

project:

This project will focus on creating a platform and social network where users can share their thoughts, news, updates and multimedia content in a fast and direct way. In this application, users will be able to participate in each other's conversations.

objectives:

Among the objectives we have for this application are:

- to provide a simple to understand interface
- basic user information should be able to be entered: name, age, profile picture, etc.
- Possibility to publish texts of up to 280 characters, links and images.
- Have a button with which users can react to each other's content.
- Have a main feed where you can see the content of other users.

User stories:

Title:	Priority:	Estimate:
User register	HIGH	
User Story: As a user I want be able to register easily using my email or social networks So that make it easier and save time.		
<ul style="list-style-type: none">-The user can register using their email or social media accounts.-The system validates the data and redirects the user to their profile after successful registration.		

Title:	Priority:	Estimate:
User post	HIGH	
User Story: As a user I want to be able to post a tweet with text, images, and links, So that my followers can see what I want to share and participate in the conversation.		
<ul style="list-style-type: none">-The user can write a tweet of up to 280 characters.-Can attach images, links, or videos to the tweet.-System displays a preview of the tweet before publishing it.		

Title:	Priority:	Estimate:
Interacción with post	MEDIUM	
User Story: As a user I want to be able to "like" other users' content, So that I can show my support or share interesting content with my followers.		
<ul style="list-style-type: none">-The user can click on a "like" icon to mark as a favorite.-Changes in the number of "likes" are reflected in real time.		

Title:	Priority:	Estimate:
Private messages	LOW	
User Story: As a user I want to be able to send private messages to other users, So that I can have private conversations without others seeing the content.		
Acceptance Criteria: -The user can send and receive direct messages privately. -Direct messages can include text, images, and videos.		

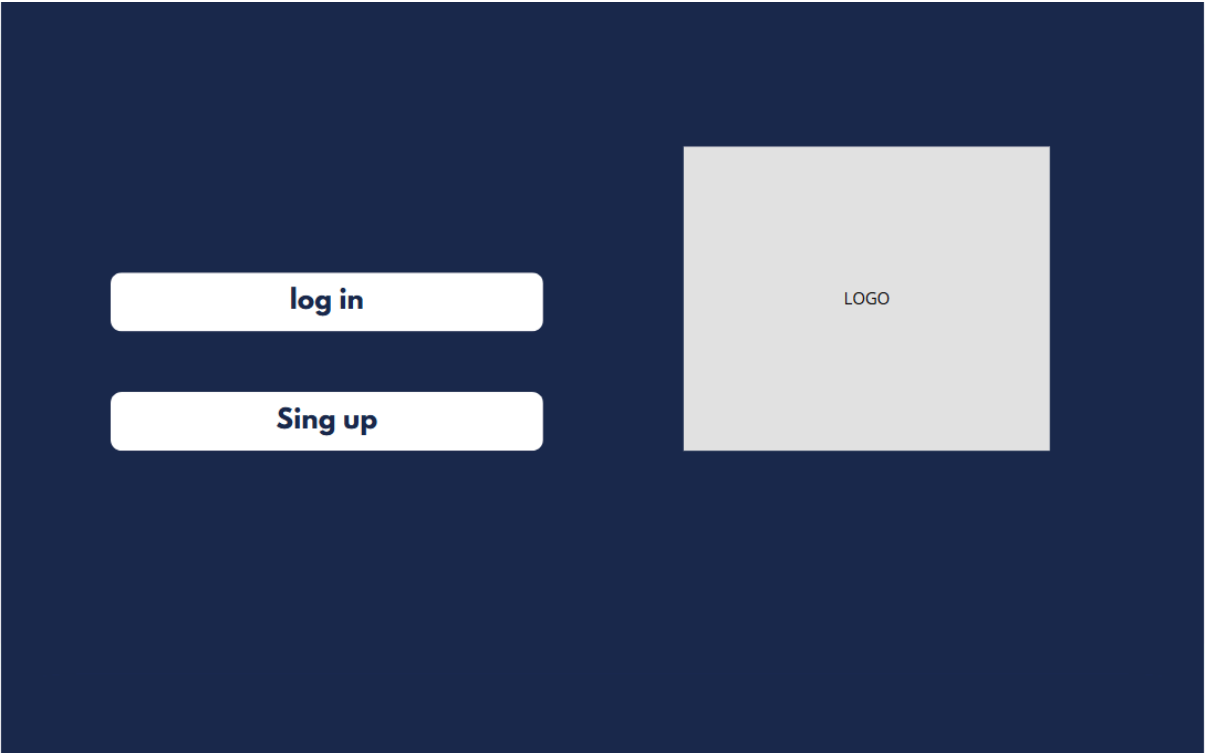
Title:	Priority:	Estimate:
Follow other users	LOW	
User Story: As a user I want to be able to follow other users to see their content in my feed, so I can keep up with their posts and participate in their conversations.		
Acceptance Criteria: -User can search and follow other users from their profile. -Users can stop following others at any time.		

Title:	Priority:	Estimate:
Notifications	LOW	
User Story: As a user I want to receive notifications when someone mentions me, “likes” my content or follows me, So that I can be aware of interaction with my account and engage in a timely manner.		
Acceptance Criteria: -The system notifies the user when he/she receives “likes”, or when someone follows him/her. -The user can configure what type of notifications he/she wants to receive.		

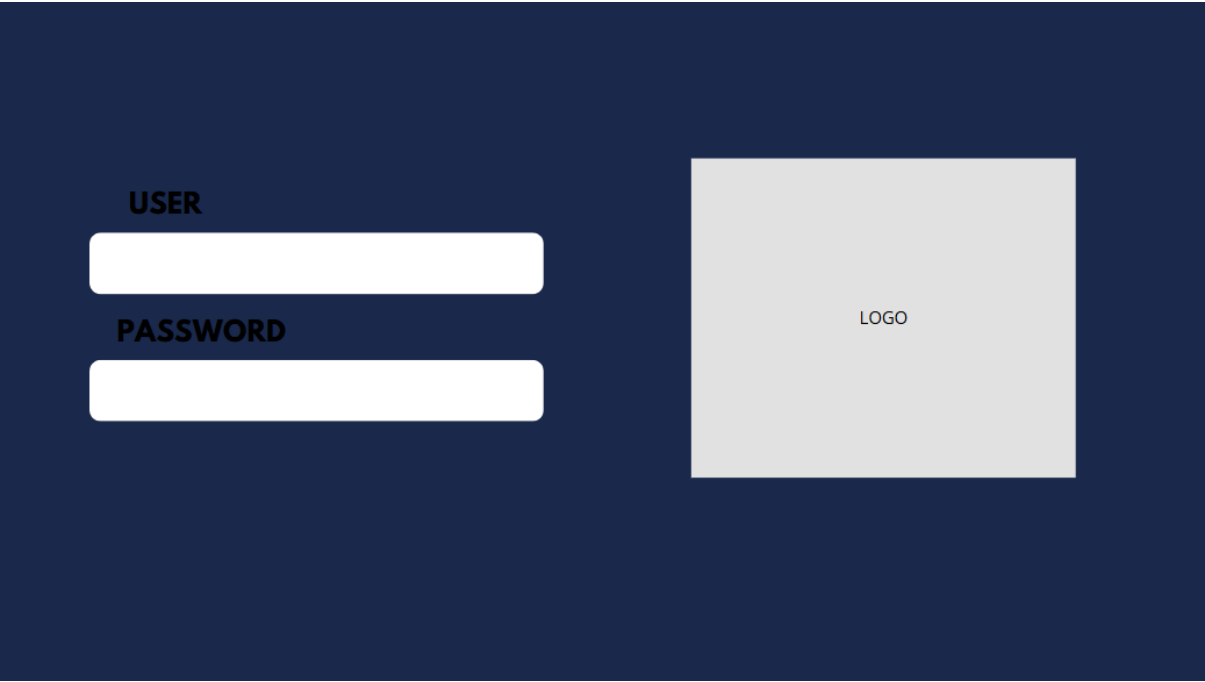
Title:	Priority:	Estimate:
Notifications	LOW	
User Story: As a user I want to be able to explore the most popular topics and trending hashtags, So I can discover relevant content and join the most important conversations.		
Acceptance Criteria: -The user can access a “Trending” section that shows the most popular hashtags and topics of discussion in real time. -The system automatically updates the trends to reflect what is most relevant based on the user's location and interest.		

mockups:

home:



log in:



Sing up:

USER

PASSWORD

EMAIL

AGE

LOGO

Feed:

LOGO

CRC Cards:

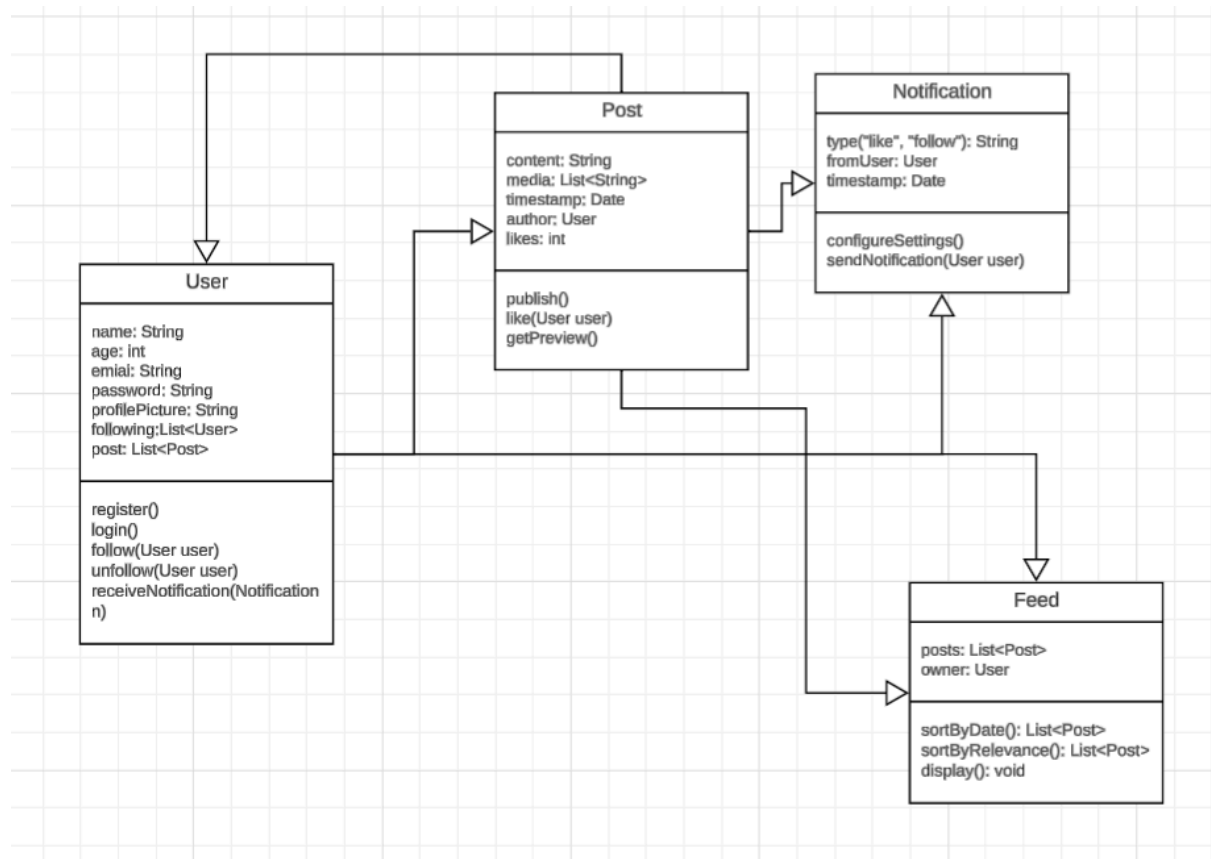
Class USER	
Responsibility	Collaborator
Store user information Create posts View the feed	Post Feed

Class POST	
Responsibility Represent a published content Store who published it and when	Collaborator User Reaction

Class FEED	
Responsibility Show posts by other users Sort posts by date or relevance	Collaborator User Post

Class REACTION	
Responsibility Represent a “like” reaction Be associated with a publication and a user.	Collaborator User Post

UML Diagrams:



-User

Encapsulation:

Attributes such as password, email and age are private or protected, accessible only through methods such as login() and register().

Access to the following and post list is controlled, avoiding direct modifications from outside the class

Abstraction:

Methods like follow(User user) and sendMessage(User to, Message msg) hide the internal logic of adding followers or managing messages.

The user does not need to know how a message is stored or how a notification is generated, just use the interface.

polymorphism:

Methods such as receiveNotification() or sendMessage() can be overwritten in subclasses to modify their behavior (e.g. users not allowing direct messages).

-Post

Encapsulation:

Attributes like likes and author are not modified directly. The likes count only changes through like(User user).

Abstraction

Methods such as publish() and getPreview() encapsulate details such as content formatting or how a post is published, exposing a clean interface.

Inheritance

It can serve as a base class for subtypes of publications such as ImagePost, VideoPost, which inherit common attributes and methods.

Polymorphism

Subclasses could override `getPreview()` to return a different view depending on the content type.

Inheritance
Can serve as a base class for subtypes of publications such as `ImagePost`, `VideoPost`, which inherit common attributes and methods.

-Notification:

Encapsulation:

Information such as type, `fromUser` and `timestamp` is hidden and handled only through public methods.

Abstraction:

`sendNotification(User user)` exposes a standard way to send notifications, without revealing the medium or format used.

Inheritance:

Notification can be a superclass for specific notifications such as `LikeNotification`

Polymorphism:

Subclasses can override `sendNotification()` to vary the content.

-Feed:

Encapsulation

The internal handling of posts and their sorting is hidden from the outside. It is accessed by methods such as `sortByDate()`.

Abstraction

Methods like `display()` or `sortByRelevance()` present a simple interface without exposing how relevance is calculated or content is filtered.

Inheritance

Feed can serve as a base class for more specific feeds such as `HashtagFeed` or `PersonalizedFeed`.

Polymorphism

Subclasses can override `sortByRelevance()` to customize the sorting logic according to the feed type.

**Relations:

User → Post: Each user can have several posts, as creating content is a main function of the system. Posts are directly associated with the author and are part of their activity within the platform.

User ↔ User: Users can follow and be followed by other users. This relationship makes it possible to build a dynamic social network where the content that appears in the feeds is determined.

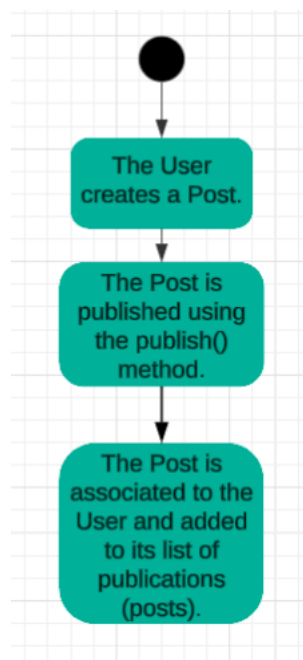
User → Notification: Users receive notifications when others interact with them, such as in cases of mentions or likes. This allows you to keep the user informed about relevant activity.

Post → Notification: Posts generate notifications when other users interact with them. This connects the content with the alerts system to improve social feedback.

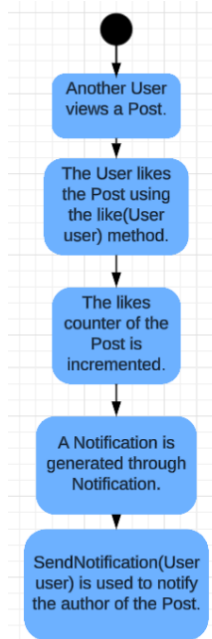
Feed → Post: The feed displays a collection of posts relevant to a user. Although the feed does not own them, it organizes and presents them according to criteria such as date or popularity.

Feed → User: Each feed belongs to a specific user and is tailored to their interests. The relationship allows the content displayed to be personalized according to the user's activity.

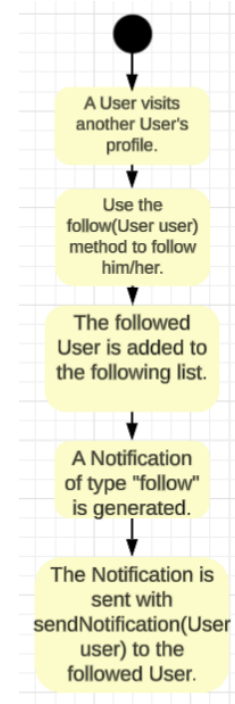
post publication:



Interaction with "Like" on a Post:



Follow between Users:



Implementation Plan for OOP Concepts:

-Encapsulation will be implemented using access modifiers, along with getters and setters:

Sensitive attributes such as password, email, and likes will be private or protected. Public methods (getEmail(), setProfilePicture(), getLikes()) will be provided to access/control their modification.

-Inheritance will be used to create hierarchies between general classes and their variants:

Post will be a base class for subclasses such as ImagePost, VideoPost. Notification will be a superclass for LikeNotification, FollowNotification and others.

-Abstraction:

Methods such as `publish()` or `sendNotification()` hide the details of how content is stored or how a notification is sent. The user or programmer only needs to know what it does, not how it does it. Classes can be defined as abstract (`Post` or `Notification`) if they are not to be instantiated directly, but only through their subclasses (`ImagePost`, `LikeNotification`, etc.).

-polymorphism:

the app has different types of notifications, "like" "new follower", each one can have its own way of sending or displaying the notification. The system simply uses the `sendNotification()` method on any notification type, and each subclass executes it in its own way.

preliminary directory structure:

-model: the classes that represent the application data are defined.

`User.java`: the user, with fields such as name, email, followers, etc.

`Post.java`: Base class for publications. It has content, author, date, etc.

`ImagePost.java` and `VideoPost.java`: Subclasses of `Post`. They represent specific publications with image or video.

`Notification.java`: General class for notifications.

`LikeNotification.java` and `FollowNotification.java`: Specific types of notifications that inherit from `Notification`.

-controller: contains the classes that encapsulate the logic that operates on the models.

`UserController.java`: it has the user's actions (registration, profile editing, following others, etc.).

`PostController.java`: Manages the creation, edition or deletion of posts.

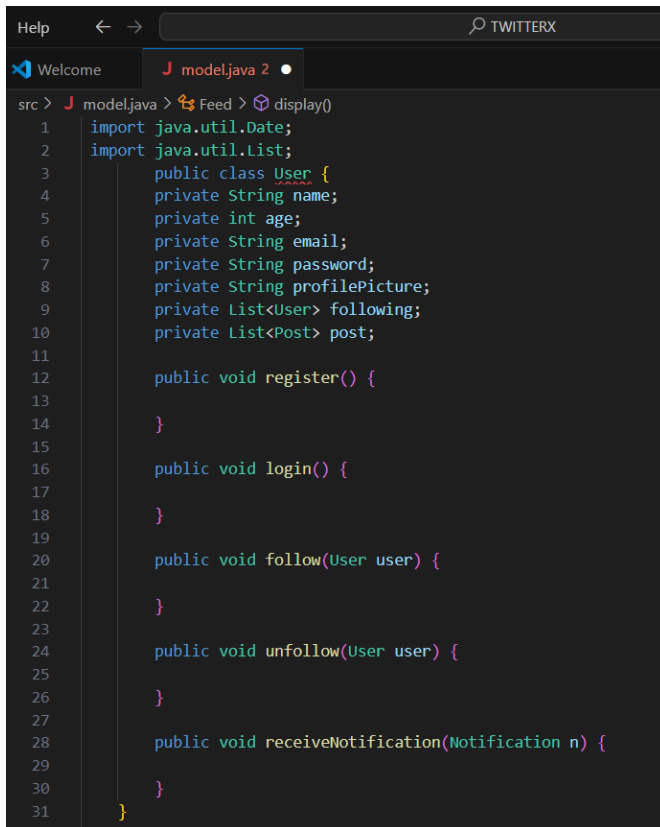
`FeedController.java`: Coordinates the obtaining of the feed seen by the user.

-view

This has the menus and graphical interfaces. In this one there are "mockups", the preliminary designs.

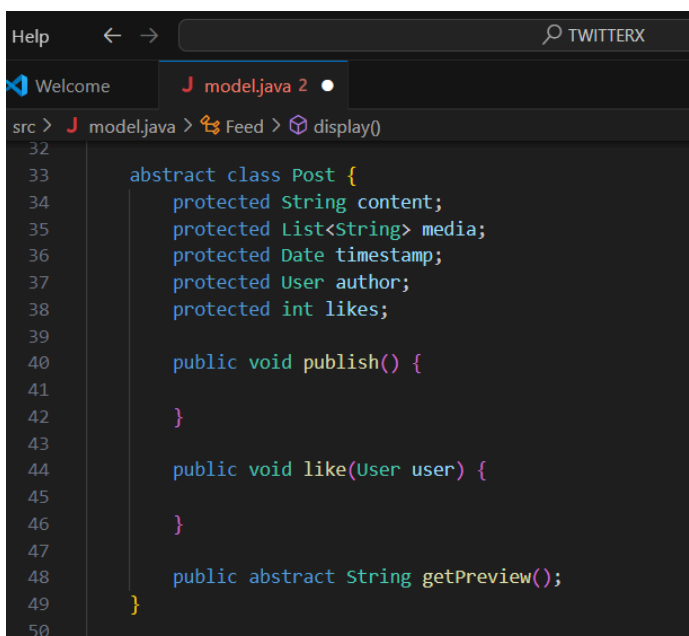
Progress Code:

The code was started by defining the classes and leaving an empty body where the placeholders will be implemented, and illustrating the abstract classes such as (`Post`) which is an abstract class, and its method `getPreview()` is also abstract, allowing the subclasses to overwrite it according to the type of content.



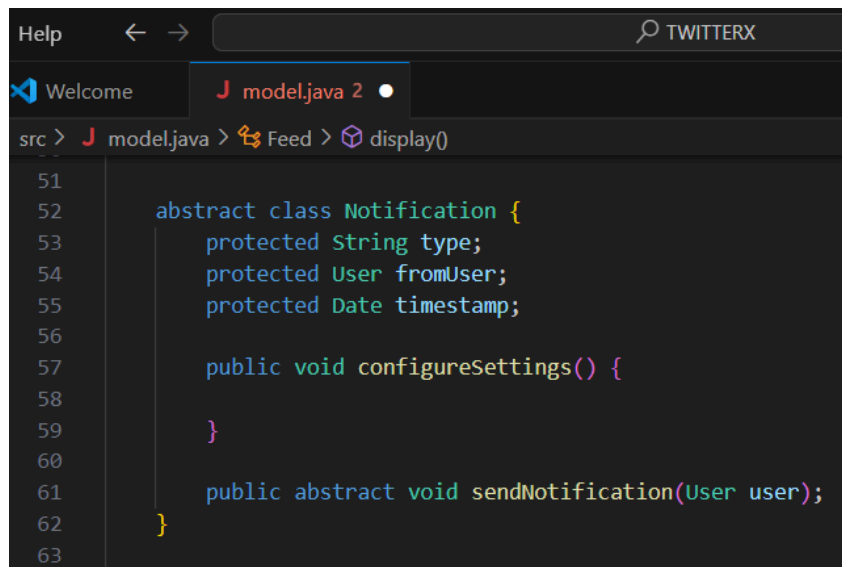
```
src > J model.java > Feed > display()
1  import java.util.Date;
2  import java.util.List;
3
4  public class User {
5      private String name;
6      private int age;
7      private String email;
8      private String password;
9      private String profilePicture;
10     private List<User> following;
11     private List<Post> post;
12
13     public void register() {
14
15     }
16
17     public void login() {
18
19     }
20
21     public void follow(User user) {
22
23     }
24
25     public void unfollow(User user) {
26
27     }
28
29     public void receiveNotification(Notification n) {
30
31     }
32 }
```

The user class is related to the UML diagram as it reflects all attributes and methods of User and also applies encapsulation.



```
src > J model.java > Feed > display()
32
33     abstract class Post {
34         protected String content;
35         protected List<String> media;
36         protected Date timestamp;
37         protected User author;
38         protected int likes;
39
40         public void publish() {
41
42         }
43
44         public void like(User user) {
45
46         }
47
48         public abstract String getPreview();
49     }
50 }
```

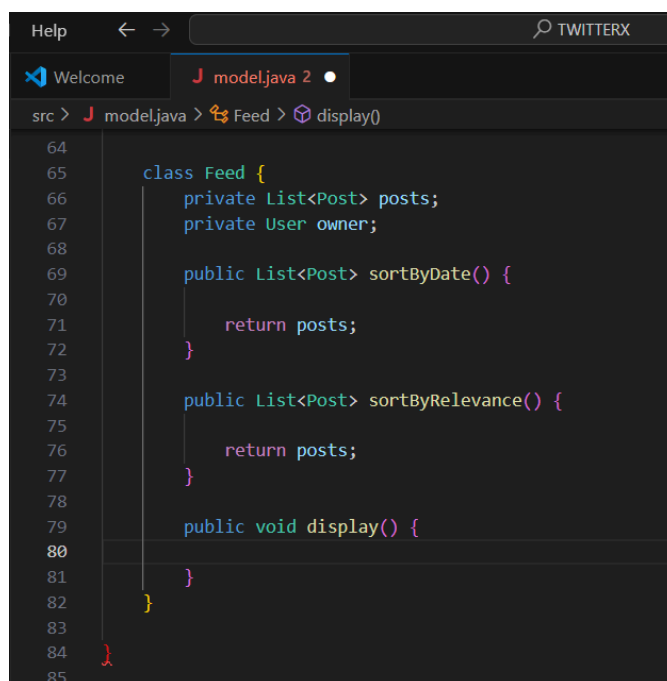
Post is an abstract class and has attributes and methods defined in the UML, hence the relationship.



The screenshot shows an IDE window with a tab labeled 'model.java 2'. The breadcrumb navigation shows the path 'src > model.java > Feed > display()'. The code editor displays the following Java code for the Notification class:

```
51
52     abstract class Notification {
53         protected String type;
54         protected User fromUser;
55         protected Date timestamp;
56
57         public void configureSettings() {
58
59         }
60
61         public abstract void sendNotification(User user);
62     }
63
```

Is related to the UML diagram as it has exact design, with sendNotification() as a polymorphic method that will be overridden in subclasses such as LikeNotification.



The screenshot shows the same IDE window with the same tab and breadcrumb navigation. The code editor displays the following Java code for the Feed class:

```
64
65     class Feed {
66         private List<Post> posts;
67         private User owner;
68
69         public List<Post> sortByDate() {
70
71             return posts;
72         }
73
74         public List<Post> sortByRelevance() {
75
76             return posts;
77         }
78
79         public void display() {
80
81         }
82     }
83
84 }
85
```

Is related to the UML as it contains the organization and visualization methods described. The Feed class uses an aggregation relationship with Post and association with User.