**1. Design for class “LoginScreen”**

A screenshot of a computer program

AI-generated content may be incorrect.

1.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Attribute Name | Data Type | Default Value | Description |
| 1 | username | string | None | Stores the username input for login |
| 2 | password | int | None | Stores the password input for login |

1.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| # | Name | Return type | Description |
| 1 | notifySuccess() | void | Notifies the user when login is successful |
| 2 | notifyFail() | void | Notifies the user when login fails |
| 3 | submitLogin(loginController: LoginController) | void | Submits login credentials for authentication |

1.3. Parameters

* loginController: The controller handling authentication logic and verification.

1.4. Exceptions

* InvalidCredentialsException if the username or password is incorrect.
* UserNotFoundException if the entered username does not exist.

1.5. How to Use Parameters / Attributes:

* username: Input field where users enter their username.
* password: Input field where users enter their password.
* loginController: Handles the authentication process.

1.6. Flow

1. The user enters a username and password.
2. The user submits the login request (submitLogin).
3. The system validates credentials using loginController.
4. If authentication is successful, notifySuccess is triggered.
5. If authentication fails, notifyFail is triggered.

**2. Design for class “LoginController”**

A yellow rectangular object with black text

AI-generated content may be incorrect.

2.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| # | Attribute Name | Data Type | Default Value | Description |
| 1 | None (stateless) | N/A | N/A | The LoginController does not store user state; it handles authentication logic. |

2.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Method Name** | **Return Type** | **Description** |
| 1 | getAccount(username: string) | Account | Retrieves account details associated with the given username. |
| 2 | authenticate(username: string, password: string) | boolean | Verifies if the provided credentials match a registered user. |

2.3. Parameters

* username: The username provided by the user attempting to log in.
* password: The password associated with the given username.

2.4. Exceptions

* UserNotFoundException if the username does not exist in the system.
* InvalidPasswordException if the password does not match the stored credentials.

2.5. How to Use Parameters / Attributes:

* username: Used to locate the corresponding account in the system.
* password: Checked against the stored credentials for authentication.

2.6. Flow

1. **Retrieving Account Information (getAccount)**
   * The system searches for an account based on username.
   * If found, it returns the Account object.
   * If not found, throws UserNotFoundException.
2. **Authentication Process (authenticate)**
   * The user enters username and password.
   * The system retrieves the account using getAccount().
   * It verifies the password against stored credentials.
   * If valid, returns true (authentication successful).
   * If invalid, returns false (authentication failed).

**3. Design for class “Account”**

**A yellow box with black text

AI-generated content may be incorrect.**

3.1. Attribute design example

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **#** | **Attribute Name** | **Data Type** | **Default Value** | **Description** |
| 1 | userId | string | None | Unique identifier for the user account. |
| 2 | username | string | None | The name used by the user for login. |
| 3 | password | string | None | Hashed password for authentication. |

3.2. Operation Design example

|  |  |  |  |
| --- | --- | --- | --- |
| **#** | **Method Name** | **Return Type** | **Description** |
| 1 | getUserId() | string | Retrieves the unique user ID. |
| 2 | getUsername() | string | Retrieves the username. |
| 3 | getPassword() | string | Retrieves the stored hashed password. |
| 4 | setUsername(username: string) | void | Updates the username. |
| 5 | setPassword(password: string) | void | Updates the password with a new hashed password. |
| 6 | verifyPassword(inputPassword: string) | boolean | Compares the input password with the stored hashed password and returns true if they match. |

3.3. Parameters

* username: The username used for authentication.
* password: The encrypted password stored securely.
* inputPassword: The password entered by the user for verification.

3.4. Exceptions

* InvalidPasswordException if password verification fails.
* UserNotFoundException if the user ID does not exist.

3.5. How to Use Parameters / Attributes:

* username: Used for login identification.
* password: Used to authenticate users securely.
* inputPassword: Used in the verifyPassword() function to check login credentials.

3.6. Flow