**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# 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**

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

|  |  |
| --- | --- |
| **NAME** | **MUHAMMAD SAAD UMER** |
| **CLASS** | **BESE 13-A** |
| **CMS** | **408485** |

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

**Introduction:**

# Lab 06: Test-First Programming - II

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.

* 1. 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.
  2. 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.

|  |
| --- |
| **SOLUTION** |
| SocialNetwork.java  **package** twitter;  **import** java.util.\*;  **public** **class** SocialNetwork {  **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();  followsGraph.putIfAbsent(author, **new** HashSet<>());  // Extract mentioned usernames from the tweet text  String[] words = tweet.getText().split("\\s+");  **for** (String word : words) {  **if** (word.startsWith("@")) {  String mentionedUser = word.substring(1).toLowerCase();  followsGraph.get(author).add(mentionedUser);  followsGraph.putIfAbsent(mentionedUser, **new** HashSet<>());  }  }  }  **return** followsGraph;  }  **public** **static** List<String> influencers(Map<String, Set<String>> followsGraph) {  Map<String, Integer> followerCounts = **new** HashMap<>();  // Count followers for each user  **for** (Set<String> followers : followsGraph.values()) {  **for** (String user : followers) {  followerCounts.put(user, followerCounts.getOrDefault(user, 0) + 1);  }  }  // Find the maximum number of followers  **int** maxFollowers = 0;  **for** (**int** count : followerCounts.values()) {  maxFollowers = Math.*max*(maxFollowers, count);  }  // Collect users with the maximum followers  List<String> topInfluencers = **new** ArrayList<>();  **for** (Map.Entry<String, Integer> entry : followerCounts.entrySet()) {  **if** (entry.getValue() == maxFollowers) {  topInfluencers.add(entry.getKey());  }  }  **return** topInfluencers;  }  }  SocialNetworkTest.java  **package** twitter;  **import** **static** org.junit.Assert.\*;  **import** java.time.Instant;  **import** java.util.\*;  **import** org.junit.Test;  **public** **class** SocialNetworkTest {  @Test(expected = AssertionError.**class**)  **public** **void** testAssertionsEnabled() {  **assert** **false**; // Ensure assertions are enabled with VM argument: -ea  }  @Test  **public** **void** testGuessFollowsGraphEmpty() {  Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(**new** ArrayList<>());  *assertTrue*("expected empty graph", followsGraph.isEmpty());  }  @Test  **public** **void** testGuessFollowsGraphNoMentions() {  List<Tweet> tweets = Arrays.*asList*(  **new** Tweet(1, "Ali", "Hello world!", Instant.*now*()),  **new** Tweet(2, "Fatima", "Good morning!", Instant.*now*())  );  Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);  *assertEquals*(2, followsGraph.size());  *assertTrue*(followsGraph.get("ali").isEmpty());  *assertTrue*(followsGraph.get("fatima").isEmpty());  }  @Test  **public** **void** testGuessFollowsGraphWithMentions() {  List<Tweet> tweets = Arrays.*asList*(  **new** Tweet(1, "Ali", "Hello @Fatima!", Instant.*now*()),  **new** Tweet(2, "Fatima", "Hi @Ali!", Instant.*now*())  );  Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);  }  @Test  **public** **void** testGuessFollowsGraphMultipleMentions() {  List<Tweet> tweets = Arrays.*asList*(  **new** Tweet(1, "Ali", "Hello @Fatima and @Zain!", Instant.*now*()),  **new** Tweet(2, "Fatima", "Hi @Ali!", Instant.*now*())  );  Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);  *assertTrue*(followsGraph.get("ali").contains("fatima"));  }  @Test  **public** **void** testGuessFollowsGraphMultipleTweetsBySameAuthor() {  List<Tweet> tweets = Arrays.*asList*(  **new** Tweet(1, "Ali", "Hello @Fatima!", Instant.*now*()),  **new** Tweet(2, "Ali", "Also, @Zain is great.", Instant.*now*())  );  Map<String, Set<String>> followsGraph = SocialNetwork.*guessFollowsGraph*(tweets);  *assertEquals*(3, followsGraph.size());  }  @Test  **public** **void** testInfluencersEmpty() {  Map<String, Set<String>> followsGraph = **new** HashMap<>();  List<String> influencers = SocialNetwork.*influencers*(followsGraph);  *assertTrue*("expected empty list", influencers.isEmpty());  }  @Test  **public** **void** testInfluencersSingleUserNoFollowers() {  Map<String, Set<String>> followsGraph = **new** HashMap<>();  followsGraph.put("Ali", **new** HashSet<>());  List<String> influencers = SocialNetwork.*influencers*(followsGraph);  *assertTrue*("expected empty list", influencers.isEmpty());  }  @Test  **public** **void** testInfluencersSingleInfluencer() {  Map<String, Set<String>> followsGraph = **new** HashMap<>();  followsGraph.put("Ali", **new** HashSet<>(Arrays.*asList*("Fatima")));  List<String> influencers = SocialNetwork.*influencers*(followsGraph);  *assertEquals*(1, influencers.size());  *assertEquals*("fatima", influencers.get(0).toLowerCase());  }  @Test  **public** **void** testInfluencersMultipleUsers() {  Map<String, Set<String>> followsGraph = **new** HashMap<>();  followsGraph.put("Ali", **new** HashSet<>(Arrays.*asList*("Fatima")));  followsGraph.put("Fatima", **new** HashSet<>(Arrays.*asList*("Ali", "Zain")));  followsGraph.put("Zain", **new** HashSet<>(Arrays.*asList*("Ali")));  List<String> influencers = SocialNetwork.*influencers*(followsGraph);  *assertEquals*(1, influencers.size());  }  @Test  **public** **void** testInfluencersEqualFollowers() {  Map<String, Set<String>> followsGraph = **new** HashMap<>();  followsGraph.put("Ali", **new** HashSet<>(Arrays.*asList*("Fatima", "Zain")));  followsGraph.put("Fatima", **new** HashSet<>(Arrays.*asList*("Ali")));  followsGraph.put("Zain", **new** HashSet<>(Arrays.*asList*("Ali")));  List<String> influencers = SocialNetwork.*influencers*(followsGraph);  *assertEquals*(1, influencers.size());  *assertEquals*("ali", influencers.get(0).toLowerCase());  }  } |

Deliverables:

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