CODE SMELLS

The 3 code smells I identified are the following:

Long Method

```java
public static void openExternalViewer(BibDatabaseContext databaseContext,
                                      PreferencesService preferencesService,
                                      String initialLink,
                                      Field initialField)
        throws IOException {
    String link = initialLink;
    Field field = initialField;
    if (StandardField.PS.equals(field) || StandardField.PDF.equals(field)) {
        // Find the default directory for this field type:
        List<Path> directories = databaseContext.getFileDirectories(preferencesService.getFilePreferences());

        Optional<Path> file = FileHelper.find(link, directories);

        // Check that the file exists:
        if (file.isEmpty() || !Files.exists(file.get())) {
            throw new IOException("File not found (" + field + "): '" + link + "'.");
        }
        link = file.get().toAbsolutePath().toString();

        // Use the correct viewer even if pdf and ps are mixed up:
        String[] split = file.get().getFileName().toString().split( regex: "\\.");
        if (split.length >= 2) {
            if ("pdf".equalsIgnoreCase(split[split.length - 1])) {
                field = StandardField.PDF;
            } else if ("ps".equalsIgnoreCase(split[split.length - 1])
                    || ((split.length >= 3) && "ps".equalsIgnoreCase(split[split.length - 2]))) {
                field = StandardField.PS;
            }
        }
    } else if (StandardField.DOI.equals(field)) {
        openDoi(link);
```

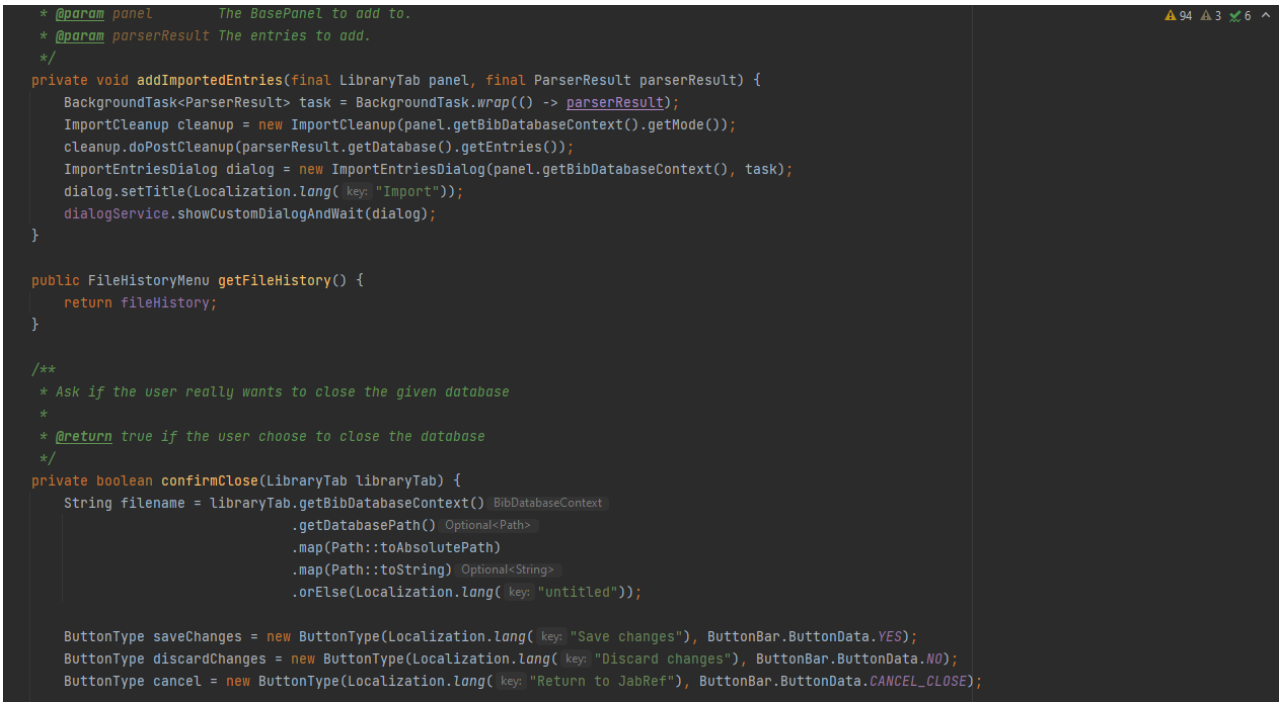
```java
            openDoi(link);
            return;
        } else if (StandardField.EPRINT.equals(field)) {
            link = ArXivIdentifier.parse(link) Optional<ArXivIdentifier>
                                  .map(ArXivIdentifier::getExternalURI) Optional<Optional<URI>>
                                  .filter(Optional::isPresent)
                                  .map(Optional::get) Optional<URI>
                                  .map(URI::toASCIIString) Optional<String>
                                  .orElse(link);
            // should be opened in browser
            field = StandardField.URL;
        }

        if (StandardField.URL.equals(field)) {
            openBrowser(link);
        } else if (StandardField.PS.equals(field)) {
            try {
                NATIVE_DESKTOP.openFile(link, StandardField.PS.getName());
            } catch (IOException e) {
                LOGGER.error("An error occurred on the command: " + link, e);
            }
        } else if (StandardField.PDF.equals(field)) {
            try {
                NATIVE_DESKTOP.openFile(link, StandardField.PDF.getName());
            } catch (IOException e) {
                LOGGER.error("An error occurred on the command: " + link, e);
            }
        } else {
            LOGGER.info("Message: currently only PDF, PS and HTML files can be opened by double clicking");
        }
    }
```

This code smell can be found in the following path: jabref -> src -> main -> java -> org.jabref -> gui -> desktop -> JabRefDesktop.java (lines 54 to 114)

The first code smell identified is of type "Long Method". As the name suggests, this code smell is related to methods that are too long and too complex, like we see in this case.

Refactoring suggestion: Given the code snippet, one way to refactor this method is by separating it into smaller methods, each one with a certain functionality. For example, we can create a separate method to find the default directory for some field type or using the correct viewer (based on the comments inside of this method).

Large Class



```
 * @param panel        The BasePanel to add to.
 * @param parserResult The entries to add.
 */
private void addImportedEntries(final LibraryTab panel, final ParserResult parserResult) {
    BackgroundTask<ParserResult> task = BackgroundTask.wrap(() -> parserResult);
    ImportCleanup cleanup = new ImportCleanup(panel.getBibDatabaseContext().getMode());
    cleanup.doPostCleanup(parserResult.getDatabase().getEntries());
    ImportEntriesDialog dialog = new ImportEntriesDialog(panel.getBibDatabaseContext(), task);
    dialog.setTitle(Localization.lang( key: "Import"));
    dialogService.showCustomDialogAndWait(dialog);
}

public FileHistoryMenu getFileHistory() {
    return fileHistory;
}

/**
 * Ask if the user really wants to close the given database
 *
 * @return true if the user choose to close the database
 */
private boolean confirmClose(LibraryTab libraryTab) {
    String filename = libraryTab.getBibDatabaseContext() BibDatabaseContext
                                .getDatabasePath() Optional<Path>
                                .map(Path::toAbsolutePath)
                                .map(Path::toString) Optional<String>
                                .orElse(Localization.lang( key: "untitled"));

    ButtonType saveChanges = new ButtonType(Localization.lang( key: "Save changes"), ButtonBar.ButtonData.YES);
    ButtonType discardChanges = new ButtonType(Localization.lang( key: "Discard changes"), ButtonBar.ButtonData.NO);
    ButtonType cancel = new ButtonType(Localization.lang( key: "Return to JabRef"), ButtonBar.ButtonData.CANCEL_CLOSE);
```

This code smell can be found in the following path: jabref -> src -> main -> java -> org.jabref -> gui -> JabRefFrame.java

This case is an example of a “Large Class”, which is also a code smell. Through this snippet alone, we can see three completely unrelated methods already, which can give us a glimpse of how diversified this class’ functionalities are. Knowing that JabRefFrame.java has about 1300 lines of code, we can conclude that this class alone has a great responsibility to the entire system, and that’s a big problem, since it can serve as a black hole, meaning that this amount of responsibilities can attract more and more responsibilities in the future.

Refactoring suggestion: Although it can become very time-consuming to fix large classes due to the numerous links they have with other classes, one way to solve this problem is by separating them into different classes, with different functionalities. For example, using the presented code snippet, if there is more methods related to imports just like

addImportedEntries(), we could create a class only dedicated to import functionalities. We could use a similar strategy for other functionalities.

Long Parameter List

```java
/**
 * @param cursor Where to insert.
 * @param pageInfo A single pageInfo for a list of entries. This is what we get from the GUI.
 */
public static void insertCitationGroup(XTextDocument doc,
                                       OOFrontend frontend,
                                       XTextCursor cursor,
                                       List<BibEntry> entries,
                                       BibDatabase database,
                                       OOBibStyle style,
                                       CitationType citationType,
                                       String pageInfo)
    throws
    NoDocumentException,
    NotRemoveableException,
    WrappedTargetException,
    PropertyVetoException,
    CreationException,
    IllegalTypeException {

    List<String> citationKeys = OOListUtil.map(entries, EditInsert::insertEntryGetCitationKey);

    final int totalEntries = entries.size();
    List<Optional<OOText>> pageInfos = OODataModel.fakePageInfos(pageInfo, totalEntries);

    List<CitationMarkerEntry> citations = new ArrayList<>(totalEntries);
    for (int i = 0; i < totalEntries; i++) {
        Citation cit = new Citation(citationKeys.get(i));
        cit.lookupInDatabases(Collections.singletonList(database));
        cit.setPageInfo(pageInfos.get(i));
        citations.add(cit);
    }
```

This code smell can be found in the following path: jabref -> src -> main -> java -> org.jabref -> logic -> openoffice -> action -> EditInsert.java (lines 59 to 66)

In this code snippet, we can see that the insertCitationGroup() method has 8 parameters, which is considered an excessive amount of parameters for just one method. Therefore we can conclude that this illustrates the "Long parameter list" code smell.

Refactoring suggestion: One way to reduce the amount of parameters is by creating parameter objects. For example, we could associate "style", "citationType" and "frontend" into just one object and passing it as a parameter.