code smell

1. Long method. This method can be divided into several parts (sub-methods) to improve readability.

\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. This class is too large for further work in it. This class should be divided into subclasses.

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";