Code Smells

First code smell: long method, ‘masked’ by too many comments

```
private static List<BibEntryDiff> compareEntries(List<BibEntry>
originalEntries, List<BibEntry> newEntries) {
    List<BibEntryDiff> differences = new ArrayList<>();

    // Create pointers that are incremented as the entries of each base are
used in
    // successive order from the beginning. Entries "further down" in the
new database
    // can also be matched.
    int positionNew = 0;

    // Create a HashSet where we can put references to entries in the new
    // database that we have matched. This is to avoid matching them twice.
    Set<Integer> used = new HashSet<>(newEntries.size());
    Set<BibEntry> notMatched = new HashSet<>(originalEntries.size());

    // Loop through the entries of the original database, looking for exact
matches in the new one.
    // We must finish scanning for exact matches before looking for near
matches, to avoid an exact
    // match being "stolen" from another entry.
    mainLoop:
    for (BibEntry originalEntry : originalEntries) {
        // First check if the similarly placed entry in the other base
matches exactly.
        if (!used.contains(positionNew) && (positionNew <
newEntries.size())) {
            double score =
DuplicateCheck.compareEntriesStrictly(originalEntry,
newEntries.get(positionNew));
            if (score > 1) {
                used.add(positionNew);
                positionNew++;
                continue;
            }
        }
        // No? Then check if another entry matches exactly.
        for (int i = positionNew + 1; i < newEntries.size(); i++) {
            if (!used.contains(i)) {
                double score =
DuplicateCheck.compareEntriesStrictly(originalEntry, newEntries.get(i));
                if (score > 1) {
                    used.add(i);
                    continue mainLoop;
                }
            }
        }

        // No? Add this entry to the list of non-matched entries.
        notMatched.add(originalEntry);
    }

    // Now we've found all exact matches, look through the remaining
entries, looking for close matches.
    for (Iterator<BibEntry> iteratorNotMatched = notMatched.iterator();
iteratorNotMatched.hasNext(); ) {
        BibEntry originalEntry = iteratorNotMatched.next();

        // These two variables will keep track of which entry most closely
matches the one we're looking at.
        double bestMatch = 0;
        int bestMatchIndex = -1;
        if (positionNew < (newEntries.size() - 1)) {
            for (int i = positionNew; i < newEntries.size(); i++) {
                if (!used.contains(i)) {
                    double score =
DuplicateCheck.compareEntriesStrictly(originalEntry, newEntries.get(i));
                    if (score > bestMatch) {
                        bestMatch = score;
                        bestMatchIndex = i;
                    }
                }
            }
        }

        if (bestMatch > MATCH_THRESHOLD) {
            used.add(bestMatchIndex);
            iteratorNotMatched.remove();

            differences.add(new BibEntryDiff(originalEntry,
newEntries.get(bestMatchIndex)));
        } else {
            differences.add(new BibEntryDiff(originalEntry, null));
        }
    }

    // Finally, look if there are still untouched entries in the new
database. These may have been added.
    for (int i = 0; i < newEntries.size(); i++) {
        if (!used.contains(i)) {
            differences.add(new BibEntryDiff(null, newEntries.get(i)));
        }
    }

    return differences;
}
```

it can be found in: **jabref > logic > bibtex > comparator > BibDatabaseDiff.java**, lines 43-122

This method is extensive and should be divided in smaller parts for easier comprehension instead of explained through a large quantity of comments.

Second code smell: Comments take a 'reminder' nature,

```
public int compare(BibEntry e1, BibEntry e2) {
    // default equals
    // TODO: with the new default equals this does not only return 0 for
identical objects,
    //       but for all objects that have the same id and same fields
    if (Objects.equals(e1, e2)) {
        return 0;
```

```java
    }

    Object f1 = e1.getField(sortField).orElse(null);
    Object f2 = e2.getField(sortField).orElse(null);

    if (binary) {
        // We just separate on set and unset fields:
        if (f1 == null) {
            return f2 == null ? (next == null ? idCompare(e1, e2) :
next.compare(e1, e2)) : 1;
        } else {
            return f2 == null ? -1 : (next == null ? idCompare(e1, e2) :
next.compare(e1, e2));
        }
    }

    // If the field is author or editor, we rearrange names so they are
    // sorted according to last name.
    if (sortField.getProperties().contains(FieldProperty.PERSON_NAMES)) {
        if (f1 != null) {
            f1 = AuthorList.fixAuthorForAlphabetization((String)
f1).toLowerCase(Locale.ROOT);
        }
        if (f2 != null) {
            f2 = AuthorList.fixAuthorForAlphabetization((String)
f2).toLowerCase(Locale.ROOT);
        }
    } else if (sortField.equals(InternalField.TYPE_HEADER)) {
        // Sort by type.
        f1 = e1.getType();
        f2 = e2.getType();
    } else if (sortField.equals(InternalField.KEY_FIELD)) {
        f1 = e1.getCitationKey().orElse(null);
        f2 = e2.getCitationKey().orElse(null);
    } else if (sortField.isNumeric()) {
        try {
            Integer i1 = Integer.parseInt((String) f1);
            Integer i2 = Integer.parseInt((String) f2);
            // Ok, parsing was successful. Update f1 and f2:
            f1 = i1;
            f2 = i2;
        } catch (NumberFormatException ex) {
            // Parsing failed. Give up treating these as numbers.
            // TODO: should we check which of them failed, and sort based
on that?
        }
    }

    if (f2 == null) {
        if (f1 == null) {
            return next == null ? idCompare(e1, e2) : next.compare(e1, e2);
        } else {
            return -1;
        }
    }

    if (f1 == null) { // f2 != null here automatically
        return 1;
    }

    int result;
    if ((f1 instanceof Integer) && (f2 instanceof Integer)) {
        result = ((Integer) f1).compareTo((Integer) f2);
    } else if (f2 instanceof Integer) {
        Integer f1AsInteger = Integer.valueOf(f1.toString());
        result = f1AsInteger.compareTo((Integer) f2);
    } else if (f1 instanceof Integer) {
        Integer f2AsInteger = Integer.valueOf(f2.toString());
        result = ((Integer) f1).compareTo(f2AsInteger);
    } else {
        String ours = ((String) f1).toLowerCase(Locale.ROOT);
        String theirs = ((String) f2).toLowerCase(Locale.ROOT);
        int comp = ours.compareTo(theirs);
        result = comp;
    }
    if (result != 0) {
        return descending ? -result : result; // Primary sort.
    }
    if (next == null) {
        return idCompare(e1, e2); // If still equal, we use the unique IDs.
    } else {
        return next.compare(e1, e2); // Secondary sort if existent.
    }
}
```

it can be found in: **jabref > logic > bibtex > comparator > EntryComparator.java**, lines 51-52 and 93-94

These comments refer to things that are yet to be done. Having this method call other methods which took care of these parts would allow for easier understanding and implementation of such parts.

Third code smell: Feature envy,

```java
/**
 * @implNote Should be kept in sync with {@link MetaData#equals(Object)}
 */
public List<String> getDifferences(PreferencesService preferences) {
    List<String> changes = new ArrayList<>();

    if (originalMetaData.isProtected() != newMetaData.isProtected()) {
        changes.add(Localization.lang("Library protection"));
    }
    if (!Objects.equals(originalMetaData.getGroups(),
newMetaData.getGroups())) {
        changes.add(Localization.lang("Modified groups tree"));
    }
    if (!Objects.equals(originalMetaData.getEncoding(),
newMetaData.getEncoding())) {
        changes.add(Localization.lang("Library encoding"));
    }
    if (!Objects.equals(originalMetaData.getSaveOrderConfig(),
newMetaData.getSaveOrderConfig())) {
        changes.add(Localization.lang("Save sort order"));
    }
    if
(!Objects.equals(originalMetaData.getCiteKeyPattern(preferences.getGlobalCi
tationKeyPattern()),
newMetaData.getCiteKeyPattern(preferences.getGlobalCitationKeyPattern())))
{
        changes.add(Localization.lang("Key patterns"));
    }
    if (!Objects.equals(originalMetaData.getUserFileDirectories(),
newMetaData.getUserFileDirectories())) {
        changes.add(Localization.lang("User-specific file directory"));
    }
    if (!Objects.equals(originalMetaData.getLatexFileDirectories(),
newMetaData.getLatexFileDirectories())) {
        changes.add(Localization.lang("LaTeX file directory"));
    }
    if (!Objects.equals(originalMetaData.getDefaultCiteKeyPattern(),
newMetaData.getDefaultCiteKeyPattern())) {
        changes.add(Localization.lang("Default pattern"));
    }
    if (!Objects.equals(originalMetaData.getSaveActions(),
newMetaData.getSaveActions())) {
        changes.add(Localization.lang("Save actions"));
    }
    if (originalMetaData.getMode() != newMetaData.getMode()) {
        changes.add(Localization.lang("Library mode"));
    }
    if (!Objects.equals(originalMetaData.getDefaultFileDirectory(),
newMetaData.getDefaultFileDirectory())) {
        changes.add(Localization.lang("General file directory"));
    }
    if (!Objects.equals(originalMetaData.getContentSelectors(),
newMetaData.getContentSelectors())) {
        changes.add(Localization.lang("Content selectors"));
    }
    return changes;
}
```

it can be found in: **jabref > logic > bibtex > comparator > MetaDataDiff.java**, lines 32-75

This method having to be kept 'in sync' manually with **jabref > model > metadata > MetaData.java** class shows an inappropriate use of classes, having MetaDataDiff more interested in MetaData than it is on itself. This class is aimed at identifying and recording the changes made to MetaData and should therefore be part of, an extension or extending the class MetaData.