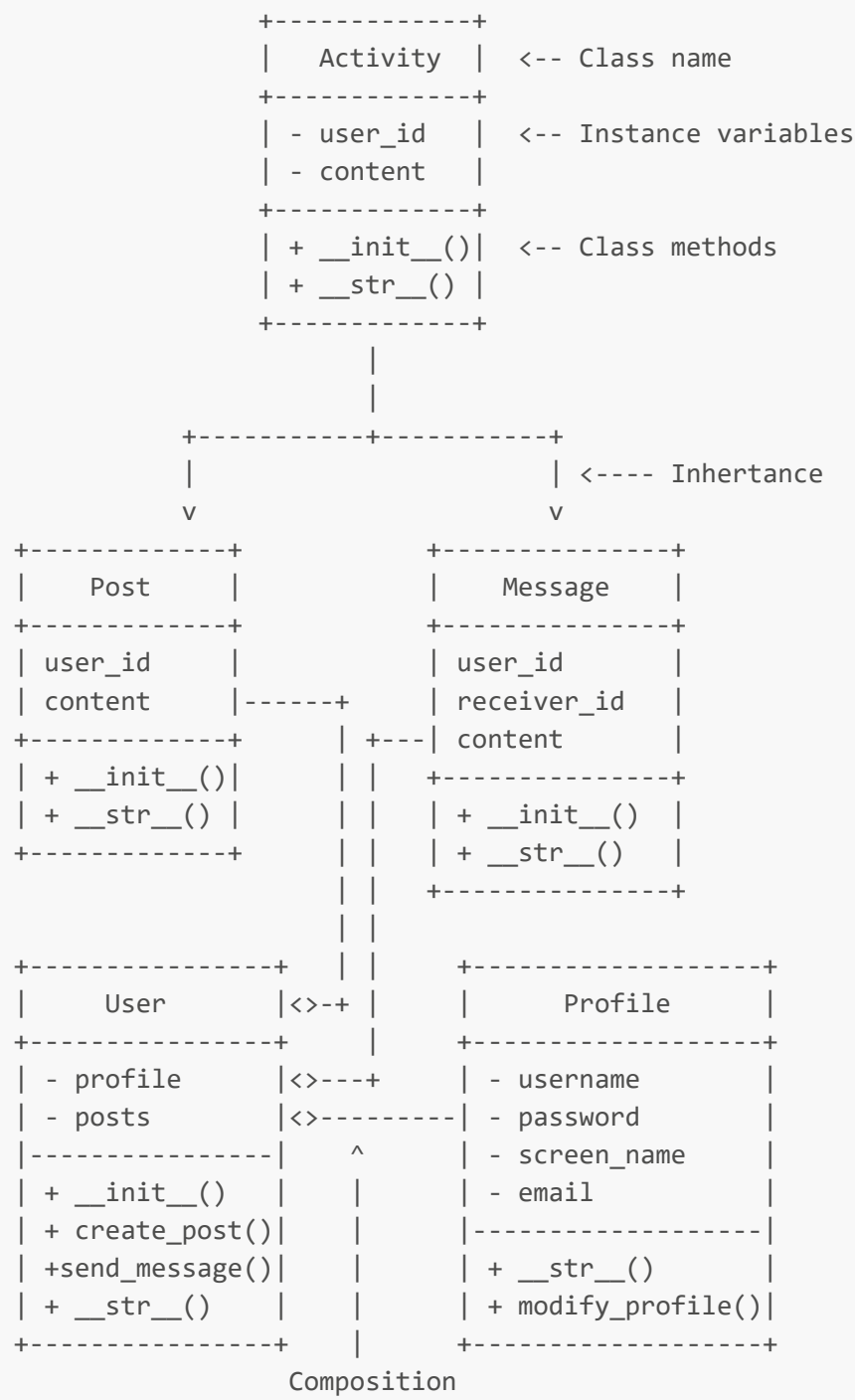
Social Network System - Homework 2

Overview

In this homework assignment, you will design and implement a basic social network system using object-oriented programming principles in Python. The focus will be on utilizing inheritance and composition relationships between classes to model the structure of the social network.



The diagram depicts the class structure social network system, featuring five classes: User, Profile, Activity, and specialized subclasses Post and Message. The User class encapsulates user-related functionalities, managing

activities like creating posts and sending messages. The Activity class acts as a base for user actions, with Post and Message inheriting from it. The design employs a combination of inheritance and composition to promote modularity and encapsulation, facilitating user interactions within the social network while maintaining a structured and scalable architecture.

Inheritance and Composition

Inheritance, represented by `-->` in the diagram above, is used to model *is-a* relationships. A `Post` *is-an* `Activity`. A `Message` *is-an* `Activity.` It makes sense conceptually for these classes to inherit all attributes related to an `Activity`.

Composition, represtended by `--<>` in the diagram above, is used to model *has-a* relationships. A `Post` *has-a* `User`, so we attach an object of type `User` in `Post.__init__`. A `Message` *has-a* `User`, so we do the same in `Message.__init__`. A `User` *has-a* `Profile`, so we attach a `Profile` object to a `User` object in `User.__init__`.

Requirements

Part 1: hw2.py - Classes

You must fully implement the following classes in the `hw2.py` file. Refer to the `TODO` comments for places where you must write your own code.

1. **Profile Class** Holds essential user information (username, password, screen name, email).
    - Variables:
        - `username` (str): username of the user.
        - `password` (str): Password associated with the account.
        - `screen_name` (str): Display name of the user.
        - `email` (str): Email address of the user.
    - Methods:
        - `__init__(self, username, password, screen_name, email)`: Initialize a Profile instance.
        - `__str__(self)`: Return a string representation of the Profile.
        - `modify_profile(self, password=None, screen_name=None, email=None)`: Modify user's profile information.
2. **Activity Class (Base Class)** Serves as the base class for user activities. Contains common attributes like the user associated with the activity and the content of the activity. Both `Post` and `Message` classes inherit from Activity.
    - Variables:
        - `user` (User): The user associated with the activity.
        - `content` (str): Content of the activity.
    - Methods:
        - `__init__(self, user, content)`: Initialize an Activity instance.
        - `__str__(self)`: Return a string representation of the Activity.
3. **Post Class (Derived from Activity)** Represents a user's post in the social network. Inherits from the Activity class.

- Additional Attributes:
  - None
- Additional Methods:
  - None

4. **Message Class (Derived from Activity)** Represents a user's message to another user. Inherits from the Activity class.

   - Additional Attributes:
     - `receiver` (User): The user receiving the message.
   - Additional Methods:
     - None

5. **User Class** Represents a user in the social network. Contains a Profile instance representing the user's details (username, password, screen name, email). Manages user activities such as creating posts and sending messages.

   - Attributes:
     - `profile` (Profile): User's profile.
     - `posts` (list of Post): List of posts created by the user.
     - `messages` (list of Message): List of messages sent by the user.
   - Methods:
     - `__init__(self, username, password, screen_name, email)`: Initialize a User instance.
     - `create_post(self, content) -> Post`: Create a new post for the user. Raise ValueError if the content of the post is empty.
     - `send_message(self, receiver, content) -> Message`: Send a message to another user. Raise ValueError if the receiver ID or message content is empty.
     - `__str__(self)`: Return a string representation of the User.

Part 2: hw2_test.py - Testing

Testing is a crucial part of software development to ensure that your classes and methods work as expected. For this homework, you are required to thoroughly test all the classes and methods you implement. Utilize the unittest framework to create test cases for each class and their respective methods. Pay close attention to edge cases and ensure that your code handles them gracefully.

Example Testing

Here's a brief example of how you might structure your test cases:

1. **Profile Class Tests**

   - Test the initialization of a Profile instance.
   - Modify the profile using the `modify_profile` method and check if the changes are reflected.

2. **Activity Class Tests**

   - Test the initialization of an Activity instance.
   - Check if the `__str__` method returns the expected string.

3. **Post Class Tests**
    - Test the creation of posts.
4. **Message Class Tests**
    - Test the creation of messages.
5. **User Class Tests**
    - Test the initialization of a User instance.
    - Test a post and a message with empty content.Expecting a ValueError to be raised.

Example Test Case

```
import unittest
from hw2 import Profile, Activity, Post, Message, User
class TestUser(unittest.TestCase):
    """Test cases for the User class."""
    def setUp(self):
        self.user = User("user1", "password1", "User One", "user1@example.com")

    def test_create_post(self):
        """Test creating a post for a user."""
        post = self.user.create_post("Test Post Content")
        # Check if the post is added to the user's posts list
        self.assertIn(post, self.user.posts)
        # Check if the user is correct
        self.assertEqual(post.user, self.user)
        # Check if the content of the post is correct
        self.assertEqual(post.content, "Test Post Content")

    # Add more test cases for other methods and classes

if __name__ == '__main__':
    unittest.main()
```

Note: Some classes and methods are given to you in the starter code

Comments and Docstrings

- Include meaningful comments throughout your code.
- Add docstrings for each class and method, providing clear and concise explanations of their purpose.

Imports

No imports are allowed on the assignments, with the following exceptions:

- `unittest` - required for creating unit tests.
- `random` - may be used for creating unit tests.
- `typing` - not required, but some students have requested it in the past.

Submission

Submit the following files:

- `hw.py`
- `test_hw2.py`

Students must submit to Gradescope individually by the due date (typically Tuesday at 11:59 pm EST) to receive credit.

Good luck!