
    }

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

```
// {{{ toLowerCase() method
/**
 * Converts the selected text to lower case.
 *
 * @since jEdit 2.7pre2
 */
public void toLowerCase() {
    if (!buffer.isEditable()) {
        getToolkit().beep();
        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);
        getToolkit().beep();
        return;
    }

    buffer.beginCompoundEdit();

    for (int i = 0; i < selection.length; i++) {
        Selection s = selection[i];
        setSelectedText(s, getSelectedText(s).toLowerCa
    }

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

4.2.4 Refactored code

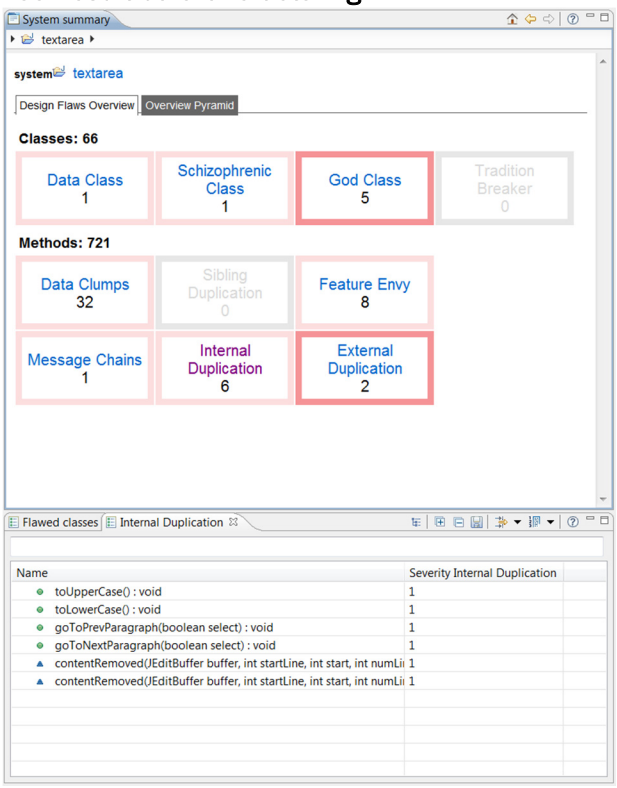
```
// amoyeen: added to remove code duplication
private final String toUpper = "UPPER";
private final String toLower = "LOWER";
                        // amoyeen: added to remove code duplication
                        // {{{ changeCase(String toCase) method
                        /**
                         * Converts the selected text to given case.
                         *
                         * @since jEdit 2.7pre2
                         */
                        private void changeCase(String toCase) {
                            if (!buffer.isEditable()) {
                                getToolkit().beep();
                                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);
                                getToolkit().beep();
                                return;
                            }

                            buffer.beginCompoundEdit();

                            for (int i = 0; i < selection.length; i++) {
                                Selection s = selection[i];
                                if (toCase == toUpper)
                                    setSelectedText(s, getSelectedText(s).toUpperCase());
                                if (toCase == toLower)
                                    setSelectedText(s, getSelectedText(s).toLowerCase());
                            }

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

4.2.5 Tests

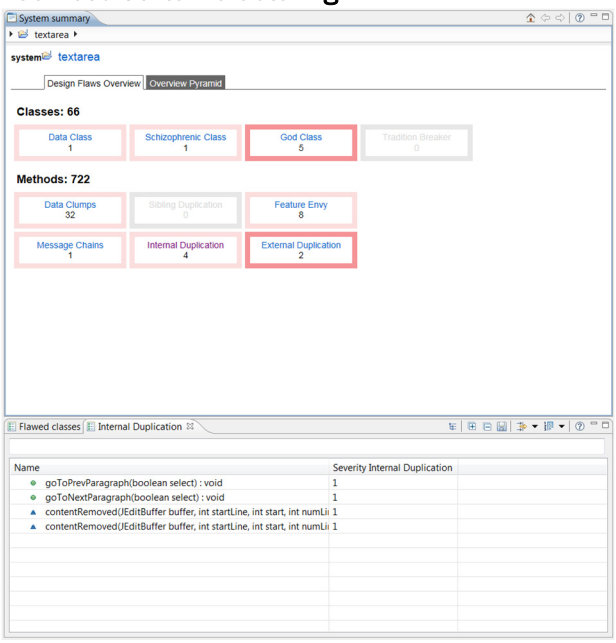
Test Case #	Description	Execution before Refactoring	Execution before Refactoring
T1	Demonstrate that program can be invoked successfully	Pass	Pass
T2	Demonstrate that an existing test file can be opened	Pass	Pass
T3	Demonstrate that part of text from open text file can be selected	Pass	Pass
T4	Demonstrate 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

4.2.6 Tool results

Tool result before refactoring



Tool result after refactoring



5 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/