CODE SMELLS

LONG METHOD

private void initialize() {

this.localDragboard = stateManager.getLocalDragboard();

viewModel = new GroupTreeViewModel(stateManager, dialogService, preferencesService, taskExecutor, localDragboard);

// Set-up groups tree

groupTree.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

dragExpansionHandler = new DragExpansionHandler();

// Set-up bindings

Platform.runLater(() ->

BindingsHelper.bindContentBidirectional(

groupTree.getSelectionModel().getSelectedItems(),

viewModel.selectedGroupsProperty(),

(newSelectedGroups) -> newSelectedGroups.forEach(this::selectNode),

this::updateSelection

));

// We try to to prevent publishing changes in the search field directly to the search task that takes some time

// for larger group structures.

final Timer searchTask = FxTimer.create(Duration.ofMillis(400), () -> {

LOGGER.debug("Run group search " + searchField.getText());

viewModel.filterTextProperty().setValue(searchField.textProperty().getValue());

});

searchField.textProperty().addListener((observable, oldValue, newValue) -> searchTask.restart());

setNewGroupButtonStyle(groupTree);

groupTree.rootProperty().bind(

EasyBind.map(viewModel.rootGroupProperty(),

group -> {

if (group == null) {

return null;

} else {

return new RecursiveTreeItem<>(

group,

GroupNodeViewModel::getChildren,

GroupNodeViewModel::expandedProperty,

viewModel.filterPredicateProperty());

}

}));

// Icon and group name

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withText(GroupNodeViewModel::getDisplayName)

.withIcon(GroupNodeViewModel::getIcon)

.withTooltip(GroupNodeViewModel::getDescription)

.install(mainColumn);

// Number of hits (only if user wants to see them)

PseudoClass anySelected = PseudoClass.getPseudoClass("any-selected");

PseudoClass allSelected = PseudoClass.getPseudoClass("all-selected");

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withGraphic(group -> {

final StackPane node = new StackPane();

node.getStyleClass().setAll("hits");

if (!group.isRoot()) {

BindingsHelper.includePseudoClassWhen(node, anySelected,

group.anySelectedEntriesMatchedProperty());

BindingsHelper.includePseudoClassWhen(node, allSelected,

group.allSelectedEntriesMatchedProperty());

}

Text text = new Text();

if (preferencesService.getDisplayGroupCount()) {

text.textProperty().bind(group.getHits().asString());

}

text.getStyleClass().setAll("text");

node.getChildren().add(text);

node.setMaxWidth(Control.USE\_PREF\_SIZE);

return node;

})

.install(numberColumn);

// Arrow indicating expanded status

new ViewModelTreeTableCellFactory<GroupNodeViewModel>()

.withGraphic(viewModel -> {

final StackPane disclosureNode = new StackPane();

disclosureNode.visibleProperty().bind(viewModel.hasChildrenProperty());

disclosureNode.getStyleClass().setAll("tree-disclosure-node");

final StackPane disclosureNodeArrow = new StackPane();

disclosureNodeArrow.getStyleClass().setAll("arrow");

disclosureNode.getChildren().add(disclosureNodeArrow);

return disclosureNode;

})

.withOnMouseClickedEvent(group -> event -> {

group.toggleExpansion();

event.consume();

})

.install(expansionNodeColumn);

// Set pseudo-classes to indicate if row is root or sub-item ( > 1 deep)

PseudoClass rootPseudoClass = PseudoClass.getPseudoClass("root");

PseudoClass subElementPseudoClass = PseudoClass.getPseudoClass("sub");

groupTree.setRowFactory(treeTable -> {

TreeTableRow<GroupNodeViewModel> row = new TreeTableRow<>();

row.treeItemProperty().addListener((ov, oldTreeItem, newTreeItem) -> {

setNewGroupButtonStyle(treeTable);

boolean isRoot = newTreeItem == treeTable.getRoot();

row.pseudoClassStateChanged(rootPseudoClass, isRoot);

boolean isFirstLevel = (newTreeItem != null) && (newTreeItem.getParent() == treeTable.getRoot());

row.pseudoClassStateChanged(subElementPseudoClass, !isRoot && !isFirstLevel);

});

// Remove disclosure node since we display custom version in separate column

// Simply setting to null is not enough since it would be replaced by the default node on every change

row.setDisclosureNode(null);

row.disclosureNodeProperty().addListener((observable, oldValue, newValue) -> row.setDisclosureNode(null));

// Add context menu (only for non-null items)

row.contextMenuProperty().bind(

EasyBind.wrapNullable(row.itemProperty())

.map(this::createContextMenuForGroup)

.orElse((ContextMenu) null));

row.addEventFilter(MouseEvent.MOUSE\_PRESSED, event -> {

if (event.getButton() == MouseButton.SECONDARY) {

// Prevent right-click to select group

event.consume();

}

});

// Drag and drop support

row.setOnDragDetected(event -> {

List<String> groupsToMove = new ArrayList<>();

for (TreeItem<GroupNodeViewModel> selectedItem : treeTable.getSelectionModel().getSelectedItems()) {

if ((selectedItem != null) && (selectedItem.getValue() != null)) {

groupsToMove.add(selectedItem.getValue().getPath());

}

}

if (groupsToMove.size() > 0) {

localDragboard.clearAll();

}

// Put the group nodes as content

Dragboard dragboard = treeTable.startDragAndDrop(TransferMode.MOVE);

// Display the group when dragging

dragboard.setDragView(row.snapshot(null, null));

ClipboardContent content = new ClipboardContent();

content.put(DragAndDropDataFormats.GROUP, groupsToMove);

dragboard.setContent(content);

event.consume();

});

row.setOnDragOver(event -> {

Dragboard dragboard = event.getDragboard();

if ((event.getGestureSource() != row) && (row.getItem() != null) && row.getItem().acceptableDrop(dragboard)) {

event.acceptTransferModes(TransferMode.MOVE, TransferMode.LINK);

// expand node and all children on drag over

dragExpansionHandler.expandGroup(row.getTreeItem());

if (localDragboard.hasBibEntries()) {

ControlHelper.setDroppingPseudoClasses(row);

} else {

ControlHelper.setDroppingPseudoClasses(row, event);

}

}

event.consume();

});

row.setOnDragExited(event -> {

ControlHelper.removeDroppingPseudoClasses(row);

});

row.setOnDragDropped(event -> {

Dragboard dragboard = event.getDragboard();

boolean success = false;

if (dragboard.hasContent(DragAndDropDataFormats.GROUP)) {

List<String> pathToSources = (List<String>) dragboard.getContent(DragAndDropDataFormats.GROUP);

List<GroupNodeViewModel> changedGroups = new LinkedList<>();

for (String pathToSource : pathToSources) {

Optional<GroupNodeViewModel> source = viewModel

.rootGroupProperty().get()

.getChildByPath(pathToSource);

if (source.isPresent()) {

source.get().draggedOn(row.getItem(), ControlHelper.getDroppingMouseLocation(row, event));

changedGroups.add(source.get());

success = true;

}

}

groupTree.getSelectionModel().clearSelection();

changedGroups.forEach(value -> selectNode(value, true));

}

if (localDragboard.hasBibEntries()) {

List<BibEntry> entries = localDragboard.getBibEntries();

row.getItem().addEntriesToGroup(entries);

success = true;

}

event.setDropCompleted(success);

event.consume();

});

return row;

});

// Filter text field

setupClearButtonField(searchField);

}

LOCATION - src/main/java/org/jabref/gui/groups/GroupTreeView.java

WHY

Almost 200 lines of code.

REFACTORING PROPOSAL

It should have auxiliary methods to improve readability and reduce its complexity.

COMMENTS

Texto

Descrição gerada automaticamente

LOCATION - src/main/java/org/jabref/gui/citationkeypattern/CitationKeyPatternPanel.java

WHY

Commented out old code that is now irrelevant making the method clogged up with too many comments and hard to read.

REFACTORING PROPOSAL

Deleting this comments or making just a small reference.

LARGE CLASS

Texto

Descrição gerada automaticamente

LOCATION - src/main/java/org/jabref/preferences/JabRefPreferences.java

WHY

Large class with almost 3000 lines of code.

REFACTORING PROPOSAL

It should be divided into several smaller classes. Grouping types of preferences.