

**Software Engineering Project**

Phase 1

Diogo Ye 56726

Tiago Vieira 57719

Pedro Ribeiro 55921

Thiago Monteiro 55355

Yuliia Bila 54605

**Use cases**

Groups

Diagrama

Descrição gerada automaticamente

**Use case: Create Group**

ID: 1

Description: Creates group

Primary actors: User

Pre-conditions: Have a library.

Main Flow:

1. The use case starts when the user clicks on the “add group” button.

2. The User enters the group name (it can leave the other details as default).

3. The User selects the type of change that he wants the system to do.

4. The User clicks in the “ok” button.

Post-conditions: A new group is added with the given name.

**Use case: Add entries**

ID: 2

Description: Adds an entry to an existing group.

Primary actors: User

Pre-conditions: Have entries.

Main Flow:

1. The User selects the entries.

2. The User drags the selected entries above the group.

Post-conditions: The selected entries are added to the group.

**Use case: Add subgroup**

ID: 3

Description: Adds a subgroup to an existing group.

Primary actors: User

Pre-conditions: Have a group.

Main Flow:

1. The User right-clicks above the group.

2. The User selects the “add subgroup” option.

3. The User enters the subgroup name (it can leave the other details as default).

Post-conditions: The subgroup is added to the group.

**Use case: Remove groups/subgroups**

ID: 4

Description: Removes existing group/subgroup

Primary actors: User

Pre-conditions: Have groups.

Main Flow:

1. The User right clicks above the group or subgroup.

2. The User selects the “remove group option” and chooses whether to keep or remove the group’s subgroups.

Post-conditions: The group is removed.

**Use case: Edit group**

ID: 5

Description: The User edits the group details.

Primary actors: User

Pre-conditions: Have groups.

Main Flow:

1. The User right clicks above the group or subgroup.

2. The User selects the “edit group” option.

3. The User edits the details that he wants: Name, Description, Color, Icon, Hierarchical Context, group Type (explicit selection, searching for a keyword, free search expression, specified keywords, cited entries) .

Post-conditions: The group details are changed.

**Use case: Search Group**

ID: 6

Description: The User searches for the group’s name.

Primary actors: User

Pre-conditions: Have groups.

Main Flow:

1. The User enters the group name in the “Filter group” text box and presses “Enter”.

Post-conditions: Only the groups that match that name appear on groups tab.

**Use case: Display union/intersection of groups**

ID: 7

Description: The User chooses to see the intersection/union of several groups.

Primary actors: User

Pre-conditions: Have more than one group.

Main Flow:

1. The User selects several groups.
2. The User clicks on the “Toggle intersection/union” button.

Post-conditions: If selected intersection the entries that are in all the selected groups are shown, if selected union all the entries in the selected groups are shown.

Edit

Diagrama

Descrição gerada automaticamente

**Use case: Compares files**

ID: 8

Description: User compare files with other sources

Primary actors: User

Secondary actors: Web sites

Pre-conditions: 1. User need to have articles.

Main Flow:

1. The use case starts when the user tries to compare the files.

2. The User compares the files with the information of other sites selected by him.

3. The User selected the type of change that he wants the system to do.

4. The system makes the changes and show the User the results.

5. The User selects if he wants to change or not.

Post-conditions: The files are filtered according with the User actions.

**Use case: Define rules**

ID: 9

Description: User customize rules

Primary actors: User

Secondary actors:

Pre-conditions: 1. User need to have articles.

Main Flow:

1. The use case starts when the user defines his preferences.

2. The system makes the changes (rename/move the files) based on the rules.

Post-conditions: The files are renamed and organize according with the rules

Cite

Diagrama

Descrição gerada automaticamente

**Use case: Inserting Citations**

ID: 10

Description: Inserts citations

Primary actors: User

Secondary actors: Microsoft word

Pre-conditions: 1Cited document in JabRef, a third party software (either BibTex/biblatex, Microsoft word, OpenOffice/LibreOffice)

Main Flow: 1. The use case starts when you select an entry to be cited in an external application

2. The system grabs the entry and inserts it into the selected document format (eg. Microsoft word)

3. Organizes bibliography.

Post-conditions: The cited document still exists.

Add entries

Diagrama

Descrição gerada automaticamente

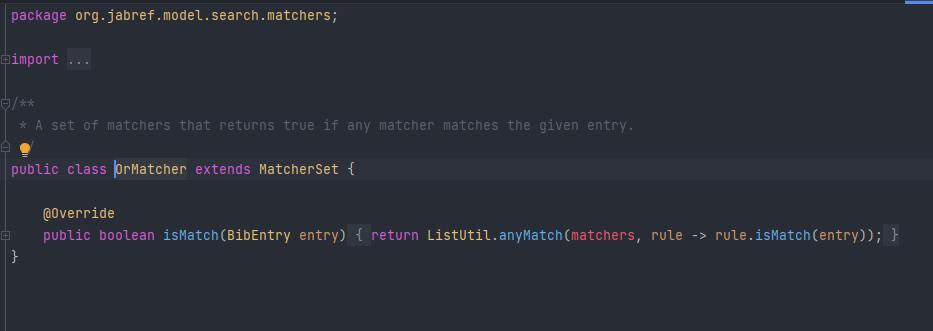
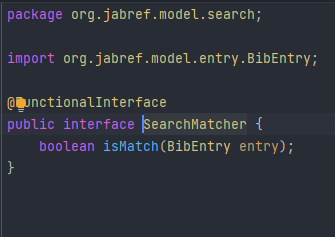
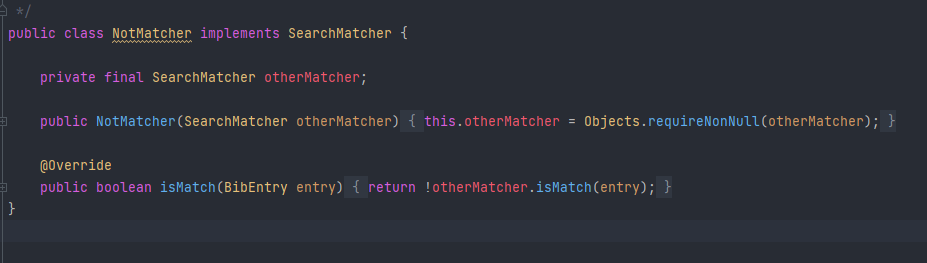
Diagrama

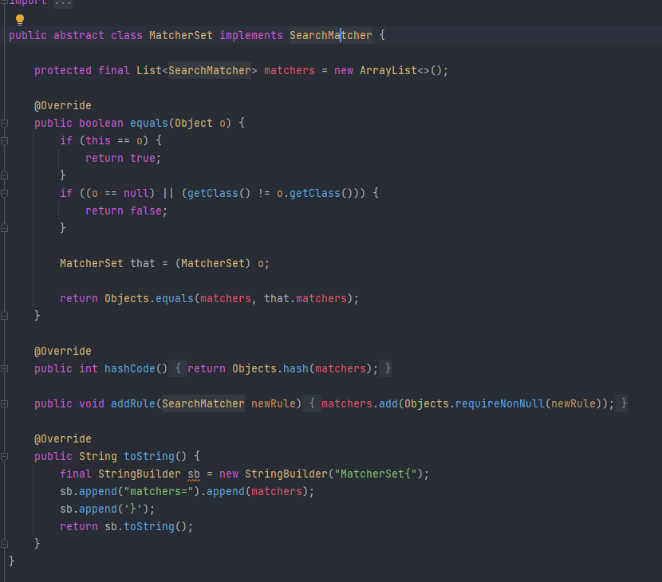
Descrição gerada automaticamente

**Design Patterns**

Diogo Ye

**Template Method Pattern**







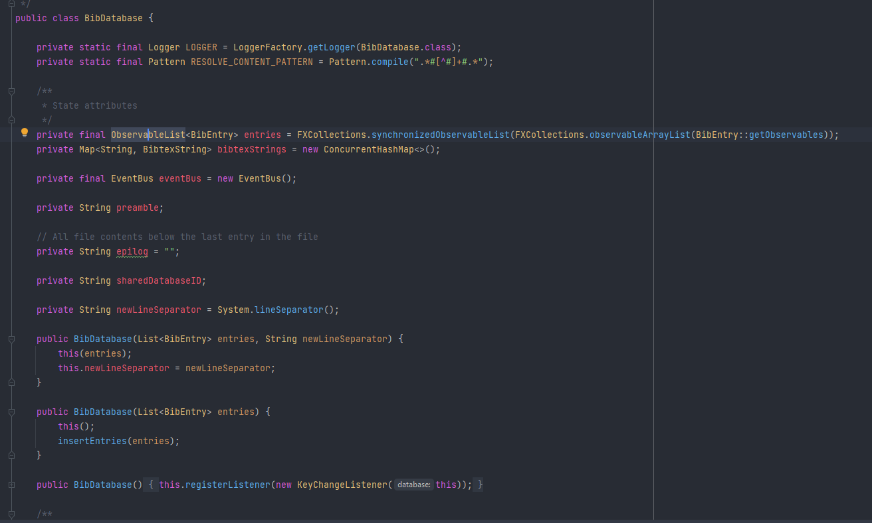
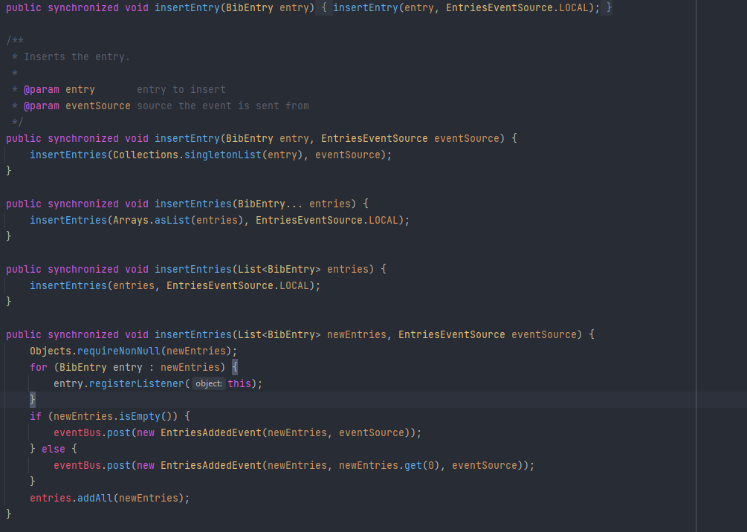
D:\IntelliJ IDEA Community Edition 2021.2.2\SE2122\_54605\_55355\_55921\_56726\_57719\src\main\java\org\jabref\model\search\matchers\MatcherSet.java

We can see a Template Method Pattern, because the subclasses have the same method implemented in different ways.

**Façade Pattern**

D:\IntelliJ IDEA Community Edition 2021.2.2\SE2122\_54605\_55355\_55921\_56726\_57719\src\main\java\org\jabref\model\groups\AbstractGroup.java

Façade Pattern contains and change all the information about groups. The creation of different groups and different methods.

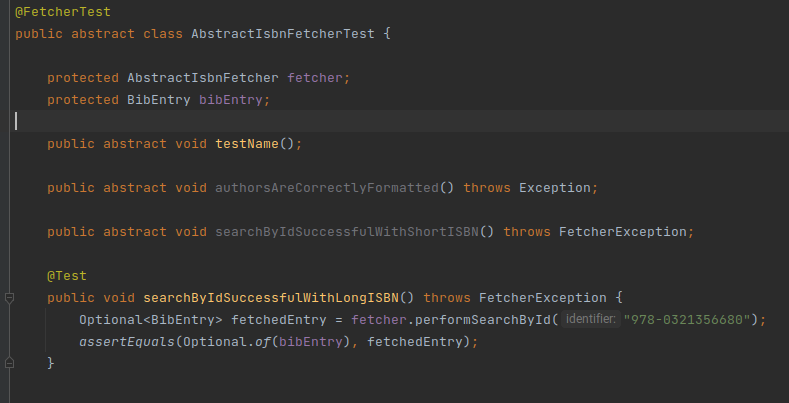
D:\IntelliJ IDEA Community Edition 2021.2.2\SE2122\_54605\_55355\_55921\_56726\_57719\src\main\java\org\jabref\model\database\BibDatabase.java

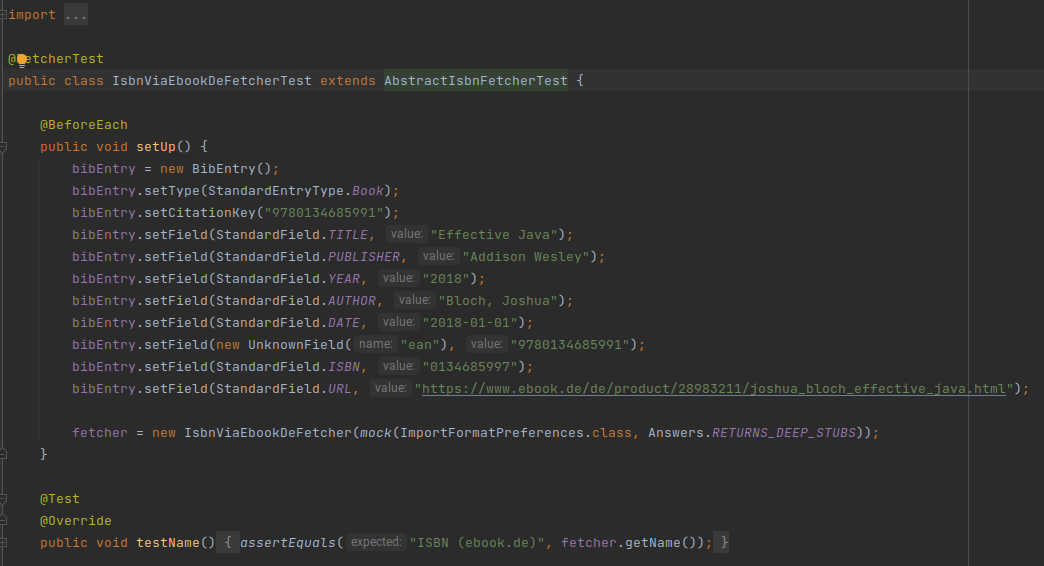
Observer Pattern, because when the list changes, the data structure changes the value automatically in the data changed.

Thiago Monteiro

**Singleton Pattern**

src/main/java/org/jabref/gui/externalfiletype

**Template Method Pattern -** src/test/java/org/jabref/logic/importer/fetcher



**Chain of Responsibility Pattern**

src/main/java/org/jabref/gui/FallbackExceptionHandler.java



Tiago Vieira

**Factory Method**

Texto

Descrição gerada automaticamente

Location - src/main/java/org/jabref/model/entry/types/EntryTypeFactory.java

Receives a type as a String and returns the type according with that. If the type does not exist it creates a new unknown type.

**Observer Pattern**

Code Snippet

Texto

Descrição gerada automaticamente

Location - src/main/java/org/jabref/gui/collab/DatabaseChangeMonitor.java

Checks if the file has changes (using ChangeScanner class). If it has changes notifies all observers (listeners).

**Builder Pattern**

Code Snippet

Texto

Descrição gerada automaticamente

Texto

Descrição gerada automaticamente

Location - src/main/java/org/jabref/model/entry/BibEntryTypeBuilder.java

It receives the attributes of the BibEntryType to return. When it receives the build() command it returns the BibEntryType with the attributes it received previously, allowing BibEntryType to be immutable.

Pedro Ribeiro

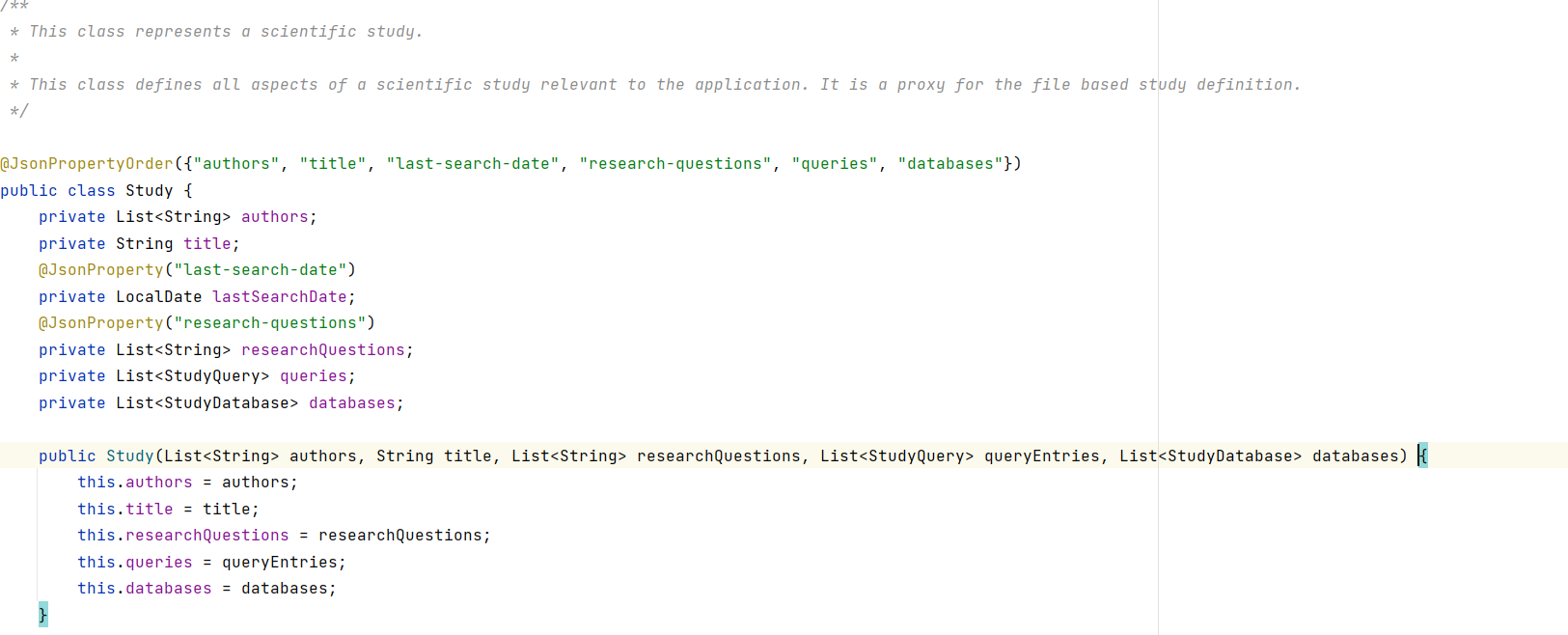
**State Pattern**

Encontramos aqui um proxy state pattern pois representa um virtual proxy para uma versão mais "leve" do que o sistema tradicional dum ficheiro study

src/main/java/org/jabref/model/study/Study.java

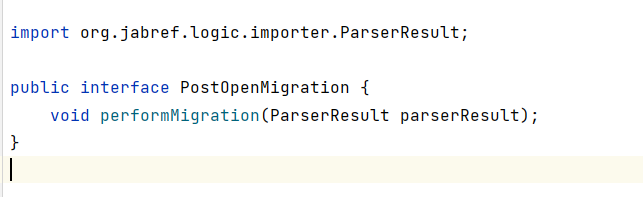
é instanciado aqui src/main/java/org/jabref/gui/slr/StartNewStudyAction.java

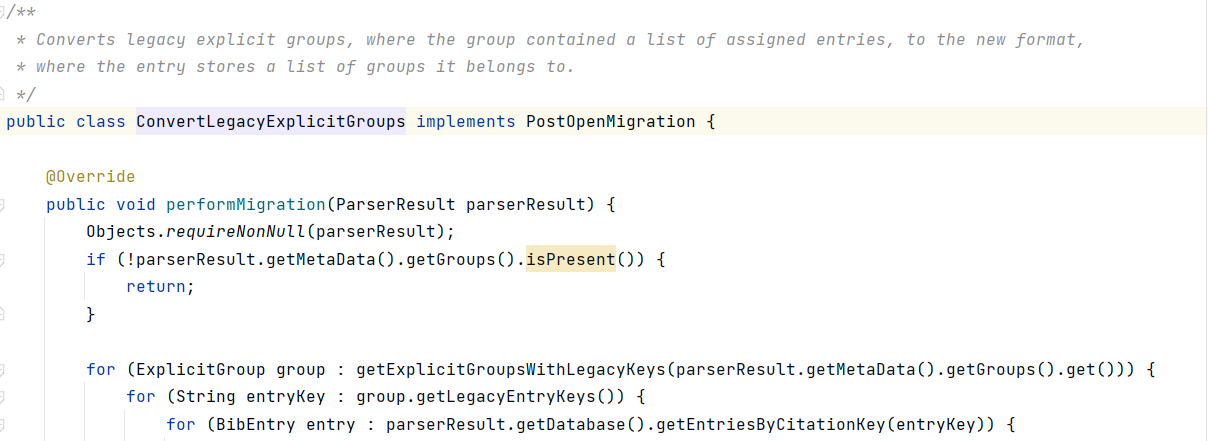
e por sua vez é chamado no menu() src/main/java/org/jabref/gui/JabRefFrame.java

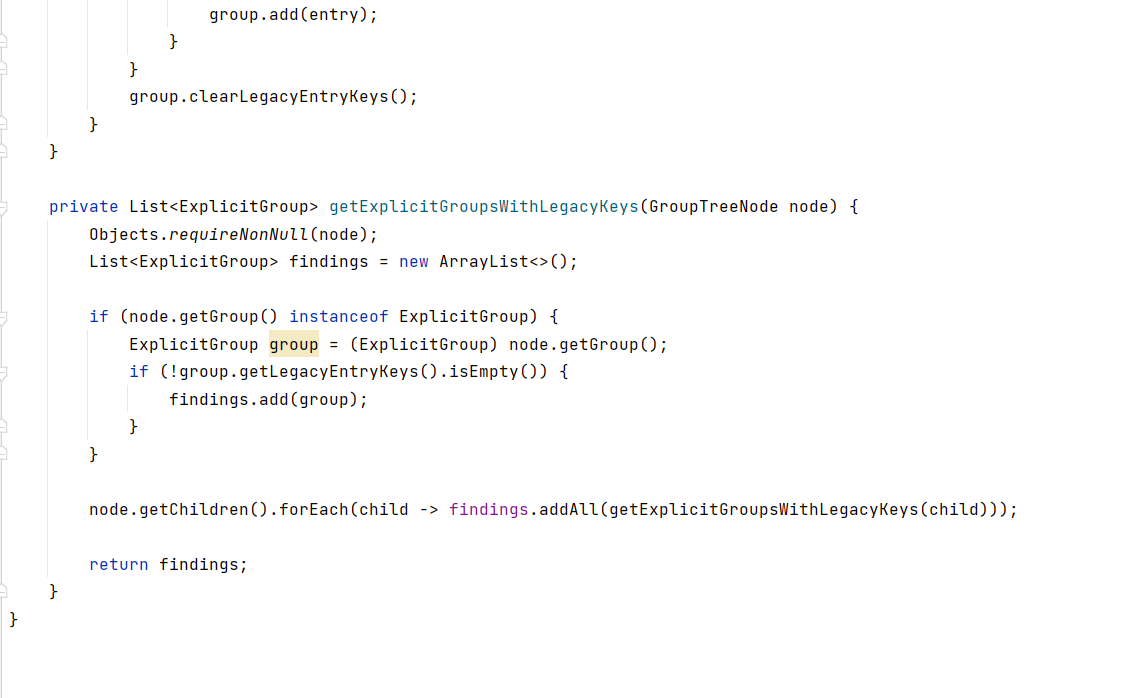




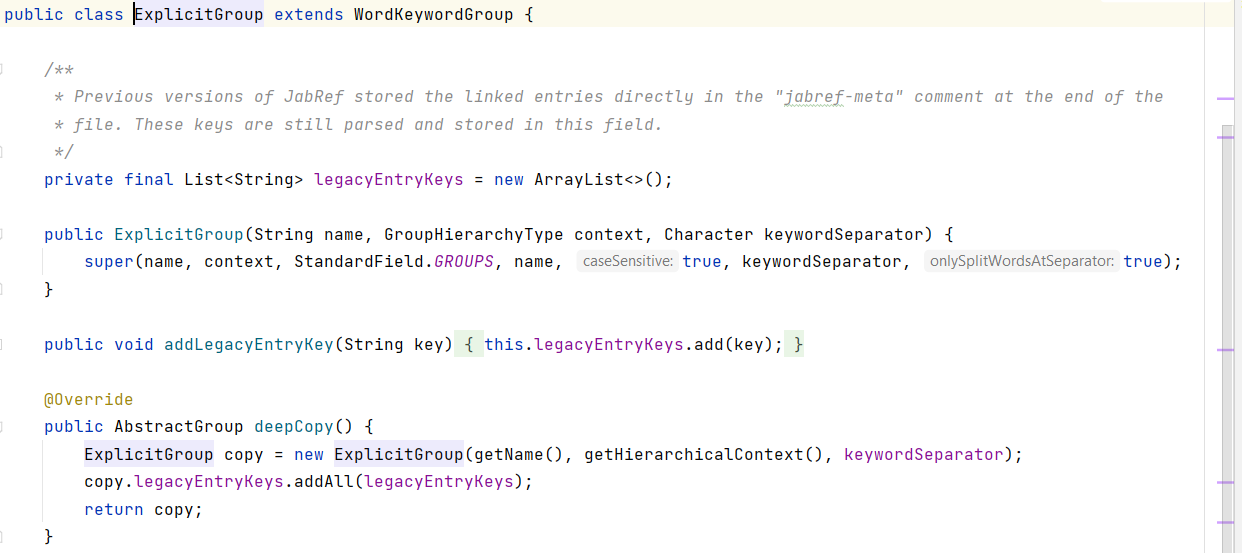
**Adapter Pattern**



src/main/java/org/jabref/migrations/PostOpenMigration.java 



src/main/java/org/jabref/migrations/ConvertLegacyExplicitGroups.java



src/main/java/org/jabref/model/groups/ExplicitGroup.java

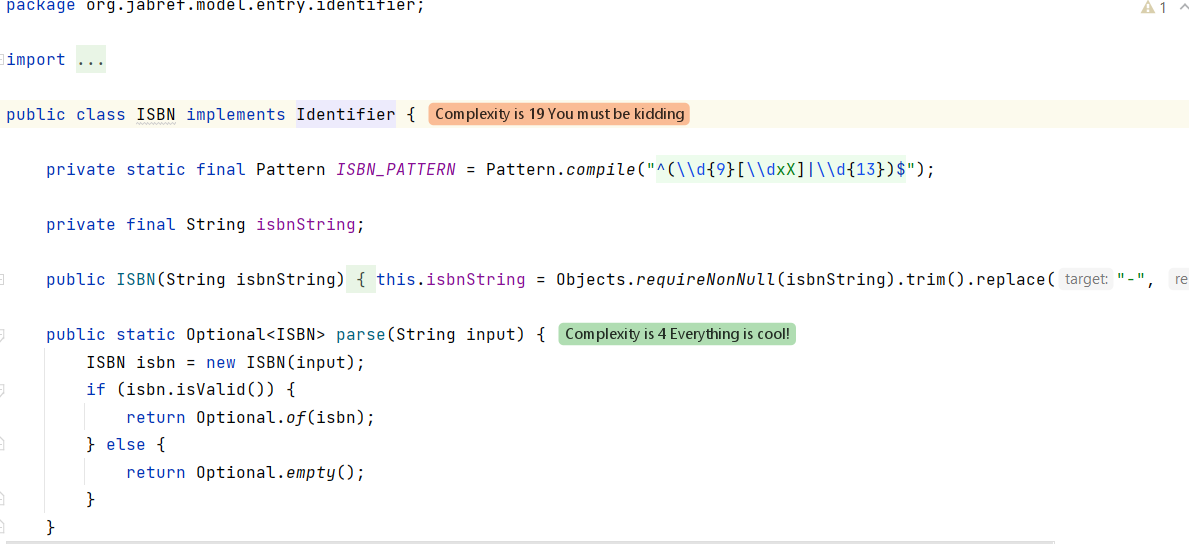
Isto é claramente um adapter pattern pois formata objectos previamente criados para um novo formato utilizado

**Bridge Design Pattern**

encontramos no identifier um bridge design pattern visto que adapta sistemas diferentes e serve como ponte para perceber ou representar sistemas independentes.

src/main/java/org/jabref/model/entry/identifier/Identifier.java



src/main/java/org/jabref/model/entry/identifier/ISBN.java

src/main/java/org/jabref/model/entry/identifier/ArXivIdentifier.java

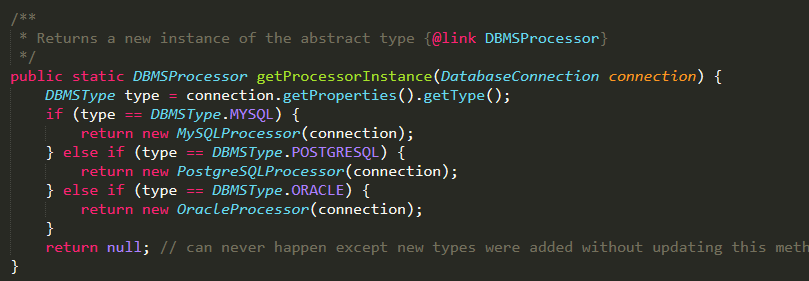


Yuliia Bila

**Design Pattern – Factory**

When getting a processor instance the getProcessorInstance function looks at the connection type and chooses the appropriate SQL processor class.

\src\main\java\org\jabref\logic\shared\DBMSProcessor.java



**Design Pattern – Template**

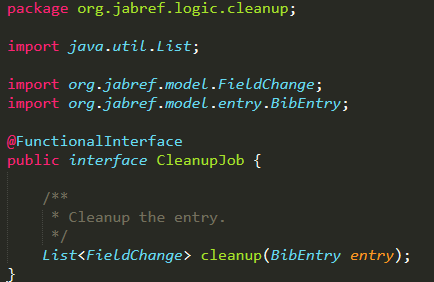
Many classes implement the methods (mainly the cleanup method) from the CleanupJob class, which then allows flexibility when running jobs from a list.

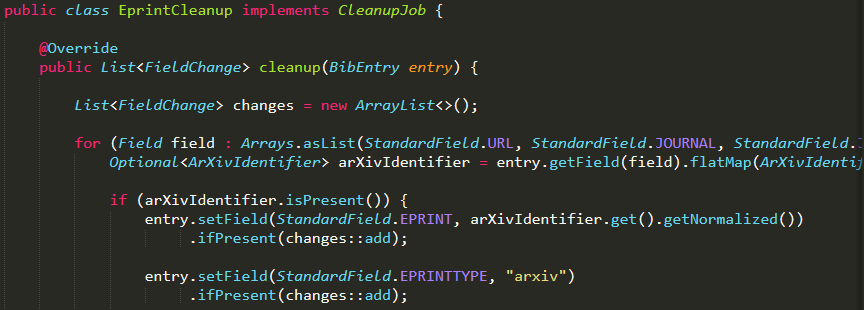
src\main\java\org\jabref\logic\cleanup\CleanupJob.java

src\main\java\org\jabref\logic\cleanup\EprintCleanup.java

src\main\java\org\jabref\logic\cleanup\FileLinksCleanup.java

and other classes that implement CleanupJob as well.







**Design Pattern – Observer**

In this file, the Observer pattern is implemented as there is a subscription to a listener.

\src\main\java\org\jabref\gui\cleanup\CleanupPresetPanel.java



**Metrics Sets**

Diogo Ye

**Chidamber-Kemerer metrics**

The set Chidamber-Kemerer metrics give us Coupling Between Objects (CBO), Depth of inheritance tree (DIT), lack of cohesion of methods (LCOM), number children (NOC), response for class (RFC), Weighted method complexity (WMC).

The CBO give us the number of classes to which a class is coupled. The DIT give use the maximum inheritance path from the class to the root class. NOC give us the number of immediate sub-classes of a class, WMC give us the number of methods defined in the class, and RFC is the number of methods in the class plus the number of remote methods called by the class.

In the collected metrics we can check that the Coupling Between Objects parameter of the BibEntry is too high and possible a code smell (Feature Envy, Inappropriate Intimacy), because the high value of the parameter describe that the class has multiples calls to other classes.

In the same class we can check that the Weighted method complexity, which is the number of methods defined in the class, is too high and creating a possibility to a code smells like Large Class.

A high value of response for class parameter could result in a complex and hard to understand class, creating a code smell.

Pedro Ribeiro

**MOOD metrics**,quinta, 2 dez 2021 17:03:40 GMT

Project,AttributeHidingFactor,AttributeInheritanceFactor,CouplingFactor,MethodHidingFactor,MethodInheritanceFactor,PolymorphismFactor

"project","75,70%","25,70%","0,99%","27,28%","23,11%","49,23%"

Mood metrics representa metrics para object oriented factors, dos quais analisamos 6 factores .

O AHF e o MHF servem para medir o uso de medidas para esconder informação do qual é suposto ajudar com: Complexidade, reduzir efeitos secundarios provocados por melhoramento da implementação, entre outros. Valores baixos de MHF uma implementação muito abstracta do projecto, enquanto um valor muito alto representaria pouca funcionabilidade no projecto .Em termos de AHF é ideal ser alto pois estes atributos só seriam acedidos pelas suas classes, pois quando o AHF aumenta a complexidade do projecto diminui.

O MIF e AIF sao medidas de herança. Isto é herança é um mecanismo que expressa a similança entre as classes que permite uma generalização e especialização entre relação das classes. É preciso ter cuidado em não só usar MIF pois isso dificultaria a testibilidade e a compreensibilidade do código.

O COF representa a percentagem de codigo que nao imputável pela herança. Podemos afirmar que queremos que as classes comuniquem pouco entre umas e outras, pois quão mais coupling relations maior a complexidade. Por tanto quão menor o COF melhor seria.

Polimorphismo represnta a capacidade de tomar diferentes formas, em sistemas orientados a objectos permite que a implementação duma operação seja dependente no objecto que está em causa. Um polimorphismo alto representaria uma maior dificuldade em fazer debug num projecto.

Podemos verificar que neste projecto tem um polimorphismo alto o que não é bem desejado pela informação dita anteriormente, também tem um AIF baixo o que explica que uma falta de simplificidade no código.

Podemos aqui verificar com valores baixos de AIF que este codigo precisa de comentarios a mais para explicar as funções do código dai levar com code smells de demasiado comentarios e com um polimorphismo alto pode representar um long method que é por sua vez outro caso de code smells.

Thiago Monteiro

### Martin Packaging Metric

Efferent Coupling (Ce) – É o número de classes de um package que dependem de outros packages. Mede a vulnerabilidade do package em questão a mudanças em packages que depende. O alto valor da métrica Ce> 20 indica instabilidade de um pacote, a mudança em qualquer uma das inúmeras classes externas pode causar a necessidade de alterações no pacote. Os valores preferidos para a métrica Ce estão na faixa de 0 a 20, valores mais altos causam problemas com cuidado e desenvolvimento de código.

Afferent Coupling (Ca) – Parecido com o Ce, porém mede outros tipos de dependências como incoming dependencies. Possibilita medir a sensibilidade dos packages remanescentes à mudanças. Altos valores de Ca métrico geralmente sugerem alta estabilidade do componente. Isso se deve ao fato de que a classe depende de muitas outras classes. Portanto, não pode ser modificado de forma significativa, pois, neste caso, aumenta a probabilidade de propagação de tais alterações. Os valores preferidos para a métrica Ca estão na faixa de 0 a 500.

Instability (I) – Essa métrica é usada para medir a suscetibilidade relativa da classe às mudanças. É a razão de todas as outgoing dependencies para todas as dependências de um package. Com base no valor da métrica I, podemos distinguir dois tipos de componentes:

Os que têm muitas dependências de saída e não muitas das de entrada (valor I próximo a 1), que são bastante instáveis devido à possibilidade de mudanças fáceis nesses pacotes;

Os que têm muitas dependências de entrada e não muitas de saída (o valor I é próximo a 0), portanto, são um pouco mais difíceis de modificar devido à sua maior responsabilidade.

Os valores preferidos para a métrica I devem estar dentro dos intervalos de 0 a 0,3 ou 0,7 a 1. Os pacotes devem ser muito estáveis ou instáveis, portanto, devemos evitar pacotes de estabilidade intermediária.

Abstractness (A) – É a proporção de classes abstratas em um package. Os valores preferidos para a métrica A devem assumir valores extremos próximos de 0 ou 1. Pacotes que são estáveis (métrica I próxima de 0), o que significa que são dependentes em um nível muito baixo de outros pacotes, também devem ser abstratos (métrica A próxima a 1). Por sua vez, os pacotes muito instáveis (métrica I próxima de 1) devem ser constituídos por classes concretas (métrica A próxima a 0). No caso ótimo, a instabilidade da classe é compensada por sua abstração, há uma equação I + A = 1.

Normalized Distance from Main Sequence (D) – Mede o balanço entre Estabilidade e Abstração. D = | A + I - 1 | (A- abstração, I – instabilidade).

O valor da métrica D deve ser o mais baixo possível para que os componentes fiquem localizados próximos à sequência principal. Além disso, são considerados os dois casos extremamente desfavoráveis:

A = 0 e I = 0, um pacote é extremamente estável e concreto, a situação é indesejável porque o pacote é muito rígido e não pode ser estendido;

A = 1 e I = 1, situação bastante impossível porque um pacote completamente abstrato deve ter alguma conexão com o exterior, para que a instância que implementa a funcionalidade definida nas classes abstratas contidas neste pacote possa ser criada.

Fonte: kariera.future-processing.pl

# Análise do JabRef

Ce- No projeto podemos identificar mais de 127 packages com valores de Ce acima do ideal. Muitos packages tem valores muito altos chegando até um valor acima de 3000. Isso mostra que o projeto é muito sensível a mudanças.

Ca- No projeto podemos identificar 17 packages com valores de Ca fora do ideal. Os resultados não são tão maus quanto os de Ce, entretanto existem packages com valores superiores a 10.000. Isso mostra que o projeto tem alta sensibilidade a mudanças.

I- Ao analisar esta métrica podemos ver que existem 34 packages de estabilidade mediana.

A- Ao analisar está metrica, podemos ver que é a que esta mais de acordo com o desejável de todas. Existem apenas 8 packages com valores longe dos extremos.

D- Ao analisar esta métrica podemos observar 76 packages com valores de D acima de 0,3. Isso é um resultado muito longe do ideal

Tiago Vieira

**Complexity Metrics**

The average Cyclomatic Complexity of the project is 2.15 which is considered a good score, meaning it is easy to understand, maintain and add new features.

The class with the highest operation complexity is *FieldNameLabel* with a score of 49.00 because of the huge switch statement, causing the method to be hard to understand for someone who is reading it for the first time and also time consuming to maintain.

The method with the highest cognitive complexity is *refreshCiteMarkersInternal* in the *OOBibBase* class with a score of 189.00. This is caused by several nested *if* statements, making it hard to understand.

Yuliia Bila

**Lines of code**

“Measuring programming progress by lines of code is like measuring aircraft building progress by weight.” -- Bill Gates

My task is to analyze the line code metrics for the folder “migrations”.

The purpose of using this type of metrics is:

1. **Check the size of code units.** The analysis of different units of the program allows us to see some of the shortcomings made in the process of writing the code. For example, a method longer than 20-30 lines can be quite difficult to understand and work with. It is also highly undesirable for classes that are too long to search for lines of code and read the code.
2. **Estimate the size of project.** We can estimate the number of logical lines of code and physical lines of code, compare them. However, applying this metric, we must understand that the number of lines of code is not equal to well-made code.

We must clearly understand that the length of the code is not an indicator of the productivity and efficiency of the programmer. It is not uncommon for 20 lines of code to be significantly worse than 100 lines of code in terms of time or space complexity.

**Method metrics**

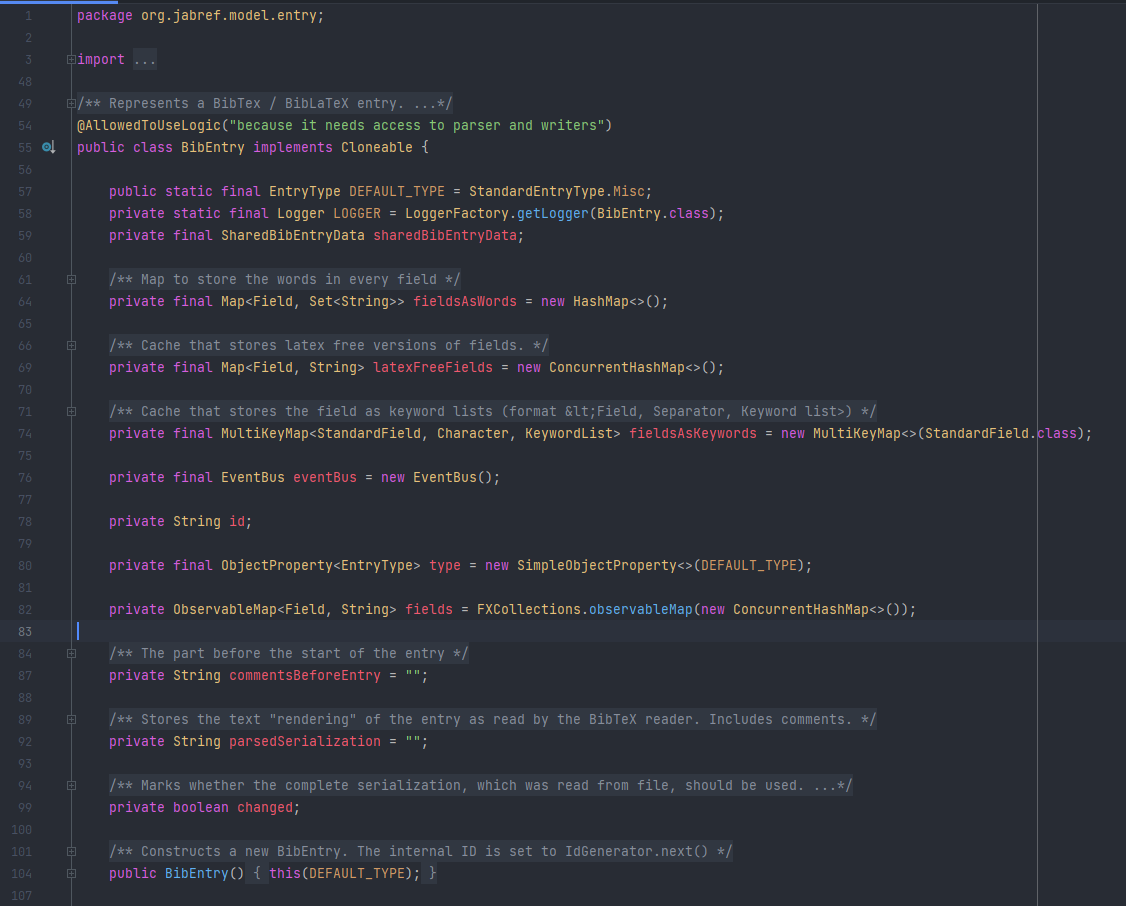
Analyzing the methods of the folder in question, we can conclude that most of the lines of code consist of actual code, rather than comments. There is a appropriate amount of comments in places where it is necessary, namely before long and more complex methods.

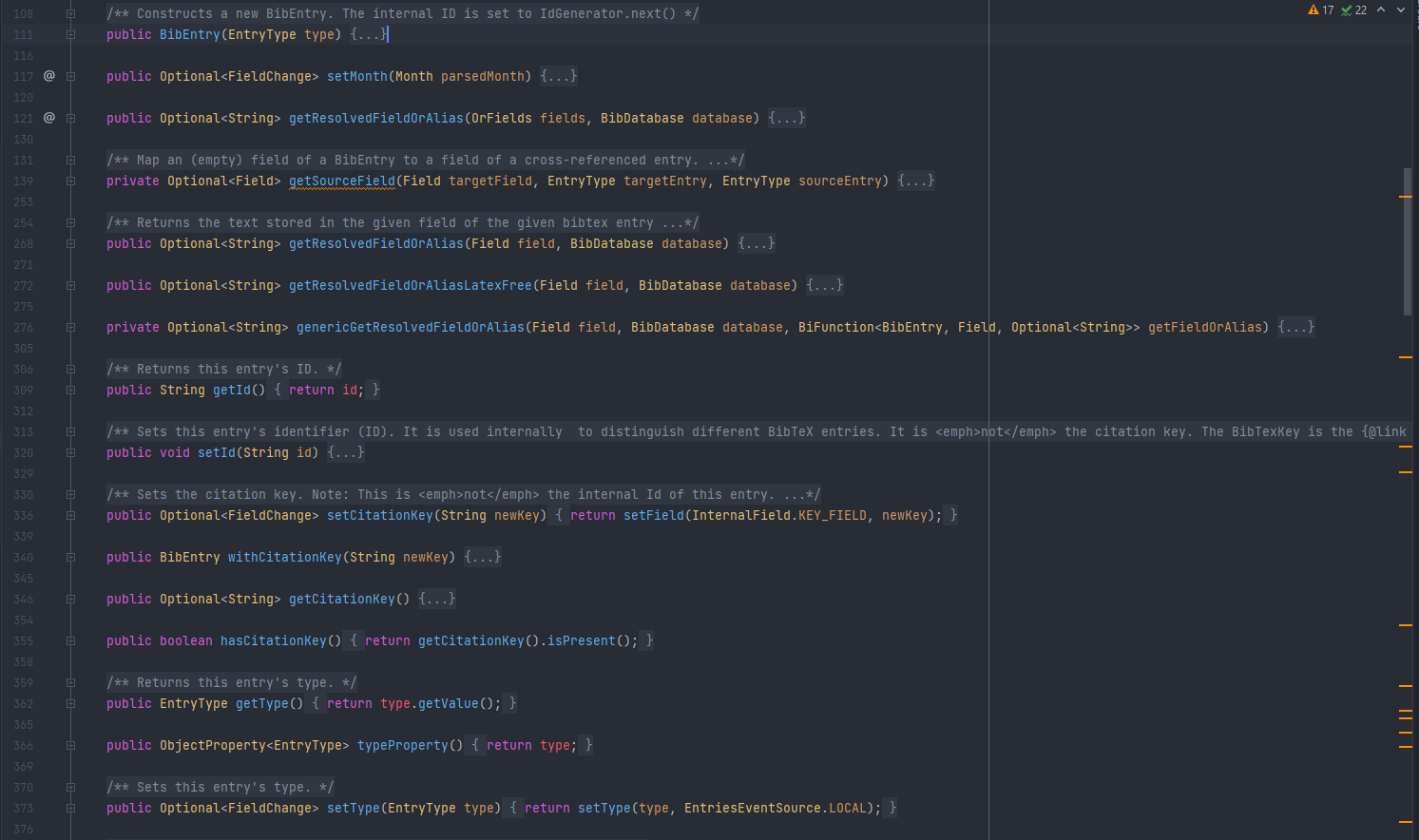
**Class metrics**

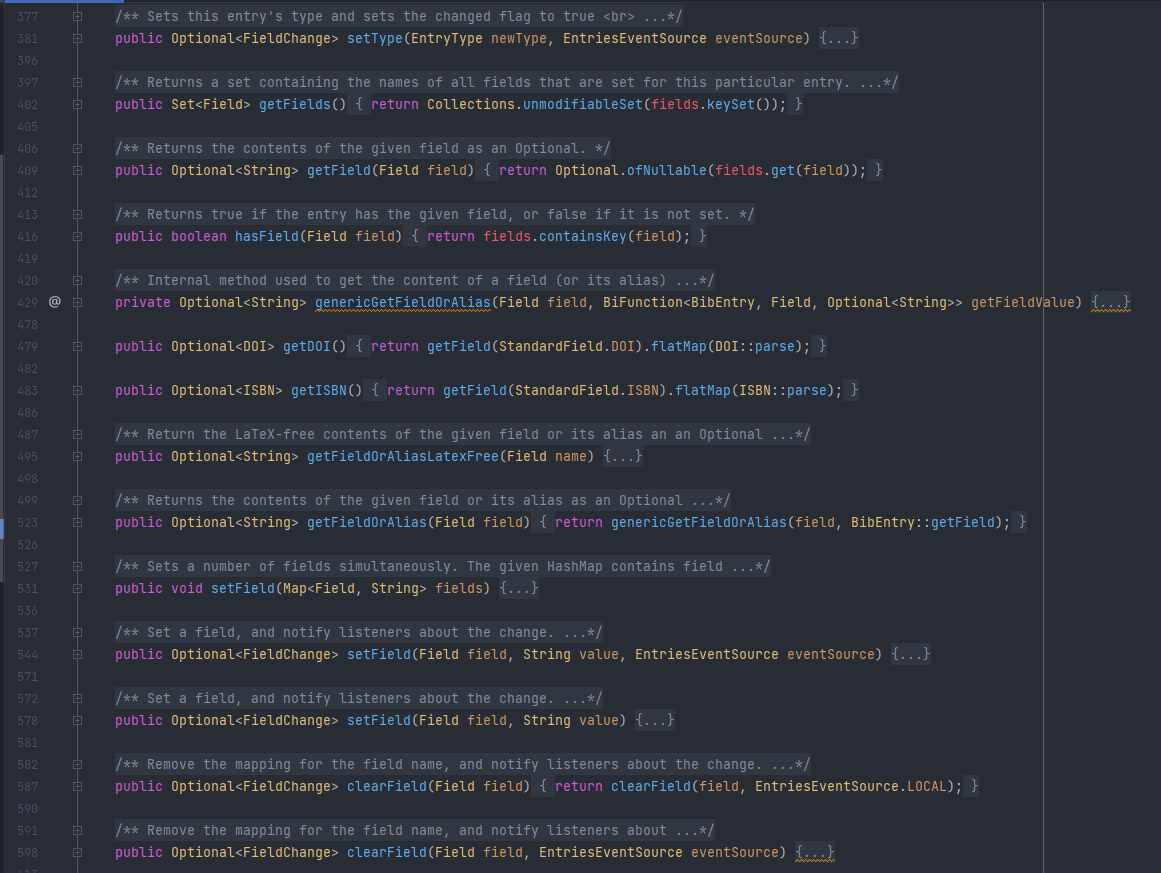
Looking at the classes, we can also say that the number of lines of code exceeds the number of comments. No unnecessary comments, everything looks structured and easy to read. In general, we can conclude that the classes and interfaces of this folder are quite balanced.

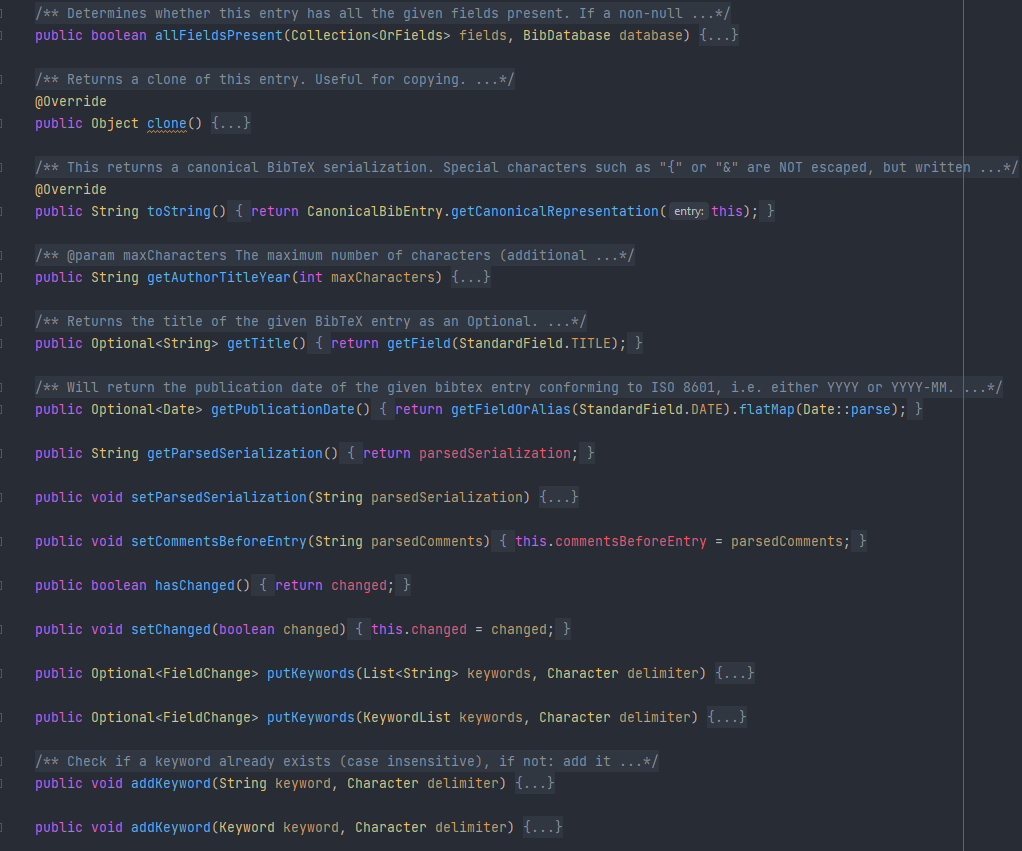
**Code smells**

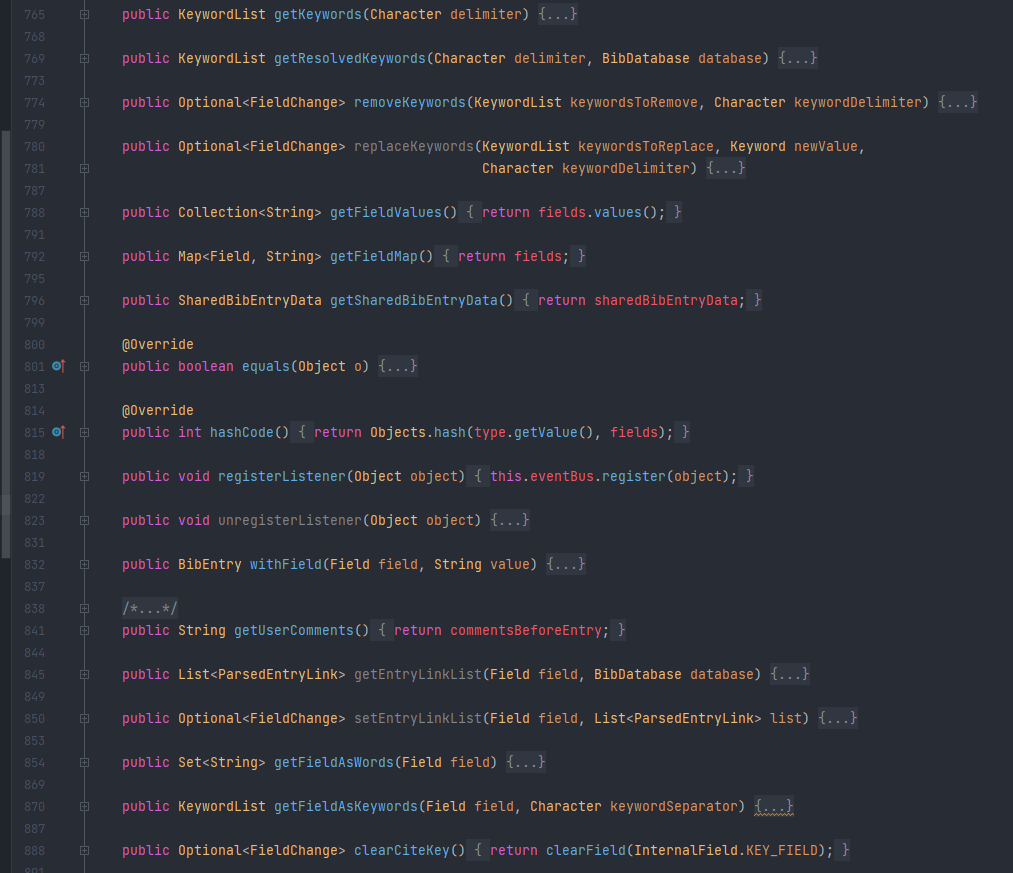
Diogo Ye

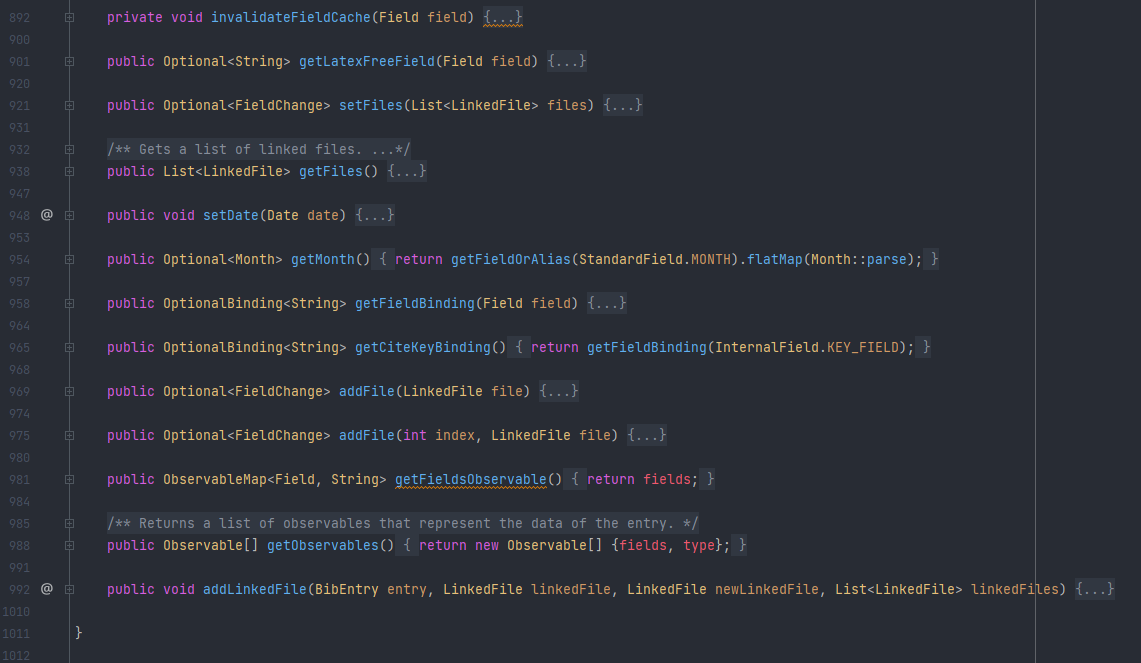






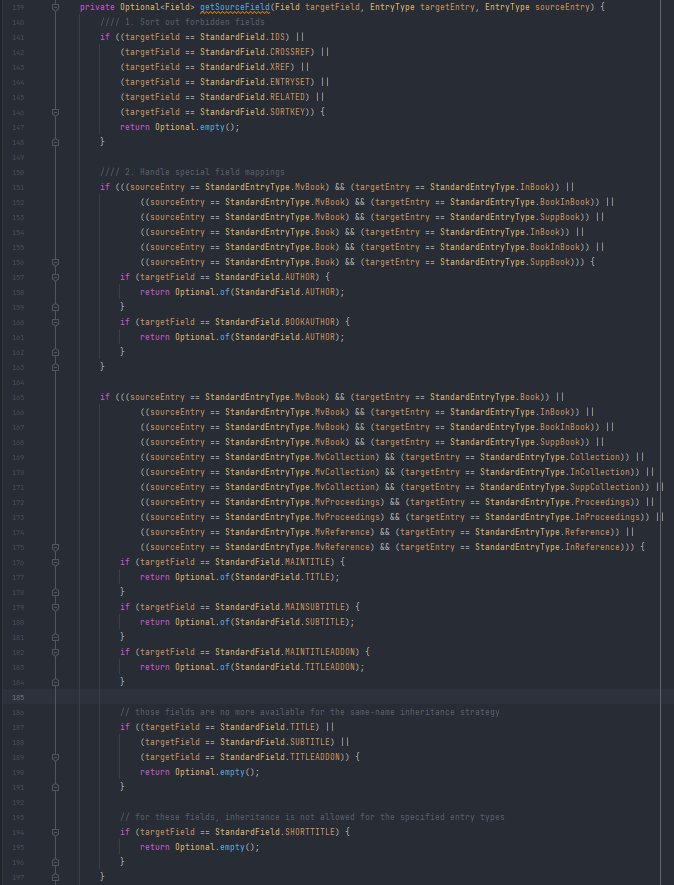






src/main/java/org/jabref/model/entry/BibEntry.java

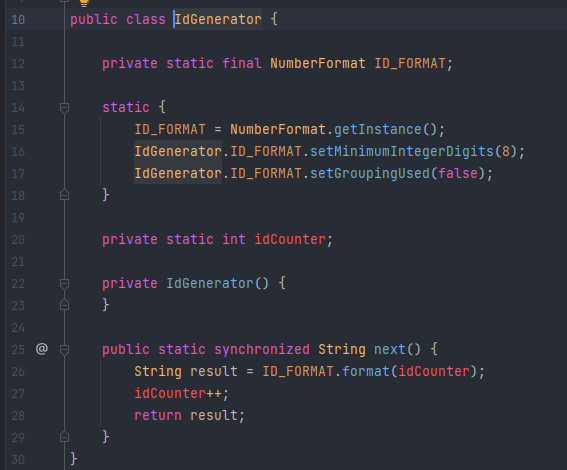
The code smell identified in this class is Large Class, we can check that the class has more than 1000 lines of code, maybe we can divide the responsibility of this class into other classes trying to decrease the number of methods and lines.





src/main/java/org/jabref/model/entry/BibEntry.java

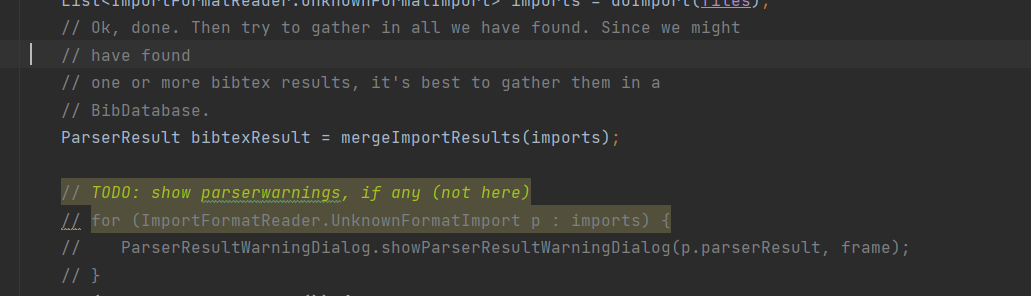
We can check that the code smells in this code is the long method, we can check that there are parts in the method which are very similar, the logic behind it is a little confusing the first time we see the code. We should divide the method into other methods making more readable and possibly refactoring the method by not using duplicated/ similar code.



src/main/java/org/jabref/model/entry/IdGenerator.java

The code smell where is Data Class, because this class only stores the information about the id. This class could be removed if we had and ID in the upper-level Class where the Id is needed and instead of making a class for it, generate the value and store in the upper-level.

Pedro Ribeiro

  
Too many comments

WHY

unecessary comments just clumping up the code and making it harder to read

where src/main/java/org/jabref/gui/importer/ImportAction.java

LONG METHOD

over 200 lines of code, should be seperated into auxilary methods to improve readability

src/main/java/org/jabref/logic/importer/fileformat/RisImporter.java  
RisImporter

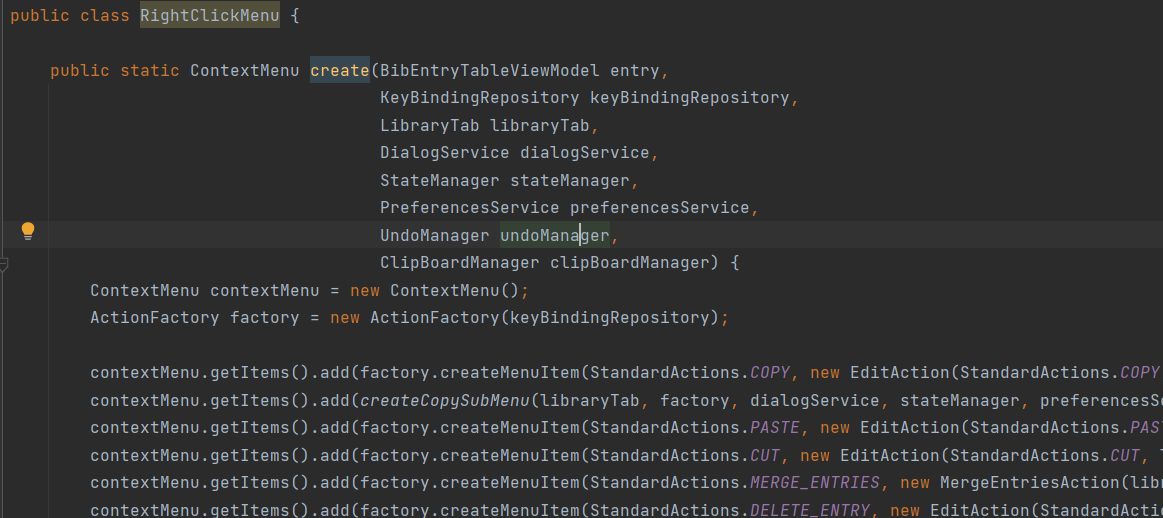
public ParserResult importDatabase(BufferedReader reader) throws IOException {  
 List<BibEntry> bibitems = new ArrayList<>();  
  
 // use optional here, so that no exception will be thrown if the file is empty  
 String linesAsString = reader.lines().reduce((line, nextline) -> line + "\n" + nextline).orElse("");  
  
 String[] entries = linesAsString.replace("\u2013", "-").replace("\u2014", "--").replace("\u2015", "--")  
 .split("ER -.\*(\\n)\*");  
  
 // stores all the date tags from highest to lowest priority  
 List<String> dateTags = Arrays.*asList*("Y1", "PY", "DA", "Y2");  
  
 for (String entry1 : entries) {  
  
 String dateTag = "";  
 String dateValue = "";  
 int datePriority = dateTags.size();  
 int tagPriority;  
  
 EntryType type = StandardEntryType.*Misc*;  
 String author = "";  
 String editor = "";  
 String startPage = "";  
 String endPage = "";  
 String comment = "";  
 Optional<Month> month = Optional.*empty*();  
 Map<Field, String> fields = new HashMap<>();  
  
 String[] lines = entry1.split("\n");  
  
 for (int j = 0; j < lines.length; j++) {  
 StringBuilder current = new StringBuilder(lines[j]);  
 boolean done = false;  
 while (!done && (j < (lines.length - 1))) {  
 if ((lines[j + 1].length() >= 6) && !" - ".equals(lines[j + 1].substring(2, 6))) {  
 if ((current.length() > 0) && !Character.*isWhitespace*(current.charAt(current.length() - 1))  
 && !Character.*isWhitespace*(lines[j + 1].charAt(0))) {  
 current.append(' ');  
 }  
 current.append(lines[j + 1]);  
 j++;  
 } else {  
 done = true;  
 }  
 }  
 String entry = current.toString();  
 if (entry.length() < 6) {  
 continue;  
 } else {  
 String tag = entry.substring(0, 2);  
 String value = entry.substring(6).trim();  
 if ("TY".equals(tag)) {  
 if ("BOOK".equals(value)) {  
 type = StandardEntryType.*Book*;  
 } else if ("JOUR".equals(value) || "MGZN".equals(value)) {  
 type = StandardEntryType.*Article*;  
 } else if ("THES".equals(value)) {  
 type = StandardEntryType.*PhdThesis*;  
 } else if ("UNPB".equals(value)) {  
 type = StandardEntryType.*Unpublished*;  
 } else if ("RPRT".equals(value)) {  
 type = StandardEntryType.*TechReport*;  
 } else if ("CONF".equals(value)) {  
 type = StandardEntryType.*InProceedings*;  
 } else if ("CHAP".equals(value)) {  
 type = StandardEntryType.*InCollection*;  
 } else if ("PAT".equals(value)) {  
 type = IEEETranEntryType.*Patent*;  
 } else {  
 type = StandardEntryType.*Misc*;  
 }  
 } else if ("T1".equals(tag) || "TI".equals(tag)) {  
 String oldVal = fields.get(StandardField.*TITLE*);  
 if (oldVal == null) {  
 fields.put(StandardField.*TITLE*, value);  
 } else {  
 if (oldVal.endsWith(":") || oldVal.endsWith(".") || oldVal.endsWith("?")) {  
 fields.put(StandardField.*TITLE*, oldVal + " " + value);  
 } else {  
 fields.put(StandardField.*TITLE*, oldVal + ": " + value);  
 }  
 }  
 fields.put(StandardField.*TITLE*, fields.get(StandardField.*TITLE*).replaceAll("\\s+", " ")); // Normalize whitespaces  
 } else if ("BT".equals(tag)) {  
 fields.put(StandardField.*BOOKTITLE*, value);  
 } else if (("T2".equals(tag) || "J2".equals(tag) || "JA".equals(tag)) && ((fields.get(StandardField.*JOURNAL*) == null) || "".equals(fields.get(StandardField.*JOURNAL*)))) {  
 // if there is no journal title, then put second title as journal title  
 fields.put(StandardField.*JOURNAL*, value);  
 } else if ("JO".equals(tag) || "J1".equals(tag) || "JF".equals(tag)) {  
 // if this field appears then this should be the journal title  
 fields.put(StandardField.*JOURNAL*, value);  
 } else if ("T3".equals(tag)) {  
 fields.put(StandardField.*SERIES*, value);  
 } else if ("AU".equals(tag) || "A1".equals(tag) || "A2".equals(tag) || "A3".equals(tag) || "A4".equals(tag)) {  
 if ("".equals(author)) {  
 author = value;  
 } else {  
 author += " and " + value;  
 }  
 } else if ("ED".equals(tag)) {  
 if (editor.isEmpty()) {  
 editor = value;  
 } else {  
 editor += " and " + value;  
 }  
 } else if ("JA".equals(tag) || "JF".equals(tag)) {  
 if (type.equals(StandardEntryType.*InProceedings*)) {  
 fields.put(StandardField.*BOOKTITLE*, value);  
 } else {  
 fields.put(StandardField.*JOURNAL*, value);  
 }  
 } else if ("LA".equals(tag)) {  
 fields.put(StandardField.*LANGUAGE*, value);  
 } else if ("CA".equals(tag)) {  
 fields.put(new UnknownField("caption"), value);  
 } else if ("DB".equals(tag)) {  
 fields.put(new UnknownField("database"), value);  
 } else if ("IS".equals(tag) || "AN".equals(tag) || "C7".equals(tag) || "M1".equals(tag)) {  
 fields.put(StandardField.*NUMBER*, value);  
 } else if ("SP".equals(tag)) {  
 startPage = value;  
 } else if ("PB".equals(tag)) {  
 if (type.equals(StandardEntryType.*PhdThesis*)) {  
 fields.put(StandardField.*SCHOOL*, value);  
 } else {  
 fields.put(StandardField.*PUBLISHER*, value);  
 }  
 } else if ("AD".equals(tag) || "CY".equals(tag) || "PP".equals(tag)) {  
 fields.put(StandardField.*ADDRESS*, value);  
 } else if ("EP".equals(tag)) {  
 endPage = value;  
 if (!endPage.isEmpty()) {  
 endPage = "--" + endPage;  
 }  
 } else if ("ET".equals(tag)) {  
 fields.put(StandardField.*EDITION*, value);  
 } else if ("SN".equals(tag)) {  
 fields.put(StandardField.*ISSN*, value);  
 } else if ("VL".equals(tag)) {  
 fields.put(StandardField.*VOLUME*, value);  
 } else if ("N2".equals(tag) || "AB".equals(tag)) {  
 String oldAb = fields.get(StandardField.*ABSTRACT*);  
 if (oldAb == null) {  
 fields.put(StandardField.*ABSTRACT*, value);  
 } else if (!oldAb.equals(value) && !value.isEmpty()) {  
 fields.put(StandardField.*ABSTRACT*, oldAb + OS.*NEWLINE* + value);  
 }  
 } else if ("UR".equals(tag) || "L2".equals(tag) || "LK".equals(tag)) {  
 fields.put(StandardField.*URL*, value);  
 } else if (((tagPriority = dateTags.indexOf(tag)) != -1) && (value.length() >= 4)) {  
  
 if (tagPriority < datePriority) {  
 String year = value.substring(0, 4);  
  
 try {  
 Year.*parse*(year, *formatter*);  
 // if the year is parsebale we have found a higher priority date  
 dateTag = tag;  
 dateValue = value;  
 datePriority = tagPriority;  
 } catch (DateTimeParseException ex) {  
 // We can't parse the year, we ignore it  
 }  
 }  
 } else if ("KW".equals(tag)) {  
 if (fields.containsKey(StandardField.*KEYWORDS*)) {  
 String kw = fields.get(StandardField.*KEYWORDS*);  
 fields.put(StandardField.*KEYWORDS*, kw + ", " + value);  
 } else {  
 fields.put(StandardField.*KEYWORDS*, value);  
 }  
 } else if ("U1".equals(tag) || "U2".equals(tag) || "N1".equals(tag)) {  
 if (!comment.isEmpty()) {  
 comment = comment + OS.*NEWLINE*;  
 }  
 comment = comment + value;  
 } else if ("M3".equals(tag) || "DO".equals(tag)) {  
 addDoi(fields, value);  
 } else if ("C3".equals(tag)) {  
 fields.put(StandardField.*EVENTTITLE*, value);  
 } else if ("N1".equals(tag) || "RN".equals(tag)) {  
 fields.put(StandardField.*NOTE*, value);  
 } else if ("ST".equals(tag)) {  
 fields.put(StandardField.*SHORTTITLE*, value);  
 } else if ("C2".equals(tag)) {  
 fields.put(StandardField.*EPRINT*, value);  
 fields.put(StandardField.*EPRINTTYPE*, "pubmed");  
 } else if ("TA".equals(tag)) {  
 fields.put(StandardField.*TRANSLATOR*, value);  
  
 // fields for which there is no direct mapping in the bibtext standard  
 } else if ("AV".equals(tag)) {  
 fields.put(new UnknownField("archive\_location"), value);  
 } else if ("CN".equals(tag) || "VO".equals(tag)) {  
 fields.put(new UnknownField("call-number"), value);  
 } else if ("DB".equals(tag)) {  
 fields.put(new UnknownField("archive"), value);  
 } else if ("NV".equals(tag)) {  
 fields.put(new UnknownField("number-of-volumes"), value);  
 } else if ("OP".equals(tag)) {  
 fields.put(new UnknownField("original-title"), value);  
 } else if ("RI".equals(tag)) {  
 fields.put(new UnknownField("reviewed-title"), value);  
 } else if ("RP".equals(tag)) {  
 fields.put(new UnknownField("status"), value);  
 } else if ("SE".equals(tag)) {  
 fields.put(new UnknownField("section"), value);  
 } else if ("ID".equals(tag)) {  
 fields.put(new UnknownField("refid"), value);  
 }  
 }  
 // fix authors  
 if (!author.isEmpty()) {  
 author = AuthorList.*fixAuthorLastNameFirst*(author);  
 fields.put(StandardField.*AUTHOR*, author);  
 }  
 if (!editor.isEmpty()) {  
 editor = AuthorList.*fixAuthorLastNameFirst*(editor);  
 fields.put(StandardField.*EDITOR*, editor);  
 }  
 if (!comment.isEmpty()) {  
 fields.put(StandardField.*COMMENT*, comment);  
 }  
  
 fields.put(StandardField.*PAGES*, startPage + endPage);  
 }  
  
 // if we found a date  
 if (dateTag.length() > 0) {  
 fields.put(StandardField.*YEAR*, dateValue.substring(0, 4));  
  
 String[] parts = dateValue.split("/");  
 if ((parts.length > 1) && !parts[1].isEmpty()) {  
 try {  
 int monthNumber = Integer.*parseInt*(parts[1]);  
 month = Month.*getMonthByNumber*(monthNumber);  
 } catch (NumberFormatException ex) {  
 // The month part is unparseable, so we ignore it.  
 }  
 }  
 }  
  
 // Remove empty fields:  
 fields.entrySet().removeIf(key -> (key.getValue() == null) || key.getValue().trim().isEmpty());  
  
 // create one here  
 // type is set in the loop above  
 BibEntry entry = new BibEntry(type);  
 entry.setField(fields);  
 // month has a special treatment as we use the separate method "setMonth" of BibEntry instead of directly setting the value  
 month.ifPresent(entry::setMonth);  
 bibitems.add(entry);  
 }  
 return new ParserResult(bibitems);  
}

Long method

why» too many parameters (8)

src/main/java/org/jabref/gui/maintable/RightClickMenu.java

should try to have global variables so it's shorter to read and has a smaller chance of messing up



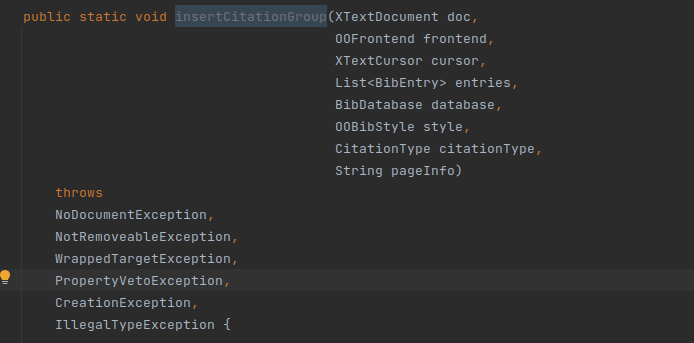
Thiago Monteiro

### Code Smells

src/main/java/org/jabref/logic/openoffice/action/EditInsert.java

line 59

8 parameters



src/main/java/org/jabref/logic/openoffice/action/EditMerge.java

line 156

Long Method

private static boolean checkAddToGroup(ScanState state, CitationGroup group, XTextRange currentRange) {  
  
 if (state.currentGroup.isEmpty()) {  
 return false;  
 }  
  
 Objects.*requireNonNull*(state.currentGroupCursor);  
 Objects.*requireNonNull*(state.cursorBetween);  
 Objects.*requireNonNull*(state.prev);  
 Objects.*requireNonNull*(state.prevRange);  
  
 // Only combine (Author 2000) type citations  
 if (group.citationType != CitationType.*AUTHORYEAR\_PAR*) {  
 return false;  
 }  
  
 if (state.prev != null) {  
  
 // Even if we combine AUTHORYEAR\_INTEXT citations, we would not mix them with AUTHORYEAR\_PAR  
 if (group.citationType != state.prev.citationType) {  
 return false;  
 }  
  
 if (!UnoTextRange.*comparables*(state.prevRange, currentRange)) {  
 return false;  
 }  
  
 // Sanity check: the current range should start later than the previous.  
 int textOrder = UnoTextRange.*compareStarts*(state.prevRange, currentRange);  
 if (textOrder != (-1)) {  
 String msg =  
 String.*format*("MergeCitationGroups:"  
 + " \"%s\" supposed to be followed by \"%s\","  
 + " but %s",  
 state.prevRange.getString(),  
 currentRange.getString(),  
 ((textOrder == 0)  
 ? "they start at the same position"  
 : "the start of the latter precedes the start of the first"));  
 *LOGGER*.warn(msg);  
 return false;  
 }  
 }  
  
 if (state.cursorBetween == null) {  
 return false;  
 }  
  
 Objects.*requireNonNull*(state.cursorBetween);  
 Objects.*requireNonNull*(state.currentGroupCursor);  
  
 // assume: currentGroupCursor.getEnd() == cursorBetween.getEnd()  
 if (UnoTextRange.*compareEnds*(state.cursorBetween, state.currentGroupCursor) != 0) {  
 *LOGGER*.warn("MergeCitationGroups: cursorBetween.end != currentGroupCursor.end");  
 throw new IllegalStateException("MergeCitationGroups failed");  
 }  
  
 /\*  
 \* Try to expand state.currentGroupCursor and state.cursorBetween by going right to reach  
 \* rangeStart.  
 \*/  
 XTextRange rangeStart = currentRange.getStart();  
 boolean couldExpand = true;  
 XTextCursor thisCharCursor =  
 (currentRange.getText().createTextCursorByRange(state.cursorBetween.getEnd()));  
  
 while (couldExpand && (UnoTextRange.*compareEnds*(state.cursorBetween, rangeStart) < 0)) {  
 //  
 // Check that we only walk through inline whitespace.  
 //  
 couldExpand = thisCharCursor.goRight((short) 1, true);  
 String thisChar = thisCharCursor.getString();  
 thisCharCursor.collapseToEnd();  
 if (thisChar.isEmpty() || "\n".equals(thisChar) || !thisChar.trim().isEmpty()) {  
 couldExpand = false;  
 if (!thisChar.isEmpty()) {  
 thisCharCursor.goLeft((short) 1, false);  
 }  
 break;  
 }  
 state.cursorBetween.goRight((short) 1, true);  
 state.currentGroupCursor.goRight((short) 1, true);  
  
 // These two should move in sync:  
 if (UnoTextRange.*compareEnds*(state.cursorBetween, state.currentGroupCursor) != 0) {  
 *LOGGER*.warn("MergeCitationGroups: cursorBetween.end != currentGroupCursor.end (during expand)");  
 throw new IllegalStateException("MergeCitationGroups failed");  
 }  
 }  
  
 return couldExpand;  
}

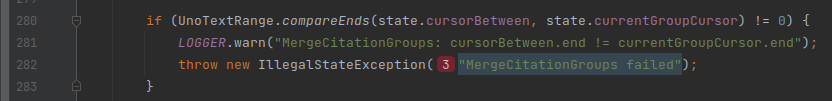
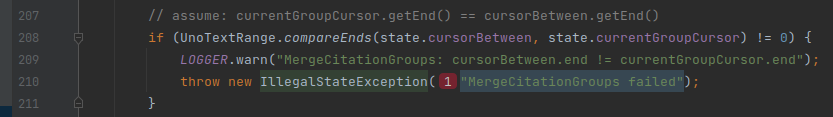
src/main/java/org/jabref/logic/openoffice/action/EditMerge.java

line 210

line 242

line 282

Shotgun Surgery



Tiago Vieira

**CODE SMELLS**

LONG METHOD

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();

// Set-up bindings

Platform.runLater(() ->

BindingsHelper.bindContentBidirectional(

groupTree.getSelectionModel().getSelectedItems(),

viewModel.selectedGroupsProperty(),

(newSelectedGroups) -> newSelectedGroups.forEach(this::selectNode),

this::updateSelection

));

// We try to to prevent publishing changes in the search field directly to the search task that takes some time

// for larger group structures.

final Timer searchTask = FxTimer.create(Duration.ofMillis(400), () -> {

LOGGER.debug("Run group search " + searchField.getText());

viewModel.filterTextProperty().setValue(searchField.textProperty().getValue());

});

searchField.textProperty().addListener((observable, oldValue, newValue) -> searchTask.restart());

setNewGroupButtonStyle(groupTree);

groupTree.rootProperty().bind(

EasyBind.map(viewModel.rootGroupProperty(),

group -> {

if (group == null) {

return null;

} else {

return new RecursiveTreeItem<>(

group,

GroupNodeViewModel::getChildren,

GroupNodeViewModel::expandedProperty,

viewModel.filterPredicateProperty());

}

}));

// Icon and group name

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withText(GroupNodeViewModel::getDisplayName)

.withIcon(GroupNodeViewModel::getIcon)

.withTooltip(GroupNodeViewModel::getDescription)

.install(mainColumn);

// Number of hits (only if user wants to see them)

PseudoClass anySelected = PseudoClass.getPseudoClass("any-selected");

PseudoClass allSelected = PseudoClass.getPseudoClass("all-selected");

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withGraphic(group -> {

final StackPane node = new StackPane();

node.getStyleClass().setAll("hits");

if (!group.isRoot()) {

BindingsHelper.includePseudoClassWhen(node, anySelected,

group.anySelectedEntriesMatchedProperty());

BindingsHelper.includePseudoClassWhen(node, allSelected,

group.allSelectedEntriesMatchedProperty());

}

Text text = new Text();

if (preferencesService.getDisplayGroupCount()) {

text.textProperty().bind(group.getHits().asString());

}

text.getStyleClass().setAll("text");

node.getChildren().add(text);

node.setMaxWidth(Control.USE\_PREF\_SIZE);

return node;

})

.install(numberColumn);

// Arrow indicating expanded status

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withGraphic(viewModel -> {

final StackPane disclosureNode = new StackPane();

disclosureNode.visibleProperty().bind(viewModel.hasChildrenProperty());

disclosureNode.getStyleClass().setAll("tree-disclosure-node");

final StackPane disclosureNodeArrow = new StackPane();

disclosureNodeArrow.getStyleClass().setAll("arrow");

disclosureNode.getChildren().add(disclosureNodeArrow);

return disclosureNode;

})

.withOnMouseClickedEvent(group -> event -> {

group.toggleExpansion();

event.consume();

})

.install(expansionNodeColumn);

// Set pseudo-classes to indicate if row is root or sub-item ( > 1 deep)

PseudoClass rootPseudoClass = PseudoClass.getPseudoClass("root");

PseudoClass subElementPseudoClass = PseudoClass.getPseudoClass("sub");

groupTree.setRowFactory(treeTable -> {

TreeTableRow<GroupNodeViewModel> row = new TreeTableRow<>();

row.treeItemProperty().addListener((ov, oldTreeItem, newTreeItem) -> {

setNewGroupButtonStyle(treeTable);

boolean isRoot = newTreeItem == treeTable.getRoot();

row.pseudoClassStateChanged(rootPseudoClass, isRoot);

boolean isFirstLevel = (newTreeItem != null) && (newTreeItem.getParent() == treeTable.getRoot());

row.pseudoClassStateChanged(subElementPseudoClass, !isRoot && !isFirstLevel);

});

// Remove disclosure node since we display custom version in separate column

// Simply setting to null is not enough since it would be replaced by the default node on every change

row.setDisclosureNode(null);

row.disclosureNodeProperty().addListener((observable, oldValue, newValue) -> row.setDisclosureNode(null));

// Add context menu (only for non-null items)

row.contextMenuProperty().bind(

EasyBind.wrapNullable(row.itemProperty())

.map(this::createContextMenuForGroup)

.orElse((ContextMenu) null));

row.addEventFilter(MouseEvent.MOUSE\_PRESSED, event -> {

if (event.getButton() == MouseButton.SECONDARY) {

// Prevent right-click to select group

event.consume();

}

});

// Drag and drop support

row.setOnDragDetected(event -> {

List<String> groupsToMove = new ArrayList<>();

for (TreeItem<GroupNodeViewModel> selectedItem : treeTable.getSelectionModel().getSelectedItems()) {

if ((selectedItem != null) && (selectedItem.getValue() != null)) {

groupsToMove.add(selectedItem.getValue().getPath());

}

}

if (groupsToMove.size() > 0) {

localDragboard.clearAll();

}

// Put the group nodes as content

Dragboard dragboard = treeTable.startDragAndDrop(TransferMode.MOVE);

// Display the group when dragging

dragboard.setDragView(row.snapshot(null, null));

ClipboardContent content = new ClipboardContent();

content.put(DragAndDropDataFormats.GROUP, groupsToMove);

dragboard.setContent(content);

event.consume();

});

row.setOnDragOver(event -> {

Dragboard dragboard = event.getDragboard();

if ((event.getGestureSource() != row) && (row.getItem() != null) && row.getItem().acceptableDrop(dragboard)) {

event.acceptTransferModes(TransferMode.MOVE, TransferMode.LINK);

// expand node and all children on drag over

dragExpansionHandler.expandGroup(row.getTreeItem());

if (localDragboard.hasBibEntries()) {

ControlHelper.setDroppingPseudoClasses(row);

} else {

ControlHelper.setDroppingPseudoClasses(row, event);

}

}

event.consume();

});

row.setOnDragExited(event -> {

ControlHelper.removeDroppingPseudoClasses(row);

});

row.setOnDragDropped(event -> {

Dragboard dragboard = event.getDragboard();

boolean success = false;

if (dragboard.hasContent(DragAndDropDataFormats.GROUP)) {

List<String> pathToSources = (List<String>) dragboard.getContent(DragAndDropDataFormats.GROUP);

List<GroupNodeViewModel> changedGroups = new LinkedList<>();

for (String pathToSource : pathToSources) {

Optional<GroupNodeViewModel> source = viewModel

.rootGroupProperty().get()

.getChildByPath(pathToSource);

if (source.isPresent()) {

source.get().draggedOn(row.getItem(), ControlHelper.getDroppingMouseLocation(row, event));

changedGroups.add(source.get());

success = true;

}

}

groupTree.getSelectionModel().clearSelection();

changedGroups.forEach(value -> selectNode(value, true));

}

if (localDragboard.hasBibEntries()) {

List<BibEntry> entries = localDragboard.getBibEntries();

row.getItem().addEntriesToGroup(entries);

success = true;

}

event.setDropCompleted(success);

event.consume();

});

return row;

});

// Filter text field

setupClearButtonField(searchField);

}

LOCATION - src/main/java/org/jabref/gui/groups/GroupTreeView.java

WHY

Almost 200 lines of code.

REFACTORING PROPOSAL

It should have auxiliary methods to improve readability and reduce its complexity.

COMMENTS

Texto

Descrição gerada automaticamente

LOCATION - src/main/java/org/jabref/gui/citationkeypattern/CitationKeyPatternPanel.java

WHY

Commented out old code that is now irrelevant making the method clogged up with too many comments and hard to read.

REFACTORING PROPOSAL

Deleting these comments or making just a small reference.

LARGE CLASS

Texto

Descrição gerada automaticamente

LOCATION - src/main/java/org/jabref/preferences/JabRefPreferences.java

WHY

Large class with almost 3000 lines of code.

REFACTORING PROPOSAL

It should be divided into several smaller classes. Grouping types of preferences.

Yuliia Bila

**Code smells**

1. Long method

\main\java\org\jabref\logic\importer\AuthorListParser.java

private Optional<Author> getAuthor() {  
 List<Object> tokens = new ArrayList<>(); // initialization  
 int vonStart = -1;  
 int lastStart = -1;  
 int commaFirst = -1;  
 int commaSecond = -1;  
  
 // First step: collect tokens in 'tokens' Vector and calculate indices  
 boolean continueLoop = true;  
 while (continueLoop) {  
 Token token = getToken();  
 switch (token) {  
 case *EOF*:  
 case AND:  
 continueLoop = false;  
 break;  
 case *COMMA*:  
 if (commaFirst < 0) {  
 commaFirst = tokens.size();  
 } else if (commaSecond < 0) {  
 commaSecond = tokens.size();  
 }  
 break;  
 case *WORD*:  
 tokens.add(original.substring(tokenStart, tokenEnd));  
 tokens.add(original.substring(tokenStart, tokenAbbrEnd));  
 tokens.add(tokenTerm);  
 tokens.add(tokenCase);  
 if (commaFirst >= 0) {  
 break;  
 }  
 if (lastStart >= 0) {  
 break;  
 }  
 if (vonStart < 0) {  
 if (!tokenCase) {  
 int previousTermToken = (tokens.size() - *TOKEN\_GROUP\_LENGTH* - *TOKEN\_GROUP\_LENGTH*) + *OFFSET\_TOKEN\_TERM*;  
 if ((previousTermToken >= 0) && tokens.get(previousTermToken).equals('-')) {  
 // We are in a first name which contained a hyphen  
 break;  
 }  
  
 int thisTermToken = previousTermToken + *TOKEN\_GROUP\_LENGTH*;  
 if ((thisTermToken >= 0) && tokens.get(thisTermToken).equals('-')) {  
 // We are in a name which contained a hyphen  
 break;  
 }  
  
 vonStart = tokens.size() - *TOKEN\_GROUP\_LENGTH*;  
 break;  
 }  
 } else if (tokenCase) {  
 lastStart = tokens.size() - *TOKEN\_GROUP\_LENGTH*;  
 break;  
 }  
 break;  
 default:  
 break;  
 }  
 }  
  
 // Second step: split name into parts (here: calculate indices  
 // of parts in 'tokens' Vector)  
 if (tokens.isEmpty()) {  
 return Optional.*empty*(); // no author information  
 }  
  
 // the following negatives indicate absence of the corresponding part  
 int firstPartStart = -1;  
 int vonPartStart = -1;  
 int lastPartStart = -1;  
 int jrPartStart = -1;  
 int firstPartEnd;  
 int vonPartEnd = 0;  
 int lastPartEnd = 0;  
 int jrPartEnd = 0;  
 if (commaFirst < 0) { // no commas  
 if (vonStart < 0) { // no 'von part'  
 lastPartEnd = tokens.size();  
 lastPartStart = tokens.size() - *TOKEN\_GROUP\_LENGTH*;  
 int index = (tokens.size() - (2 \* *TOKEN\_GROUP\_LENGTH*)) + *OFFSET\_TOKEN\_TERM*;  
 if (index > 0) {  
 Character ch = (Character) tokens.get(index);  
 if (ch == '-') {  
 lastPartStart -= *TOKEN\_GROUP\_LENGTH*;  
 }  
 }  
 firstPartEnd = lastPartStart;  
 if (firstPartEnd > 0) {  
 firstPartStart = 0;  
 }  
 } else { // 'von part' is present  
 if (lastStart >= 0) {  
 lastPartEnd = tokens.size();  
 lastPartStart = lastStart;  
 vonPartEnd = lastPartStart;  
 } else {  
 vonPartEnd = tokens.size();  
 }  
 vonPartStart = vonStart;  
 firstPartEnd = vonPartStart;  
 if (firstPartEnd > 0) {  
 firstPartStart = 0;  
 }  
 }  
 } else {  
 // commas are present: it affects only 'first part' and 'junior part'  
 firstPartEnd = tokens.size();  
 if (commaSecond < 0) {  
 // one comma  
 if (commaFirst < firstPartEnd) {  
 firstPartStart = commaFirst;  
 }  
 } else {  
 // two or more commas  
 if (commaSecond < firstPartEnd) {  
 firstPartStart = commaSecond;  
 }  
 jrPartEnd = commaSecond;  
 if (commaFirst < jrPartEnd) {  
 jrPartStart = commaFirst;  
 }  
 }  
 if (vonStart == 0) { // 'von part' is present  
 if (lastStart < 0) {  
 vonPartEnd = commaFirst;  
 } else {  
 lastPartEnd = commaFirst;  
 lastPartStart = lastStart;  
 vonPartEnd = lastPartStart;  
 }  
 vonPartStart = 0;  
 } else { // no 'von part'  
 lastPartEnd = commaFirst;  
 if (lastPartEnd > 0) {  
 lastPartStart = 0;  
 }  
 }  
 }  
  
 if ((firstPartStart == -1) && (lastPartStart == -1) && (vonPartStart != -1)) {  
 // There is no first or last name, but we have a von part. This is likely  
 // to indicate a single-entry name without an initial capital letter, such  
 // as "unknown".  
 // We make the von part the last name, to facilitate handling by last-name formatters:  
 lastPartStart = vonPartStart;  
 lastPartEnd = vonPartEnd;  
 vonPartStart = -1;  
 vonPartEnd = -1;  
 }  
  
 // Third step: do actual splitting, construct Author object  
 String firstPart = firstPartStart < 0 ? null : concatTokens(tokens, firstPartStart, firstPartEnd, *OFFSET\_TOKEN*, false);  
 String firstAbbr = firstPartStart < 0 ? null : concatTokens(tokens, firstPartStart, firstPartEnd, *OFFSET\_TOKEN\_ABBR*, true);  
 String vonPart = vonPartStart < 0 ? null : concatTokens(tokens, vonPartStart, vonPartEnd, *OFFSET\_TOKEN*, false);  
 String lastPart = lastPartStart < 0 ? null : concatTokens(tokens, lastPartStart, lastPartEnd, *OFFSET\_TOKEN*, false);  
 String jrPart = jrPartStart < 0 ? null : concatTokens(tokens, jrPartStart, jrPartEnd, *OFFSET\_TOKEN*, false);  
  
 if ((firstPart != null) && (lastPart != null) && lastPart.equals(lastPart.toUpperCase(Locale.*ROOT*)) && (lastPart.length() < 5)  
 && (Character.UnicodeScript.*of*(lastPart.charAt(0)) != Character.UnicodeScript.*HAN*)) {  
 // The last part is a small string in complete upper case, so interpret it as initial of the first name  
 // This is the case for example in "Smith SH" which we think of as lastname=Smith and firstname=SH  
 // The length < 5 constraint should allow for "Smith S.H." as input  
 return Optional.*of*(new Author(lastPart, lastPart, vonPart, firstPart, jrPart));  
 } else {  
 return Optional.*of*(new Author(firstPart, firstAbbr, vonPart, lastPart, jrPart));  
 }  
}

1. Long Parameter List

src\main\java\org\jabref\logic\util\UpdateField.java

public static void setAutomaticFields(BibEntry entry, boolean overwriteOwner, boolean overwriteTimestamp,  
OwnerPreferences ownerPreferences, TimestampPreferences timestampPreferences) {  
 String defaultOwner = ownerPreferences.getDefaultOwner();  
 String timestamp = timestampPreferences.now();  
 boolean setOwner = ownerPreferences.isUseOwner() && (overwriteOwner || (!entry.hasField(StandardField.*OWNER*)));  
 boolean setTimeStamp = timestampPreferences.shouldAddCreationDate();  
  
 *setAutomaticFields*(entry, setOwner, defaultOwner, setTimeStamp, timestamp);  
}

1. Large class

src\main\java\org\jabref\preferences\JabRefPreferences.java

This class contain 2793 lines

1. public class JabRefPreferences implements PreferencesService {  
     
    // Push to application preferences  
    public static final String *PUSH\_EMACS\_PATH* = "emacsPath";  
    public static final String *PUSH\_EMACS\_ADDITIONAL\_PARAMETERS* = "emacsParameters";  
    public static final String *PUSH\_LYXPIPE* = "lyxpipe";  
    public static final String *PUSH\_TEXSTUDIO\_PATH* = "TeXstudioPath";  
    public static final String *PUSH\_WINEDT\_PATH* = "winEdtPath";  
    public static final String *PUSH\_TEXMAKER\_PATH* = "texmakerPath";  
    public static final String *PUSH\_VIM\_SERVER* = "vimServer";  
    public static final String *PUSH\_VIM* = "vim";