**WEEK 4**

* **Class Diagram with Applied Design Patterns :**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

* **Explanation of Responsibilities per Class :**

**1.User**:

* **Responsibilities**:
  + Manages user account creation (register), authentication (login), and profile updates (updateProfile).
  + Stores user data (ID, username, email, password) in the SQLite database.
  + Acts as the central entity for interactions like creating posts, comments, and receiving notifications.
* **Purpose**: Represents a student using the app, enabling personalized features like posting and messaging.

**2.** **Post**:

* **Responsibilities**:
  + Handles creation (createPost) and deletion (deletePost) of posts.
  + Retrieves associated comments (getComments).
  + Stores post data (content, timestamp, user ID) in SQLite.
* **Purpose**: Represents user-generated content (e.g., study tips, event announcements) to foster collaboration.

**3.** **Comment**:

* **Responsibilities**:
  + Manages adding (addComment) and removing (deleteComment) comments on posts.
  + Stores comment data (content, timestamp, user ID, post ID) in SQLite.
* **Purpose**: Enables threaded discussions to enhance student interaction.

**4.Notification**:

* **Responsibilities**:
  + Sends notifications to users (sendNotification) for events like new comments or mentions.
  + Stores notification data (message, timestamp, user ID) in SQLite.
* **Purpose**: Keeps students informed of relevant activities, supporting real-time engagement.

**5. DatabaseConnection (Singleton)**:

* **Responsibilities**:
  + Ensures a single SQLite database connection instance (getInstance) across the app.
  + Executes queries (query) to interact with SQLite.
  + Private constructor prevents multiple instances.
* **Purpose**: Optimizes resource usage by maintaining one database connection, critical for SQLite’s file-based nature.

**6.ContentFactory (Factory Interface)**:

* **Responsibilities**:
  + Defines a method (createContent) for creating content objects (posts, comments).
* **Purpose**: Abstracts content creation, allowing flexibility in adding new content types (e.g., polls) without modifying client code.

**7.PostFactory (Factory)**:

* **Responsibilities**:
  + Implements createContent to instantiate Post objects.
* **Purpose**: Encapsulates post creation logic, adhering to the Factory pattern for modularity.

**8.** **CommentFactory (Factory)**:

* **Responsibilities**:
  + Implements createContent to instantiate Comment objects.
* **Purpose**: Encapsulates comment creation, promoting code reuse and scalability.

**9**.**Content (Interface)**:

* **Responsibilities**:
  + Defines a save method for content objects (posts, comments) to persist to SQLite.
* **Purpose**: Ensures a common interface for content types, supporting Factory pattern extensibility.

**10.Observer (Interface)**:

* **Responsibilities**:
  + Defines an update method for observers to handle notifications.
* **Purpose**: Part of the Observer pattern, enabling loose coupling between subjects and observers.

**11.Subject (Interface)**:

* **Responsibilities**:
  + Defines methods for managing observers (attach, detach, notify).
* **Purpose**: Allows subjects to notify observers of state changes, used for notifications.

**12.NotificationSubject (Observer)**:

* **Responsibilities**:
  + Maintains a list of observers and notifies them (notify) when a new notification occurs.
  + Manages observer subscriptions (attach, detach).
* **Purpose**: Implements the Subject role in the Observer pattern, coordinating real-time notifications.

**13.UserObserver (Observer)**:

* **Responsibilities**:
  + Receives notifications via update and delivers them to the associated User.
* **Purpose**: Acts as an observer for a specific user, ensuring they receive relevant notifications.

**Relationships Between Classes**

**1.User → Post (1-to-many)**:

* A User can create multiple Post objects (association).
* Represented by a solid line with an arrow from User to Post.
* **Purpose**: Links users to their posts, enabling content attribution.

**2.** **User → Comment (1-to-many)**:

* A User can create multiple Comment objects.
* Solid line with an arrow from User to Comment.
* **Purpose**: Associates comments with their authors.

**3.** **Post → Comment (1-to-many)**:

* A Post can have multiple Comment objects.
* Solid line with an arrow from Post to Comment.
* **Purpose**: Represents the hierarchical structure of posts and their comments.

**4.** **User → Notification (1-to-many)**:

* A User can receive multiple Notification objects.
* Solid line with an arrow from User to Notification.
* **Purpose**: Ensures users receive relevant alerts (e.g., new comment on their post).

**5**. **NotificationSubject → Notification (Association)**:

* NotificationSubject triggers Notification objects to be sent.
* Solid line from NotificationSubject to Notification.
* **Purpose**: Part of the Observer pattern, where the subject initiates notifications.

**6.NotificationSubject ↔ UserObserver (Composition, Observer Pattern)**:

* NotificationSubject maintains a list of UserObserver objects (composition, shown as a diamond-ended line).
* UserObserver implements the Observer interface, receiving updates from NotificationSubject.
* **Purpose**: Enables real-time notifications using the Observer pattern, e.g., notifying users of new comments.

**7.** **UserObserver → User (Association)**:

* Each UserObserver is tied to a specific User to deliver notifications.
* Solid line from UserObserver to User.
* **Purpose**: Ensures notifications reach the correct user.

**8.** **PostFactory, CommentFactory → ContentFactory (Inheritance)**:

* PostFactory and CommentFactory implement the ContentFactory interface.
* Dashed line with an arrow (generalization) from factories to ContentFactory.
* **Purpose**: Factory pattern allows polymorphic creation of content objects.

**9.Post, Comment → Content (Implementation)**:

* Post and Comment implement the Content interface.
* Dashed line with an arrow from Post and Comment to Content.
* **Purpose**: Ensures posts and comments share a common interface for saving to SQLite.

**10.DatabaseConnection → SQLite (Dependency)**:

* DatabaseConnection uses SQLite for querying (via the query method).
* Dashed line from DatabaseConnection to SQLite.
* **Purpose**: Singleton pattern ensures a single connection to SQLite, optimizing database access.