1) Package Overview

Packages and Classes

- **Package:** guiStudentHome
    - **Model:** ModelStudentHome
    - **View:** ViewStudentHome
    - **Controller:** ControllerStudentHome

Purpose
Implements the **Student Home** GUI for the Student Discussion System using the **Model-View-Controller** pattern. The UI allows students to create posts, search/filter, view feeds, reply, edit, and delete posts or replies, and logout. The GUI employs the **entityClasses.Post** class (Task 3) through collection adapters to satisfy the **Students User Stories** end-to-end.

Alignment with provided GUI pattern

- Controller owns event handlers and orchestration.
- View owns only UI composition and node references; no business logic.
- Model holds UI-adjacent data structures (reply collection cache).
- Naming and packaging align to the provided FoundationsF25 GUI convention: gui<Feature>/<Model|View|Controller><Feature>.java.

---

2) How MVC Collaborates (Concept of Operations)

1. **View renders UI** (top search bar, center feed, bottom “create post”).
2. **Controller wires events** (create, search, filter, show all, my posts only, edit/delete, reply, logout).
3. **Controller delegates** to PostCollection and ReplyCollection for CRUD operations.

4. **Model** retains a shared, in-memory ReplyCollection (non-persistent), supporting quick UI updates.

5. **Controller refreshes the feed** and View rebinds content (rendering Post cards + toggled Reply lists).

This flow mirrors the provided GUI packages' responsibilities split and uses the **Post** entity from Task 3.

---

3) Class Documentation

3.1 ViewStudentHome

Purpose
Composes and displays the Student Home scene: search and thread filter controls, feed area, and create-post panel.

Key Attributes (with rationale)

- label_PageTitle — Screen title for context.
- tfSearch, btnSearch, btnShowAll — Keyword search and quick reset for discoverability.
- tfThreadFilter — Thread/category filter to implement "navigate by topic."
- cbMyPostsOnly — Focus on personal contributions for participation tracking.
- tfThreadName, tfPostTitle, taPostContent, btnCreatePost — Create-post inputs; maps directly to Post's title/content/thread.
- vbFeed, spFeed — Scrollable container for dynamic post cards.

Operations

- displayStudentHome(Stage, User) boots the screen.
- start(Stage, User) constructs scene, sets title, and invokes controller initialization.

- buildTop(), buildCenter(), buildBottom(): isolate layout concerns for readability and maintainability.

How this enables user stories

- Students can create, browse, and search posts from a single page.
- Dedicated input fields guarantee valid mapping to Post attributes.

3.2 ControllerStudentHome

Purpose
Central coordinator for Student Home. Owns event wiring, validation, and orchestration of PostCollection/ReplyCollection calls; updates the View feed accordingly.

Key Attributes (with rationale)

- currentUser: Needed for authorship checks (edit/delete permissions).
- postCollection, replyCollection: Data access/adapter classes enabling CRUD operations for posts and replies.
- replyReads (Map<String, Set<String>>) : Lightweight read/unread state by reply and user (UX cue for attention).
- cachedPosts: In-memory list for filtering/search without requery.
- view: Reference to View for pushing UI updates.

Operations (event handlers & helpers)

- **Initialization & refresh**
    - initialize(ViewStudentHome): wires all UI event handlers.
    - refreshPosts(boolean reload): reloads from PostCollection and applies filters.
- **Create/Search/Filter**

    - onCreatePost(): validates inputs and calls postCollection.createPost(...).
    - onSearch(): queries postCollection.searchPosts(keyword), optionally filters “my posts only.”
    - Thread filter listener (tfThreadFilter): narrows cached feed by thread name.
- **Feed rendering**
    - renderFeed(List<Post>): clears and repopulates UI cards.
    - buildPostCard(Post): composes a single post card with title/thread/author/content and action buttons.
- **Reply flows**
    - onReply(Post): modal to submit a reply and persist via replyCollection.createReply(...).
    - populateReplies(Post, VBox): retrieves replies, flags unread, binds edit/delete per ownership.
    - onEditReply(Reply, Post), onDeleteReply(Reply, Post): guarded by author checks.
- **Post edit/delete**
    - onEditPost(Post): controlled content update via postCollection.updatePostContent(...).
    - onDeletePost(Post): confirmation + postCollection.deletePost(...) (soft delete).
- **Logout**
    - performLogout(): prompt + stage close; returns to FoundationsMain.
- **Helpers**
    - usernameOf(User): robust username lookup via reflection (supports getUsername()/getUserName()).
    - filterMine(List<Post>): filters by current user for “My posts only.”

    - isUnread(Reply), markRead(Reply): per-user unread state.

How this enables user stories

- **Create**: onCreatePost() maps directly to Post creation.
- **Read/Browse**: refreshPosts(), renderFeed() display active posts.
- **Update**: onEditPost(), onEditReply() provide safe editing.
- **Delete**: onDeletePost(), onDeleteReply() implement soft delete flows.
- **Search/Filter**: onSearch(), thread filter, and “My posts only” fulfill discovery/ownership views.
- **Replies**: onReply() and populateReplies() provide threaded discussion.

3.3 ModelStudentHome

Purpose
Holds model-side cache for replies used by Student UI. Scope is limited and non-persistent (per assignment constraints).

Key Attributes (with rationale)

- protected static ReplyCollection allReplies — a single in-memory reply collection to facilitate quick UI access without DB roundtrips.

Operations

- None beyond data holding (the controller is the only client). This separation clarifies MVC boundaries for future expansion (e.g., adding observable lists or bindings).

How this enables user stories

- Efficient reply rendering supports responsive UI for reading and participation.

---

4) Requirements Coverage (Student User Stories ↔ Code)

REQ ID	User Story / GUI Capability	Where Implemented	Evidence
REQ-UI-CreatePost	Students can create posts	ControllerStudentHome.onCreatePost, ViewStudentHome input fields	Javadoc + screenshot of create handler
REQ-UI-ViewFeed	Students can view a list of posts	renderFeed, buildPostCard, ViewStudentHome.vbFeed/spFeed	Feed screenshot
REQ-UI-Search	Students can search posts by keyword	onSearch, tfSearch, btnSearch, btnShowAll	Search bar screenshot
REQ-UI-FilterThread	Filter by thread/category	Thread filter listener, tfThreadFilter	Filter screenshot
REQ-UI-MyPostsOnly	Show only posts created by me	cbMyPostsOnly, filterMine	Checkbox + filtered feed screenshot
REQ-UI-Reply	Students can reply to posts	onReply, populateReplies, ReplyCollection.createReply	“Reply” dialog screenshot
REQ-UI-EditPost	Students can edit their posts	onEditPost + author checks	Edit dialog screenshot
REQ-UI-DeletePost	Students can delete their posts	onDeletePost + confirmation	Confirm dialog screenshot
REQ-UI-EditReply	Students can edit their replies	onEditReply + author checks	Edit reply screenshot
REQ-UI-DeleteReply	Students can delete their replies	onDeleteReply + confirmation	Delete reply screenshot

REQ-UI-UnreadCue	Unread replies highlighted	isUnread, markRead, style application	Reply block highlight screenshot
REQ-UI-Logout	Students can log out	performLogout, btnLogout	Logout confirm screenshot

---

5) Attribute Tables by Class

5.1 ViewStudentHome — UI Attributes

Attribute	Type	Purpose / Rationale
label_PageTitle	Label	Screen context and branding
tfSearch, btnSearch, btnShowAll	TextField, Button, Button	Keyword search UX and quick reset
tfThreadFilter	TextField	Topic-based filtering
cbMyPostsOnly	CheckBox	Focus on student's own posts
tfThreadName, tfPostTitle, taPostContent, btnCreatePost	Inputs + action	Map directly to Post(title/content/thread)
vbFeed, spFeed	VBox, ScrollPane	Dynamic feed area for posts and nested replies

5.2 ControllerStudentHome — State & Adapters

Attribute	Type	Purpose / Rationale
currentUser	User	Authorship and permissions
postCollection	PostCollection	Post CRUD backend
replyCollection	ReplyCollection	Reply CRUD backend
replyReads	Map<String, Set<String>>	Read/unread indicator per user
cachedPosts	List<Post>	Fast in-memory filtering
view	ViewStudentHome	Push UI updates from controller

5.3 ModelStudentHome — Data

Attribute	Type	Purpose / Rationale
allReplies	ReplyCollection	Shared model cache for replies

---

6) Javadoc-Style Method Summaries (Extract)

ControllerStudentHome

- initialize(ViewStudentHome v): wires all event handlers and triggers initial feed load.
- onCreatePost(): validates inputs and persists via postCollection.createPost.
- onSearch(): searches posts; integrates “my posts only.”
- refreshPosts(boolean reload): reloads and applies thread/ownership filters.
- renderFeed(List<Post>), buildPostCard(Post): constructs UI cards.
- onReply(Post), populateReplies(Post, VBox): reply dialog and rendering.
- onEditPost(Post), onDeletePost(Post): guarded by author checks and confirmations.
- onEditReply(Reply, Post), onDeleteReply(Reply, Post): reply editing/deletion with ownership checks.
- performLogout(): confirmation and stage close.

ViewStudentHome

- displayStudentHome(Stage, User): entry point.
- start(Stage, User): scene build + controller init.
- buildTop(), buildCenter(), buildBottom(): layout composition.

ModelStudentHome

- Holds ReplyCollection for view-adjacent access (non-persistent).

7) Screenshot Appendix

Figure 4.1 — Package & Classes (Task 4.1)

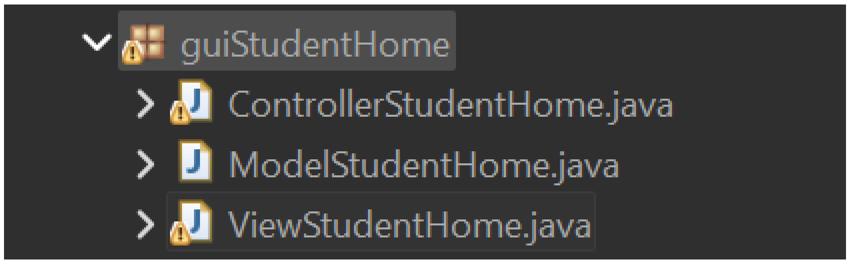
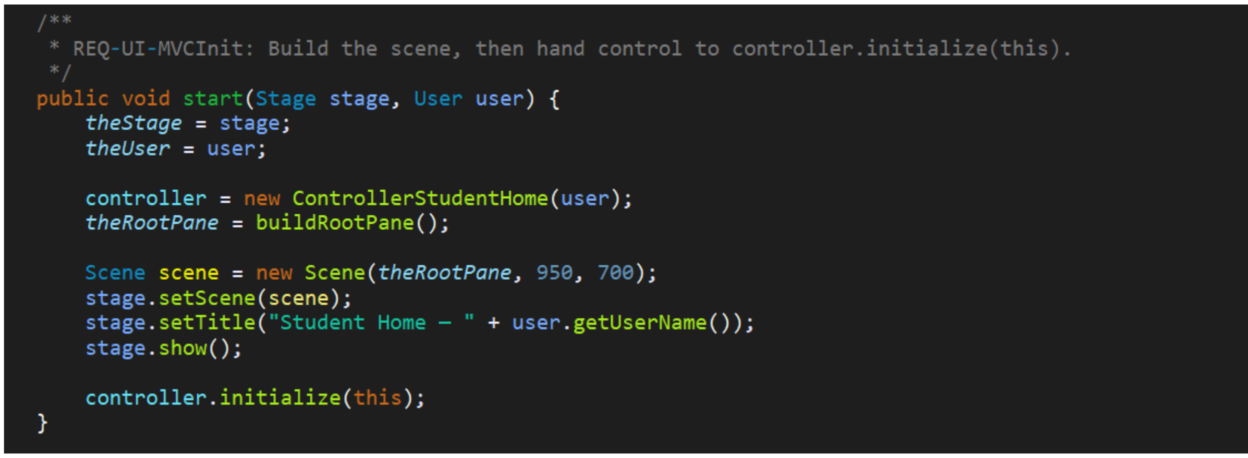


Figure 4.2 — MVC Alignment (Task 4.2)

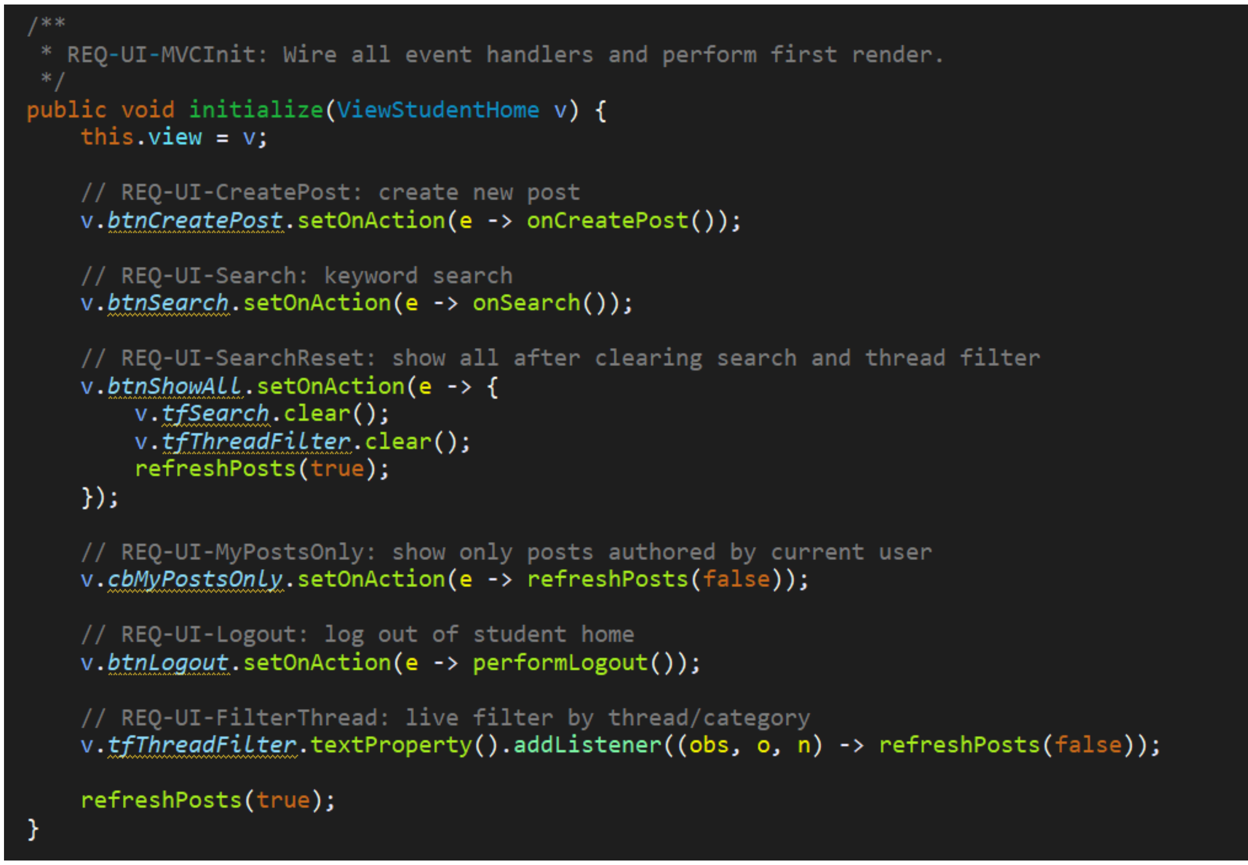
- Screenshots:

  1. ViewStudentHome.start(...) calling controller.initialize(this)



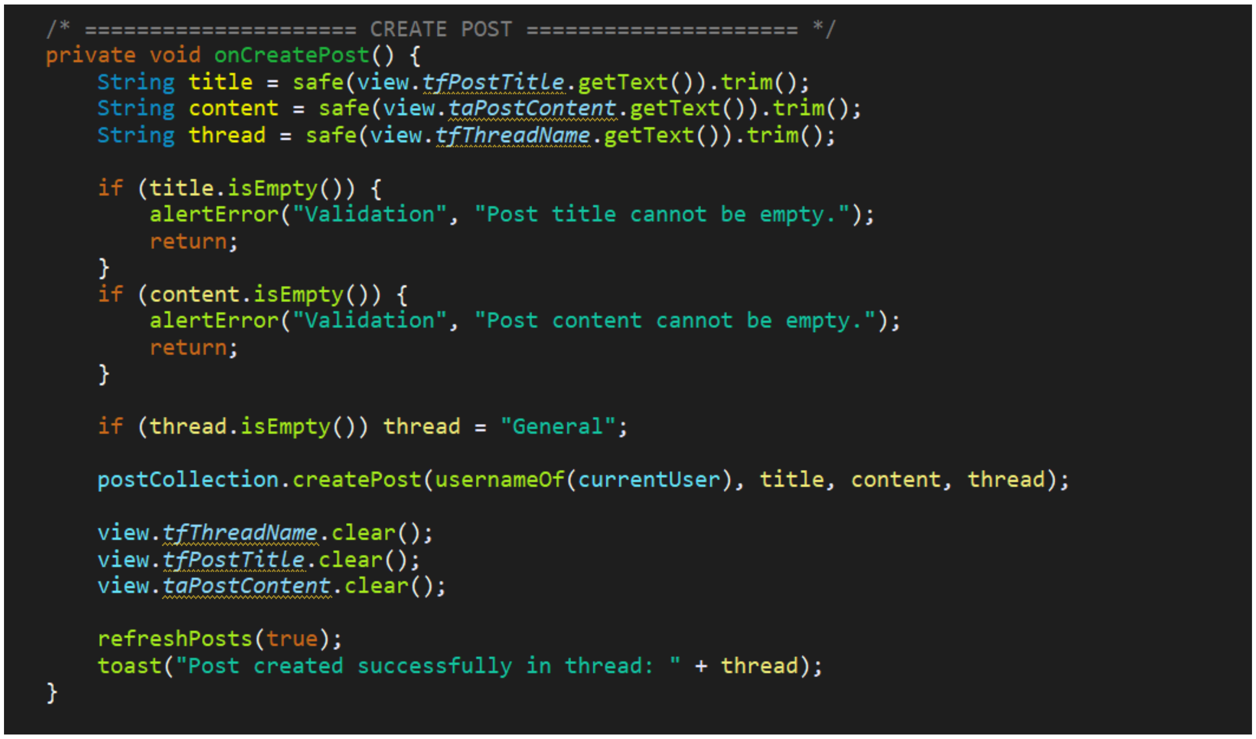
```
/**
 * REQ-UI-MVCInit: Build the scene, then hand control to controller.initialize(this).
 */
public void start(Stage stage, User user) {
    theStage = stage;
    theUser = user;

    controller = new ControllerStudentHome(user);
    theRootPane = buildRootPane();

    Scene scene = new Scene(theRootPane, 950, 700);
    stage.setScene(scene);
    stage.setTitle("Student Home — " + user.getUserName());
    stage.show();

    controller.initialize(this);
}
```

  2. ControllerStudentHome.initialize(...)



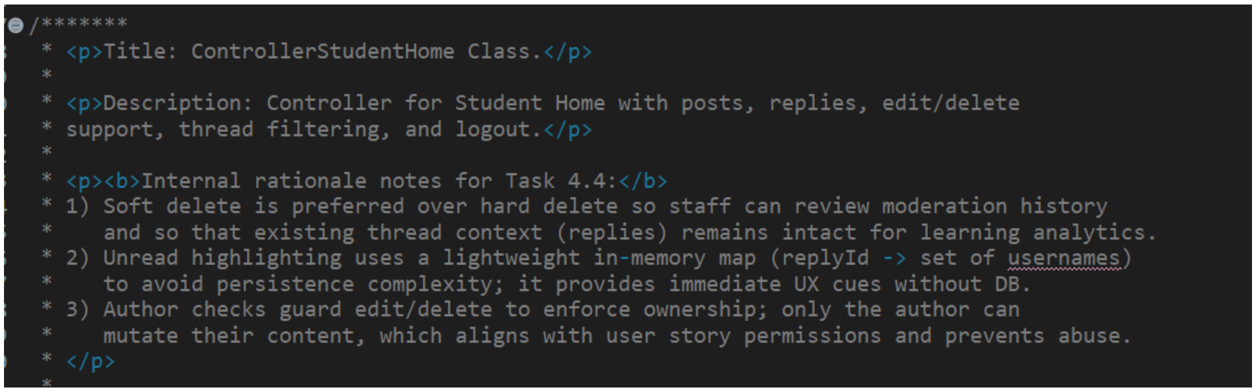
```
/**
 * REQ-UI-MVCInit: Wire all event handlers and perform first render.
 */
public void initialize(ViewStudentHome v) {
    this.view = v;

    // REQ-UI-CreatePost: create new post
    v.btnCreatePost.setOnAction(e -> onCreatePost());

    // REQ-UI-Search: keyword search
    v.btnSearch.setOnAction(e -> onSearch());

    // REQ-UI-SearchReset: show all after clearing search and thread filter
    v.btnShowAll.setOnAction(e -> {
        v.tfSearch.clear();
        v.tfThreadFilter.clear();
        refreshPosts(true);
    });

    // REQ-UI-MyPostsOnly: show only posts authored by current user
    v.cbMyPostsOnly.setOnAction(e -> refreshPosts(false));

    // REQ-UI-Logout: log out of student home
    v.btnLogout.setOnAction(e -> performLogout());

    // REQ-UI-FilterThread: live filter by thread/category
    v.tfThreadFilter.textProperty().addListener((obs, o, n) -> refreshPosts(false));

    refreshPosts(true);
}
```

Figure 4.3 — User Story Coverage (Task 4.3)

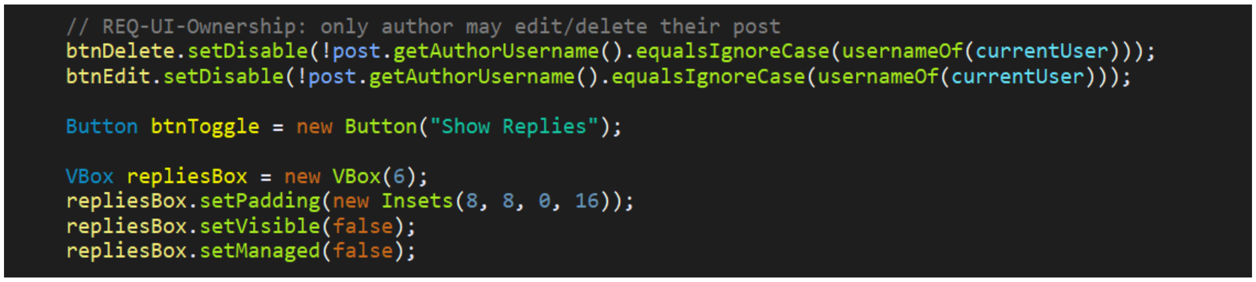


```java
/* ===================== CREATE POST ===================== */
private void onCreatePost() {
    String title = safe(view.tfPostTitle.getText()).trim();
    String content = safe(view.taPostContent.getText()).trim();
    String thread = safe(view.tfThreadName.getText()).trim();

    if (title.isEmpty()) {
        alertError("Validation", "Post title cannot be empty.");
        return;
    }
    if (content.isEmpty()) {
        alertError("Validation", "Post content cannot be empty.");
        return;
    }

    if (thread.isEmpty()) thread = "General";

    postCollection.createPost(usernameOf(currentUser), title, content, thread);

    view.tfThreadName.clear();
    view.tfPostTitle.clear();
    view.taPostContent.clear();

    refreshPosts(true);
    toast("Post created successfully in thread: " + thread);
}
```

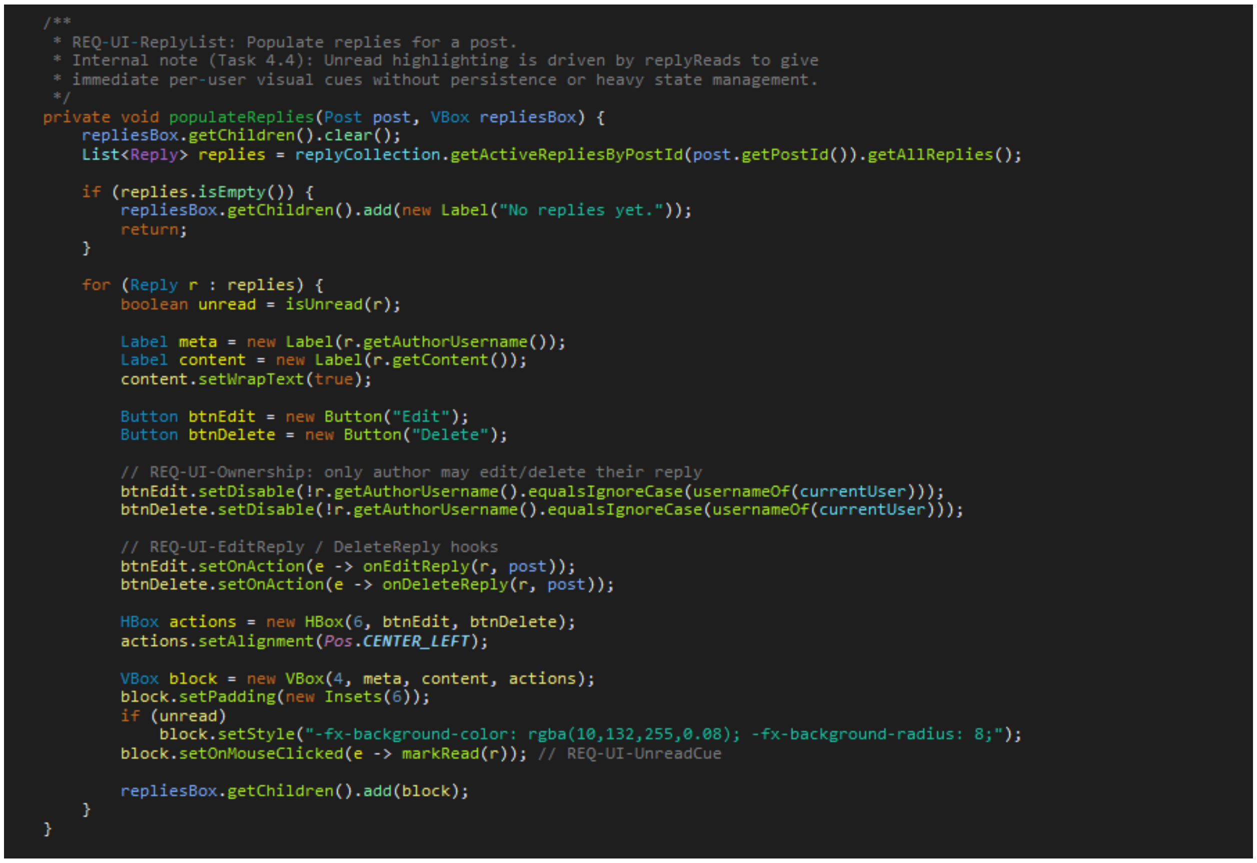
Figure 4.4 — Internal Comments / Rationale (Task 4.4)



```java
/*******
 * <p>Title: ControllerStudentHome Class.</p>
 *
 * <p>Description: Controller for Student Home with posts, replies, edit/delete
 * support, thread filtering, and logout.</p>
 *
 * <p><b>Internal rationale notes for Task 4.4:</b>
 * 1) Soft delete is preferred over hard delete so staff can review moderation history
 *    and so that existing thread context (replies) remains intact for learning analytics.
 * 2) Unread highlighting uses a lightweight in-memory map (replyId -> set of usernames)
 *    to avoid persistence complexity; it provides immediate UX cues without DB.
 * 3) Author checks guard edit/delete to enforce ownership; only the author can
 *    mutate their content, which aligns with user story permissions and prevents abuse.
 * </p>
 *
```

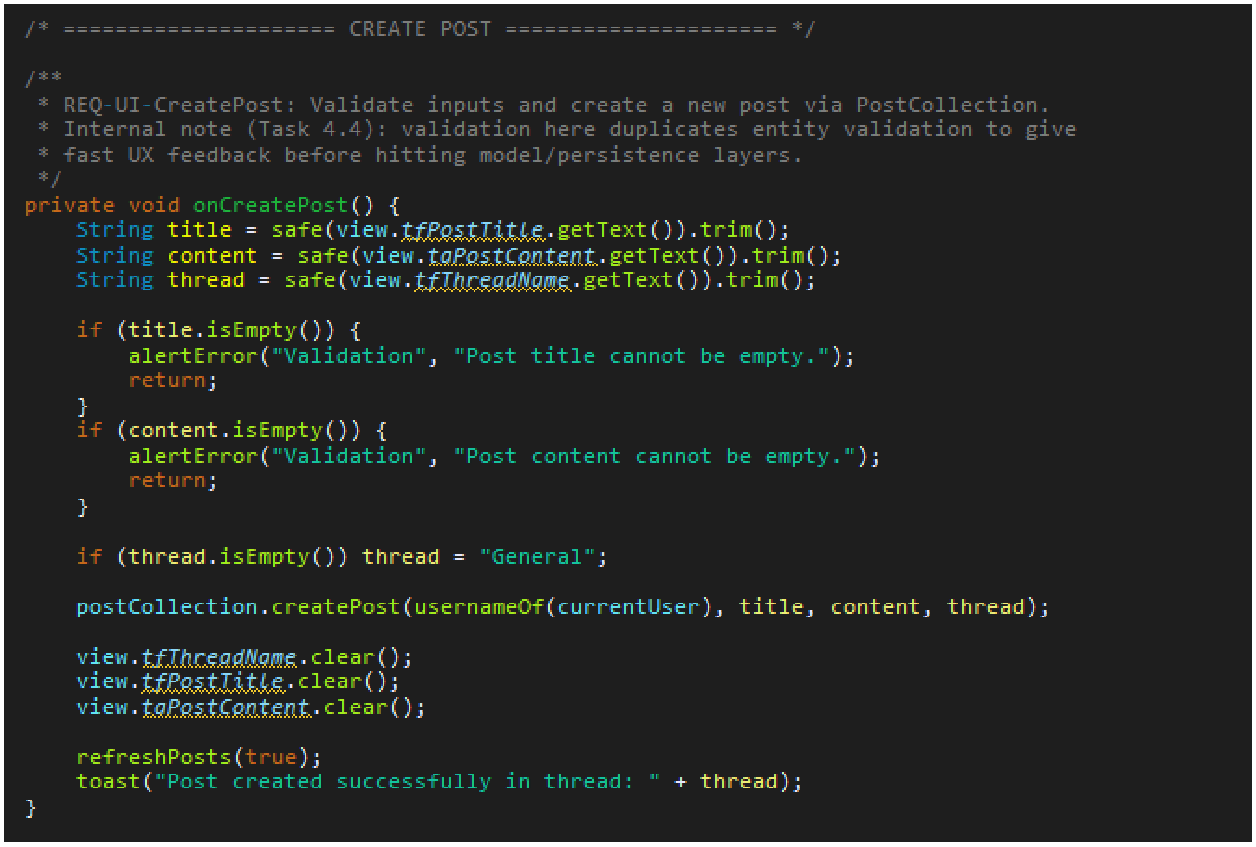


```java
// REQ-UI-Ownership: only author may edit/delete their post
btnDelete.setDisable(!post.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));
btnEdit.setDisable(!post.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));

Button btnToggle = new Button("Show Replies");

VBox repliesBox = new VBox(6);
repliesBox.setPadding(new Insets(8, 8, 0, 16));
repliesBox.setVisible(false);
repliesBox.setManaged(false);
```

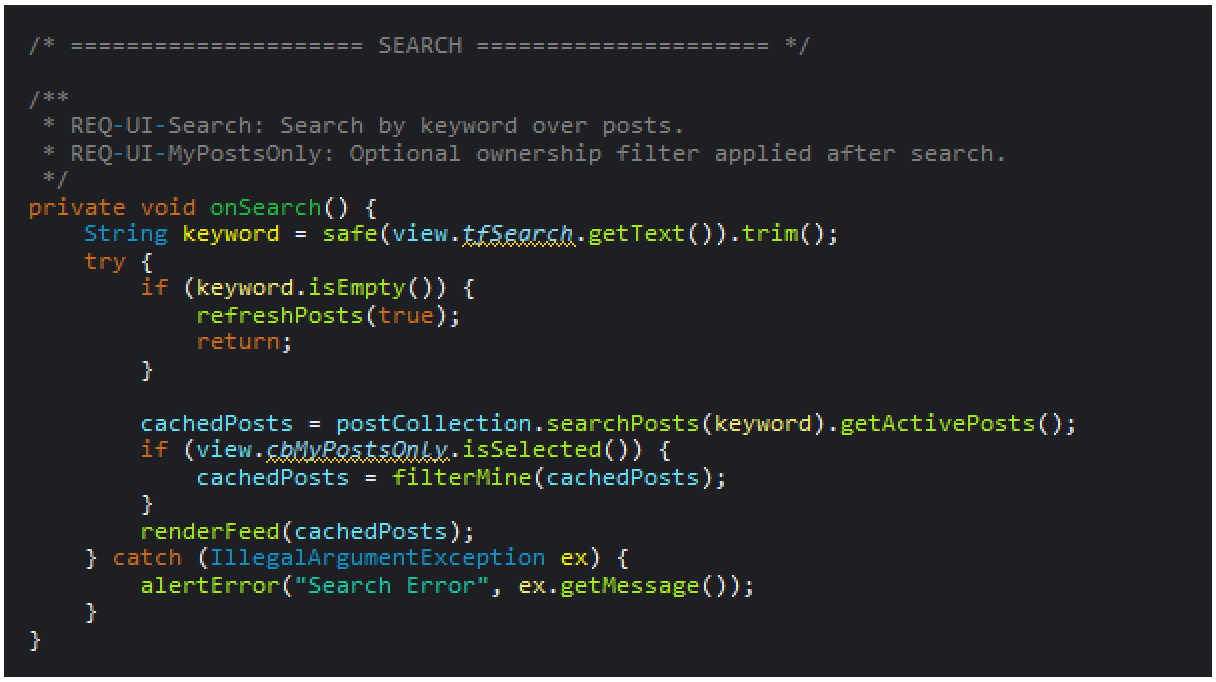


```
/**
 * REQ-UI-ReplyList: Populate replies for a post.
 * Internal note (Task 4.4): Unread highlighting is driven by replyReads to give
 * immediate per-user visual cues without persistence or heavy state management.
 */
private void populateReplies(Post post, VBox repliesBox) {
    repliesBox.getChildren().clear();
    List<Reply> replies = replyCollection.getActiveRepliesByPostId(post.getPostId()).getAllReplies();

    if (replies.isEmpty()) {
        repliesBox.getChildren().add(new Label("No replies yet."));
        return;
    }

    for (Reply r : replies) {
        boolean unread = isUnread(r);

        Label meta = new Label(r.getAuthorUsername());
        Label content = new Label(r.getContent());
        content.setWrapText(true);

        Button btnEdit = new Button("Edit");
        Button btnDelete = new Button("Delete");

        // REQ-UI-Ownership: only author may edit/delete their reply
        btnEdit.setDisable(!r.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));
        btnDelete.setDisable(!r.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));

        // REQ-UI-EditReply / DeleteReply hooks
        btnEdit.setOnAction(e -> onEditReply(r, post));
        btnDelete.setOnAction(e -> onDeleteReply(r, post));

        HBox actions = new HBox(6, btnEdit, btnDelete);
        actions.setAlignment(Pos.CENTER_LEFT);

        VBox block = new VBox(4, meta, content, actions);
        block.setPadding(new Insets(6));
        if (unread)
            block.setStyle("-fx-background-color: rgba(10,132,255,0.08); -fx-background-radius: 8;");
        block.setOnMouseClicked(e -> markRead(r)); // REQ-UI-UnreadCue

        repliesBox.getChildren().add(block);
    }
}
```

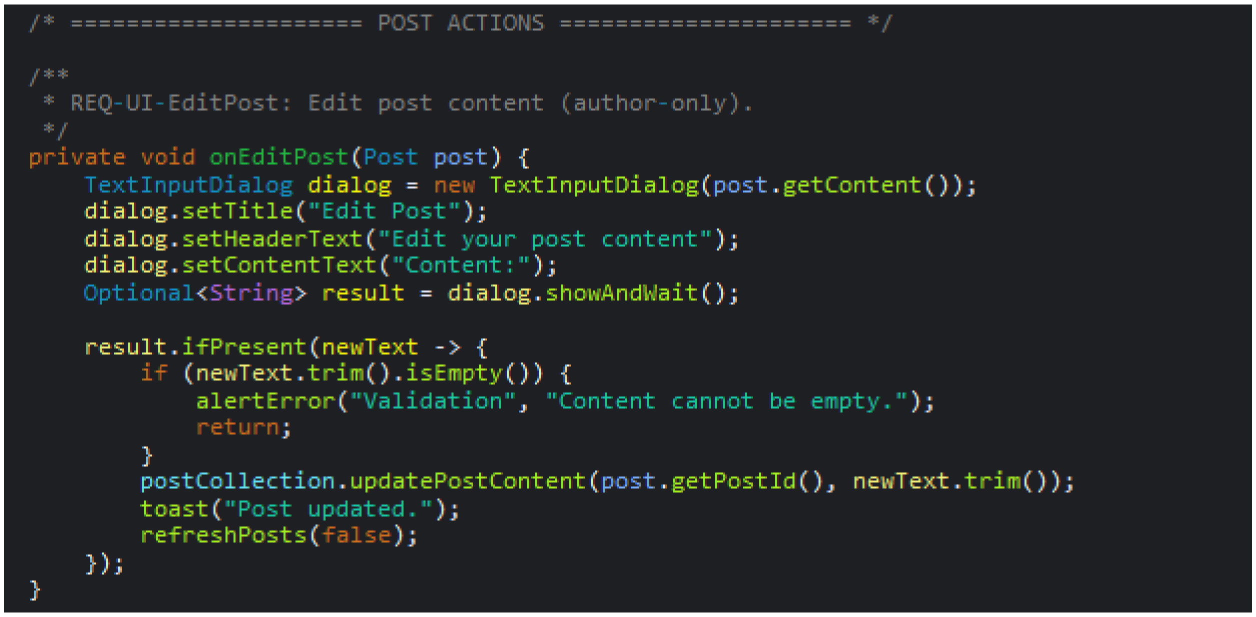
Figure 4.5 — Requirement Tags (Task 4.5)

- 

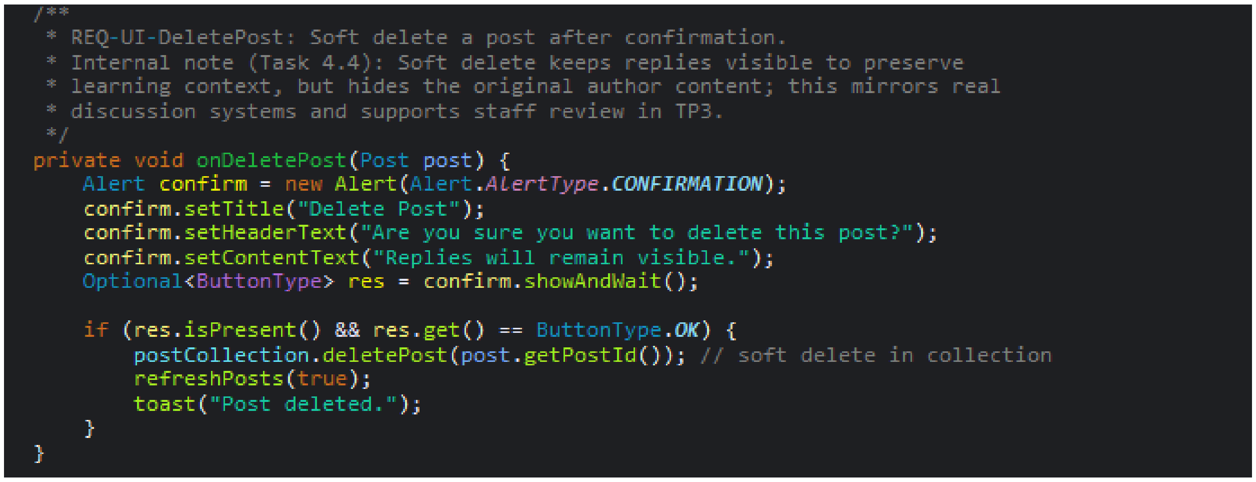
```java
/* ==================== CREATE POST ==================== */

/**
 * REQ-UI-CreatePost: Validate inputs and create a new post via PostCollection.
 * Internal note (Task 4.4): validation here duplicates entity validation to give
 * fast UX feedback before hitting model/persistence layers.
 */
private void onCreatePost() {
    String title = safe(view.tfPostTitle.getText()).trim();
    String content = safe(view.taPostContent.getText()).trim();
    String thread = safe(view.tfThreadName.getText()).trim();

    if (title.isEmpty()) {
        alertError("Validation", "Post title cannot be empty.");
        return;
    }
    if (content.isEmpty()) {
        alertError("Validation", "Post content cannot be empty.");
        return;
    }

    if (thread.isEmpty()) thread = "General";

    postCollection.createPost(usernameOf(currentUser), title, content, thread);

    view.tfThreadName.clear();
    view.tfPostTitle.clear();
    view.taPostContent.clear();

    refreshPosts(true);
    toast("Post created successfully in thread: " + thread);
}
```



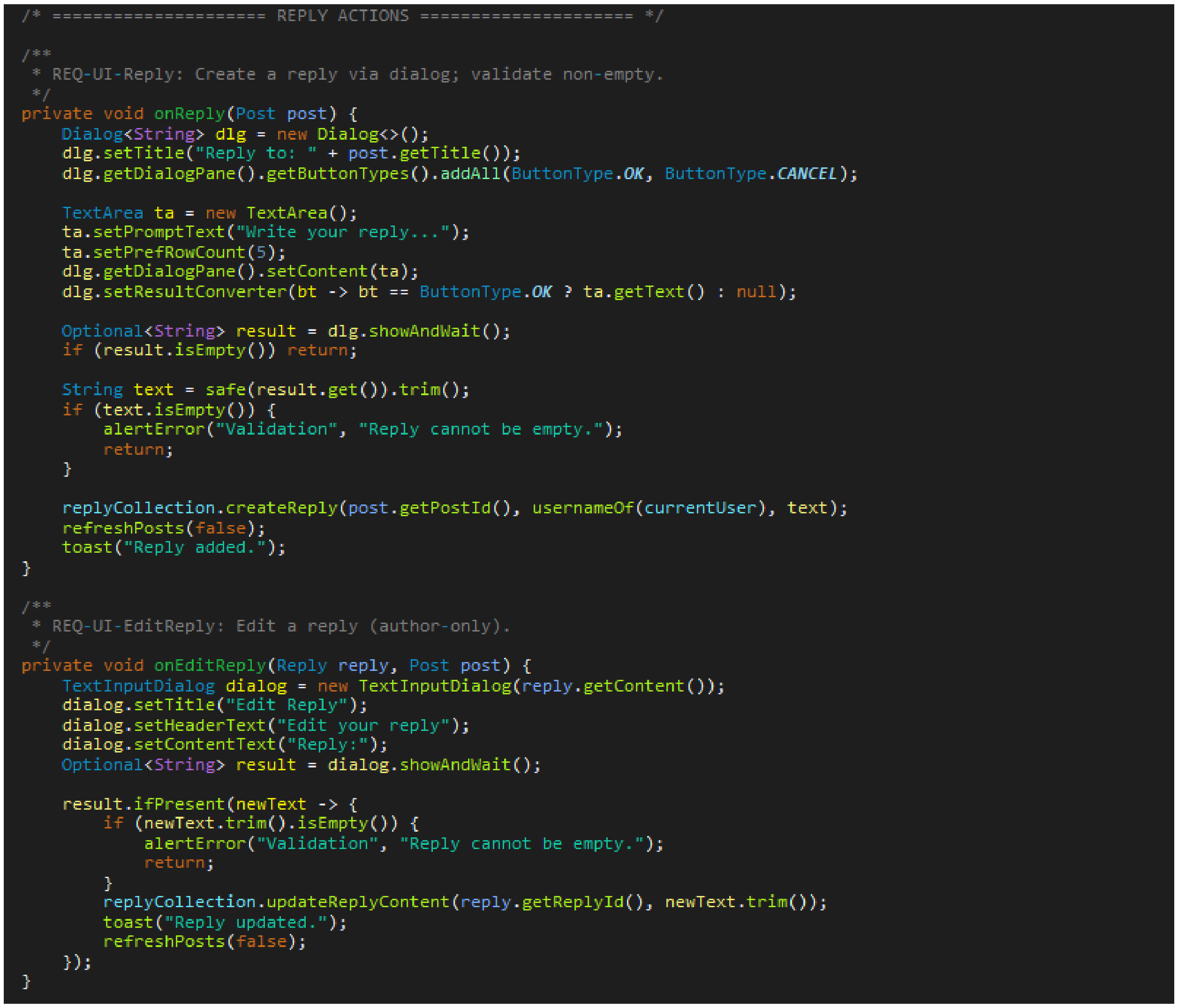
```java
/* ==================== SEARCH ==================== */

/**
 * REQ-UI-Search: Search by keyword over posts.
 * REQ-UI-MyPostsOnly: Optional ownership filter applied after search.
 */
private void onSearch() {
    String keyword = safe(view.tfSearch.getText()).trim();
    try {
        if (keyword.isEmpty()) {
            refreshPosts(true);
            return;
        }

        cachedPosts = postCollection.searchPosts(keyword).getActivePosts();
        if (view.cbMyPostsOnly.isSelected()) {
            cachedPosts = filterMine(cachedPosts);
        }
        renderFeed(cachedPosts);
    } catch (IllegalArgumentException ex) {
        alertError("Search Error", ex.getMessage());
    }
}
```



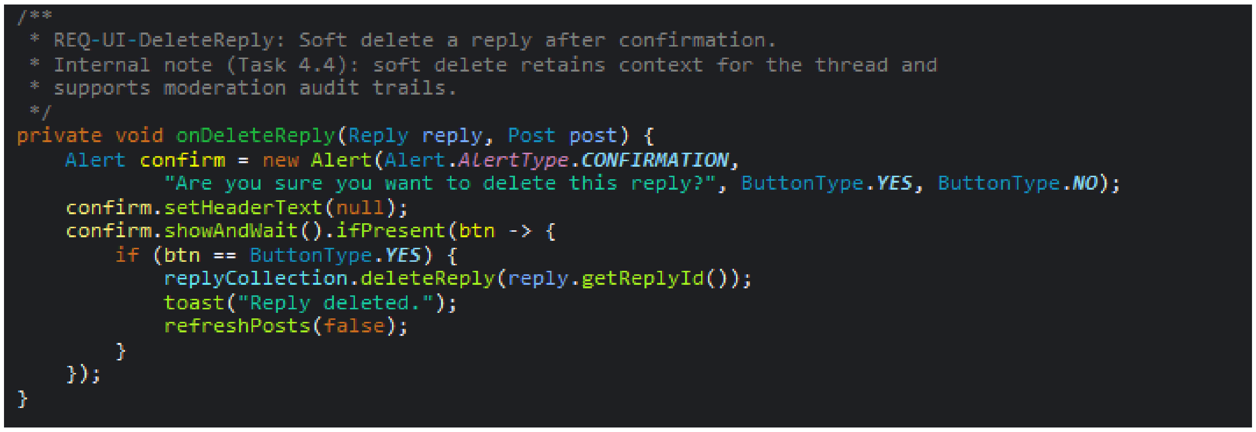
```java
/* ==================== POST ACTIONS ==================== */

/**
 * REQ-UI-EditPost: Edit post content (author-only).
 */
private void onEditPost(Post post) {
    TextInputDialog dialog = new TextInputDialog(post.getContent());
    dialog.setTitle("Edit Post");
    dialog.setHeaderText("Edit your post content");
    dialog.setContentText("Content:");
    Optional<String> result = dialog.showAndWait();

    result.ifPresent(newText -> {
        if (newText.trim().isEmpty()) {
            alertError("Validation", "Content cannot be empty.");
            return;
        }
        postCollection.updatePostContent(post.getPostId(), newText.trim());
        toast("Post updated.");
        refreshPosts(false);
    });
}
```



```java
/**
 * REQ-UI-DeletePost: Soft delete a post after confirmation.
 * Internal note (Task 4.4): Soft delete keeps replies visible to preserve
 * learning context, but hides the original author content; this mirrors real
 * discussion systems and supports staff review in TP3.
 */
private void onDeletePost(Post post) {
    Alert confirm = new Alert(Alert.AlertType.CONFIRMATION);
    confirm.setTitle("Delete Post");
    confirm.setHeaderText("Are you sure you want to delete this post?");
    confirm.setContentText("Replies will remain visible.");
    Optional<ButtonType> res = confirm.showAndWait();

    if (res.isPresent() && res.get() == ButtonType.OK) {
        postCollection.deletePost(post.getPostId()); // soft delete in collection
        refreshPosts(true);
        toast("Post deleted.");
    }
}
```

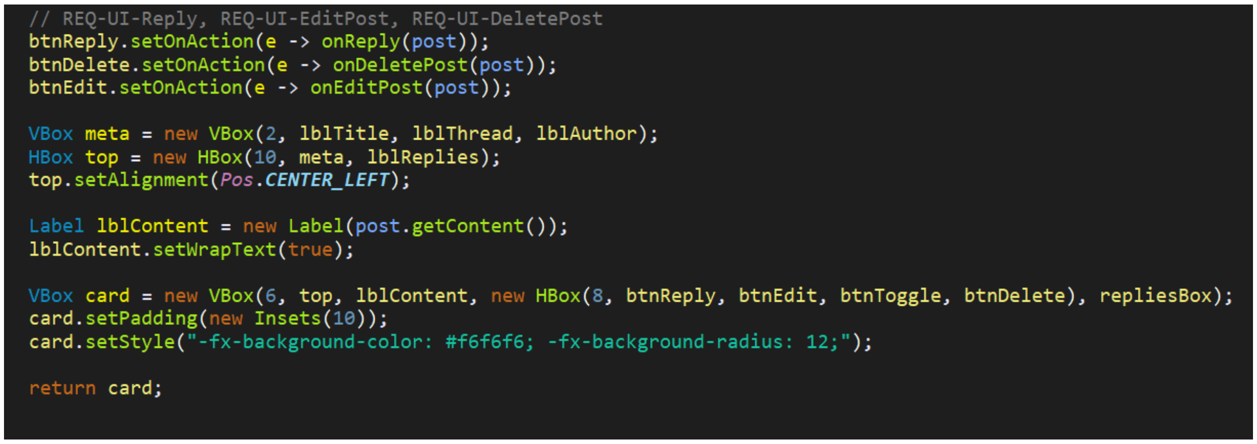


```
/* ==================== REPLY ACTIONS ==================== */

/**
 * REQ-UI-Reply: Create a reply via dialog; validate non-empty.
 */
private void onReply(Post post) {
    Dialog<String> dlg = new Dialog<>();
    dlg.setTitle("Reply to: " + post.getTitle());
    dlg.getDialogPane().getButtonTypes().addAll(ButtonType.OK, ButtonType.CANCEL);

    TextArea ta = new TextArea();
    ta.setPromptText("Write your reply...");
    ta.setPrefRowCount(5);
    dlg.getDialogPane().setContent(ta);
    dlg.setResultConverter(bt -> bt == ButtonType.OK ? ta.getText() : null);

    Optional<String> result = dlg.showAndWait();
    if (result.isEmpty()) return;

    String text = safe(result.get()).trim();
    if (text.isEmpty()) {
        alertError("Validation", "Reply cannot be empty.");
        return;
    }

    replyCollection.createReply(post.getPostId(), usernameOf(currentUser), text);
    refreshPosts(false);
    toast("Reply added.");
}

/**
 * REQ-UI-EditReply: Edit a reply (author-only).
 */
private void onEditReply(Reply reply, Post post) {
    TextInputDialog dialog = new TextInputDialog(reply.getContent());
    dialog.setTitle("Edit Reply");
    dialog.setHeaderText("Edit your reply");
    dialog.setContentText("Reply:");
    Optional<String> result = dialog.showAndWait();

    result.ifPresent(newText -> {
        if (newText.trim().isEmpty()) {
            alertError("Validation", "Reply cannot be empty.");
            return;
        }
        replyCollection.updateReplyContent(reply.getReplyId(), newText.trim());
        toast("Reply updated.");
        refreshPosts(false);
    });
}
```



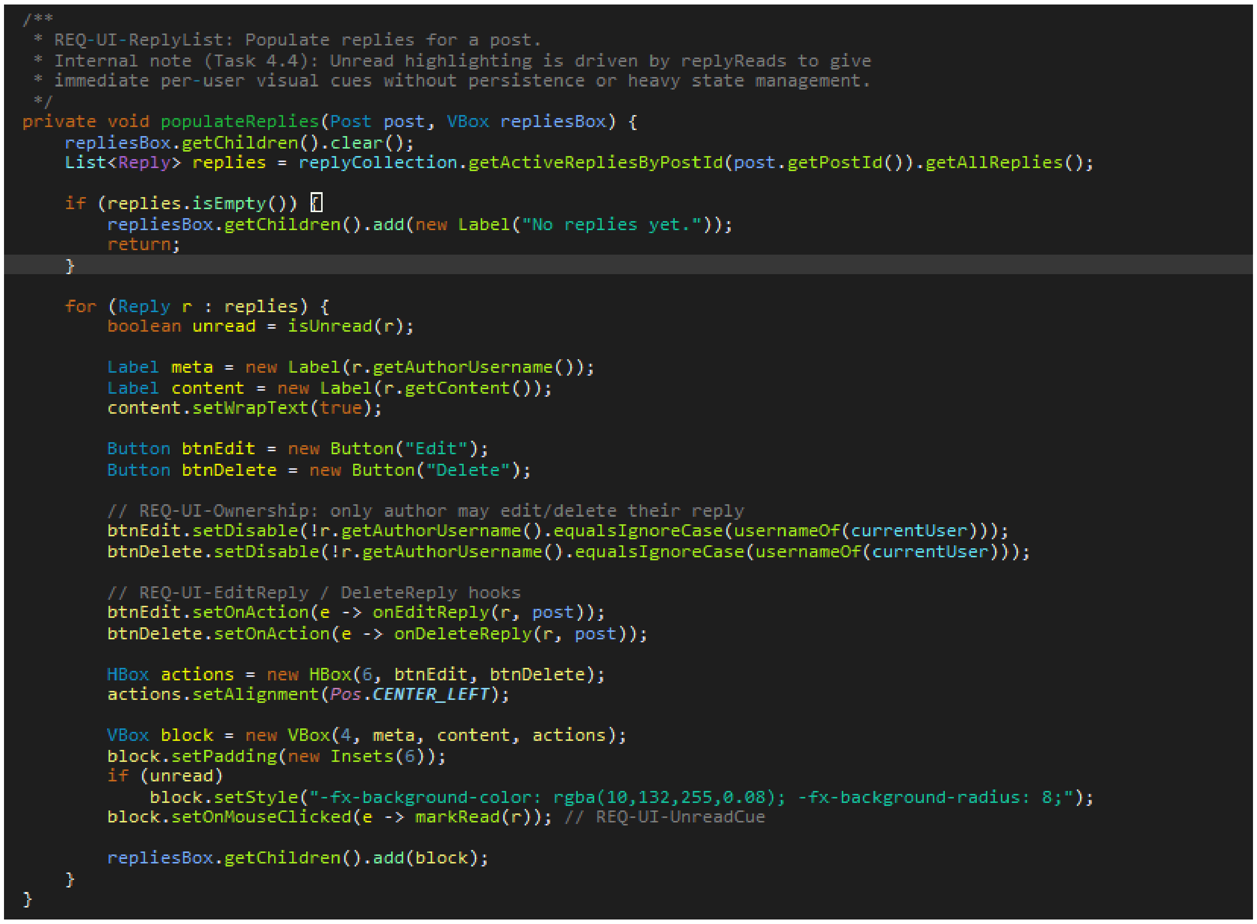
```
/**
 * REQ-UI-DeleteReply: Soft delete a reply after confirmation.
 * Internal note (Task 4.4): soft delete retains context for the thread and
 * supports moderation audit trails.
 */
private void onDeleteReply(Reply reply, Post post) {
    Alert confirm = new Alert(Alert.AlertType.CONFIRMATION,
            "Are you sure you want to delete this reply?", ButtonType.YES, ButtonType.NO);
    confirm.setHeaderText(null);
    confirm.showAndWait().ifPresent(btn -> {
        if (btn == ButtonType.YES) {
            replyCollection.deleteReply(reply.getReplyId());
            toast("Reply deleted.");
            refreshPosts(false);
        }
    });
}
```

- **Figure 4.6 — Reply Flow:**



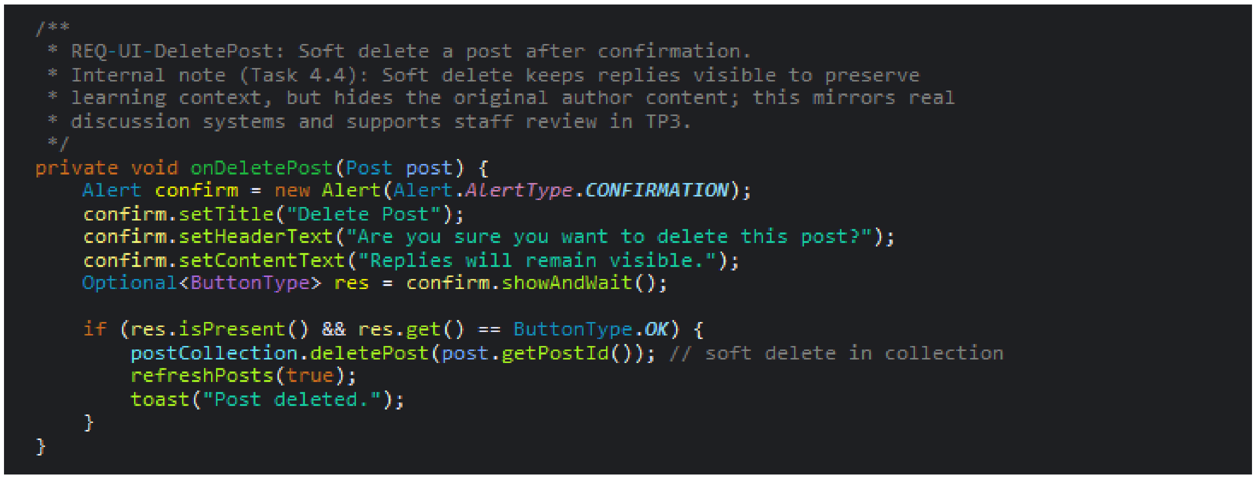
```
// REQ-UI-Reply, REQ-UI-EditPost, REQ-UI-DeletePost
btnReply.setOnAction(e -> onReply(post));
btnDelete.setOnAction(e -> onDeletePost(post));
btnEdit.setOnAction(e -> onEditPost(post));

VBox meta = new VBox(2, lblTitle, lblThread, lblAuthor);
HBox top = new HBox(10, meta, lblReplies);
top.setAlignment(Pos.CENTER_LEFT);

Label lblContent = new Label(post.getContent());
lblContent.setWrapText(true);

VBox card = new VBox(6, top, lblContent, new HBox(8, btnReply, btnEdit, btnToggle, btnDelete), repliesBox);
card.setPadding(new Insets(10));
card.setStyle("-fx-background-color: #f6f6f6; -fx-background-radius: 12;");

return card;
```

-

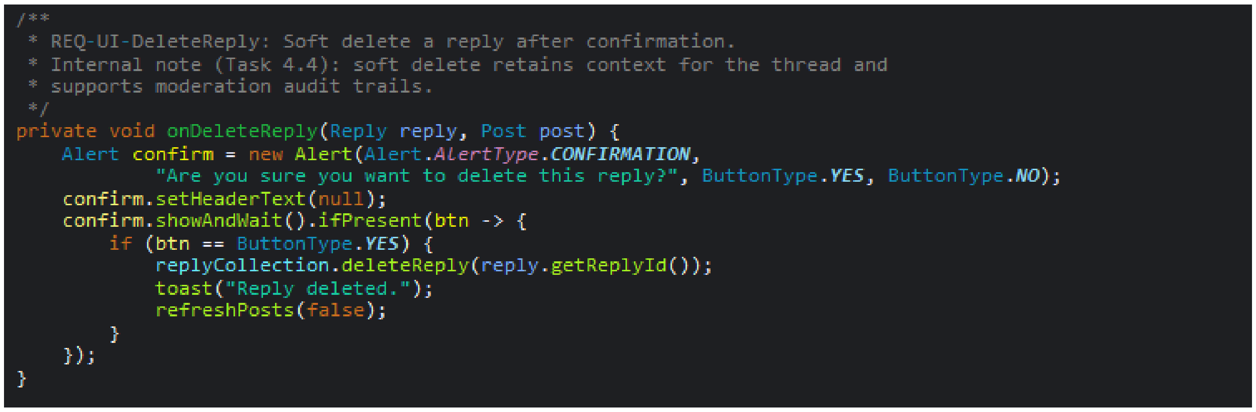


```
/**
 * REQ-UI-ReplyList: Populate replies for a post.
 * Internal note (Task 4.4): Unread highlighting is driven by replyReads to give
 * immediate per-user visual cues without persistence or heavy state management.
 */
private void populateReplies(Post post, VBox repliesBox) {
    repliesBox.getChildren().clear();
    List<Reply> replies = replyCollection.getActiveRepliesByPostId(post.getPostId()).getAllReplies();

    if (replies.isEmpty()) {
        repliesBox.getChildren().add(new Label("No replies yet."));
        return;
    }

    for (Reply r : replies) {
        boolean unread = isUnread(r);

        Label meta = new Label(r.getAuthorUsername());
        Label content = new Label(r.getContent());
        content.setWrapText(true);

        Button btnEdit = new Button("Edit");
        Button btnDelete = new Button("Delete");

        // REQ-UI-Ownership: only author may edit/delete their reply
        btnEdit.setDisable(!r.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));
        btnDelete.setDisable(!r.getAuthorUsername().equalsIgnoreCase(usernameOf(currentUser)));

        // REQ-UI-EditReply / DeleteReply hooks
        btnEdit.setOnAction(e -> onEditReply(r, post));
        btnDelete.setOnAction(e -> onDeleteReply(r, post));

        HBox actions = new HBox(6, btnEdit, btnDelete);
        actions.setAlignment(Pos.CENTER_LEFT);

        VBox block = new VBox(4, meta, content, actions);
        block.setPadding(new Insets(6));
        if (unread)
            block.setStyle("-fx-background-color: rgba(10,132,255,0.08); -fx-background-radius: 8;");
        block.setOnMouseClicked(e -> markRead(r)); // REQ-UI-UnreadCue

        repliesBox.getChildren().add(block);
    }
}
```

- **Figure 4.7 — Delete Confirm:** the confirmation dialogs for post/reply deletion.



```
/**
 * REQ-UI-DeletePost: Soft delete a post after confirmation.
 * Internal note (Task 4.4): Soft delete keeps replies visible to preserve
 * learning context, but hides the original author content; this mirrors real
 * discussion systems and supports staff review in TP3.
 */
private void onDeletePost(Post post) {
    Alert confirm = new Alert(Alert.AlertType.CONFIRMATION);
    confirm.setTitle("Delete Post");
    confirm.setHeaderText("Are you sure you want to delete this post?");
    confirm.setContentText("Replies will remain visible.");
    Optional<ButtonType> res = confirm.showAndWait();

    if (res.isPresent() && res.get() == ButtonType.OK) {
        postCollection.deletePost(post.getPostId()); // soft delete in collection
        refreshPosts(true);
        toast("Post deleted.");
    }
}
```



```
/**
 * REQ-UI-DeleteReply: Soft delete a reply after confirmation.
 * Internal note (Task 4.4): soft delete retains context for the thread and
 * supports moderation audit trails.
 */
private void onDeleteReply(Reply reply, Post post) {
    Alert confirm = new Alert(Alert.AlertType.CONFIRMATION,
            "Are you sure you want to delete this reply?", ButtonType.YES, ButtonType.NO);
    confirm.setHeaderText(null);
    confirm.showAndWait().ifPresent(btn -> {
        if (btn == ButtonType.YES) {
            replyCollection.deleteReply(reply.getReplyId());
            toast("Reply deleted.");
            refreshPosts(false);
        }
    });
}
```

---

8) Conclusion

The guiStudentHome MVC set:

- Introduces **properly named** MVC packages/classes (4.1),
- Demonstrates **MVC alignment** with the provided GUI pattern (4.2),
- **Covers all Students User Stories** with visible UI flows (4.3),
- Provides **Javadoc and internal comments** detailing purposes, attributes, operations, and rationale (4.4), and
- Includes **requirement-linked screenshots** that clarify which lines of code satisfy which requirement (4.5).