Department of Computer Science

NOVA School of Sciences and Technology

THE S TEAM

Phase 1

Bruna Arroja nº 56751

Claúdia Santos nº 57049

Pedro Nunes nº 57854

Daniel Cavalheiro nº57869

António Brejo nº58152

User Case Descriptions

We decided to make our user case diagram based on the part of the code that treats the creation of new entries of references.

We created the base diagram. Added two actors:

1. The User, that is using the jabref application and adding these references.
2. The Reference Database, which is will, be used when the user wants to retrieve a certain reference in the database.

Added the use case “Add Entry” that generalizes the use cases “Manually” and “With ID”,

- Manually, is when a user enters manually a new entry reference, so it is associated with the user actor.
- With ID, refers to when a user inputs a I’d like for example a ISBN (book identifier), this in turn will access the reference data base to get the reference. This means that this use case will have an association with the user and the reference database actors and will have an extend relation with another use case “Invalid ID” that will occur when the id is invalid.

The use case “Reference Text” is used when a user adds an entry with a reference text. This then includes a case “Parses Data” which will parse the text.

The use case “Using online Service” is used when the user searches for a reference in one of the References Database. This includes a use case which is “List Found References”, this lists all the references found within the search for the user to choose which one he wants to add.

Added the use case “Using PDFs” that includes a case “Parses data”.

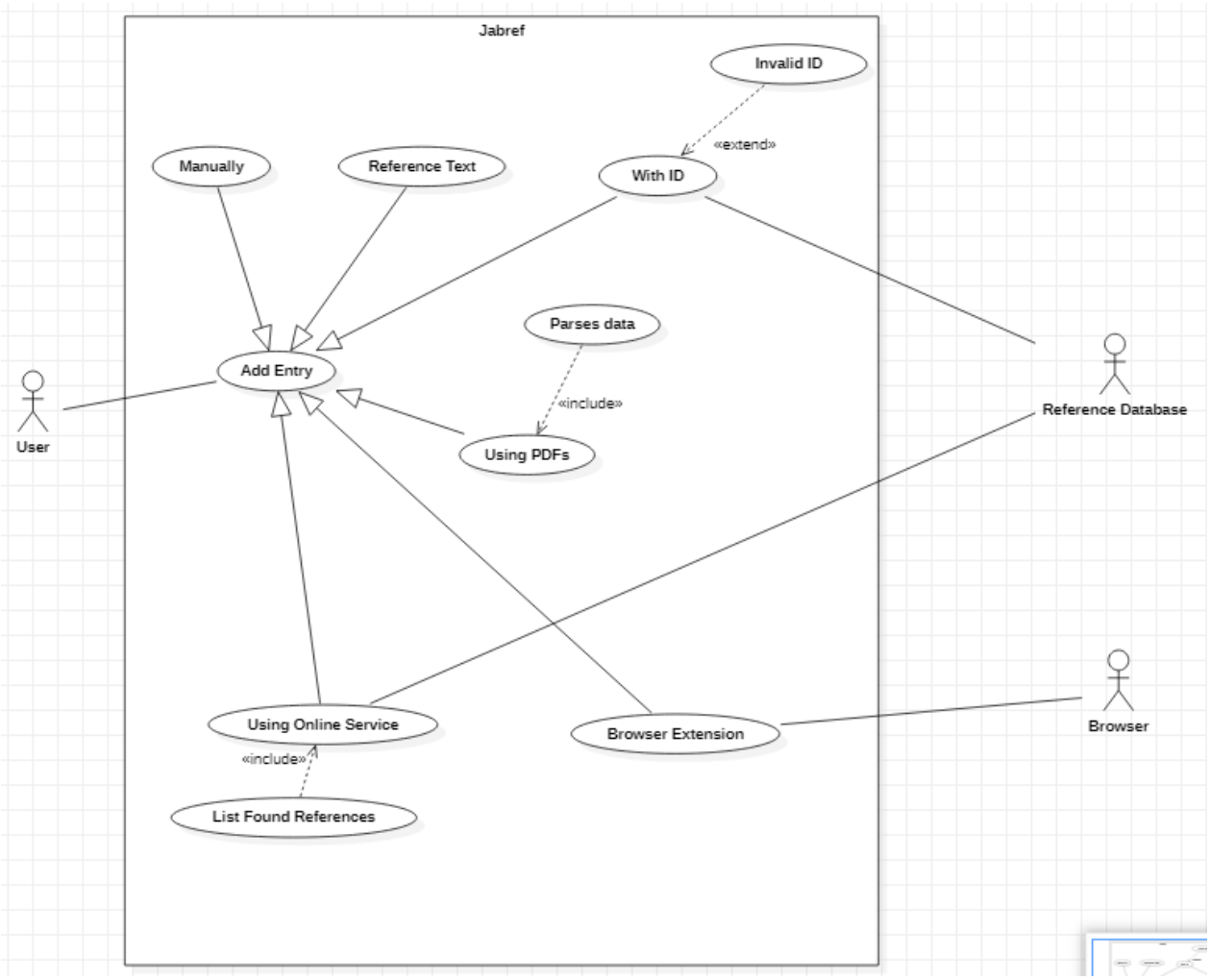
“Using PDFs” -> One way to create a new entry is using PDFs, JabRef can search automatically for the PDF files, even lets you choose the important ones, and then convert them into new entries. “Using PDFs” is a generalization of “Add entry”.

“Parses data” -> The program parses the data to use it. It’s linked to “Using PDFs” with a <<include>> because that always happens.

“Browser Extension” -> A way of adding new literature to JabRef is by using a browser extension. This extension automatically identifies any bibliographic information from a website, sending directly into JabRef. This is a generalization of the addEntry use case.

“Browser” -> This secondary actor entity interacts with the system by initiating the browser extension and enabling it to export references into JabRef.

Daniel added the “add Entry” use case, the “manually” use case and the “with ID” use case. António added the “Reference Text” and the “Using online Service” use cases. Bruna added the “Using PDFs” use case and Pedro added the “Browser Extension” use case.


Jabref
Invalid ID
«extend»
With ID
Manually
Reference Text
Parses data
Add Entry
«include»
Using PDFs
User
Reference Database
Using Online Service
«include»
List Found References
Browser Extension
Browser

António Brejo Nº58152

Command Design

```java
/**
 * A simple command that does not track progress of the action.
 */
public abstract class SimpleCommand extends CommandBase {

    protected ReadOnlyStringWrapper statusMessage = new ReadOnlyStringWrapper( initialValue: "");

    public String getStatusMessage() { return statusMessage.get(); }

    public ReadOnlyStringProperty statusMessageProperty() { return statusMessage.getReadOnlyProperty(); }

    @Override
    public double getProgress() { return 0; }

    @Override
    public ReadOnlyDoubleProperty progressProperty() { return null; }

    public void setExecutable(boolean executable) { this.executable.bind(BindingsHelper.constantOf(executable)); }
}
```

This code is located on src/main/java/org/jabref/gui/actions/SimpleCommand.java

```java
public class UndoRedoAction extends SimpleCommand {

    private static final Logger LOGGER = LoggerFactory.getLogger(UndoRedoAction.class);

    private final StandardActions action;
    private final JabRefFrame frame;
    private DialogService dialogService;

    public UndoRedoAction(StandardActions action, JabRefFrame frame, DialogService dialogService, StateManager stateManager) {
        this.action = action;
        this.frame = frame;
        this.dialogService = dialogService;

        // ToDo: Rework the UndoManager to something like the following, if it had a property.
        //  this.executable.bind(frame.getCurrentBasePanel().getUndoManager().canUndo())
        this.executable.bind(ActionHelper.needsDatabase(stateManager));
    }

    @Override
    public void execute() {
        LibraryTab libraryTab = frame.getCurrentLibraryTab();
        if (action == StandardActions.UNDO) {
            try {
                libraryTab.getUndoManager().undo();
                libraryTab.markBaseChanged();
                dialogService.notify(Localization.lang( key: "Undo"));
            } catch (CannotUndoException ex) {
                dialogService.notify(Localization.lang( key: "Nothing to undo") + '.');
            }
            frame.getCurrentLibraryTab().markChangedOrUnChanged();
        } else if (action == StandardActions.REDO) {
            try {
                libraryTab.getUndoManager().redo();
                libraryTab.markBaseChanged();
                dialogService.notify(Localization.lang( key: "Redo"));
            } catch (CannotRedoException ex) {
                dialogService.notify(Localization.lang( key: "Nothing to redo") + '.');
            }

            libraryTab.markChangedOrUnChanged();
        } else {
            LOGGER.debug("No undo/redo action: " + action.name());
        }
```

This code is located on src/main/java/org/jabref/gui/undo/UndoRedoAction.java

This is an example of the command pattern where the work is delegated from, in this case, the abstract class to a subclass which executes the command.

Observer design pattern.

```java
/**
 * Manages the RemoteListenerServerThread through typical life cycle methods.
 * <p/>
 * open -> start -> stop
 * openAndStart -> stop
 * <p/>
 * Observer: isOpen, isNotStartedBefore
 */
public class RemoteListenerServerLifecycle implements AutoCloseable {

    private static final Logger LOGGER = LoggerFactory.getLogger(RemoteListenerServerLifecycle.class);

    private RemoteListenerServerThread remoteListenerServerThread;

    public void stop() {
        if (isOpen()) {
            remoteListenerServerThread.interrupt();
            remoteListenerServerThread = null;
            JabRefExecutorService.INSTANCE.stopRemoteThread();
        }
    }
```

This class can be found in src\main\java\org\jabref\logic\remote\server\RemoteListenerServerLifecycle.java.
It is an event manager which manages listeners of RemoteListenerServerThread type.

```java
/**
 This thread wrapper is required to be able to interrupt the remote listener while listening on a port.
 */
public class RemoteListenerServerThread extends Thread {

    private static final Logger LOGGER = LoggerFactory.getLogger(RemoteListenerServerThread.class);

    private final RemoteListenerServer server;

    public RemoteListenerServerThread(MessageHandler messageHandler, int port, PreferencesService preferencesServi
        this.server = new RemoteListenerServer(messageHandler, port, preferencesService);
        this.setName("JabRef - Remote Listener Server on port " + port);
    }
```

This class is implements in src\main\java\org\jabref\logic\remote\server\RemoteListenerServerThread.java, this in turn calls a Remote Listener server which has the listeners.

```java
public class RemoteListenerServer implements Runnable {
    private static final Logger LOGGER = LoggerFactory.getLogger(RemoteListenerServer.class);

    private static final int BACKLOG = 1;

    private static final int TIMEOUT = 1000;

    private final MessageHandler messageHandler;
    private final ServerSocket serverSocket;
    private final PreferencesService preferencesService;

    public RemoteListenerServer(MessageHandler messageHandler, int port, PreferencesService preferencesService) thr
        this.serverSocket = new ServerSocket(port, BACKLOG, RemotePreferences.getIpAddress());
        this.messageHandler = messageHandler;
        this.preferencesService = preferencesService;
    }
```

Review:
Reviewer: Pedro Nunes

I agree with everything and also suggest to go more in detail about the observer pattern.

Facade design pattern

```java
/**
 * Facade to unify the access to the citation style engine. Use these methods if you need rendered BibTeX item(s)
 * given journal style. This class uses {@link CSLAdapter} to create output.
 */
public class CitationStyleGenerator {

    private static final Logger LOGGER = LoggerFactory.getLogger(CitationStyleGenerator.class);
    private static final CSLAdapter CSL_ADAPTER = new CSLAdapter();

    private CitationStyleGenerator() {
    }
```

This class can be found in:
src\main\java\org\jabref\logic\citationstyle\CitationStyleGenerator.java.
It is used to unify the access to the citation style engine.

```java
    /**
     * Generates the citation for multiple entries at once.
     *
     * @implNote The citations are generated using JavaScript which may take some time, better call it from outside the main thread.
     */
    public static List<String> generateCitations(List<BibEntry> bibEntries, String style, CitationStyleOutputFormat outputFormat) {
        try {
            return CSL_ADAPTER.makeBibliography(bibEntries, style, outputFormat);
        } catch (IllegalArgumentException ignored) {
            LOGGER.error("Could not generate BibEntry citation. The CSL engine could not create a preview for your item.", ignored);
            return Collections.singletonList(Localization.lang("Cannot generate preview based on selected citation style."));
        } catch (IOException | ArrayIndexOutOfBoundsException e) {
            LOGGER.error("Could not generate BibEntry citation", e);
            return Collections.singletonList(Localization.lang("Cannot generate preview based on selected citation style."));
        } catch (TokenMgrException e) {
            LOGGER.error("Bad character inside BibEntry", e);
            // sadly one cannot easily retrieve the bad char from the TokenMgrError
            return Collections.singletonList(Localization.lang("Cannot generate preview based on selected citation style.") +
                    outputFormat.getLineSeparator() +
                    Localization.lang("Bad character inside entry") +
                    outputFormat.getLineSeparator() +
                    e.getLocalizedMessage());
        }
    }
}
```

Review:
Reviewer: Daniel Cavalheiro

The first snippet reveals a comment that literally calls this class a Facade and after close inspection it does apper to unify complex subsystems relative to the citation style engine.
A suggestion would be to go in to more detail on the subsystem unified by this facade design pattern.

MOOD
Metrics

project	AHF	AIF	CF	MHF	MIF	PF
groups	95.33%	20.63%	129.41%	20.77%	7.88%	100.00%
auximport	100.00%	35.29%	800.00%	28.57%	50.00%	100.00%

MOOD Metrics

The mood metrics set is composed of 6 different metrics: MHF, AHF, MIF, AIF, PF and CF.

MHF is the inverse percentage of MethodsVisible. Where MethodsVisible is equal to the sum of the number of other classes where the methos is visible divided by number of classes minus 1 divided by the number of methods.

AHF is the same as MHF but relative to attributes.

MIF is the number of inherited methods divide by the total number of methods available in classes.

AIF is the same as MIF but relative to attributes.

PF or polymorphism factor is equal to the number of overrides divided by the maximum number of possible overrides.

CF or coupling factor is the number of actual couplings divided by the maximum number of possible couplings. A coupling happens when class A calls methods or accesses variables of class B and in turn class B calls methods or accesses variables of class A.

In both packages all values seem normal except for the coupling factor which Is a bit high for groups package but is abnormally high on the auximport package. I used this website as reference for my values: https://www.aivosto.com/project/help/pm-oo-mood.html

The package auximport, under the paths rc/main/java/org/jabref/gui/auximport, has high CF which means it has a lot of coupling classes, this causes complexity, reduces encapsulation and potential reuse, this is related to one of the code smells found in this package which is inappropriate intimacy. This code smells occurs when two classes depend too much on one another.

MOOD Metrics Review

Reviewer: Daniel Cavalheiro

MOOD Metrics

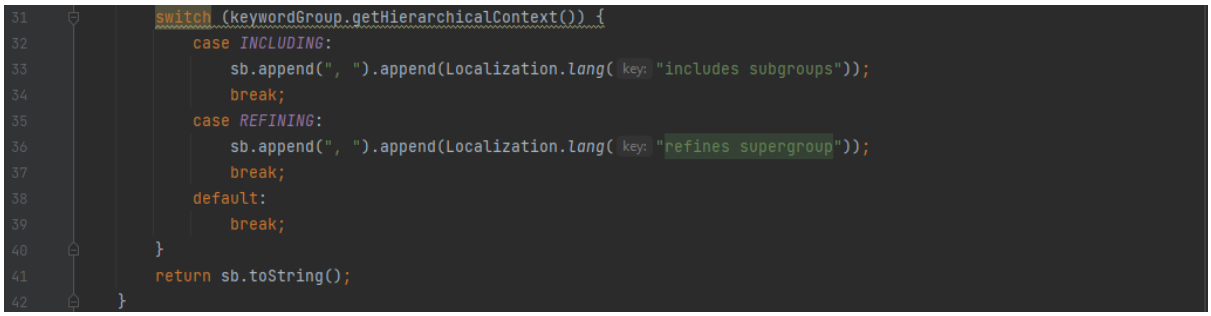
The linked website is very useful to understand these metrics in detail. Though the explanation given is also good.

These metrics were used in two different packages. In the first one my results were also similar and did not raise any alarms. But in the second package the CF (coupling factor) is way above than the average in the auximport class which is where I found an inappropriate intimacy code smell.

All this is referenced in the doc so and matches with my findings leading me to believe that it is correct.

Duplicated Code

```
14 @    public static String getShortDescriptionKeywordGroup(KeywordGroup keywordGroup, boolean showDynamic) {
15          StringBuilder sb = new StringBuilder();
16          sb.append("<b>");
17          if (showDynamic) {
18              sb.append("<i>").append(StringUtil.quoteForHTML(keywordGroup.getName())).append("</i>");
19          } else {
20              sb.append(StringUtil.quoteForHTML(keywordGroup.getName()));
21          }
22          sb.append("</b> - ");
23          sb.append(Localization.lang( key: "dynamic group"));
```



```
31          switch (keywordGroup.getHierarchicalContext()) {
32              case INCLUDING:
33                  sb.append(", ").append(Localization.lang( key: "includes subgroups"));
34                  break;
35              case REFINING:
36                  sb.append(", ").append(Localization.lang( key: "refines supergroup"));
37                  break;
38              default:
39                  break;
40          }
41          return sb.toString();
42      }
```

On the class GroupDescriptions.java, located on src\main\java\org.jabref\gui\groups, there are two occurrences of duplicated code. The second snippet appears two more times while the first snippet appears one more time.

This code can be refactored by creating two private methods one for each code snippet.

Review:

Reviewer: Daniel Cavalheiro

There is in fact two instantces duplicate code.

Could have taken a snippet of the other instance of duplicate could but not really necessary, all and all looks good.

Long method

The class located on src/main/java/org/jabref/cli/ArgumentProcessor.java, has a method which is 92 lines long, this method can be refactored into smaller private methods.

```
181    private List<ParserResult> processArguments() {
182
183        if (!cli.isBlank() && cli.isDebugLogging()) {
184            JabRefLogger.setDebug();
185        }
186
187        if ((startupMode == Mode.INITIAL_START) && cli.isShowVersion()) {
188            cli.displayVersion();
189        }
190
191        if ((startupMode == Mode.INITIAL_START) && cli.isHelp()) {
192            JabRefCLI.printUsage();
193            noGUINeeded = true;
194            return Collections.emptyList();
195        }
196
197        // Check if we should reset all preferences to default values:
198        if (cli.isPreferencesReset()) {
199            resetPreferences(cli.getPreferencesReset());
200        }
201
202        // Check if we should import preferences from a file:
203        if (cli.isPreferencesImport()) {
204            importPreferences();
205        }
206
207        // List to put imported/loaded database(s) in.
208        List<ParserResult> loaded = importAndOpenFiles();
209
210        if (!cli.isBlank() && cli.isFetcherEngine()) {
211            fetch(cli.getFetcherEngine()).ifPresent(loaded::add);
212        }
213
214        if (cli.isExportMatches()) {
215            if (!loaded.isEmpty()) {
216                if (!exportMatches(loaded)) {
217                    return Collections.emptyList();
218                }
219            } else {
```

Review:

Reviewer: Bruna Arroja

Reviewed: António Brejo

There is in fact a long method code smell, the method processArguments() has too many lines. Could have said that the complexity would decrease and would be easier to read the code due to solving this problem.

Large Class

```
52
53  public class IconTheme {
54
55      public static final Color DEFAULT_DISABLED_COLOR = Color.web("#c8c8c8");
56      public static final Color SELECTED_COLOR = Color.web("#50618F");
57      private static final String DEFAULT_ICON_PATH = "/images/external/red.png";
58      private static final Logger LOGGER = LoggerFactory.getLogger(IconTheme.class);
59      private static final Map<String, String> KEY_TO_ICON = readIconThemeFile(IconTheme.class.getResource( name: "/images/Icons
60      private static final Set<Ikon> ICON_NAMES = new HashSet<>();
61
62      public static Color getDefaultGroupColor() { return Color.web("#8a8a8a"); }
65
66  @  public static Optional<JabRefIcon> findIcon(String code, Color color) {
67          if (ICON_NAMES.isEmpty()) {
68              loadAllIkons();
69          }
70          return ICON_NAMES.stream().filter(icon -> icon.toString().equals(code.toUpperCase(Locale.ENGLISH)))
71                           .map(internalMat -> new InternalMaterialDesignIcon(internalMat).withColor(color)).findFirst();
72      }
```

This class is located on src/main/java/org/jabref/gui/icon/IconTheme.java. It has a huge enumerator inside of it which might make it hard to understand and read. This enumerator should be extracted to a file on its own, this will significantly decrease the size of the class and make things a lot more clearer.

Review:

Reviewer: Pedro Nunes

I agree that this nested enum should be removed as it is not necessary and is bloating the code.

Bruna Arroja Nº56751

DESIGN PATTERN

➔ SINGLETON

Location in the code:
src/main/java/org/jabref/gui/externalfiletype/ExternalFileTypes

The singleton design pattern presented in this class makes sure that there's only one instance of ExternalFileTypes.

```
private ExternalFileTypes() { updateExternalFileTypes(); }

public static ExternalFileTypes getInstance() {
    if (ExternalFileTypes.singleton == null) {
        ExternalFileTypes.singleton = new ExternalFileTypes();
    }
    return ExternalFileTypes.singleton;
}
```

DESIGN PATTERN

➔ FACTORY METHOD

Location in the code: src/main/java/org/jabref/gui/desktop/JabRefDesktop

The factory JabRefDesktop is on charge to create multiple types of desktops.

```
public class JabRefDesktop {

    private static final Logger LOGGER = LoggerFactory.getLogger(JabRefDesktop.class);

    private static final NativeDesktop NATIVE_DESKTOP = getNativeDesktop();
    private static final Pattern REMOTE_LINK_PATTERN = Pattern.compile("[a-z]+://.*");

    private JabRefDesktop() {
    }
```

The static method getNativeDesktop() is the method that creates the desktops.

```
    // TODO: Move to OS.java
    public static NativeDesktop getNativeDesktop() {
        if (OS.WINDOWS) {
            return new Windows();
        } else if (OS.OS_X) {
            return new OSX();
        } else if (OS.LINUX) {
            return new Linux();
        }
        return new DefaultDesktop();
    }
```

Review:

Reviewer: António Brejo

I think it is a factory method since we have creator class, which creates objects which in turn implement an object interface, which in this case is the NativeDesktop interface.

DESIGN PATTERN

➔ OBSERVER METHOD

Location in the code: src/main/java/org/jabref/gui/util/uithreadaware/ UiThreadListChangeListener

The observer class named UiThreadListChangeListener, notifies if a change happened to the list.

```
class UiThreadListChangeListener<E> implements ListChangeListener<E> {

    private final ListChangeListener<E> delegate;

    public UiThreadListChangeListener(ListChangeListener<E> delegate) { this.delegate = delegate; }

    @Override
    public void onChanged(Change<? extends E> c) {
        UiThreadHelper.ensureUiThreadExecution(() -> delegate.onChanged(c));
    }

    @Override
    public boolean equals(Object o) { return delegate.equals(o); }

    @Override
    public int hashCode() { return delegate.hashCode(); }
}
```

Review:

Reviewer: Pedro Nunes

I agree, this class observes any changes done to the list, making it a case of an observer pattern

Chidamber and Kemerer

Class	CBO	DIT	LCOM	NOC	RFC	WMC
org.jabref.gui.groups.GroupDescriptions	8.0	1.0	2.0	0.0	16.0	16.0
org.jabref.gui.groups.GroupDialogHeader	5.0		2.0		2.0	0.0
org.jabref.gui.groups.GroupDialogView	18.0	3.0	2.0	0.0	117.0	12.0
org.jabref.gui.groups.GroupDialogView.IkonliCell	1.0	9.0	1.0	0.0	24.0	2.0
org.jabref.gui.groups.GroupDialogViewModel	38.0	1.0	3.0	0.0	150.0	75.0
org.jabref.gui.groups.GroupModeViewModel	4.0	1.0	1.0	0.0	7.0	7.0
org.jabref.gui.groups.GroupNodeViewModel	30.0	1.0	7.0	0.0	127.0	56.0
org.jabref.gui.groups.GroupSidePane	16.0	2.0	3.0	0.0	28.0	12.0
org.jabref.gui.groups.GroupsPreferences	5.0	1.0	4.0	0.0	5.0	5.0
org.jabref.gui.groups.GroupTreeNodeViewModel	17.0	1.0	1.0	0.0	65.0	36.0
org.jabref.gui.groups.GroupTreeView	19.0	6.0	1.0	0.0	187.0	32.0
org.jabref.gui.groups.GroupTreeView.DragExpansionHandler	2.0	1.0	1.0	0.0	5.0	3.0
org.jabref.gui.groups.GroupTreeViewModel	19.0	2.0	4.0	0.0	84.0	34.0
org.jabref.gui.groups.GroupViewMode	7.0		2.0		2.0	0.0
org.jabref.gui.groups.MoveGroupChange	2.0	1.0	4.0	0.0	5.0	5.0
org.jabref.gui.groups.UndoableAddOrRemoveGroup	6.0	3.0	1.0	0.0	24.0	22.0

org.jabref.gui.groups.UndoableChangeEntriesOfGroup	5.0	1.0	1.0	0.0	7.0	4.0
org.jabref.gui.groups.UndoableModifySubtree	5.0	3.0	2.0	0.0	17.0	6.0
org.jabref.gui.groups.UndoableMoveGroup	6.0	3.0	2.0	0.0	18.0	4.0
Total						331.0
Average	11.21052631578947	2.352941176470588	2.315789473684210	0.0	46.84210526315791	17.42105263157895

Chidamber and Kemerer Metrics

The Chimdamber and Kemerer Metrics consists in the following six metrics calculated for each class:

- Coupling Between Object (CBO) - number of classes to which a class is coupled;
- Depth of inheritance Tree (DIT) - maximum inheritance path from the class to the root class;
- Lack of Cohesion (LCOM) – number of pairs of methods which do not share instance variables, minus the number of pairs of methods that share instance variables of the class;
- Number Of Children (NOC) - number of immediate sub-classes of a class;
- Response For a Class (RFC) – number of methods of a class plus the number of the invoked methods by those class methods;
- Weighted Methods per Class (WMC) – number of methods defined in class.

Extracted values of Chimdamber and Kemerer metrics to the package “groups”.

Class metrics

class	CBO	DIT	LCOM	NOC	RFC	WMC
org.jabref.gui.groups.GroupDescriptions	8	1	2	0	16	16
org.jabref.gui.groups.GroupDialogHeader	5		2		2	0
org.jabref.gui.groups.GroupDialogView	18	3	2	0	117	12
org.jabref.gui.groups.GroupDialogView.IkonliCell	1	9	1	0	24	2
org.jabref.gui.groups.GroupDialogViewModel	38	1	3	0	150	75
org.jabref.gui.groups.GroupModeViewModel	4	1	1	0	7	7
org.jabref.gui.groups.GroupNodeViewModel	30	1	7	0	127	56
org.jabref.gui.groups.GroupSidePane	16	2	3	0	28	12
org.jabref.gui.groups.GroupsPreferences	5	1	4	0	5	5
org.jabref.gui.groups.GroupTreeNodeViewModel	17	1	1	0	65	36
org.jabref.gui.groups.GroupTreeView	19	6	1	0	187	32
org.jabref.gui.groups.GroupTreeView.DragExpansionHa	2	1	1	0	5	3
org.jabref.gui.groups.GroupTreeViewModel	19	2	4	0	84	34
org.jabref.gui.groups.GroupViewMode	7		2		2	0
org.jabref.gui.groups.MoveGroupChange	2	1	4	0	5	5
org.jabref.gui.groups.UndoableAddOrRemoveGroup	6	3	1	0	24	22
org.jabref.gui.groups.UndoableChangeEntriesOfGroup	5	1	1	0	7	4
org.jabref.gui.groups.UndoableModifySubtree	5	3	2	0	17	6
org.jabref.gui.groups.UndoableMoveGroup	6	3	2	0	18	4
Total						**331**
Average	11,21	2,35	2,32	0,00	46,84	17,42

CBO (number of classes to which a class is coupled)

The higher the value of CBO metrics is in a class, the higher is the difficult to reuse it in another app, and harder to test due to the complexity.

The value of CBO in a class should not be higher than 14.

Checking the results of CBO values above, we can see that in the class "GroupsDialogViewModel" the CBO value is 38, picking this as an example, this value is too high. The reuse of this class will be complicated.

DIT (maximum inheritance path from the class to the root class)

The deeper is a class in hierarchy, the more methods and variables it will inherit, and due to that, the more complex it becomes.

The value of DIT in a class should not be higher than 5.

If we check the results above, we can find 2 classes that have a higher number than 5 in the DIT value, the class "IkonliCell" and the class "GroupTreeView", with respectively 9 and 6 DIT values. They become too complex to develop.

LCOM (number of pairs of methods which do not share instance variables, minus the number of pairs of methods that share instance variables of the class)

- LCOM = 0
  Means that the class is cohesive.
- LCOM > 0
  Means that the class needs to be separate into two or more classes.
  (Because its variables belong in disjoint sets)

In the results above, all the classes have the LCOM value greater than 0, and the higher this value is, the more likely is this classes to failure.

NOC (number of immediate sub-classes of a class)

A high NOC value seems to indicate less problems and indicate a greater reuse of base class, however it will need more testing.

We should be aware that a high value of NOC combined with a high value of WMC can be a sign of poor design due to complexity.

In the results above, all the classes have a NOC value of 0.

RFC (number of methods of a class plus the number of the invoked methods by those class methods)

The higher the RFC value is, the more likely to have problems, is very complex and harder to understand.

In the results above we can find some classes with really big numbers of RFC, like “GroupTreeView” or “GroupDialogViewModel” with the values 187 and 150, respectively.

WMC (number of methods defined in class)

The higher the WMC value is, the more likely to have problems, the less reusable it gets and the more time it needs to develop and maintain.

A good way to limit the WMC value is that only 10% of the classes can have more than 24 methods.

In the chosen package “groups”, we have 19 classes, 10% of 19 is 1.9, so that means we can only have 1 class with 24 or more methods. As we can see, that doesn’t applies, 5 classes have more than 24 methods.

Checking the class “GroupNodeViewModel”, we can find a ‘Long method’ code smell, assuming that we would solve this problem, we would have to split the method into two or more methods, and that would increase the number of methods defined in class (WMC).

Reviewer: Pedro Nunes

The explanation of the collected metrics is straight to the point and explains each one concisely. This is accompanied by a screenshot of the metrics collected of the package org.jabref.gui.groups and a deeper dive into the meaning of each metric and how they can be interpreted.

In the CBO explanation there should have been a more profound explanation on why the CBO value affects the reuse of this class and maybe an example in that class. In the following explanations of each metric, it becomes more apparent the cause and effect of each metric, but there could have been more examples to better understand these effects.

CODE SMELL

➔ LONG METHOD

Location in the code:
src/main/java/org/jabref/gui/groups/GroupTreeView

The method initialize() on the class GroupTreeView has 198 lines of code.

This method starts at the line 140.

```
private void initialize() {
    this.localDragboard = stateManager.getLocalDragboard();
    viewModel = new GroupTreeViewModel(stateManager, dialogService, preferencesService, taskExecutor, localDragboard);

    // Set-up groups tree
    groupTree.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);
    dragExpansionHandler = new DragExpansionHandler();
```

And ends at the line 338.

```
            event.consume();
        });

        return row;
    });

    // Filter text field
    setupClearButtonField(searchField);
}
```

The initialize() method, with this many lines of code, is very complicated to understand. Maybe divide this method into smaller ones would decrease the complexity and make it easier to read.

Review:

Reviewer: Daniel Cavalheiro

Identified a Long method code smell that is just shy of 200 lines, in average a method should just have 30 lines.

Refactoring suggestion is correct.

Everything looks good.

CODE SMELL

➔ LONG PARAMETER LIST

Location in the code:
src/main/java/org/jabref/gui/groups/GroupNodeViewModel

The constructor of the class GroupNodeViewModel has to many parameters.

```
public GroupNodeViewModel(BibDatabaseContext databaseContext, StateManager stateManager, TaskExecutor taskExecutor,
                          GroupTreeNode groupNode, CustomLocalDragboard localDragBoard, PreferencesService preferencesService) {
    this.databaseContext = Objects.requireNonNull(databaseContext);
    this.taskExecutor = Objects.requireNonNull(taskExecutor);
    this.stateManager = Objects.requireNonNull(stateManager);
    this.groupNode = Objects.requireNonNull(groupNode);
    this.localDragBoard = Objects.requireNonNull(localDragBoard);
    this.preferencesService = preferencesService;
```

Refactoring may be a good way to solve it, and may even reveal previously unnoticed duplicate code.

Review:

Reviewer: Pedro Nunes

I agree, there are to many parameters in this method.

Also i would like to add that refactoring may lead to decreasing the

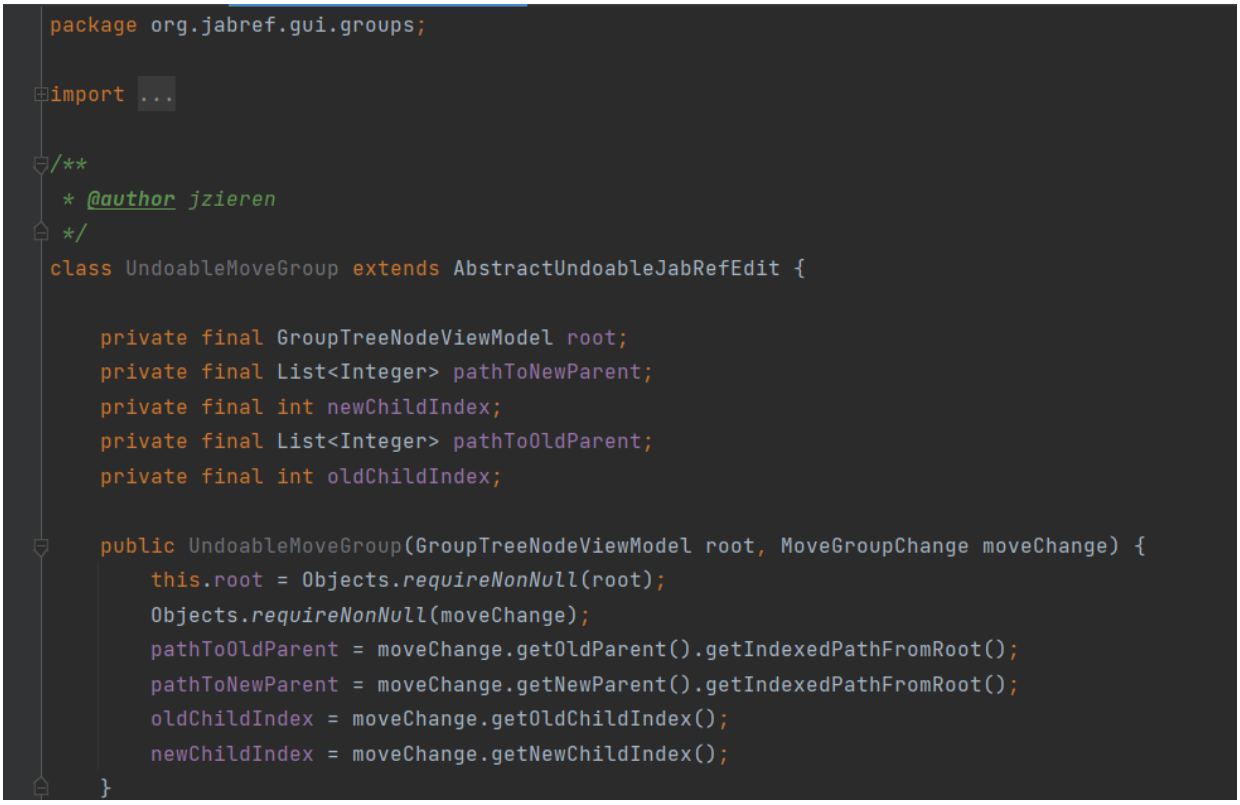
overall complexity of this class.

CODE SMELL

➔ SPECULATIVE GENERALITY

Location in the code:
src/main/java/org/jabref/gui/groups/MoveGroupChange

The class UndoableMoveGroup is never used.



```
package org.jabref.gui.groups;

import ...

/**
 * @author jzieren
 */
class UndoableMoveGroup extends AbstractUndoableJabRefEdit {

    private final GroupTreeNodeViewModel root;
    private final List<Integer> pathToNewParent;
    private final int newChildIndex;
    private final List<Integer> pathToOldParent;
    private final int oldChildIndex;

    public UndoableMoveGroup(GroupTreeNodeViewModel root, MoveGroupChange moveChange) {
        this.root = Objects.requireNonNull(root);
        Objects.requireNonNull(moveChange);
        pathToOldParent = moveChange.getOldParent().getIndexedPathFromRoot();
        pathToNewParent = moveChange.getNewParent().getIndexedPathFromRoot();
        oldChildIndex = moveChange.getOldChildIndex();
        newChildIndex = moveChange.getNewChildIndex();
    }
```

Perhaps if we erase this class, the code smell will disappear.

Review:

Reviewer: António Brejo

I confirm that this class is never used and can be deleted.

Daniel Cavalheiro 57869

Composite design pattern

Location: src/main/java/org/jabref/gui/groups/ GroupNodeViewModel

In this class we can find a variable list of GroupNodeViewModel called “children” and a variable of type GroupTreeNode leading me to believe that the composite design pattern is present

```
public class GroupNodeViewModel {

    private final String displayName;
    private final boolean isRoot;
    private final ObservableList<GroupNodeViewModel> children;
    private final BibDatabaseContext databaseContext;
    private final StateManager stateManager;
    private final GroupTreeNode groupNode;
    private final ObservableList<BibEntry> matchedEntries = FXCollections.observableArrayList();
    private final SimpleBooleanProperty hasChildren;
    private final SimpleBooleanProperty expandedProperty = new SimpleBooleanProperty();
    private final BooleanBinding anySelectedEntriesMatched;
    private final BooleanBinding allSelectedEntriesMatched;
    private final TaskExecutor taskExecutor;
    private final CustomLocalDragboard localDragBoard;
    private final ObservableList<BibEntry> entriesList;
    private final PreferencesService preferencesService;
    private final InvalidationListener onInvalidatedGroup = (listener) -> refreshGroup();
```

Review

Reviewer: Bruna Arroja

Reviewed: Daniel Cavalheiro

Nothing to add, looks good.

Factory design pattern

Location: src/main/java/org/jabref/gui/fieldeditors

The factory class is called FieldEditors and is responsible for the creation of various types of editors that are in this package.

```
public class FieldEditors {

    private static final Logger LOGGER = LoggerFactory.getLogger(FieldEditors.class);

    public static FieldEditorFX getForField(final Field field,
                                            final TaskExecutor taskExecutor,
                                            final DialogService dialogService,
                                            final JournalAbbreviationRepository journalAbbreviationRepository,
                                            final PreferencesService preferences,
                                            final BibDatabaseContext databaseContext,
                                            final EntryType entryType,
                                            final SuggestionProviders suggestionProviders,
                                            final UndoManager undoManager) {
        final Set<FieldProperty> fieldProperties = field.getProperties();

        final SuggestionProvider<?> suggestionProvider = getSuggestionProvider(field, suggestionProviders, databas

        final FieldCheckers fieldCheckers = new FieldCheckers(
```

The getForField static method is actually the method that creates all of de editors witch implement the interface FieldEditorFX.

Some examples of the objects it creates:

```
if (preferences.getTimestampPreferences().getTimestampField().equals(field)) {
    return new DateEditor(field, DateTimeFormatter.ofPattern(preferences.getTimestampPreferences().getTime
} else if (fieldProperties.contains(FieldProperty.DATE)) {
    return new DateEditor(field, DateTimeFormatter.ofPattern("[uuuu][-MM][-dd]"), suggestionProvider, fiel
} else if (fieldProperties.contains(FieldProperty.EXTERNAL)) {
    return new UrlEditor(field, dialogService, suggestionProvider, fieldCheckers, preferences);
} else if (fieldProperties.contains(FieldProperty.JOURNAL_NAME)) {
    return new JournalEditor(field, journalAbbreviationRepository, preferences, suggestionProvider, fieldC
} else if (fieldProperties.contains(FieldProperty.DOI) || fieldProperties.contains(FieldProperty.EPRINT)
    return new IdentifierEditor(field, taskExecutor, dialogService, suggestionProvider, fieldCheckers, pre
} else if (field == StandardField.OWNER) {
    return new OwnerEditor(field, preferences, suggestionProvider, fieldCheckers);
} else if (fieldProperties.contains(FieldProperty.FILE_EDITOR)) {
    return new LinkedFilesEditor(field, dialogService, databaseContext, taskExecutor, suggestionProvider,
} else if (fieldProperties.contains(FieldProperty.YES_NO)) {
    return new OptionEditor<>(new YesNoEditorViewModel(field, suggestionProvider, fieldCheckers));
} else if (fieldProperties.contains(FieldProperty.MONTH)) {
    return new OptionEditor<>(new MonthEditorViewModel(field, suggestionProvider, databaseContext.getMode
} else if (fieldProperties.contains(FieldProperty.GENDER)) {
    return new OptionEditor<>(new GenderEditorViewModel(field, suggestionProvider, fieldCheckers));
} else if (fieldProperties.contains(FieldProperty.EDITOR_TYPE)) {
    return new OptionEditor<>(new EditorTypeEditorViewModel(field, suggestionProvider, fieldCheckers));
} else if (fieldProperties.contains(FieldProperty.PAGINATION)) {
```

Review

Reviewer: Bruna Arroja

Reviewed: Daniel Cavalheiro

I do believe that there is a composite design pattern due to the variable's names. I have nothing to add.

Singleton design patten

Location: src/main/java/org/jabref/preferences/JabRefPreferences

To ensure there is only one instance of JabRefPreferences the constructor is made private and there is this method:

```java
/**
 * @return Instance of JaRefPreferences
 * @deprecated Use {@link PreferencesService} instead
 */
@Deprecated
public static JabRefPreferences getInstance() {
    if (JabRefPreferences.singleton == null) {
        JabRefPreferences.singleton = new JabRefPreferences();
    }
    return JabRefPreferences.singleton;
}
```

So when the unique instance is needed we use this method to get it.

Review

Reviewer: António Brejo

This singleton pattern seems to be in accord to the material given in classes.

Lines of CODE Metrics

Classes	Lines of CODE
AppearancePreferences	45
ExportComparator	9
ExternalApplicationsPreferences	52
FilePreferences	82
GeneralPreferences	82
GuiPreferences	118
ImportExportPreferences	107
JabRefPreferences	2130
MrDlibPreferences	51
PreferencesFilter	79
PreferencesService	165
PreviewPreferences	102
PushToApplicationPreferences	33
SearchPreferences	83
SidePanePreferences	40
TelemetryPreferences	30
VersionPreferences	11
Total in preferences package	**3219**

Lines of code metrics is used to determine the size of a program based on the number of lines in the source code. This metric is useful to predict the complexity of a program and the amount of effort a programmer will need to tackle developing and maintaining the code.

I was unable to use the Lines of Code metric in the metrics reloaded plug in due to an error that me and many of my colleagues encountered. However, after searching online I saw a recommendation to the “Statistic” plug in that would allow me to attain the same result.

My lab teacher recommended my group to pick a package and run the plug in there instead of the whole project. I chose this directory: src/main/java/org/jabref/preferences.

After doing some research, I encountered that in this book “Refactoring in Large Software Projects” by Martin Lippert and Stephen Roock there is rule called “Rule of 30” that states: A class should contain an average of less than 30 methods, resulting in up to 900 lines of code. The data that I collect in the package falls under this rule except for the JabRefPreferences class which has 2130 lines of code. This is a big amount of code for a single class raising a major red flag that could be challenging for a person to understand and refactor later down the line.

Also, there is a long parameter list in the GuiPreferences, the refactoring of this code smell will lead to a creation of a new class lowering the number of lines of code in this

class but in turn will probably lead to the increase in total lines of code due to the newly created class.

Lines of CODE Metrics Review

Reviewer: António Brejo

The line of code metrics uses the number of lines in each class. When the class gets too long it becomes complex and hard to read, it also makes it hard to refactor later. After checking the sources used by my colleague, I think he used the right method to assess the results of the metrics. The class JabRefPreferences class really is an outlier, especially when compared with the classes in the same package.

Refactoring the GuiPreferences by creating a new class would lower the number of lines per class which is what is intended, even if the total number of lines increases.

Inapropriate intimacy code smell

Location: src/main/java/org.jabref/gui/auximport/AuxParserResultViewModel

This class only uses methods of a different class called auxParserResult resulting in a inappropriate intimacy code smell.

```
public class AuxParserResultViewModel {

    private AuxParserResult auxParserResult;

    public AuxParserResultViewModel(AuxParserResult auxParserResult) { this.auxParserResult = auxParserResult; }

    /** Prints parsing statistics */
    public String getInformation(boolean includeMissingEntries) {
        StringBuilder result = new StringBuilder();

        result.append(Localization.lang( key: "keys in library")).append(' ').append(this.auxParserResult.getMasterDatabase().getEntryCount()).append('\n'
              .append(Localization.lang( key: "found in AUX file")).append(' ').append(this.auxParserResult.getFoundKeysInAux()).append('\n')
              .append(Localization.lang( key: "resolved")).append(' ').append(this.auxParserResult.getResolvedKeysCount()).append('\n')
              .append(Localization.lang( key: "not found")).append(' ').append(this.auxParserResult.getUnresolvedKeysCount()).append('\n')
              .append(Localization.lang( key: "crossreferenced entries included")).append(' ').append(this.auxParserResult.getCrossRefEntriesCount()).appe
              .append(Localization.lang( key: "strings included")).append(' ').append(this.auxParserResult.getInsertedStrings()).append('\n');

        if (includeMissingEntries && (this.auxParserResult.getUnresolvedKeysCount() > 0)) {
            for (String entry : this.auxParserResult.getUnresolvedKeys()) {
                result.append(entry).append('\n');
            }
        }
        if (this.auxParserResult.getNestedAuxCount() > 0) {
            result.append(Localization.lang( key: "nested AUX files")).append(' ').append(this.auxParserResult.getNestedAuxCount());
        }
        return result.toString();
    }
```

To correct this code smell the class auxParserResult should have a method getInformation() that builds the string that this class is building.

Review

Reviewer - António Brejo

It looks good to me, nothing to add. I can say that this is in accordance with the results of

my analysis of the MOOD metrics.

Long Method Code Smell

Location: src/main/java/org/jabref/gui/openoffice/OOBibBase

This class has a method called refreshCiteMarkersInternal() which has more than 200 lines of code.

Begging of the method:

```
427  @   private List<String> refreshCiteMarkersInternal(List<BibDatabase> databases, OOBibStyle style)
428          throws WrappedTargetException, IllegalArgumentException, NoSuchElementException,
429          UndefinedCharacterFormatException, UnknownPropertyException, PropertyVetoException,
430          CreationException, BibEntryNotFoundException {
431
432      List<String> cited = findCitedKeys();
433      Map<String, BibDatabase> linkSourceBase = new HashMap<>();
434      Map<BibEntry, BibDatabase> entries = findCitedEntries(databases, cited, linkSourceBase);
435
```

End of the method:

```
714                  insertBookMark(OOBibBase.BIB_SECTION_NAME, cursor);
715              }
716          }
717
718          List<String> unresolvedKeys = new ArrayList<>();
719          for (BibEntry entry : entries.keySet()) {
720              if (entry instanceof UndefinedBibtexEntry) {
721                  String key = ((UndefinedBibtexEntry) entry).getKey();
722                  if (!unresolvedKeys.contains(key)) {
723                      unresolvedKeys.add(key);
724                  }
725              }
726          }
727          return unresolvedKeys;
728      }
729
```

Even though the method has some comments explain what it is doing, if it was broken up in to smaller methods maybe every time it does a cycle like in the snippet each with their own description then this code would be easier to read.

Review

Reviewer: Pedro Nunes

I agree with this remarks and also think this function should be chopped up into smaller subroutines to make it more readable.

Long Parameter List Code Smell

Location: src/main/java/org/jabref/preferences/GuiPreferences

```java
public class GuiPreferences {
    private final DoubleProperty positionX;
    private final DoubleProperty positionY;
    private final DoubleProperty sizeX;
    private final DoubleProperty sizeY;

    private final BooleanProperty windowMaximised;

    private final BooleanProperty shouldOpenLastEdited;
    private final ObservableList<String> lastFilesOpened;
    private final ObjectProperty<Path> lastFocusedFile;
    private final DoubleProperty sidePaneWidth;

    public GuiPreferences(double positionX,
                          double positionY,
                          double sizeX,
                          double sizeY,
                          boolean windowMaximised,
                          boolean shouldOpenLastEdited,
                          List<String> lastFilesOpened,
                          Path lastFocusedFile, double sidePaneWidth) {
        this.positionX = new SimpleDoubleProperty(positionX);
        this.positionY = new SimpleDoubleProperty(positionY);
        this.sizeX = new SimpleDoubleProperty(sizeX);
        this.sizeY = new SimpleDoubleProperty(sizeY);
        this.windowMaximised = new SimpleBooleanProperty(windowMaximised);
        this.shouldOpenLastEdited = new SimpleBooleanProperty(shouldOpenLastEdited);
```

The constructor of this class has to many parameters. An easy refactor would be to create a class that groups up the parameters positionX, position, sizeX and size Y making the parameter list smaller and easier to read.

Review

Reviewer: Pedro Nunes

Nothing to add here seems good.

Pedro Nunes nº57854

Observer pattern

Path : src\main\java\org\jabref\gui\collab\DatabaseChangeMonitor.java

```java
@Override
public void fileUpdated() {
    // File on disk has changed, thus look for notable changes and notify listeners in case there are
    ChangeScanner scanner = new ChangeScanner(database, preferencesService, stateManager);
    BackgroundTask.wrap(scanner::scanForChanges)
                  .onSuccess(changes -> {
                      if (!changes.isEmpty()) {
                          listeners.forEach(listener -> listener.databaseChanged(changes));
                      }
                  })
                  .onFailure(e -> LOGGER.error("Error while watching for changes", e))
                  .executeWith(taskExecutor);
}
```

This method, as explained by the comment above, notifies the listeners when a file has been changed and calls the databaseChanged function for each listener.

Factory pattern

Path : src\main\java\org\jabref\gui\util\ViewModelTableRowFactory

```java
public class ViewModelTableRowFactory<S> implements Callback<TableView<S>, TableRow<S>> {

    private BiConsumer<S, ? super MouseEvent> onMouseClickedEvent;
    private Function<S, ContextMenu> contextMenuFactory;
    private TriConsumer<TableRow<S>, S, ? super MouseEvent> toOnDragDetected;
    private TriConsumer<TableRow<S>, S, ? super DragEvent> toOnDragDropped;
    private BiConsumer<S, ? super DragEvent> toOnDragEntered;
    private TriConsumer<TableRow<S>, S, ? super DragEvent> toOnDragExited;
    private TriConsumer<TableRow<S>, S, ? super DragEvent> toOnDragOver;
    private TriConsumer<TableRow<S>, S, ? super MouseDragEvent> toOnMouseDragEntered;
    private Callback<S, String> toTooltip;
```

This class is used to construct a TableRow object which represents a single row in a TreeTableView object.

It also provides a lot of context in which the row was created in (example: onMouseClicked) and changes it’s behaviour according to it.

Review

Reviewer: Bruna Arroja

Reviewed: Pedro Nunes

The design pattern seems to be the factory pattern, nothing special to add.

Builder pattern

Path : src\main\java\org\jabref\gui\util\DirectoryDialogConfiguration.java

```java
public class DirectoryDialogConfiguration {

    private final Path initialDirectory;

    private DirectoryDialogConfiguration(Path initialDirectory) { this.initialDirectory = initialDirectory

    public Optional<Path> getInitialDirectory() { return Optional.ofNullable(initialDirectory); }

    public static class Builder {

        private Path initialDirectory;

        public DirectoryDialogConfiguration build() { return new DirectoryDialogConfiguration(initialDirec

        public Builder withInitialDirectory(Path directory) {

            directory = directory.toAbsolutePath();
            // Dir must be a folder, not a file
            if (!Files.isDirectory(directory)) {
                directory = directory.getParent();
            }
            // The lines above work also if the dir does not exist at all!
            // NULL is accepted by the filechooser as no inital path

            if (!Files.exists(directory)) {

                directory = null;
            }
            initialDirectory = directory;
            return this;
        }

        public Builder withInitialDirectory(String directory) {
            withInitialDirectory(Path.of(directory));
            return this;
        }
    }
}
```

This nested class within this class is used to build an object DirectoryDialogConfiguration with a String has a parameter or an object of the Class Path.

And so, the DirectoryDialogConfiguration class is being built in a separate class and can be considered a Builder design pattern.

Reviewer: António Brejo

I agree that this is a builder design pattern since it builds an object step by step.

Package	A	Ca	Ce	D	I
java.org.jabref.logic.journals	0	0	0	1	1
org.jabref	0	0	6	0	1
org.jabref.architecture	0,6	0	0	0,4	1
org.jabref.benchmarks	0	0	78	0	1
org.jabref.cli	0	16	472	0,03	0,98
org.jabref.gui	0,05	2253	1939	0,49	0,45
org.jabref.gui.actions	0,29	1106	865	0,28	0,44
org.jabref.gui.autocompleter	0,1	151	463	0,14	0,78
org.jabref.gui.auximport	0	3	70	0,04	0,96
org.jabref.gui.bibtexextractor	0	2	108	0,02	0,98
org.jabref.gui.citationkeypattern	0	7	207	0,03	0,97
org.jabref.gui.cleanup	0	2	223	0,01	0,99
org.jabref.gui.collab	0,12	6	375	0,11	0,98
org.jabref.gui.commonfxcontrols	0	53	235	0,18	0,82
org.jabref.gui.contentselector	0	1	196	0,01	0,99
org.jabref.gui.copyfiles	0	2	140	0,01	0,99
org.jabref.gui.customentrytypes	0	1	176	0,01	0,99
org.jabref.gui.desktop	0	58	133	0,3	0,7
org.jabref.gui.desktop.os	0,2	20	41	0,13	0,67
org.jabref.gui.dialogs	0	7	30	0,19	0,81
org.jabref.gui.documentviewer	0,23	5	74	0,17	0,93
org.jabref.gui.duplicationFinder	0	57	155	0,27	0,86
org.jabref.gui.edit	0	47	744	0,06	0,94
org.jabref.gui.entryeditor	0,08	58	780	0,01	0,93
org.jabref.gui.entryeditor.fileannotationtab	0	2	189	0,01	0,99
org.jabref.gui.errorconsole	0	1	78	0,01	0,99
org.jabref.gui.exporter	0	63	727	0,08	0,93
org.jabref.gui.externalfiles	0	37	586	0,06	0,94
org.jabref.gui.externalfiletype	0,1	228	189	0,45	0,45
org.jabref.gui.fieldeditors	0,11	92	1563	0,05	0,94
org.jabref.gui.fieldeditors.contextmenu	0	19	133	0,12	0,87
org.jabref.gui.groups	0	42	1152	0,04	0,96
org.jabref.gui.help	0	31	143	0,18	0,82
org.jabref.gui.icon	0,12	947	4	0,87	0,01
org.jabref.gui.importer	0	76	667	0,1	0,9
org.jabref.gui.importer.actions	0,2	15	140	0,1	0,9
org.jabref.gui.importer.fetcher	0	4	175	0,02	0,98
org.jabref.gui.integrity	0	1	89	0,01	0,99
org.jabref.gui.journals	0	4	130	0,03	0,97
org.jabref.gui.keyboard	0	370	317	0,54	0,46
org.jabref.gui.libraryproperties	0	1	152	0,01	0,99
org.jabref.gui.linkedfile	0	2	116	0,02	0,98
org.jabref.gui.logging	0	0	7	0	1
org.jabref.gui.maintable	0	434	848	0,34	0,71
org.jabref.gui.maintable.columns	0	30	299	0,09	0,91
org.jabref.gui.menus	0	12	87	0,12	0,88
org.jabref.gui.mergeentries	0	33	427	0,07	0,93
org.jabref.gui.metadata	0	2	81	0,02	0,98
org.jabref.gui.openoffice	0	1	834	0	1

org.jabref.gui.preferences	0,38	382	204	0,28	0,35
org.jabref.gui.preferences.appearance	0	1	102	0,01	0,99
org.jabref.gui.preferences.citationkeypattern	0	1	99	0,01	0,99
org.jabref.gui.preferences.customexporter	0	1	55	0,02	0,98
org.jabref.gui.preferences.customimporter	0	1	103	0,01	0,99
org.jabref.gui.preferences.entryeditor	0	1	82	0,01	0,99
org.jabref.gui.preferences.entryeditortabs	0	1	49	0,02	0,98
org.jabref.gui.preferences.external	0	1	99	0,01	0,99
org.jabref.gui.preferences.file	0	1	42	0,02	0,98
org.jabref.gui.preferences.general	0	1	129	0,01	0,99
org.jabref.gui.preferences.groups	0	1	33	0,03	0,97
org.jabref.gui.preferences.importexport	0	1	71	0,01	0,99
org.jabref.gui.preferences.journals	0	1	259	0	1
org.jabref.gui.preferences.keybindings	0	33	113	0,23	0,77
org.jabref.gui.preferences.keybindings.presets	0,33	11	69	0,2	0,86
org.jabref.gui.preferences.linkedfiles	0	1	90	0,01	0,99
org.jabref.gui.preferences.nameformatter	0	1	53	0,02	0,98
org.jabref.gui.preferences.network	0	1	146	0,01	0,99
org.jabref.gui.preferences.preview	0	1	301	0	1
org.jabref.gui.preferences.protectedterms	0	1	210	0	0,99
org.jabref.gui.preferences.table	0	1	277	0	1
org.jabref.gui.preferences.xmp	0	1	112	0,01	0,99
org.jabref.gui.preview	0	47	561	0,08	0,92
org.jabref.gui.push	0,14	39	236	0	0,86
org.jabref.gui.remote	0	2	13	0,13	0,87
org.jabref.gui.search	0,08	39	732	0,03	0,95
org.jabref.gui.search.rules.describer	0,2	16	184	0,12	0,85
org.jabref.gui.shared	0	6	369	0,02	0,98
org.jabref.gui.sidepane	0,2	88	43	0,47	0,31
org.jabref.gui.slr	0	2	141	0,01	0,99
org.jabref.gui.specialfields	0	133	219	0,38	0,62
org.jabref.gui.texparser	0	4	187	0,02	0,98
org.jabref.gui.undo	0	193	241	0,44	0,56
org.jabref.gui.util	0,03	1950	241	0,86	0,12
org.jabref.gui.util.comparator	0	3	42	0,07	0,93
org.jabref.gui.util.component	0	4	16	0,2	0,8
org.jabref.gui.util.uithreadaware	0	5	0	1	0
org.jabref.logic	0	52	68	0,43	0,57
org.jabref.logic.autosaveandbackup	0	40	57	0,41	0,59
org.jabref.logic.auxparser	0,2	40	89	0,11	0,69
org.jabref.logic.bibtex	0	163	934	0,15	0,85
org.jabref.logic.bibtex.comparator	0	94	822	0,1	0,9
org.jabref.logic.bst	0,07	4	177	0,05	0,98
org.jabref.logic.citationkeypattern	0,08	245	1051	0,11	0,83
org.jabref.logic.citationstyle	0	71	119	0,37	0,56
org.jabref.logic.cleanup	0,06	524	1620	0,19	0,78
org.jabref.logic.crawler	0	6	434	0,01	0,99
org.jabref.logic.database	0	37	845	0,04	0,96
org.jabref.logic.exporter	0,05	278	3047	0,04	0,92
org.jabref.logic.externalfiles	0	18	50	0,26	0,74

org.jabref.logic.formatter	0	12	79	0,13	0,87
org.jabref.logic.formatter.bibtexfields	0	101	172	0,37	0,63
org.jabref.logic.formatter.casechanger	0	41	47	0,47	0,53
org.jabref.logic.formatter.minifier	0	3	10	0,23	0,77
org.jabref.logic.git	0	33	2	0,94	0,06
org.jabref.logic.groups	0	12	6	0,67	0,33
org.jabref.logic.help	0	92	2	0,98	0,02
org.jabref.logic.importer	0,36	2077	508	0,45	0,2
org.jabref.logic.importer.fetcher	0,05	165	4854	0,01	0,97
org.jabref.logic.importer.fetcher.transformers	0,22	64	20	0,54	0,24
org.jabref.logic.importer.fileformat	0	302	5580	0,05	0,95
org.jabref.logic.importer.util	0	53	640	0,08	0,93
org.jabref.logic.integrity	0,04	77	1025	0,03	0,93
org.jabref.logic.journals	0	165	0	1	0
org.jabref.logic.l10n	0	4109	0	1	0
org.jabref.logic.layout	0,23	584	285	0,44	0,33
org.jabref.logic.layout.format	0	140	677	0,17	0,83
org.jabref.logic.logging	0	11	0	1	0
org.jabref.logic.msbib	0	6	362	0,02	0,98
org.jabref.logic.net	0	164	52	0,76	0,24
org.jabref.logic.openoffice	0	109	29	0,79	0,21
org.jabref.logic.openoffice.action	0	0	211	0	1
org.jabref.logic.openoffice.backend	0	15	212	0,07	0,93
org.jabref.logic.openoffice.frontend	0	40	265	0,13	0,87
org.jabref.logic.openoffice.style	0	142	1107	0,11	0,88
org.jabref.logic.pdf	0,14	13	126	0,05	0,91
org.jabref.logic.pdf.search.indexing	0	32	236	0,12	0,88
org.jabref.logic.pdf.search.retrieval	0	12	74	0,14	0,86
org.jabref.logic.preferences	0	119	3	0,98	0,02
org.jabref.logic.preview	1	126	2	0,02	0,02
org.jabref.logic.protectedterms	0	96	12	0,89	0,11
org.jabref.logic.remote	0	27	71	0,28	0,72
org.jabref.logic.remote.client	0	23	24	0,49	0,51
org.jabref.logic.remote.server	0,25	52	30	0,38	0,37
org.jabref.logic.remote.shared	0	36	0	1	0
org.jabref.logic.search	0,2	68	371	0,05	0,85
org.jabref.logic.shared	0,17	141	654	0	0,82
org.jabref.logic.shared.event	0	19	15	0,56	0,44
org.jabref.logic.shared.exception	0	18	9	0,67	0,33
org.jabref.logic.shared.listener	0	5	8	0,38	0,62
org.jabref.logic.shared.prefs	0	44	15	0,75	0,25
org.jabref.logic.shared.security	0	8	0	1	0
org.jabref.logic.texparser	0,17	7	449	0,15	0,98
org.jabref.logic.undo	0	5	0	1	0
org.jabref.logic.util	0,06	2181	420	0,78	0,16
org.jabref.logic.util.io	0,05	225	262	0,41	0,57
org.jabref.logic.util.strings	0	114	46	0,71	0,29
org.jabref.logic.xmp	0	78	340	0,19	0,81
org.jabref.migrations	0,08	16	822	0,06	0,98
org.jabref.model	0,25	364	23	0,69	0,06

org.jabref.model.database	0	2775	533	0,84	0,16
org.jabref.model.database.event	0	43	16	0,73	0,27
org.jabref.model.entry	0	10388	1868	0,85	0,15
org.jabref.model.entry.event	0,2	116	22	0,64	0,16
org.jabref.model.entry.field	0,06	13210	18	0,94	0
org.jabref.model.entry.identifier	0,08	221	20	0,84	0,08
org.jabref.model.entry.types	0,09	2155	3133	0,32	0,59
org.jabref.model.event	0	14	15	0,48	0,52
org.jabref.model.groups	0,19	909	464	0,47	0,32
org.jabref.model.groups.event	0	2	4	0,33	0,67
org.jabref.model.metadata	0	577	57	0,91	0,09
org.jabref.model.metadata.event	0	3	5	0,38	0,62
org.jabref.model.openoffice	0	25	0	1	0
org.jabref.model.openoffice.backend	1	27	9	0,25	0,25
org.jabref.model.openoffice.ootext	0	209	95	0,69	0,28
org.jabref.model.openoffice.rangesort	0,15	50	54	0,33	0,52
org.jabref.model.openoffice.style	0,3	556	82	0,57	0,13
org.jabref.model.openoffice.uno	0	241	0	1	0
org.jabref.model.openoffice.util	0	121	0	1	0
org.jabref.model.paging	0	23	0	1	0
org.jabref.model.pdf	0	132	0	1	0
org.jabref.model.pdf.search	0	79	5	0,94	0,06
org.jabref.model.push	0	34	0	1	0
org.jabref.model.search	0,5	35	17	0,17	0,33
org.jabref.model.search.matchers	0,14	33	39	0,32	0,61
org.jabref.model.search.rules	0,07	457	145	0,69	0,41
org.jabref.model.strings	0	525	0	1	0
org.jabref.model.study	0	92	3	0,97	0,03
org.jabref.model.texparser	0	218	6	0,97	0,03
org.jabref.model.util	0,2	349	6	0,78	0,02
org.jabref.performance	0	0	0	1	1
org.jabref.preferences	0,05	1806	848	0,63	0,32
org.jabref.styletester	0	0	14	0	1
org.jabref.support	0,5	0	1	0,5	1
org.jabref.testutils	0	0	4	0	1
org.jabref.testutils.category	1	0	0	0	1

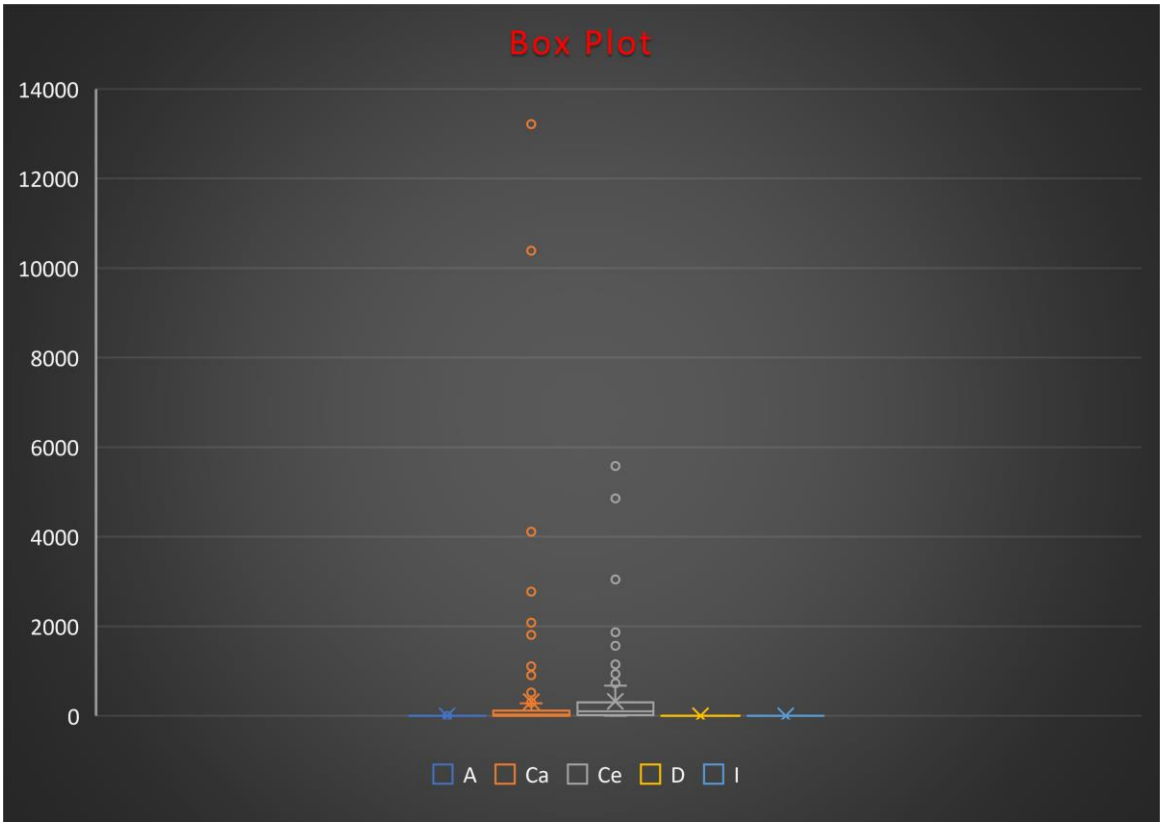
	A	Ca	Ce	D	I
Minimum	0	0	0	0	0
Q1	0	3	17	0,02	0,33
Median	0	33	102	0,17	0,85
Q3	0,06	119	299	0,54	0,98
Maximum	1	13210	5580	1	1
Mean	0,06389	316,87	321,724	0,32422	0,65611
Range	1	13210	5580	1	1
IQR	0,06	116	282	0,52	0,65
	0,09	174	423	0,78	0,975

Lower bound	-0,09	-171	-406	-0,76	-0,645
Upper bound	0,15	293	722	1,32	1,955

Values outside of these boundaries are considered outliers

Metrics of package \src\main\java\org\jabref\logic\importer\fetcher

A	Ca	Ce	D	I
0,05	165	4854	0,01	0,97



Metrics – Martin Packaging Metrics

Package metrics are used to help programmers increase the quality of the design and reduce the cost of development of software. These metrics help determine which packages are hard to maintain and/or lack abstraction, stability, coupling. The Martin Packaging Metrics, proposed by Robert Martin, these metrics are focused on identifying poorly designed packages and measure a few characteristics.

Abstractness (A) -> The ratio between the number of abstract classes and the number of overall classes in the package. Ranges between 0 and 1, where 0 is package devoid of any abstraction and 1 a totally abstract one.

Afferent Coupling (Ca) -> The number of classes outside of the package that depend on the package.

Efferent Coupling (Ce) -> The number of classes outside of the package in which the package depends on.

Instability (I) -> Represented by the following equation: $\boldsymbol{I = \frac{Ce}{Ca + Ce}}$, ranging between 0 and 1, where 0 is a indicates a stable package whereas 1 indicates maximum instability.

Distance from the main sequence (D) -> $D = |A + I - 1|$, ranging between 0 and 1 where 0 stands for a package that coincides with the main sequence and 1 a package as far from the main sequence as possible.

Looking at the data collected from all the JabRef packages and comparing to the \logic\importer\fetcher package, we can observe that the Ce value is a statistical outlier. Which means that there are a lot of classes outside of the package that it depends on. This coupled with the fact that the Abstractness is also a low number explains why the ComplexSearchQuery.java code smell exists. The code should be reformatted to improve abstraction and reduce efferent coupling to ultimately reduce complexity.

Metrics Review

Reviewer: Bruna Arroja

Reviewed: Pedro Nunes

Martin Packaging Metrics

These metrics were used throughout the program, with a special attention in the package ‘fetcher’ which has a ‘long parameter list’ code smell.

After analyzing the results, we can indeed relate the code smell to one of the Martin’s metrics, Abstractness, due to the low value that the package presents.

I think the metrics are well explained, starting on the introduction, then defining the metrics and at the end the relationship between the results and one code smell.

I have nothing to correct or add.

Code smell – Primitive obsession

Path: src\main\java\org\jabref\logic\importer\fileformat\RisImporter.java

```
String author = "";
String editor = "";
String startPage = "";
String endPage = "";
String comment = "";
Optional<Month> month = Optional.empty();
Map<Field, String> fields = new HashMap<>();
```

These string instances could be replaced by a StringBuilder because they all take advantage of concatenation.

```
if (!comment.isEmpty()) {
    comment = comment + OS.NEWLINE;
}
comment = comment + value;
```

When a String is appended at the end of another String it must make a conversion to an intermediate object and then it is converted back into being a String.

Therefore, using String Builder would be more efficient in this case.

Reviewer: António Brejo

I also think this class has a primitive obssession code smell,

and the refactor looks good.

Code smell – Switch Statement

Path : src\main\java\org\jabref\logic\bst\BibtexCaseChanger.java

```
switch (format) {
    case TITLE_LOWERS:
    case ALL_LOWERS:
        if ("L O OE AE AA".contains(s)) {
            sb.append(s.toLowerCase(Locale.ROOT));
        } else {
            sb.append(s);
        }
        break;
    case ALL_UPPERS:
        if ("l o oe ae aa".contains(s)) {
            sb.append(s.toUpperCase(Locale.ROOT));
        } else if ("i j ss".contains(s)) {
            sb.deleteCharAt(sb.length() - 1); // Kill backslash
            sb.append(s.toUpperCase(Locale.ROOT));
            while ((pos < c.length) && Character.isWhitespace(c[pos])) {
                pos++;
            }
        } else {
            sb.append(s);
        }
        break;
    default:
        LOGGER.info("convertAccented - Unknown format: " + format);
        break;
}
```

```
switch (format) {
    case TITLE_LOWERS:
        if ((i == 0) || (prevColon && Character.isWhitespace(c[i - 1]))) {
            sb.append(c[i]);
        } else {
            sb.append(Character.toLowerCase(c[i]));
        }
        if (c[i] == ':') {
            prevColon = true;
        } else if (!Character.isWhitespace(c[i])) {
            prevColon = false;
        }
        break;
    case ALL_LOWERS:
        sb.append(Character.toLowerCase(c[i]));
        break;
    case ALL_UPPERS:
        sb.append(Character.toUpperCase(c[i]));
        break;
    default:
        LOGGER.info("convertCharIfBraceLevelIsZero - Unknown format: " + format);
        break;
}
```

The use of these three switch statements in a row is a code smellbecause if it is needed to add a new condition, it will require to change every switch statement. Therefore, it should be replaced with some polymorphism to generalize these formats.

Review

Reviewer: Bruna Arroja

Reviewed: Pedro Nunes

I can identify the switch statement code smell on the BibtexCaseChanger class, and his justification is correct.

Code smell - Long parameter list

Path: src\main\java\org\jabref\logic\importer\fetcher\ComplexSearchQuery.java

```java
public class ComplexSearchQuery {
    // Field for non-fielded search
    private final List<String> defaultField;
    private final List<String> authors;
    private final List<String> titlePhrases;
    private final List<String> abstractPhrases;
    private final Integer fromYear;
    private final Integer toYear;
    private final Integer singleYear;
    private final String journal;
    private final String doi;

    private ComplexSearchQuery(List<String> defaultField, List<String> authors, List<String> titlePhrases, List<String> ab
        this.defaultField = defaultField;
        this.authors = authors;
        this.titlePhrases = titlePhrases;
        this.abstractPhrases = abstractPhrases;
        this.fromYear = fromYear;
        // Some APIs do not support, or not fully support, year based search. In these cases, the non applicable parameter
        this.toYear = toYear;
        this.journal = journal;
        this.singleYear = singleYear;
        this.doi = doi;
    }
```

This constructor method has way too many parameters and it's apparent that some of these attributes have relationships between each other. Perhaps there should be more constructor methods with less parameters.

Review

Reviewer: Bruna Arroja

Reviewed: Pedro Nunes

For sure a long parameter list code smell, fixing this problem may involve some refactoring and even reveal some unnoticed duplicate code.