CSC148 - Composition of Classes

Here is the documentation for our `Tweet` class:

```
class Tweet:
    """A tweet, like in Twitter.

    === Attributes ===
    content: the contents of the tweet.
    userid: the id of the user who wrote the tweet.
    created_at: the date the tweet was written.
    likes: the number of likes this tweet has received.
    """
    content: str
    userid: str
    created_at: date
    likes: int

    def __init__(self, who: str, when: date, what: str) -> None:
        """Initialize a new Tweet.
        """

    def like(self, n: int) -> None:
        """Record the fact that this tweet received <n> likes.

        These likes are in addition to the ones <self> already has.
        """

    def edit(self, new_content: str) -> None:
        """Replace the contents of this tweet with the new message.
        """
```

for programmers writing client code

Here's the start of another class that will interact with class `Tweet`, and will represent a *user* of Twitter.

```
class User:
    """A Twitter user.

    === Attributes ===
    userid: the userid of this Twitter user.
    bio: the bio of this Twitter user.
    tweets: the tweets that this user has made.
    """
    # Attribute types
    userid: str
    bio: str
    tweets: List[Tweet]
```

1. Implement an initializer for this class according to the docstring below.

```
    def __init__(self, userid: str, bio: str) -> None:
        """Initialize a new user with the given id and bio.

        The new user initially has no tweets.
        """
```

self.userid = userid
self.bio = bio
self.tweets = []

2. Implement the `User` method `tweet` according to the docstring below.

```
def tweet(self, message: str) -> None:
    """Record that this User made a tweet with the given content.

    Use date.today() to get the current date for the newly-created tweet.
    """
```

t = Tweet(self.userid, date.today(), message)

self.tweets.append(t)

3. Suppose we want another `User` method that will record the fact that the user follows another user with a given userid. We'll used a modified version of the *Function Design Recipe* to design it.

   First, write down a sample call to this method. (Do *not* worry about how you'll implement it yet!)

4. Write a header and docstring for this method. Don't forget to include type annotations for the parameters and return value. (We've left space for the method body, but don't write it yet.)

5. Decide what attribute(s) you will use to store information about who this user follows. Then, update the class docstring and attribute type annotations (on the previous page) to record your decisions.

6. Finally, implement your new method according to its docstring.