## Patterns\_58210

**Design Patterns:**

1- **Factory**

File: src/gui/action/ActionFactory

Lines: 102-106

public MenuItem createMenuItem(Action action, Command command) {

MenuItem menuItem = new MenuItem();

configureMenuItem(action, command, menuItem);

return menuItem;

}

As we can see from both the class name and the one of the methods from said class this is a clear case of the factory method in this case to create a new menu item.

2- **Singleton**

File: src/gui/desktop/JabRefDesktop

Lines: 45-49

private static final NativeDesktop NATIVE\_DESKTOP = getNativeDesktop();

private static final Pattern REMOTE\_LINK\_PATTERN = Pattern.compile("[a-z]+://.\*");

private JabRefDesktop() {

}

As we can see from the absence of instance variables that were replaced by "private static final"'s, and the empty constructor meaning that only one instance of nativeDesktop can be created, we come to the conclusion that this is the Singleton design Pattern.

3- **Observer**

File: src/gui/collab/DatabaseChangeMonitor

Lines: 51-64

public void fileUpdated() {

// File on disk has changed, thus look for notable changes and notify listeners in case there are such changes

ChangeScanner scanner = new ChangeScanner(database, preferencesService, stateManager);

BackgroundTask.wrap(scanner::scanForChanges)

.onSuccess(changes -> {

if (!changes.isEmpty()) {

listeners.forEach(listener -> listener.databaseChanged(changes));

}

})

.onFailure(e -> LOGGER.error("Error while watching for changes", e))

.executeWith(taskExecutor);

}

public void addListener(DatabaseChangeListener listener) {

listeners.add(listener);

}

Here we can find the Observer design patter as this class stores instances of "DatabaseChangeListener" that it notifies upon, in this case, a file is updated.