P07 Twitter Feed

Overview

Thanks to a controversial deal, the world has been watching popular microblogging platform Twitter slowly catch on metaphorical fire. If you're somehow unfamiliar, Twitter collects short posts (280 characters or less) which other users interact with by liking or retweeting.

A classic 2012 tweet from Twitter user @dril

In this program, you will model a simple Twitter feed as a linked list of tweets. You will be able to iterate through these tweets in reverse-chronological order, or show only tweets from users with important blue checkmarks, or with only tweets with more than a certain ratio of likes to total interactions

Grading Rubric

5 points	**Pre-assignment Quiz**: accessible through Canvas until 11:59PM on **04/10**.
15 points	**Immediate Automated Tests**: accessible by submission to Gradescope. You will receive feedback from these tests *before* the submission deadline and may make changes to your code in order to pass these tests. Passing all immediate automated tests does **not** guarantee full credit for the assignment.
20 points	**Additional Automated Tests**: these will also run on submission to Gradescope, but you will not receive feedback from these tests until after the submission deadline.
10 points	**Manual Grading Feedback**: TAs or graders will manually review your code, focusing on commenting and style.

Learning Objectives

After completing this assignment, you should be able to:

- **Explain** how data is stored in a singly-linked list, including how additional elements are added in locations other than the head or tail of the list
- **Contrast** multiple different approaches to implementation of the Iterator interface
- **Identify** the ways in which an Iterable class can be used

Additional Assignment Requirements and Notes

- Pair programming is **NOT ALLOWED** for this assignment. You must complete and submit P07 individually.

- The ONLY external libraries you may use in any of your classes are:
  ```
  java.util.Calendar, java.util.Date
  java.util.Iterator
  java.util.NoSuchElementException
  ```
  Use of *any* other packages (outside of java.lang) is NOT permitted.

- You are allowed to define any **local** variables you may need to implement the methods in this specification (inside methods). You are NOT allowed to define any additional instance or static variables or constants beyond those specified in the write-up, except for public static helper methods.

- Exceptions do not NEED to have error messages, but a descriptive error message may help you debug your code and is *recommended*.

- You are allowed to define additional **private** helper methods.

- Only your `TwiteratorTester.java` file may include a **main** method.

- All classes and methods must have their own Javadoc-style method header comments in accordance with the CS 300 Course Style Guide.

- Any source code provided in this specification may be included verbatim in your program without attribution.

- **Run your program locally before you submit to Gradescope**. If it doesn't work on your computer, *it will not work on Gradescope*.

Need More Help?

Check out the resources available to CS 300 students here:
https://canvas.wisc.edu/courses/344658/pages/resources

CS 300 Assignment Requirements

You are responsible for following the requirements listed on both of these pages on all CS 300 assignments, whether you've read them recently or not. Take a moment to review them if it's been a while:

- Academic Conduct Expectations and Advice, which addresses such questions as:
  - How much can you talk to your classmates?
  - How much can you look up on the internet?
  - What do I do about hardware problems?
  - and more!
- Course Style Guide, which addresses such questions as:
  - What should my source code look like?
  - How much should I comment?
  - and more!

Getting Started

1. Create a new project in Eclipse, called something like **P07 Twitter Feed**.
   a. Ensure this project uses Java 17. Select "JavaSE-17" under "Use an execution environment JRE" in the New Java Project dialog box.
   b. Do **not** create a project-specific package; use the default package.
2. Add the **TimelineMode.java** enum and **ListADT.java** interface from the assignment page to your project.
3. FOR NOW, create three (3) Java source files within that project's src folder:
   a. **User.java** (does NOT include a main method)
   b. **Tweet.java** (does NOT include a main method)
   c. **TwiteratorTester.java** (includes a main method)

You will create more files later! But let's start here.

1. What You'll Store in the Data Structure

To begin this program, we need objects to store in our data structure. We will model the users of our fake-Twitter with the `User` class (where we'll store information about, for example, their username and whether they have an important blue checkmark) and the tweets that they make will be modeled in the `Tweet` class.

The relationship between these two classes is "has-a" – every `Tweet` object has a `User` data field. So I recommend beginning with the `User` class! Both of these are very simple objects with just a few accessors and mutators; see the javadocs: User and Tweet

1.1 Testing Your Objects

In your `TwiteratorTester` class, create two methods to verify that the `User` and `Tweet` classes are working properly:

```
public static boolean testUser()
public static boolean testTweet()
```

Make sure to initialize the `dateGenerator` static field in `Tweet` before testing by calling the `setCalendar()` method; if you care about a specific timestamp (for example, if you are testing `toString()`) then see below for how to get a specific time. Otherwise, you can just use `Calendar.getInstance()` for the current time.

1.2 toString() examples

Both the `User` and `Tweet` classes contain an overridden implementation of Object's `toString()` method. We'll format `User`s so that they look like typical Twitter usernames (except with an asterisk * instead of an important-looking blue checkmark for verified users):

Verified User with username "uwmadison"	**Un-verified** User with username "dril"
`"@uwmadison*"`	`"@dril"`

(Be sure to implement the behavior that usernames are not allowed to contain * characters! We don't want people *faking* verification, do we?)

`Tweet`s will have this format (this is @dril's tweet from the screenshot on the cover page):

```
tweet from @dril at Tue May 22 14:46:03 CDT 2012:
-- IF THE ZOO BANS ME FOR HOLLERING AT THE ANIMALS I WILL FACE GOD AND WALK
BACKWARDS INTO HELL
-- likes: 5892
-- retweets: 4523
```

That is: all `Tweet`s begin with the text `"tweet from"` and the String representation of their `User`, followed by `"at"` and the String representation of their timestamp. The next line (which begins `"-- "`) contains the text of the tweet; the next two lines (also beginning with `"-- "`) display the number of likes and retweets on the tweet. There is NO newline character at the end of this String.

✅ To get this specific timestamp, I added the following lines to my code before I created the `Tweet`:

```
Calendar test = Calendar.getInstance();
test.set(2012, Calendar.MAY, 22, 14, 46, 03);
Tweet.setCalendar(test);
```

2. Creating Your Data Structure

1. Create two (2) more files in your project's `src` folder:
    a. **TweetNode.java** (does NOT include a main method)
    b. **TwitterFeed.java** (implements the ListADT interface; does NOT include a main method)

(That's still not everything for this project, but we're getting there.)

A `TweetNode` is a singly-linked list node containing a `Tweet` object. Per the javadocs, this must have accessors and mutators for the reference to the next node in the list, but *only* an accessor for its `Tweet` value. If you want to replace the value at a certain location in the list, you'll have to create an entirely new node to do so.

`TwitterFeed` is our singly-linked list and implements the provided ListADT interface. Note that ListADT is a generic interface! What type should go in the <> in `TwitterFeed`'s class header in order to make its methods match the javadocs? (Note: you can ignore the `Iterable` interface for now. We'll add it later.)

2.1 Testing Your Data Structure

In your `TwiteratorTester` class, create the following methods to verify that the `TweetNode` and `TwitterFeed` classes are working properly. The following are just some suggestions to get you started testing and do NOT include all edge cases – you'll need to add more to verify your code works.

```
public static boolean testNode()
public static boolean testAddTweet()
public static boolean testInsertTweet()
public static boolean testDeleteTweet()
```

For `testNode()`, create at least two (2) `TweetNode`s and connect one to the other. Verify that the accessor methods of the two nodes are working properly.

For `testAddTweet()`, create a `TwitterFeed` and verify that it is empty and has size 0. Use `addFirst()`/`addLast()` to add a `Tweet` to it. Verify that it is no longer empty, has size 1, `contains()` the `Tweet` you just added, and that `get(0)` matches that `Tweet`. Try this a few more times, and also test `getHead()`/`getTail()`.

For `testInsertTweet()`, create a `TwitterFeed` and several `Tweet` objects. Add them using add() with various indexes. Verify that the size is correct, and that `get()` with various indexes returns the `Tweet`s you expect. You may also wish to test `getHead()`/`getTail()`.

For `testDeleteTweet()`, create a `TwitterFeed` and add at least five (5) `Tweet` objects. Try removing the last `Tweet` and verify that `getTail()` has been updated correctly; try removing the first `Tweet` and verify that `getHead()` has been updated correctly. Then try removing a `Tweet` from a middle index (like 1) and verify that when you `get()` that index, it returns the value you expect.

3. Iterating Through Your Data Structure

1. Create the last three (3) files in your project's src folder:
   a. **ChronoTwiterator.java** (javadocs)
   b. **RatioTwiterator.java** (javadocs)
   c. **VerifiedTwiterator.java** (javadocs)
2. All of these classes MUST implement the `Iterator` interface (from java.util.Iterator) on `Tweet` objects, and include exactly one (1) data field:

```
private TweetNode next; // the next linked node in the list
```

3.1 ChronoTwiterator

This object iterates through `TweetNode` objects using the links between them. To complete this class:

1. Implement a single-argument constructor that expects a `TweetNode` and sets up the data field `next` to that provided value.

2. Override Iterator's `hasNext()` method to return true if and only if there are more elements to iterate through. NOTE: until a `Tweet` has actually been returned by the `next()` method, it is still waiting to be iterated through!

3. Override Iterator's `next()` method to return the `Tweet` in `next`'s `TweetNode` and advance `next` to the next `TweetNode` in the list. This method must throw a NoSuchElementException if there are no more values to return (make sure you import it).

Once you have the `ChronoTwiterator` completed, modify your `TwitterFeed` to ALSO implement the `Iterable<Tweet>` interface and add the following method to the class:

```
@Override
public Iterator<Tweet> iterator() {
  return new ChronoTwiterator(this.head);
}
```

Add a descriptive Javadoc comment! You'll be modifying this method slightly in the future.

3.2 Testing ChronoTwiterator

In your `TwiteratorTester` class, create the following methods to verify that `ChronoTwiterator` is working properly (remember to initialize `dateGenerator`):

```
public static boolean testChronoTwiterator()
```

For this method, DO NOT test `ChronoTwiterator` directly! Instead, create a `TwitterFeed` object and add at least three (3) `Tweet`s to it. Then, **use an enhanced-for loop** (also called a **for-each loop**) to iterate directly through the `TwitterFeed` and verify that all `Tweet`s are returned in the correct order.

3.3 VerifiedTwiterator

This object iterates through `TweetNode` objects similarly to `ChronoTwiterator`, but skips any `Tweet`s made by un-verified users.

1. The single-argument constructor has the same signature as `ChronoTwiterator`, but rather than just setting `next` to be the provided parameter directly, it must check whether the provided `TweetNode` contains a `Tweet` from a verified `User`. If it doesn't, it should move to the next linked `TweetNode` and check again, until it finds a `Tweet` from a verified `User` or runs out of `TweetNode`s.

2. `hasNext()` and `next()` also work similarly to `ChronoTwiterator`, but remember to skip over any `Tweet`s from unverified `User`s.

3. Once `VerifiedTwiterator` is complete, **modify** `TwitterFeed`'s `iterator()` method to return a `ChronoTwiterator` when the mode is CHRONOLOGICAL, a `VerifiedTwiterator` when the mode is VERIFIED_ONLY, and null otherwise.

In your `TwiteratorTester` class, create the following methods to verify that `VerifiedTwiterator` is working properly (remember to initialize `dateGenerator`):

```
public static boolean testVerifiedTwiterator()
```

and test it similarly to your `ChronoTwiterator`. Be sure to add at least one `Tweet` from a verified `User` to your `TwitterFeed` when you test!

3.4 RatioTwiterator

This object iterates through `TweetNode` objects similarly to `ChronoTwiterator`, but skips any `Tweet`s with a value from `getLikesRatio()` that is below a given threshold.

1. Add a `private final double THRESHOLD` data field to this class - this is the only iterator which will have TWO data fields.

2. Implement a two-argument constructor that expects a `TweetNode` and a **double**. The double value can go directly into the `THRESHOLD` constant; as with `VerifiedTwiterator`, only set `next` to be a `TweetNode` containing a `Tweet` whose likes ratio is >= `THRESHOLD`. If the provided one does not qualify, move to the next node in the list until you find one that does qualify (or you run out of nodes).

3. `hasNext()` and `next()` also work similarly to `VerifiedTwiterator`, remembering to skip over `Tweet`s whose likes ratio is below the `THRESHOLD`.

4. Once `RatioTwiterator` is complete, **modify** `TwitterFeed`'s `iterator()` method to return a `ChronoTwiterator` when the mode is CHRONOLOGICAL, a `VerifiedTwiterator` when

the mode is VERIFIED_ONLY, a `RatioTwiterator` using the class variable ratio as a threshold when the mode is LIKE_RATIO, and null otherwise.

In your `TwiteratorTester` class, create the following methods to verify that `RatioTwiterator` is working properly (remember to initialize `dateGenerator`):

```
public static boolean testRatioTwiterator()
```

and test it similarly to your `ChronoTwiterator`. Be sure to like/retweet some of the `Tweet`s in your `TwitterFeed` when you test, so there will be interesting values for the likes ratio!

4. Twitter Feed Demo

To give you a better idea of how to use these classes, here's an example of some code using the completed project and the expected output.

```
User u1 = new User("uwmadison");
u1.verify();
User u2 = new User("dril");
Calendar test = Calendar.getInstance();
test.set(2023, Calendar.APRIL, 3, 17, 07, 50);
Tweet.setCalendar(test);
TwitterFeed feed = new TwitterFeed();
feed.addFirst(new Tweet(u1, "Join us for Bucky's Big Event next Wednesday! " +
    "#CelebrateUW"));
feed.addFirst(new Tweet(u2, "type \"Gaming is back \", if you think gaming is " +
    "back"));
feed.addFirst(new Tweet(u1, "It's a GREAT day for @BadgerMHockey! #OnWisconsin"));
feed.addFirst(new Tweet(u1, "Welcome to the University of Wisconsin-Madison, " +
    "#FutureBadger!"));
feed.addLast(new Tweet(u2, "got botulism from a pair of bad jeans"));
Tweet tmp = feed.get(2);
for (int i=0; i<5243; i++) tmp.like();
for (int i=0; i<4702; i++) tmp.retweet();
tmp = feed.get(3);
tmp.like();
tmp = feed.get(4);
for (int i=0; i<307; i++) tmp.like();
for (int i=0; i<4015; i++) tmp.retweet();
System.out.println("---------- \"REVERSE CHRONOLOGICAL\" ORDER ----------");
for (Tweet t : feed) { System.out.println(t); }
System.out.println("\n---------- VERIFIED ONLY ----------");
feed.setMode(TimelineMode.VERIFIED_ONLY);
for (Tweet t : feed) { System.out.println(t); }
System.out.println("\n---------- HIGH LIKES RATIO ONLY ----------");
feed.setMode(TimelineMode.LIKE_RATIO);
for (Tweet t : feed) { System.out.println(t); }
```

This code should produce the following output:

```
---------- "REVERSE CHRONOLOGICAL" ORDER -----------
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- Welcome to the University of Wisconsin-Madison, #FutureBadger!
-- likes: 0
-- retweets: 0
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- It's a GREAT day for @BadgerMHockey! #OnWisconsin
-- likes: 0
-- retweets: 0
tweet from @dril at Mon Apr 03 17:07:50 CDT 2023:
-- type "Gaming is back ", if you think gaming is back
-- likes: 5243
-- retweets: 4702
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- Join us for Bucky's Big Event next Wednesday! #CelebrateUW
-- likes: 1
-- retweets: 0
tweet from @dril at Mon Apr 03 17:07:50 CDT 2023:
-- got botulism from a pair of bad jeans
-- likes: 307
-- retweets: 4015

---------- VERIFIED ONLY -----------
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- Welcome to the University of Wisconsin-Madison, #FutureBadger!
-- likes: 0
-- retweets: 0
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- It's a GREAT day for @BadgerMHockey! #OnWisconsin
-- likes: 0
-- retweets: 0
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- Join us for Bucky's Big Event next Wednesday! #CelebrateUW
-- likes: 1
-- retweets: 0

---------- HIGH LIKES RATIO ONLY -----------
tweet from @dril at Mon Apr 03 17:07:50 CDT 2023:
-- type "Gaming is back ", if you think gaming is back
-- likes: 5243
-- retweets: 4702
tweet from @uwmadison* at Mon Apr 03 17:07:50 CDT 2023:
-- Join us for Bucky's Big Event next Wednesday! #CelebrateUW
-- likes: 1
-- retweets: 0
```

You should also try adding `Tweet`s to the *middle* of your feed, removing some, or play around with whether a `User` is verified.

Assignment Submission

Once you're satisfied with your work, both in terms of adherence to this specification and the academic conduct and style guide requirements, submit your source code through Gradescope. For full credit, please submit ONLY the following files (source code, *not* .class files):

- ChronoTwiterator.java
- RatioTwiterator.java
- VerifiedTwiterator.java

(Iterators)

- User.java
- Tweet.java
- TweetNode.java
- TwitterFeed.java

(Linked List objects)

- TwiteratorTester.java

(Tester Class)

You do NOT need to submit **ListADT.java** or **TimelineMode.java**; these will be provided for you.

Your score for this assignment will be based on the submission marked "**active**" prior to the deadline. You may select which submission to mark active at any time, but by default this will be your most recent submission.

Copyright Notice