National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Faculty of Computing

SE-314: Software Construction

Class: BESE 13AB

Farzan Saqib

417633

BESE13 A

Lab 06: Test First Programming - II

CLO-03: Design and develop solutions based on Software Construction principles.
CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

Date: 21st Oct 2024

National University of Sciences and Technology (NUST)
School of Electrical Engineering and Computer Science

Time: 10:00 AM - 12:50 PM
02:00 PM – 04:50 PM

Instructor: Dr. Mehvish Rashid
Lab Engineer: Mr. Aftab Farooq

Lab 06: Test-First Programming - II

Introduction:

Students will have hands-on experience of test-first programming. Given a set of specifications, you will write unit tests that check for compliance with the specifications, and then implement code that meets the specifications.

Material:

https://ocw.mit.edu/ans7870/6/6.005/s16/psets/ps1/

Lectures on LMS regarding designing Specifications

Lab Tasks

Solve problem 3 of problem set 1 listed on the link. The goal of problem set is to build a toolbox of methods that can extract information from a set of tweets downloaded from Twitter.

Test-First Programming:

1. Study the specification of the method carefully.
2. Write JUnit tests for the method according to the spec.
3. Implement the method according to the spec.
4. Revise your implementation and improve your test cases until your implementation passes all your tests.

Task1:

Inferring a social network

In this problem, you will test and implement the methods in *SocialNetwork.java* . The *guessFollowsGraph()* method creates a social network over the people who are mentioned in a list of tweets. The social network is an approximation to who is following whom on Twitter, based only on the evidence found in the tweets.
Hint : The method analyzes a list of tweets to build a social network graph where each user is connected to the users they follow, based on the mentions in their tweets.

The *influencers()* method returns a list of people sorted by their influence (total number of followers).
Hint : The method analyzes a social network to find out who has the most followers and returns a list of those users, sorted from most to least influential.

a Devise, document, and implement test cases for *guessFollowsGraph()* and *influencers()* , and put them in SocialNetworkTest.java . Be careful that your test cases for *guessFollowsGraph()* respect its underdetermined postcondition.

b Implement *guessFollowsGraph()* and *influencers()* , and make sure your tests pass. For now, implement only the minimum required behaviour for *guessFollowsGraph()* , which infers that Ernie follows Bert if Ernie @-mentions Bert.

Test Cases should be :

1). Verifies that when an empty list of tweets is provided, the guessFollowsGraph function returns an empty graph.
2). Ensures that when there are tweets with no mentions in the list, the guessFollowsGraph function returns an empty graph.
3). Checks that the guessFollowsGraph function correctly identifies users who are mentioned in tweets, and associates them with the mentioned users in the graph.
4). Validates that the guessFollowsGraph function correctly associates multiple mentioned users with the authors of the tweets.
5). Verifies that the guessFollowsGraph function handles multiple tweets by the same author and associates all mentioned users with the author in the graph.
6). Checks that when the input followsGraph is empty, the influencers function returns an empty list of influencers.
7). Verifies that when there's only one user in the followsGraph with no followers, the influencers function returns an empty list.
8). Ensures that when there's a single influencer in the followsGraph, the influencers function correctly identifies and returns that influencer as the only user in the list.
9). Checks that the influencers function can correctly identify and return the top influencers when there are multiple users in the followsGraph with varying numbers of followers.
10). Validates that the influencers function correctly handles cases where multiple users have an equal number of followers and returns them in any order.

If you want to see your code work on a live sample of tweets, run Main.java . It will print the top 10 most-followed people according to the social network you generated. You can search for them on Twitter to see if their actual number of followers has a similar order.

Code:

```
```

```
/* Copyright (c) 2007-2016 MIT 6.005 course staff, all rights reserved.
 * Redistribution of original or derived work requires permission of course
staff.
 */
package twitter;

import java.util.*;
import java.util.stream.Collectors;

/**
 * SocialNetwork provides methods that operate on a social network.
 *
 * A social network is represented by a Map<String, Set<String>> where map[A]
is
 * the set of people that person A follows on Twitter, and all people are
 * represented by their Twitter usernames. Users can't follow themselves. If
A
 * doesn't follow anybody, then map[A] may be the empty set, or A may not
even exist
 * as a key in the map; this is true even if A is followed by other people in
the network.
 * Twitter usernames are not case sensitive, so "ernie" is the same as
"ERNie".
 * A username should appear at most once as a key in the map or in any given
 * map[A] set.
 *
 * DO NOT change the method signatures and specifications of these methods,
but
 * you should implement their method bodies, and you may add new public or
 * private methods or classes if you like.
 */
public class SocialNetwork {

    /**
     * Guess who might follow whom, from evidence found in tweets.
     *
     * @param tweets
     *            a list of tweets providing the evidence, not modified by
this
     *            method.
     * @return a social network (as defined above) in which Ernie follows
Bert
```

```
     *         if and only if there is evidence for it in the given list of
     *         tweets.
     *         One kind of evidence that Ernie follows Bert is if Ernie
     *         @-mentions Bert in a tweet. This must be implemented. Other
kinds
     *         of evidence may be used at the implementor's discretion.
     *         All the Twitter usernames in the returned social network must
be
     *         either authors or @-mentions in the list of tweets.
     */
    public static Map<String, Set<String>> guessFollowsGraph(List<Tweet>
tweets) {
        Map<String, Set<String>> followsGraph = new HashMap<>();

        for (Tweet tweet : tweets) {
            String author = tweet.getAuthor().toLowerCase();
            Set<String> mentionedUsers =
Extract.getMentionedUsers(Arrays.asList(tweet));

            followsGraph.putIfAbsent(author, new HashSet<>());
            followsGraph.get(author).addAll(mentionedUsers);
        }

        return followsGraph;
    }


    /**
     * Find the people in a social network who have the greatest influence,
in
     * the sense that they have the most followers.
     *
     * @param followsGraph
     *            a social network (as defined above)
     * @return a list of all distinct Twitter usernames in followsGraph, in
     *         descending order of follower count.
     */
    public static List<String> influencers(Map<String, Set<String>>
followsGraph) {
        Map<String, Integer> followerCounts = new HashMap<>();

        // Count followers
        for (Set<String> followedUsers : followsGraph.values()) {
            for (String user : followedUsers) {
                followerCounts.put(user, followerCounts.getOrDefault(user, 0)
+ 1);
            }
        }
```

```
        // Sort users by follower count
        return followerCounts.entrySet()
                .stream()
                .sorted((e1, e2) -> e2.getValue() - e1.getValue())
                .map(Map.Entry::getKey)
                .collect(Collectors.toList());
    }


}
```

```

## SocialNetworkTest.java

```
public class SocialNetworkTest {

    @Test(expected=AssertionError.class)
    public void testAssertionsEnabled() {
        assert false; // ensure assertions are enabled
    }

    // 1. Verifies that an empty list of tweets returns an empty graph
    @Test
    public void testGuessFollowsGraphEmptyList() {
        Map<String, Set<String>> followsGraph =
SocialNetwork.guessFollowsGraph(new ArrayList<>());
        assertTrue("Expected empty graph", followsGraph.isEmpty());
    }

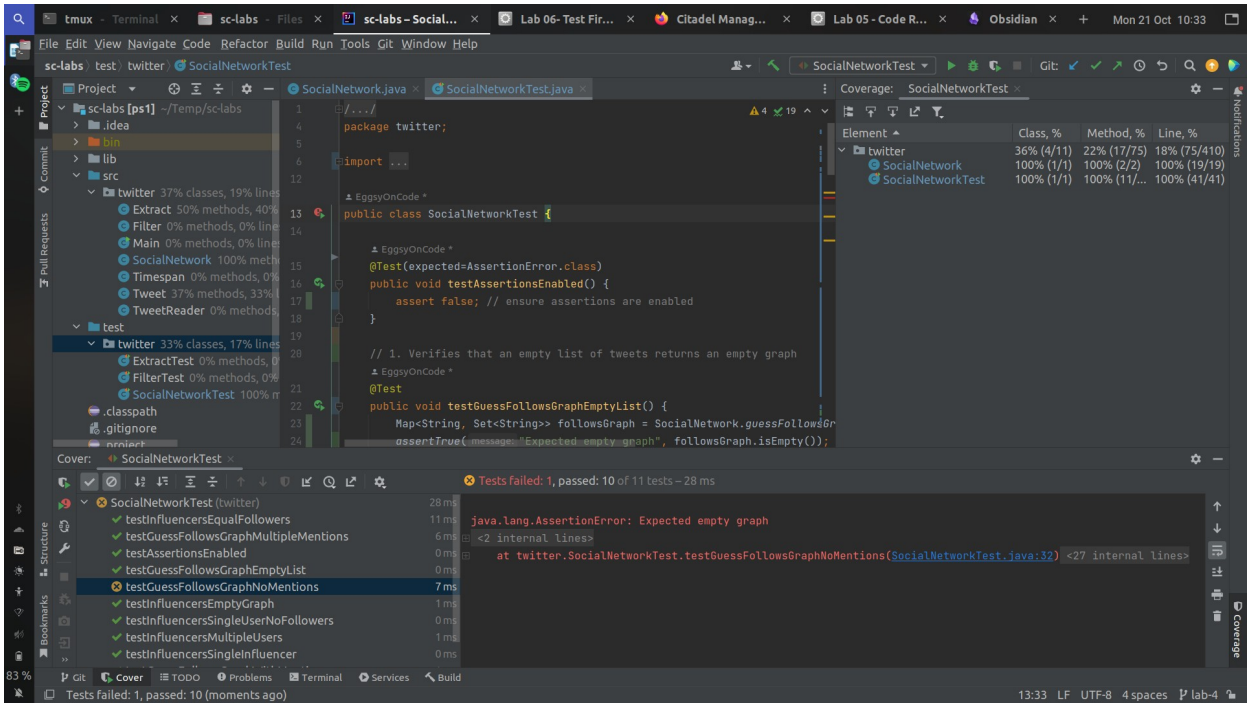
    // 2. Ensures tweets with no mentions return an empty graph
    @Test
    public void testGuessFollowsGraphNoMentions() {
        List<Tweet> tweets = Arrays.asList(new Tweet(1, "user1", "Hello
world", Instant.now()));
        Map<String, Set<String>> followsGraph =
SocialNetwork.guessFollowsGraph(tweets);
        assertTrue("Expected empty graph", followsGraph.isEmpty());
    }

    // 3. Checks correct identification and association of mentioned users
    @Test
    public void testGuessFollowsGraphWithMentions() {
        List<Tweet> tweets = Arrays.asList(new Tweet(1, "user1", "@user2
Hello", Instant.now()));
```
```

```
        Map<String, Set<String>> followsGraph =
SocialNetwork.guessFollowsGraph(tweets);
        assertTrue("user1 should follow user2",
followsGraph.get("user1").contains("user2"));
    }

    // 4. Validates multiple mentioned users associated with the author
    @Test
    public void testGuessFollowsGraphMultipleMentions() {
        List<Tweet> tweets = Arrays.asList(new Tweet(1, "user1", "@user2
@user3 Hello", Instant.now()));
        Map<String, Set<String>> followsGraph =
SocialNetwork.guessFollowsGraph(tweets);
        assertTrue("user1 should follow user2 and user3",
                followsGraph.get("user1").containsAll(Arrays.asList("user2",
"user3")));
    }

    // 5. Verifies multiple tweets from the same author
    @Test
    public void testGuessFollowsGraphMultipleTweetsSameAuthor() {
        List<Tweet> tweets = Arrays.asList(
                new Tweet(1, "user1", "@user2 Hello", Instant.now()),
                new Tweet(2, "user1", "@user3 Hi again", Instant.now())
        );
        Map<String, Set<String>> followsGraph =
SocialNetwork.guessFollowsGraph(tweets);
        assertTrue("user1 should follow user2 and user3",
                followsGraph.get("user1").containsAll(Arrays.asList("user2",
"user3")));
    }

    // 6. Empty followsGraph returns an empty influencers list
    @Test
    public void testInfluencersEmptyGraph() {
        Map<String, Set<String>> followsGraph = new HashMap<>();
        List<String> influencers = SocialNetwork.influencers(followsGraph);
        assertTrue("Expected empty list of influencers",
influencers.isEmpty());
    }

    // 7. Single user with no followers returns an empty influencers list
    @Test
    public void testInfluencersSingleUserNoFollowers() {
        Map<String, Set<String>> followsGraph = new HashMap<>();
        followsGraph.put("user1", new HashSet<>());
        List<String> influencers = SocialNetwork.influencers(followsGraph);
        assertTrue("Expected empty list of influencers",
```

```
influencers.isEmpty());
    }

    // 8. Single influencer in the graph
    @Test
    public void testInfluencersSingleInfluencer() {
        Map<String, Set<String>> followsGraph = new HashMap<>();
        followsGraph.put("user1", Set.of("user2"));
        List<String> influencers = SocialNetwork.influencers(followsGraph);
        assertEquals("Expected user2 as the only influencer", "user2",
influencers.get(0));
    }

    // 9. Multiple users with varying followers
    @Test
    public void testInfluencersMultipleUsers() {
        Map<String, Set<String>> followsGraph = new HashMap<>();
        followsGraph.put("user1", Set.of("user2", "user3"));
        followsGraph.put("user2", Set.of("user3"));
        List<String> influencers = SocialNetwork.influencers(followsGraph);
        assertEquals("Expected user3 as top influencer", "user3",
influencers.get(0));
    }

    // 10. Multiple users with equal followers
    @Test
    public void testInfluencersEqualFollowers() {
        Map<String, Set<String>> followsGraph = new HashMap<>();
        followsGraph.put("user1", Set.of("user2"));
        followsGraph.put("user3", Set.of("user2"));
        List<String> influencers = SocialNetwork.influencers(followsGraph);
        assertTrue("user2 should be the influencer",
influencers.contains("user2"));
    }

}
```

Test Execution :



The test is failing because the `SocialNetwork.guessFollowsGraph()` method might still be adding the tweet authors to the graph, even when no mentions are present.

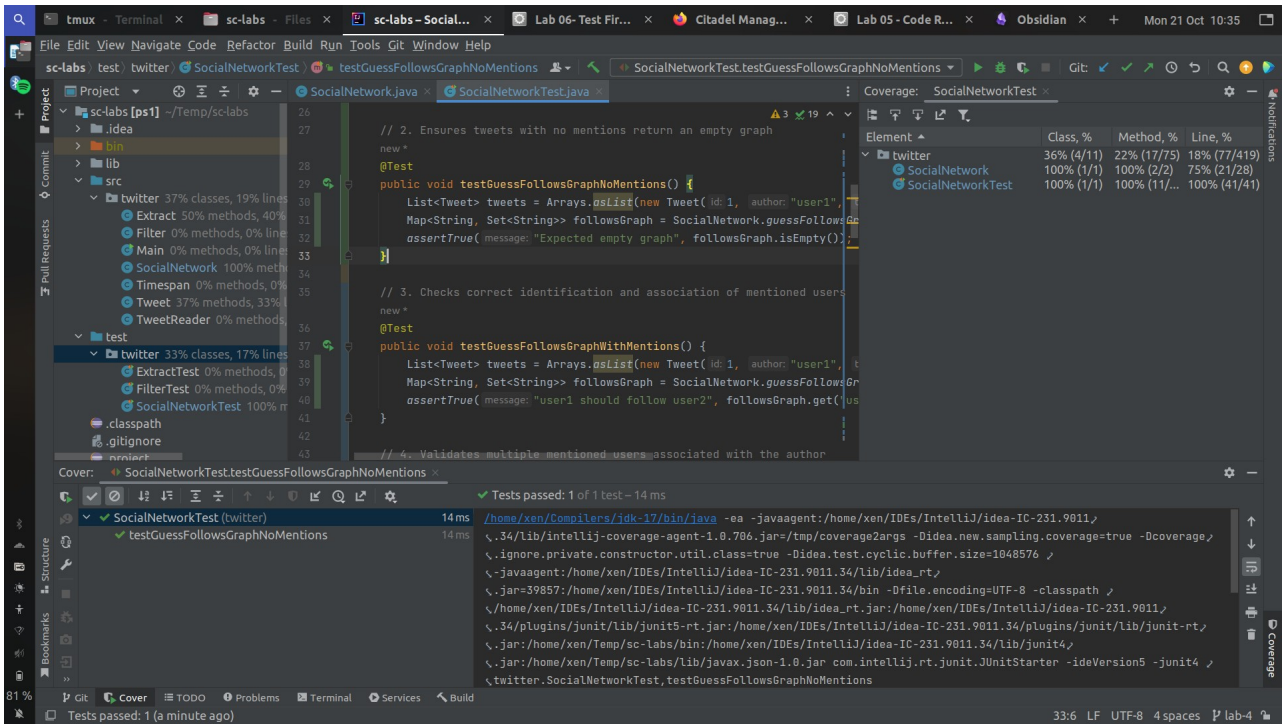
To fix this, we ensure that the `guessFollowsGraph()` method in `SocialNetwork.java` only adds an entry for a user if they mention someone in their tweet.

Updated Implementation:

```

```

```
public static Map<String, Set<String>> guessFollowsGraph(List<Tweet> tweets) {
    Map<String, Set<String>> followsGraph = new HashMap<>();

    for (Tweet tweet : tweets) {
        String author = tweet.getAuthor().toLowerCase();
        Set<String> mentions = Extract.getMentionedUsers(Arrays.asList(tweet));
```

```
        // Only add to the graph if the tweet contains mentions
        if (!mentions.isEmpty()) {
            followsGraph.putIfAbsent(author, new HashSet<>());
            followsGraph.get(author).addAll(mentions);
        }
    }

    return followsGraph;
}
```

```

# 2nd Test Run:




Test is passing!
```

Breakdown:

Methods:

1. **`guessFollowsGraph()`**: This method analyzes a list of tweets and builds a social network where users are linked to those they "follow," inferred from the users they mention in their tweets. It creates a graph (Map) where keys are users and values are sets of mentioned users.

2. **`influencers()`**: This method returns a sorted list of users by their influence, measured by how many people follow them.

Test Cases:

1. **Empty List of Tweets**: Ensures that an empty list results in an empty graph.
2. **Tweets Without Mentions**: Verifies that tweets with no mentions do not add entries to the graph.
3. **Single Mention**: Tests whether a user who mentions someone is correctly added to the graph.
4. **Multiple Mentions**: Checks if multiple mentioned users are linked to the tweet author.
5. **Multiple Tweets from One User**: Ensures that repeated mentions from the same user are captured.
6. **Empty Graph for `influencers()`**: Verifies that no users yield an empty influencer list.
7. **Single User Without Followers**: Tests that a user without followers yields no influencers.
8. **Single Influencer**: Verifies correct identification of the only influencer.
9. **Multiple Influencers**: Tests for correct influencer ordering.
10. **Tied Influence**: Ensures equal influencers are handled correctly.

Solution
Please provide the link to your GitHub repository in a Word document. Additionally, paste your source code into the same document. Once completed, upload the file to the LMS.

Deliverables:

In case of any problems with submissions on LMS, submit your Lab assignments by emailing it to aftab.farooq@seecs.edu.pk.