DESIGN PATTERNS

Singleton Structure – Creational Pattern

Diagrama

Descripción generada automáticamente

private static ExternalFileTypes *singleton*;  
  
  
private ExternalFileTypes() {  
 updateExternalFileTypes();  
}  
  
public static ExternalFileTypes getInstance() {  
 if (ExternalFileTypes.*singleton* == null) {  
 ExternalFileTypes.*singleton* = new ExternalFileTypes();  
 }  
 return ExternalFileTypes.*singleton*;  
}

The exact location on the codebase:

package org.jabref.gui.externalfiletype;

public class ExternalFileTypes {

Clients access a Singleton instance solely through Singleton’s Instance operation. Provides global access to a class that is restricted to one instance. In general, this is achieved by having a private constructor ExternalFileType , with a public method that instantiates the class “if” it is not already instantiated. This is useful when exactly one object is needed to coordinate actions across the system.

Facade Pattern (structural design pattern)

Diagrama

Descripción generada automáticamente

The interface:

package org.jabref.gui.search.rules.describer;  
  
import javafx.scene.text.TextFlow;  
  
@FunctionalInterface  
public interface SearchDescriber {  
  
 TextFlow getDescription();  
}

Classes with the interface implemented:

public class GrammarBasedSearchRuleDescriber implements SearchDescriber

public class ContainsAndRegexBasedSearchRuleDescriber implements SearchDescriber

Façade class that wrap the classes tha implement the interface:

public class SearchDescribers {  
  
 private SearchDescribers() {  
 }  
public static SearchDescriber getSearchDescriberFor(SearchQuery searchQuery) {  
 if (searchQuery.getRule() instanceof GrammarBasedSearchRule grammarBasedSearchRule) {  
 return new GrammarBasedSearchRuleDescriber(grammarBasedSearchRule.getSearchFlags(), grammarBasedSearchRule.getTree());  
 } else if (searchQuery.getRule() instanceof ContainBasedSearchRule containBasedSearchRule) {  
 return new ContainsAndRegexBasedSearchRuleDescriber(containBasedSearchRule.getSearchFlags(), searchQuery.getQuery());  
 } else if (searchQuery.getRule() instanceof RegexBasedSearchRule regexBasedSearchRule) {  
 return new ContainsAndRegexBasedSearchRuleDescriber(regexBasedSearchRule.getSearchFlags(), searchQuery.getQuery());  
 } else {  
 throw new IllegalStateException("Cannot find a describer for searchRule " + searchQuery.getRule() + " and query " + searchQuery.getQuery());  
 }  
 }  
}

The class GlobalSearchBar use de façade class to access the subsystem

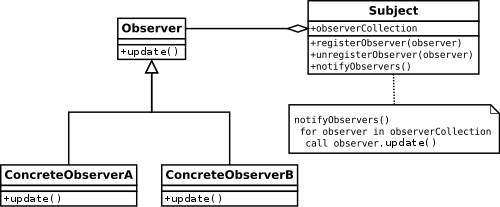
private void updateSearchResultsForQuery(SearchQuery query) {  
 updateResults(this.stateManager.getSearchResultSize().intValue(), SearchDescribers.*getSearchDescriberFor*(query).getDescription(),  
 query.isGrammarBasedSearch());  
}

The exact location on the codebase:

package org.jabref.gui.search;

SearchDescribers is a wrapper class that encapsulates a subsystem in order to hide the subsystem’s complexity, and acts as a point of entry into a subsystem without adding more functionality in itself.

Observer Pattern (Behavioral design pattern)



Subject class:

public class EntryEditor extends BorderPane {  
  
  
 private final LibraryTab libraryTab;  
 private final BibDatabaseContext databaseContext;  
 private final EntryEditorPreferences entryEditorPreferences;  
 private final ExternalFilesEntryLinker fileLinker;  
 /\*  
 \* Tabs which can apply filter, but seems non-sense  
 \* \*/  
 private final List<EntryEditorTab> tabs;  
 private BibEntry entry;  
 /\*  
   
 private final List<EntryEditorTab> entryEditorTabs = new LinkedList<>();

public EntryEditor(LibraryTab libraryTab, ExternalFileTypes externalFileTypes) {  
 this.libraryTab = libraryTab;  
 this.databaseContext = libraryTab.getBibDatabaseContext();  
  
 ViewLoader.*view*(this)  
 .root(this)  
 .load();  
  
 this.entryEditorPreferences = preferencesService.getEntryEditorPreferences();  
 this.fileLinker = new ExternalFilesEntryLinker(externalFileTypes, preferencesService.getFilePreferences(),  
 databaseContext);  
  
 EasyBind.*subscribe*(tabbed.getSelectionModel().selectedItemProperty(), tab -> {  
 EntryEditorTab activeTab = (EntryEditorTab) tab;  
 if (activeTab != null) {  
 activeTab.notifyAboutFocus(entry);  
 }  
 });  
  
 setupKeyBindings();  
  
 this.tabs = createTabs();  
  
 this.setOnDragOver(event -> {  
 if (event.getDragboard().hasFiles()) {  
 event.acceptTransferModes(TransferMode.*COPY*, TransferMode.*MOVE*, TransferMode.*LINK*);  
 }  
 event.consume();  
 });  
  
 this.setOnDragDropped(event -> {  
 BibEntry entry = this.getEntry();  
 boolean success = false;  
  
 if (event.getDragboard().hasContent(DataFormat.*FILES*)) {  
 List<Path> files = event.getDragboard().getFiles().stream().map(File::toPath).collect(Collectors.*toList*());  
 switch (event.getTransferMode()) {  
 case *COPY* -> {  
 *LOGGER*.debug("Mode COPY");  
 fileLinker.copyFilesToFileDirAndAddToEntry(entry, files);  
 }  
 case *MOVE* -> {  
 *LOGGER*.debug("Mode MOVE");  
 fileLinker.moveFilesToFileDirAndAddToEntry(entry, files);  
 }  
 case *LINK* -> {  
 *LOGGER*.debug("Mode LINK");  
 fileLinker.addFilesToEntry(entry, files);  
 }  
 }  
 success = true;  
 }  
  
 event.setDropCompleted(success);  
 event.consume();  
 });  
}

private void deleteEntry() {  
 libraryTab.delete(entry);  
}

The observer class:

public abstract class EntryEditorTab extends Tab {  
  
 protected BibEntry currentEntry;  
  
 */\*\*  
 \* Needed to track for which type of entry this tab was build and to rebuild it if the type changes  
 \*/* private EntryType currentEntryType;  
  
 */\*\*  
 \* Decide whether to show this tab for the given entry.  
 \*/* public abstract boolean shouldShow(BibEntry entry);  
  
 */\*\*  
 \* Updates the view with the contents of the given entry.  
 \*/* protected abstract void bindToEntry(BibEntry entry);  
  
 */\*\*  
 \* The tab just got the focus. Override this method if you want to perform a special action on focus (like selecting  
 \* the first field in the editor)  
 \*/* protected void handleFocus() {  
 // Do nothing by default  
 }  
  
 */\*\*  
 \* Notifies the tab that it got focus and should display the given entry.  
 \*/* public void notifyAboutFocus(BibEntry entry) {  
 if (!entry.equals(currentEntry) || !entry.getType().equals(currentEntryType)) {  
 currentEntry = entry;  
 currentEntryType = entry.getType();  
 bindToEntry(entry);  
 }  
 handleFocus();  
 }

Concreate observerA (there are more):

public class SourceTab extends EntryEditorTab {

private void updateCodeArea() {  
 DefaultTaskExecutor.*runAndWaitInJavaFXThread*(() -> {  
 if (codeArea == null) {  
 setupSourceEditor();  
 }  
  
 codeArea.clear();  
 try {  
 codeArea.appendText(getSourceString(currentEntry, mode, fieldWriterPreferences));  
 codeArea.setEditable(true);  
 highlightSearchPattern();  
 } catch (IOException ex) {  
 codeArea.setEditable(false);  
 codeArea.appendText(ex.getMessage() + "\n\n" +  
 Localization.*lang*("Correct the entry, and reopen editor to display/edit source."));  
 *LOGGER*.debug("Incorrect entry", ex);  
 }  
 });  
}

The exact location on the codebase:

package org.jabref.gui.entryeditor;

The observer pattern is a software design pattern in which an object, named the subject (Entry editor) , maintains a list of EntryEditorTab, called observers, and notifies them automatically of any state changes, usually by calling one of their methods. The observer pattern will be used when an element wants to be aware of another, without having to continuously check if it has changed or not.

*EntryEditor also registers itself to the event bus, receiving events whenever a field of the entry changes, enabling  
\* the text fields to update themselves if the change is made from somewhere else.*

Factory Method (Creational patterns)

Diagrama

Descripción generada automáticamente

The Creator class that requires a Product object doesn't instantiate the Product1 class directly. Instead, the Creator refers to a separate factoryMethod() to create a product object, which makes the Creator independent of which concrete class is instantiated. Subclasses of Creator can redefine which class to instantiate. In this example, the Creator1 subclass implements the abstract factoryMethod() by instantiating the Product1 class.

The product is the class Exporter

public abstract class Exporter {  
  
 private final String id;  
 private final String displayName;  
 private final FileType fileType;  
  
 public Exporter(String id, String displayName, FileType extension) {  
 this.id = id;  
 this.displayName = displayName;  
 this.fileType = extension;  
 }  
  
 */\*\*  
 \* Returns a one-word ID (used, for example, to identify the exporter in the console).  
 \*/* public String getId() {  
 return id;  
 }  
  
 */\*\*  
 \* Returns the name of the exporter (to display to the user).  
 \*/* public String getName() {  
 return displayName;  
 }  
  
 */\*\*  
 \* Returns the type of files this exporter creates.  
 \*/* public FileType getFileType() {  
 return fileType;  
 }  
  
 @Override  
 public String toString() {  
 return displayName;  
 }

Creator class (create product objects of particular types by adding them to the arraylist exporters). The method getExporters() returns a list instances of several posssible classes

public class ExporterFactory {  
  
@Deprecated public static int entryNumber;  
  
 private final List<Exporter> exporters;  
  
 private ExporterFactory(List<Exporter> exporters) {  
 this.exporters = Objects.*requireNonNull*(exporters);  
 }  
  
 public static ExporterFactory create(List<TemplateExporter> customFormats,  
 LayoutFormatterPreferences layoutPreferences,  
 SavePreferences savePreferences,  
 XmpPreferences xmpPreferences,  
 BibDatabaseMode bibDatabaseMode,  
 BibEntryTypesManager entryTypesManager) {  
  
 List<Exporter> exporters = new ArrayList<>();  
  
 // Initialize build-in exporters  
  
 // Initialize build-in exporters  
 exporters.add(new TemplateExporter("HTML", "html", "html", null, StandardFileType.*HTML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter(Localization.*lang*("Simple HTML"), "simplehtml", "simplehtml", null, StandardFileType.*HTML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("DocBook 5.1", "docbook5", "docbook5", null, StandardFileType.*XML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("DocBook 4", "docbook4", "docbook4", null, StandardFileType.*XML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("DIN 1505", "din1505", "din1505winword", "din1505", StandardFileType.*RTF*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("BibO RDF", "bibordf", "bibordf", null, StandardFileType.*RDF*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter(Localization.*lang*("HTML table"), "tablerefs", "tablerefs", "tablerefs", StandardFileType.*HTML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter(Localization.*lang*("HTML list"), "listrefs", "listrefs", "listrefs", StandardFileType.*HTML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter(Localization.*lang*("HTML table (with Abstract & BibTeX)"), "tablerefsabsbib", "tablerefsabsbib", "tablerefsabsbib", StandardFileType.*HTML*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("Harvard RTF", "harvard", "harvard", "harvard", StandardFileType.*RTF*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("ISO 690 RTF", "iso690rtf", "iso690RTF", "iso690rtf", StandardFileType.*RTF*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("ISO 690", "iso690txt", "iso690", "iso690txt", StandardFileType.*TXT*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("Endnote", "endnote", "EndNote", "endnote", StandardFileType.*TXT*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("OpenOffice/LibreOffice CSV", "oocsv", "openoffice-csv", "openoffice", StandardFileType.*CSV*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("RIS", "ris", "ris", "ris", StandardFileType.*RIS*, layoutPreferences, savePreferences, BlankLineBehaviour.*DELETE\_BLANKS*).withEncoding(StandardCharsets.*UTF\_8*));  
 exporters.add(new TemplateExporter("MIS Quarterly", "misq", "misq", "misq", StandardFileType.*RTF*, layoutPreferences, savePreferences));  
 exporters.add(new TemplateExporter("CSL YAML", "yaml", "yaml", null, StandardFileType.*YAML*, layoutPreferences, savePreferences, BlankLineBehaviour.*DELETE\_BLANKS*));  
 exporters.add(new BibTeXMLExporter());  
 exporters.add(new OpenOfficeDocumentCreator());  
 exporters.add(new OpenDocumentSpreadsheetCreator());  
 exporters.add(new MSBibExporter());  
 exporters.add(new ModsExporter());  
 exporters.add(new XmpExporter(xmpPreferences));  
 exporters.add(new XmpPdfExporter(xmpPreferences));  
 exporters.add(new EmbeddedBibFilePdfExporter(bibDatabaseMode, entryTypesManager, savePreferences.getFieldWriterPreferences()));  
  
 // Now add custom export formats  
 exporters.addAll(customFormats);  
  
 return new ExporterFactory(exporters);  
 }  
  
 public List<Exporter> getExporters() {  
 return Collections.*unmodifiableList*(exporters);  
 }

The concrete creator hold the actual code that instantiates the product object. The concrete products are the classes TemplateExporters, BibTeXMLExporter, OpenOfficeDocumentCreator, MSBibExporter…

public class ExportToClipboardAction extends SimpleCommand {  
  
  
 private JabRefFrame frame;  
 private final DialogService dialogService;  
 private LibraryTab panel;  
 private final List<BibEntry> entries = new ArrayList<>();  
 private final ExporterFactory exporterFactory;  
 private final ClipBoardManager clipBoardManager;  
 private final TaskExecutor taskExecutor;  
 private final PreferencesService preferences;  
  
 }  
  
 @Override  
 public void execute() {  
 if (panel == null) {  
 panel = frame.getCurrentLibraryTab();  
 }  
  
 if (panel.getSelectedEntries().isEmpty()) {  
 dialogService.notify(Localization.*lang*("This operation requires one or more entries to be selected."));  
 return;  
 }  
  
 List<Exporter> exporters = exporterFactory.getExporters().stream()  
 .sorted(Comparator.*comparing*(Exporter::getName))  
 .filter(exporter -> *SUPPORTED\_FILETYPES*.contains(exporter.getFil eType()))  
 .collect(Collectors.*toList*());  
  
 // Find default choice, if any  
 Exporter defaultChoice = exporters.stream()  
 .filter(exporter -> exporter.getName().equals(preferences.getImportExportPreferences().getLastExportExtension()))  
 .findAny()  
 .orElse(null);  
  
 Optional<Exporter> selectedExporter = dialogService.showChoiceDialogAndWait(  
 Localization.*lang*("Export"), Localization.*lang*("Select export format"),  
 Localization.*lang*("Export"), defaultChoice, exporters);  
  
 selectedExporter.ifPresent(exporter -> BackgroundTask.*wrap*(() -> exportToClipboard(exporter))  
 .onSuccess(this::setContentToClipboard)  
 .onFailure(ex -> {  
 *LOGGER*.error("Error exporting to clipboard", ex);  
 dialogService.showErrorDialogAndWait("Error exporting to clipboard", ex);  
 })  
 .executeWith(taskExecutor));  
 }

Template method pattern (Behavioural pattern)

Diagrama

Descripción generada automáticamente

It is use when you have two separate classes with very similar functionality and order of operations. Template Method lets subclasses redefine certain steps of an algorithm without changing the alorithm's structure.

Abstract class

public abstract class SuggestionProvider<T> {  
  
 protected abstract Equivalence<T> getEquivalence();  
  
 public Collection<T> getPossibleSuggestions() {  
 Comparator<T> comparator = getComparator().reversed();  
 Equivalence<T> equivalence = getEquivalence();  
 return getSource().map(equivalence::wrap) // Need to do a bit of acrobatic as there is no distinctBy method  
 .distinct()  
 .map(Equivalence.Wrapper::get)  
 .sorted(comparator)  
 .collect(Collectors.*toList*());  
 }  
  
 */\*\*  
 \* Get the comparator to order the suggestions  
 \*/* protected abstract Comparator<T> getComparator();  
  
 */\*\*  
 \* Check the given candidate is a match (ie a valid suggestion)  
 \*/* protected abstract boolean isMatch(T candidate, ISuggestionRequest request);  
  
 public abstract Stream<T> getSource();  
}

Concrete class EmptySuggestionProvider

public class EmptySuggestionProvider extends SuggestionProvider<String> {  
 @Override  
 protected Equivalence<String> getEquivalence() {  
 return Equivalence.*equals*().onResultOf(value -> value);  
 }  
  
 @Override  
 protected Comparator<String> getComparator() {  
 return Comparator.*naturalOrder*();  
 }  
  
 @Override  
 protected boolean isMatch(String candidate, AutoCompletionBinding.ISuggestionRequest request) {  
 return false;  
 }  
  
 @Override  
 public Stream<String> getSource() {  
 return Stream.*empty*();  
 }  
}

Concrete class BibEntrySuggestionProvider

public class BibEntrySuggestionProvider extends SuggestionProvider<BibEntry> {  
  
 private final BibDatabase database;  
  
 public BibEntrySuggestionProvider(BibDatabase database) {  
 this.database = database;  
 }  
  
 @Override  
 protected Equivalence<BibEntry> getEquivalence() {  
 return Equivalence.*equals*().onResultOf(BibEntry::getCitationKey);  
 }  
  
 @Override  
 protected Comparator<BibEntry> getComparator() {  
 return new EntryComparator(false, true, InternalField.*KEY\_FIELD*);  
 }  
  
 @Override  
 protected boolean isMatch(BibEntry entry, AutoCompletionBinding.ISuggestionRequest request) {  
 String userText = request.getUserText();  
 return entry.getCitationKey()  
 .map(key -> StringUtil.*containsIgnoreCase*(key, userText))  
 .orElse(false);  
 }  
  
 @Override  
 public Stream<BibEntry> getSource() {  
 return database.getEntries().parallelStream();  
 }  
}

The Adapter Pattern (Structural)



Target interface is used by the client to send a request to the adapter.

public interface JabRefIcon {  
  
 Node getGraphicNode();  
  
 String name();  
  
 JabRefIcon withColor(Color color);  
  
 JabRefIcon disabled();  
  
 Ikon getIkon();  
}

The adaptee class defines an existing interface that needs to be adapted.

public class FontIcon extends Text implements Icon {  
 private static final double *EPSILON* = 1.0E-6D;  
 protected StyleableIntegerProperty iconSize;  
 protected StyleableObjectProperty<Paint> iconColor;  
 private StyleableObjectProperty<Ikon> iconCode;

public static FontIcon of(Ikon ikon) {  
 return *of*(ikon, 8, Color.*BLACK*);  
 }

The adapter class sits between the client and the adaptee.

Conforms to what the client is expecting to see, by implementing a target interface JabRefIcon.

Translates the client request into a message that the adaptee will understand, and returns the translated request to the adaptee, using the specific request FontIcon.of(icon)

public class InternalMaterialDesignIcon implements JabRefIcon {  
  
 private final List<Ikon> icons;  
 private Optional<Color> color;  
 private final String unicode;  
  
  
 @Override  
 public Node getGraphicNode() {  
 Ikon icon = icons.get(0);  
 FontIcon fontIcon = FontIcon.*of*(icon);  
 fontIcon.getStyleClass().add("glyph-icon");  
  
// Override the default color from the css files  
 color.ifPresent(color -> fontIcon.setStyle(fontIcon.getStyle() +  
 String.*format*("-fx-fill: %s;", ColorUtil.*toRGBCode*(color)) +  
 String.*format*("-fx-icon-color: %s;", ColorUtil.*toRGBCode*(color))));  
  
 return fontIcon;  
 }  
  
 @Override  
 public JabRefIcon disabled() {  
 return new InternalMaterialDesignIcon(IconTheme.*DEFAULT\_DISABLED\_COLOR*, icons);  
 }  
  
 @Override  
 public JabRefIcon withColor(Color color) {  
 return new InternalMaterialDesignIcon(color, icons);  
 }  
  
 @Override  
 public String name() {  
 return icons.get(0).toString();  
 }  
  
 public String getCode() {  
 return this.unicode;  
 }  
  
 @Override  
 public Ikon getIkon() {  
 return icons.get(0);  
 }

The client class is the part of your system that wants to use a third-party library or external systems. Use and only knows the target interface JabRefIcon

public class IconTheme {

private final JabRefIcon icon;  
  
 JabRefIcons(Ikon... icons) {  
 icon = new InternalMaterialDesignIcon(icons);  
 }  
  
 JabRefIcons(Color color, Ikon... icons) {  
 icon = new InternalMaterialDesignIcon(color, icons);  
 }  
  
 @Override  
 public Ikon getIkon() {  
 return icon.getIkon();  
 }  
  
 @Override  
 public Node getGraphicNode() {  
 return icon.getGraphicNode();  
 }  
  
 public Button asButton() {  
 Button button = new Button();  
 button.setGraphic(getGraphicNode());  
 button.getStyleClass().add("icon-button");  
 return button;  
 }  
  
 public ToggleButton asToggleButton() {  
 ToggleButton button = new ToggleButton();  
 button.setGraphic(getGraphicNode());  
 button.getStyleClass().add("icon-button");  
 return button;  
 }  
  
 @Override  
 public JabRefIcon withColor(Color color) {  
 return icon.withColor(color);  
 }  
  
 @Override  
 public JabRefIcon disabled() {  
 return icon.disabled();  
 }  
 }