User Authentication and Signup

Description

This application facilitates user authentication and account management by allowing users to log in and sign up. It employs the Singleton design pattern to ensure a single instance manages all authentication processes, providing consistency and efficiency in user data handling.

Design Pattern Used: Singleton Pattern

- **Purpose**: The Singleton pattern guarantees that only one instance of the UserAuthentication class exists throughout the application. This instance manages user credentials and authentication processes.
- **Implementation**: The UserAuthentication class is implemented with a private static instance and a public static method (getInstance()) for accessing it. This ensures only one instance is created.
- **Why Used**:
    - **Consistency**: Ensures a single, consistent point of access for user authentication across the application.
    - **Resource Management**: Optimizes resource usage by maintaining a single instance for user data and file operations.
    - **Global Access**: Provides a centralized and globally accessible instance for authentication functionality.
    - **Prevents Duplication**: Avoids issues related to multiple instances managing user data, ensuring data integrity and synchronization.

Process Flow:

1. **Login Attempt**:
    - Users enter their username and password.
    - If credentials match, login is successful.
    - If the username exists but the password is incorrect, the user is informed of the incorrect password.
    - If the username does not exist, the user is prompted to sign up.
2. **Signup Process**:
    - If the user opts to sign up, new credentials are added to the system.
    - The user is informed of successful signup and prompted to log in again.
3. **Exit**:
    - If the user declines to sign up, the application exits with a goodbye message.

Benefits:

- **Centralized Management**: Ensures consistent user authentication management.
- **Efficiency**: Reduces overhead by maintaining a single instance.