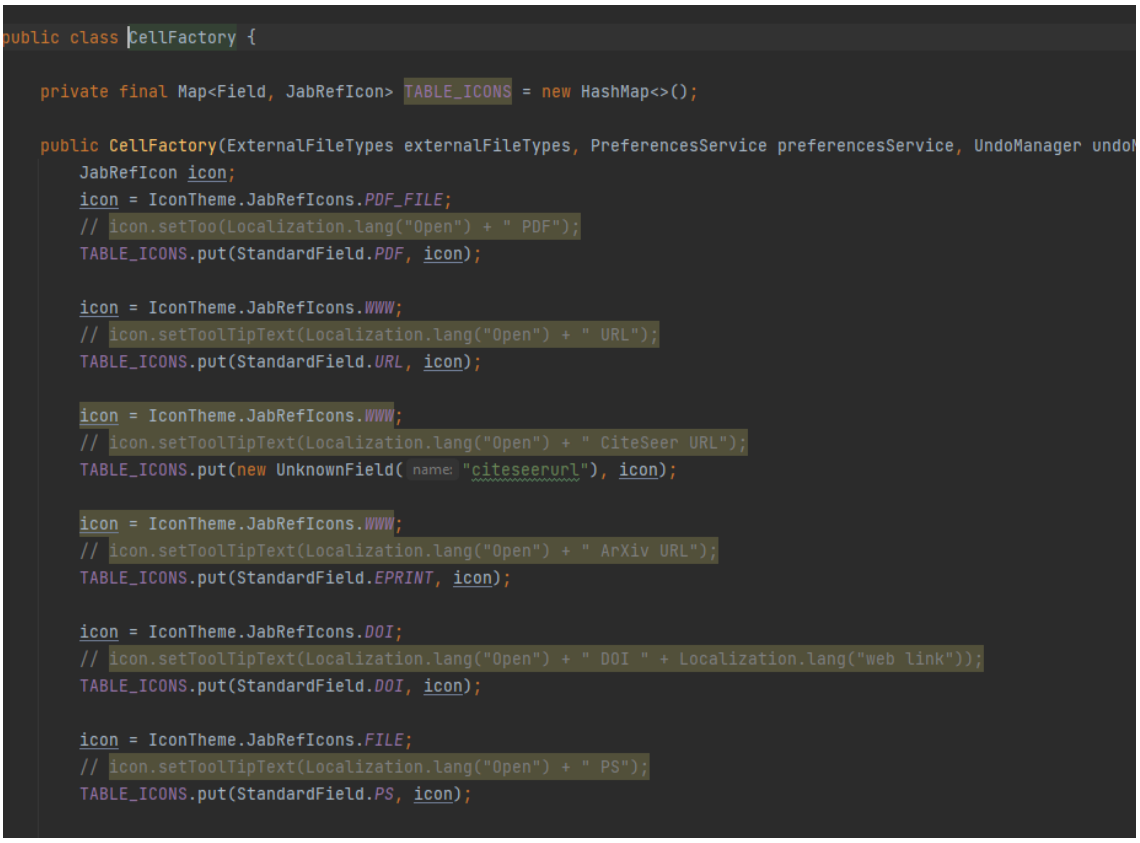
DESIGN PATTERNS

The 3 Design Patterns I identified were the following:

FACTORY METHOD PATTERN

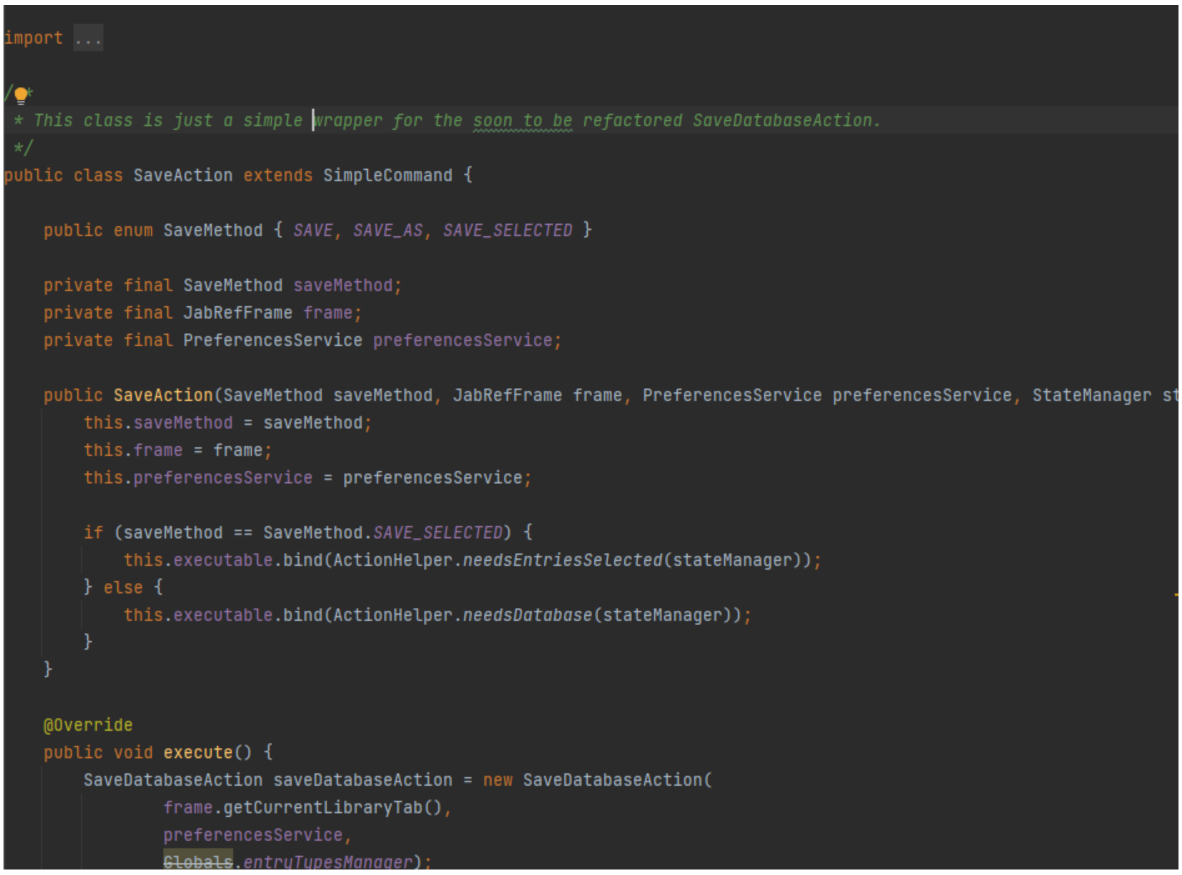


```
public class CellFactory {

    private final Map<Field, JabRefIcon> TABLE_ICONS = new HashMap<>();

    public CellFactory(ExternalFileTypes externalFileTypes, PreferencesService preferencesService, UndoManager undoM
        JabRefIcon icon;
        icon = IconTheme.JabRefIcons.PDF_FILE;
        // icon.setToo(Localization.lang("Open") + " PDF");
        TABLE_ICONS.put(StandardField.PDF, icon);

        icon = IconTheme.JabRefIcons.WWW;
        // icon.setToolTipText(Localization.lang("Open") + " URL");
        TABLE_ICONS.put(StandardField.URL, icon);

        icon = IconTheme.JabRefIcons.WWW;
        // icon.setToolTipText(Localization.lang("Open") + " CiteSeer URL");
        TABLE_ICONS.put(new UnknownField( name: "citeseerurl"), icon);

        icon = IconTheme.JabRefIcons.WWW;
        // icon.setToolTipText(Localization.lang("Open") + " ArXiv URL");
        TABLE_ICONS.put(StandardField.EPRINT, icon);

        icon = IconTheme.JabRefIcons.DOI;
        // icon.setToolTipText(Localization.lang("Open") + " DOI " + Localization.lang("web link"));
        TABLE_ICONS.put(StandardField.DOI, icon);

        icon = IconTheme.JabRefIcons.FILE;
        // icon.setToolTipText(Localization.lang("Open") + " PS");
        TABLE_ICONS.put(StandardField.PS, icon);
```

This “Factory Method” pattern can be found in jabref -> src -> main ->java -> org.jabref -> gui -> maintainable -> CellFactory.java

We can say that CellFactory.java uses a “Factory Method” pattern since it was designed for hiding the whole process of creating certain types of instances, such as icons, external file types or special fields, for example.

FAÇADE PATTERN

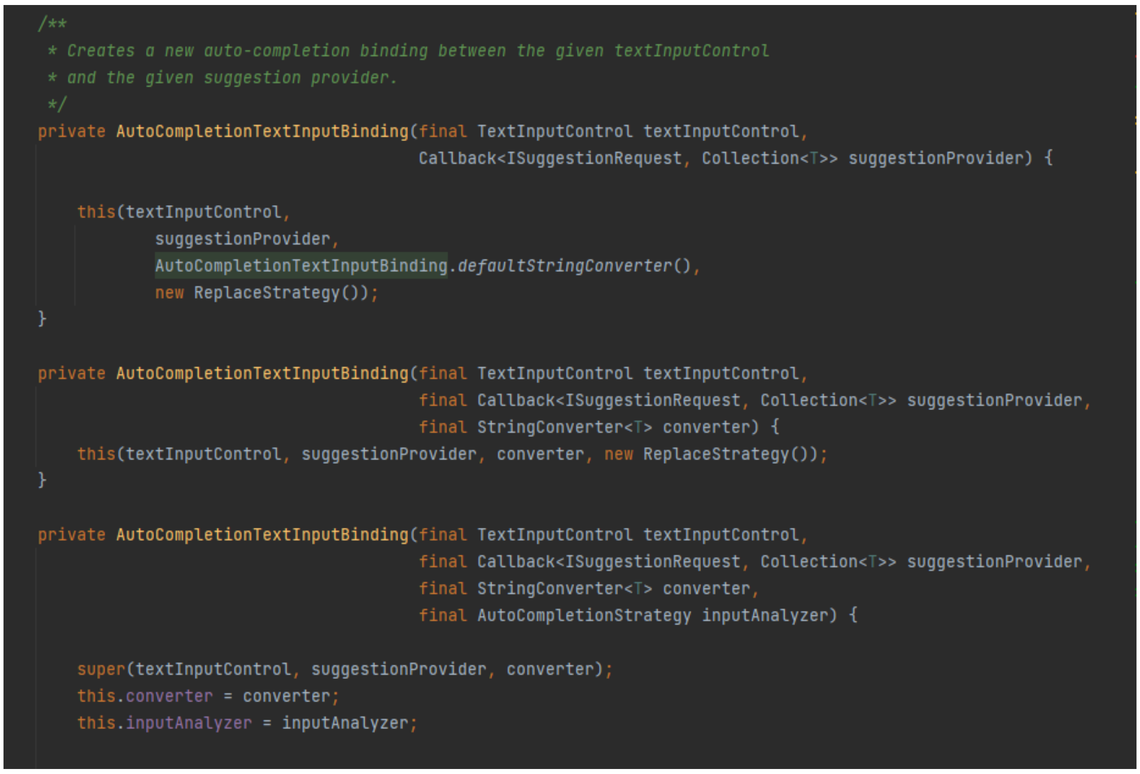


```java
import ...

/**
 * This class is just a simple wrapper for the soon to be refactored SaveDatabaseAction.
 */
public class SaveAction extends SimpleCommand {

    public enum SaveMethod { SAVE, SAVE_AS, SAVE_SELECTED }

    private final SaveMethod saveMethod;
    private final JabRefFrame frame;
    private final PreferencesService preferencesService;

    public SaveAction(SaveMethod saveMethod, JabRefFrame frame, PreferencesService preferencesService, StateManager st
        this.saveMethod = saveMethod;
        this.frame = frame;
        this.preferencesService = preferencesService;

        if (saveMethod == SaveMethod.SAVE_SELECTED) {
            this.executable.bind(ActionHelper.needsEntriesSelected(stateManager));
        } else {
            this.executable.bind(ActionHelper.needsDatabase(stateManager));
        }
    }

    @Override
    public void execute() {
        SaveDatabaseAction saveDatabaseAction = new SaveDatabaseAction(
                frame.getCurrentLibraryTab(),
                preferencesService,
                ~~Globals~~.entryTypesManager);
```

This "Façade" pattern can be found in jabref -> src -> main -> java -> org.jabref -> gui -> exporter -> SaveAction.java

We can say that this class uses a "Façade" pattern since its whole purpose is to hide the complexity of a class (in this case, SaveDatabaseAction.java) and its communication with other classes (in this case, JabRefFrame.java or SaveMethod.java, for example) by providing a much simpler interface.

SINGLETON PATTERN



```
/**
 * Creates a new auto-completion binding between the given textInputControl
 * and the given suggestion provider.
 */
private AutoCompletionTextInputBinding(final TextInputControl textInputControl,
                                       Callback<ISuggestionRequest, Collection<T>> suggestionProvider) {

    this(textInputControl,
            suggestionProvider,
            AutoCompletionTextInputBinding.defaultStringConverter(),
            new ReplaceStrategy());
}

private AutoCompletionTextInputBinding(final TextInputControl textInputControl,
                                       final Callback<ISuggestionRequest, Collection<T>> suggestionProvider,
                                       final StringConverter<T> converter) {
    this(textInputControl, suggestionProvider, converter, new ReplaceStrategy());
}

private AutoCompletionTextInputBinding(final TextInputControl textInputControl,
                                       final Callback<ISuggestionRequest, Collection<T>> suggestionProvider,
                                       final StringConverter<T> converter,
                                       final AutoCompletionStrategy inputAnalyzer) {

    super(textInputControl, suggestionProvider, converter);
    this.converter = converter;
    this.inputAnalyzer = inputAnalyzer;
```

This “Singleton” pattern can be found in jabref -> src -> main -> java -> org.jabref->gui->autocompleter-> AutoCompletionTextInputBinding.java

By analysing this snippet of code, we can see that this class uses a “Singleton” pattern. We can conclude this from the use of the “private” keyword on these 3 constructors. Its purpose is to create 3 different and unique objects which all together will contribute to the auto-completion binding mechanism.

Pedro Simões, Nº 58674