EGE UNIVERSITY

FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

ADVANCED OBJECT ORIENTED PROGRAMMING

2023–2024 SPRING SEMESTER

PROJECT REPORT

DELIVERY DATE

23/05/2024

PREPARED BY

05220000295,ALTUĞ EMRE TOSUN

05220000316,FURKAN İYEM

05220000380,YAREN KILIÇ

05210000266,GÜLNAZ KARACA

Table of Contents page

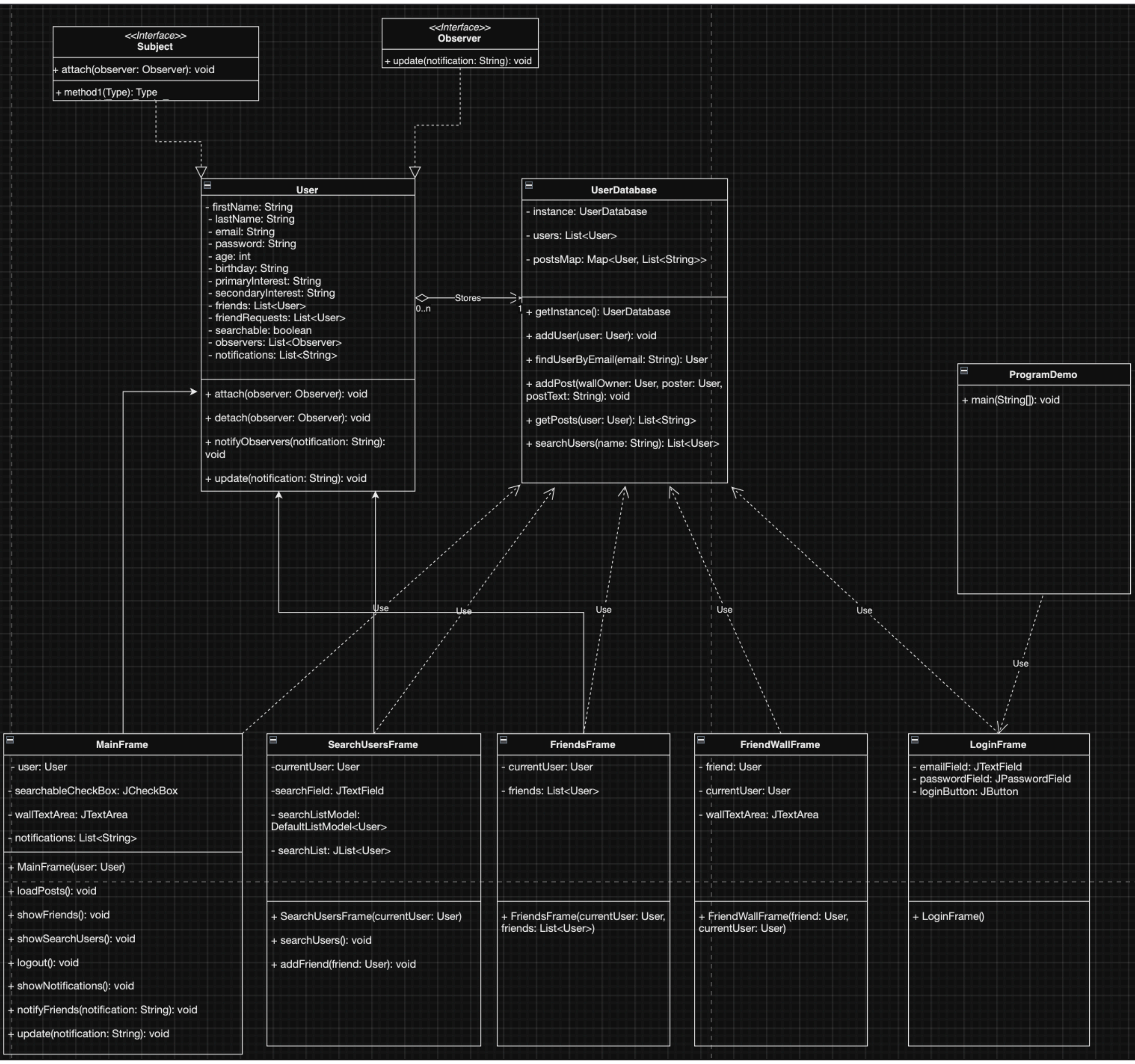
PROJECT IN SHORT

A simple Facebook-like social network software was designed and implemented using design patterns and the Java language on a single machine.

Social networks are social structures consisting of individuals connected to each other through one or more types of relationships (such as friendship, common interests, family relationships, etc.) and individual constraints defined in context (such as allowing or disallowing anyone to add someone as a friend, etc.).

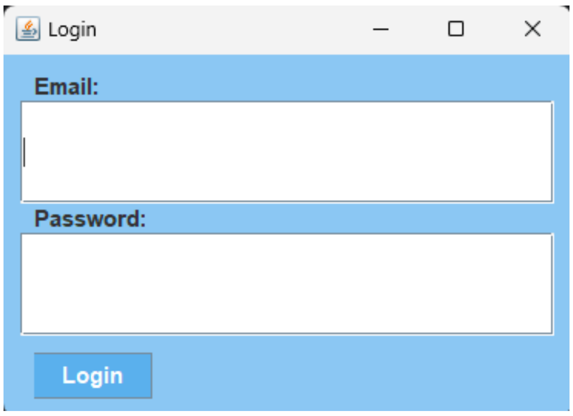
In this context, the social network software we developed allows its members to be connected to each other through various relationships, primarily friendship, and enables members to search for and find each other. Additionally, members who do not wish to appear in searches can define this as a constraint. Each member can have a wall where they can share links and/or textual information.

UML CLASS DIAGRAM

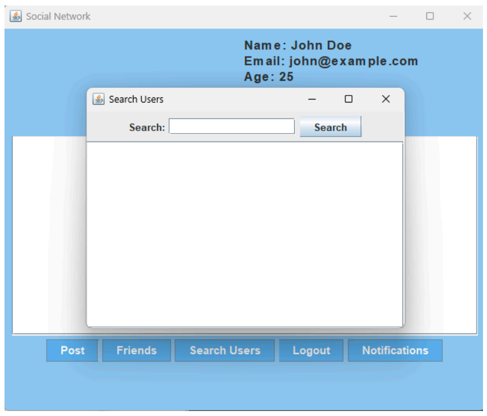

<<Interface>> Subject
+ attach(observer: Observer): void
+ method1(Type): Type
<<Interface>> Observer
+ update(notification: String): void
User
- firstName: String
- lastName: String
- email: String
- password: String
- age: int
- birthday: String
- primaryInterest: String
- secondaryInterest: String
- friends: List<User>
- friendRequests: List<User>
- searchable: boolean
- observers: List<Observer>
- notifications: List<String>
+ attach(observer: Observer): void
+ detach(observer: Observer): void
+ notifyObservers(notification: String): void
+ update(notification: String): void
Stores
0..n
1
UserDatabase
- instance: UserDatabase
- users: List<User>
- postsMap: Map<User, List<String>>
+ getInstance(): UserDatabase
+ addUser(user: User): void
+ findUserByEmail(email: String): User
+ addPost(wallOwner: User, poster: User, postText: String): void
+ getPosts(user: User): List<String>
+ searchUsers(name: String): List<User>
ProgramDemo
+ main(String[]): void
Use
Use
Use
Use
Use
Use
MainFrame
- user: User
- searchableCheckBox: JCheckBox
- wallTextArea: JTextArea
- notifications: List<String>
+ MainFrame(user: User)
+ loadPosts(): void
+ showFriends(): void
+ showSearchUsers(): void
+ logout(): void
+ showNotifications(): void
+ notifyFriends(notification: String): void
+ update(notification: String): void
SearchUsersFrame
-currentUser: User
-searchField: JTextField
- searchListModel: DefaultListModel<User>
- searchList: JList<User>
+ SearchUsersFrame(currentUser: User)
+ searchUsers(): void
+ addFriend(friend: User): void
FriendsFrame
- currentUser: User
- friends: List<User>
+ FriendsFrame(currentUser: User, friends: List<User>)
FriendWallFrame
- friend: User
- currentUser: User
- wallTextArea: JTextArea
+ FriendWallFrame(friend: User, currentUser: User)
LoginFrame
- emailField: JTextField
- passwordField: JPasswordField
- loginButton: JButton
+ LoginFrame()

USER INTERFACE

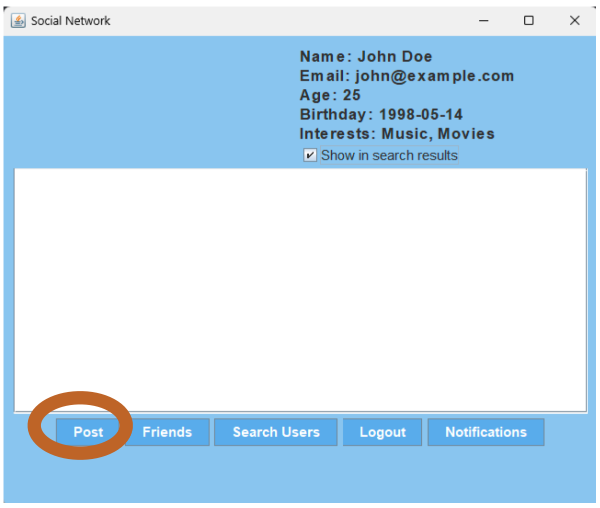
Log in frame



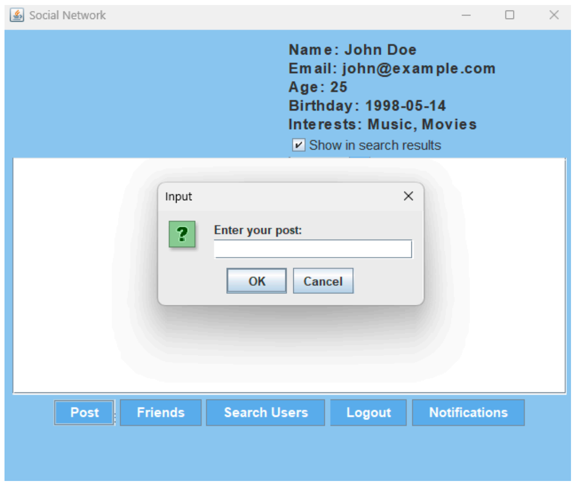
Search for a friend


Name: John Doe
Email: john@example.com
Age: 25

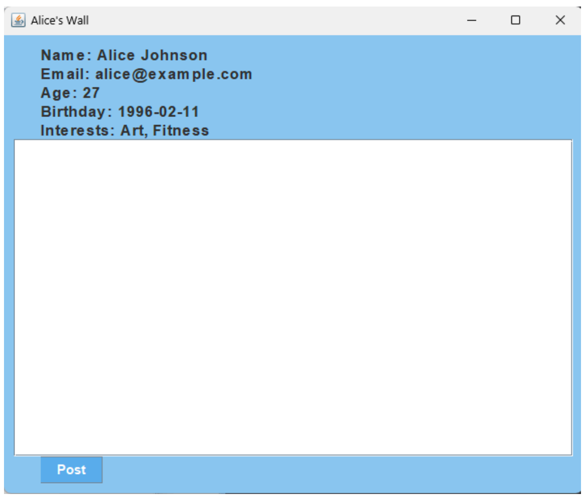
Main Frame


Name: John Doe
Email: john@example.com
Age: 25
Birthday: 1998-05-14
Interests: Music, Movies
✓ Show in search results

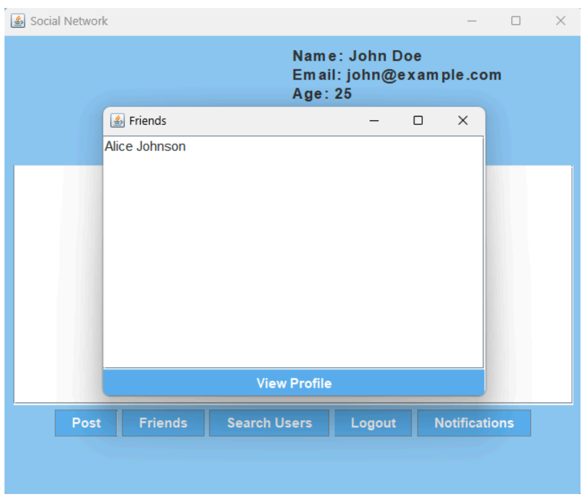
Posting


Name: John Doe
Email: john@example.com
Age: 25
Birthday: 1998-05-14
Interests: Music, Movies
✓ Show in search results

User’s Wall


Name: Alice Johnson
Email: alice@example.com
Age: 27
Birthday: 1996-02-11
Interests: Art, Fitness

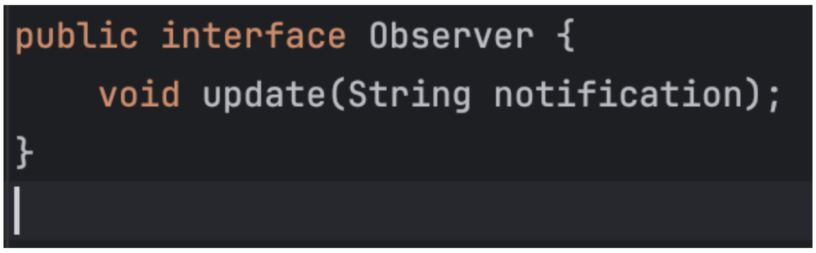
Friends List


Name: John Doe
Email: john@example.com
Age: 25
Alice Johnson

DESIGN PATTERNS

Observer:

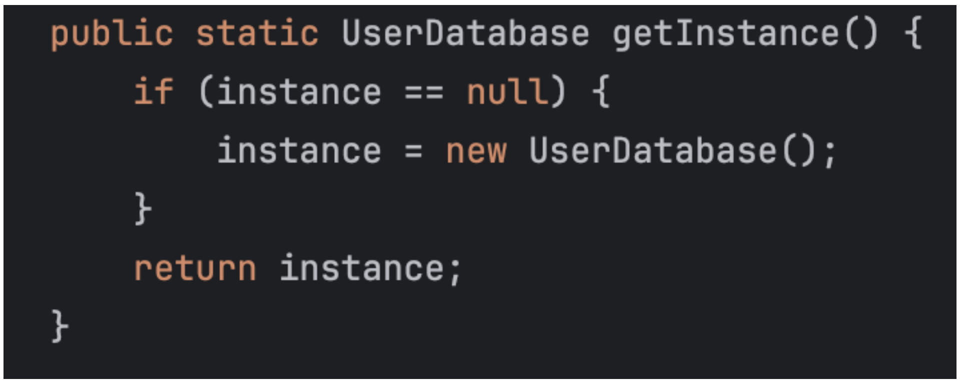
if users have methods to update each other (e.g., notifying friends about a new post), it is considered as implementing the observer pattern



```
public interface Observer {
    void update(String notification);
}
```

Singleton:

The UserDatabase class is implemented as a singleton. This is evident from the getInstance() method which ensures that there is only one instance of UserDatabase in the application. This pattern is used to manage the centralized user data and posts.



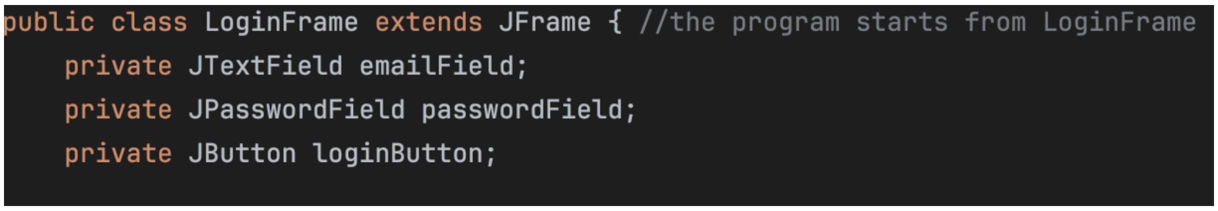
```
  public static UserDatabase getInstance() {
      if (instance == null) {
          instance = new UserDatabase();
      }
      return instance;
  }
```

MVC Pattern (Model-View-Controller):

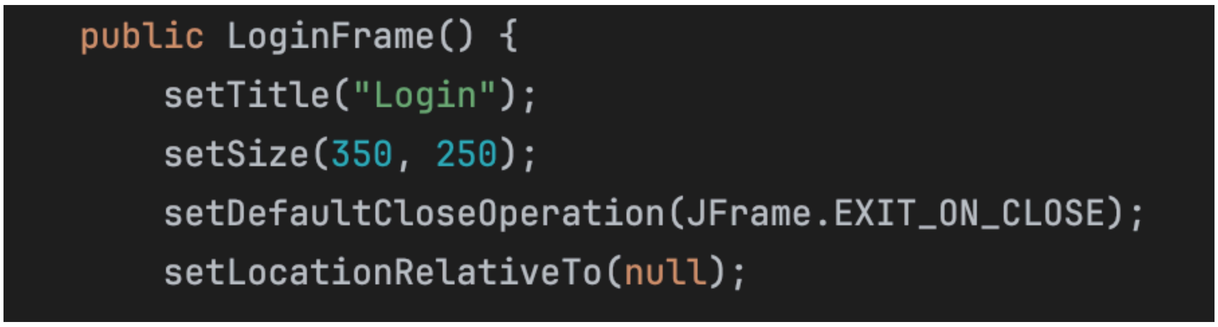
Model: User, UserDatabase

View: LoginFrame, MainFrame, SearchUsersFrame, FriendsFrame, FriendWallFrame

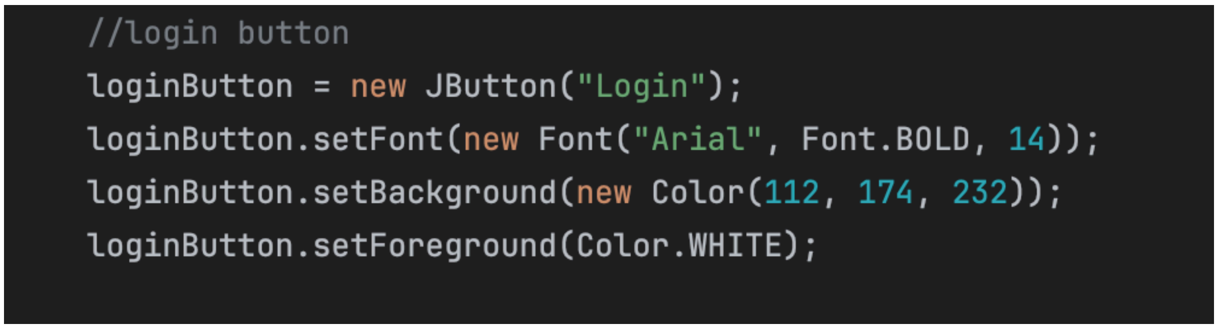
Controller: Implicitly handled in the action listeners and interaction logic within the frames.



```java
public class LoginFrame extends JFrame { //the program starts from LoginFrame
    private JTextField emailField;
    private JPasswordField passwordField;
    private JButton loginButton;
```



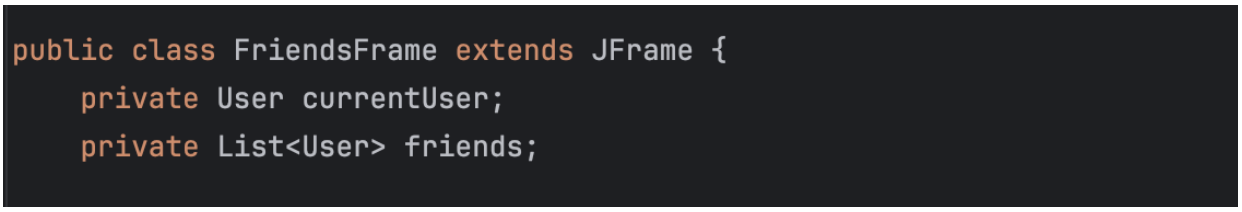
```java
    public LoginFrame() {
        setTitle("Login");
        setSize(350, 250);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
```



```java
        //login button
        loginButton = new JButton("Login");
        loginButton.setFont(new Font("Arial", Font.BOLD, 14));
        loginButton.setBackground(new Color(112, 174, 232));
        loginButton.setForeground(Color.WHITE);
```

Composite Pattern:

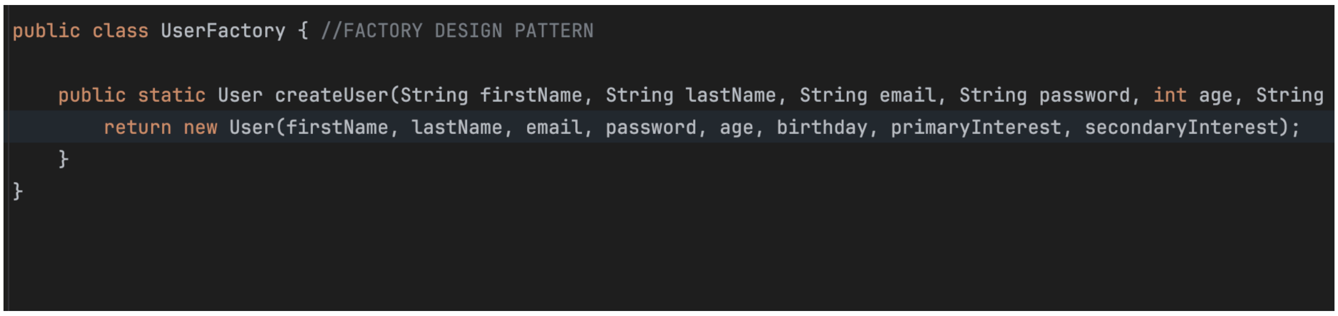
JPanel and JComponent: The Swing components like JPanel and JComponent often use the composite pattern to allow components to contain other components.



```java
public class FriendsFrame extends JFrame {
    private User currentUser;
    private List<User> friends;
```

Factory:

Factory allows us to create User without doing it directly from class.



```java
public class UserFactory { //FACTORY DESIGN PATTERN

    public static User createUser(String firstName, String lastName, String email, String password, int age, String
        return new User(firstName, lastName, email, password, age, birthday, primaryInterest, secondaryInterest);
    }
}
```

END