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

Faculty of Computing

SE-314: Software Construction

Class: BESE 13B

LAB 04: TEST FIRST PROGRAMMING - I

Ibrahim Qaiser

CMS: 405459

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

TABLE OF CONTENTS

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

LAB 04: TEST- FIRST PROGRAMMING: TWEET TWEET

LAB TASKS:

Solve problem 1 and 2 of problem set 1 listed on the link. The goal of the 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.

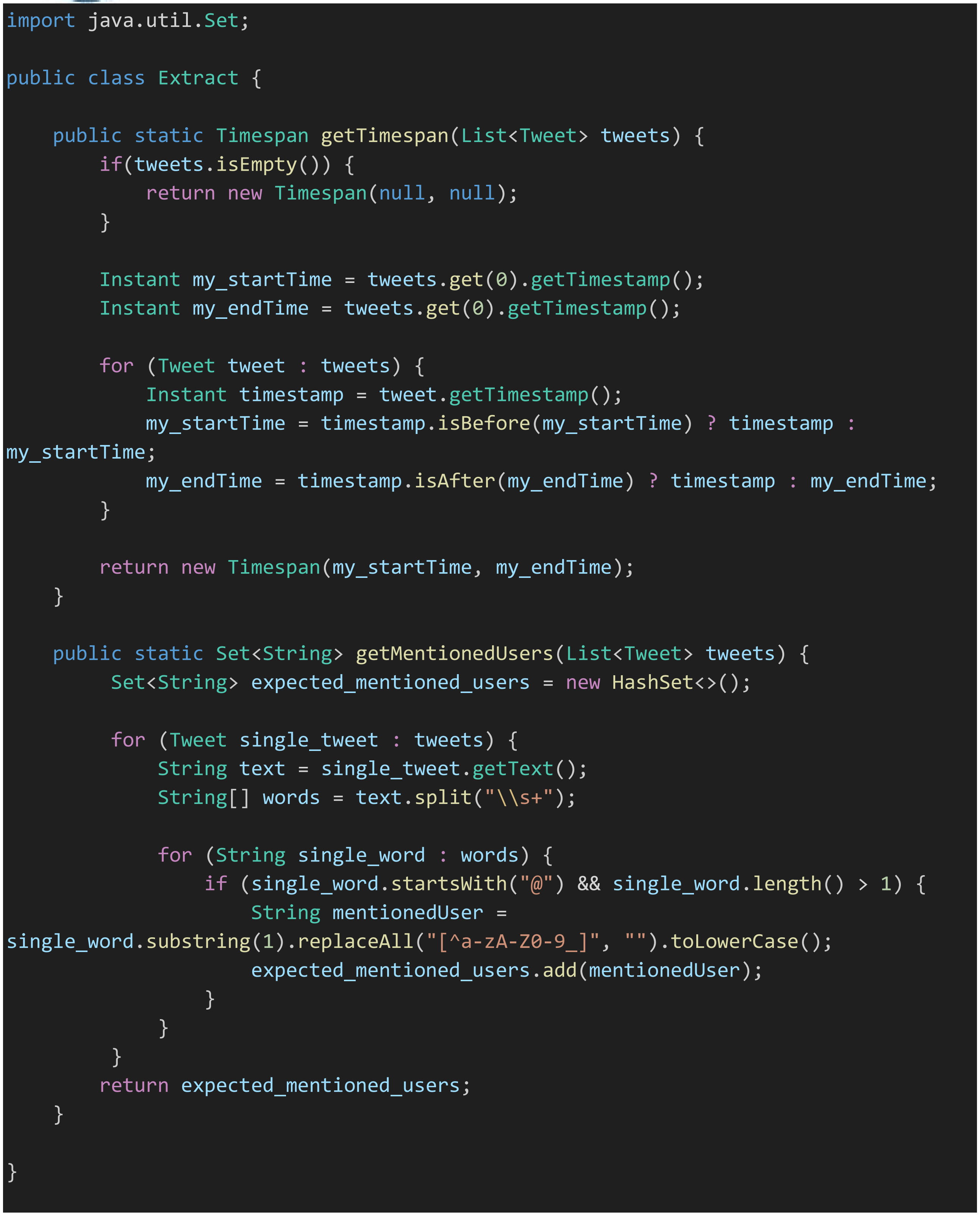
Task1: Extracting data from Tweets

In this problem, you will test and implement the methods in **Extract.java**. You'll find **Extract.java** in the **src** folder, and a JUnit test class **ExtractTest.java** in the test folder. Separating implementation code from test code is a common practice in development projects. It makes the implementation code easier to understand, uncluttered by tests, and easier to package up for release

a. Devise, document, and implement test cases for **getTimespan()** and **getMentionedUsers()** , and put them in **ExtractTest.java .**
b. Implement **getTimespan()** and **getMentionedUsers()** , and make sure your tests pass.

Extract.java file

```
/* 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.time.Instant;
import java.util.HashSet;
import java.util.List;
```

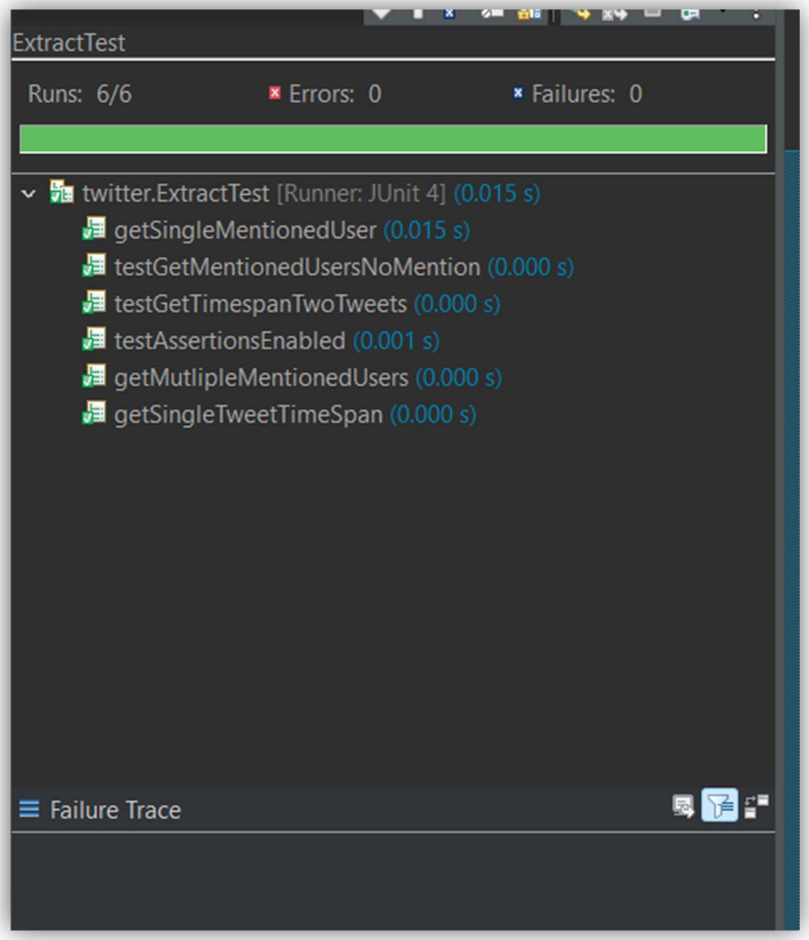


```java
import java.util.Set;

public class Extract {

    public static Timespan getTimespan(List<Tweet> tweets) {
        if(tweets.isEmpty()) {
            return new Timespan(null, null);
        }

        Instant my_startTime = tweets.get(0).getTimestamp();
        Instant my_endTime = tweets.get(0).getTimestamp();

        for (Tweet tweet : tweets) {
            Instant timestamp = tweet.getTimestamp();
            my_startTime = timestamp.isBefore(my_startTime) ? timestamp :
my_startTime;
            my_endTime = timestamp.isAfter(my_endTime) ? timestamp : my_endTime;
        }

        return new Timespan(my_startTime, my_endTime);
    }

    public static Set<String> getMentionedUsers(List<Tweet> tweets) {
         Set<String> expected_mentioned_users = new HashSet<>();

         for (Tweet single_tweet : tweets) {
             String text = single_tweet.getText();
             String[] words = text.split("\\s+");

             for (String single_word : words) {
                 if (single_word.startsWith("@") && single_word.length() > 1) {
                     String mentionedUser =
single_word.substring(1).replaceAll("[^a-zA-Z0-9_]", "").toLowerCase();
                     expected_mentioned_users.add(mentionedUser);
                 }
             }
         }
        return expected_mentioned_users;
    }

}
```

ExtractTest.java file

```java
/* 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 static org.junit.Assert.*;

import java.time.Instant;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashSet;
import java.util.Set;

import org.junit.Test;

public class ExtractTest {
    private static final Instant d1 = Instant.parse("2016-02-17T10:00:00Z");
    private static final Instant d2 = Instant.parse("2016-02-17T11:00:00Z");

    private static final Tweet tweet1 = new Tweet(1, "alyssa", "is it reasonable
to talk about rivest so much?", d1);
    private static final Tweet tweet2 = new Tweet(2, "bbitdiddle", "rivest talk in
30 minutes #hype", d2);

    @Test(expected=AssertionError.class)
    public void testAssertionsEnabled() {
        assert false; // make sure assertions are enabled with VM argument: -ea
    }

    @Test
    public void testGetTimespanTwoTweets() {
        Timespan timespan = Extract.getTimespan(Arrays.asList(tweet1, tweet2));

        assertEquals("expected start", d1, timespan.getStart());
        assertEquals("expected end", d2, timespan.getEnd());
    }

    @Test
    public void testGetMentionedUsersNoMention() {
        Set<String> mentionedUsers =
Extract.getMentionedUsers(Arrays.asList(tweet1));

        assertTrue("expected empty set", mentionedUsers.isEmpty());
    }

    @Test
    public void getSingleTweetTimeSpan() {
        // THis test asserts when only single tweet is available in the input list
of tweets
        Timespan timespan =
Extract.getTimespan(Collections.singletonList(tweet1));
        assertEquals( d1, timespan.getStart());
        assertEquals( d1, timespan.getEnd());
    }

    @Test
    public void getSingleMentionedUser() {
        // This test asserts when certain tweet metnions a user in it
        Tweet tweet = new Tweet(3, "nigger", "Hey @nigger, how are you?",
Instant.parse("2024-02-25T12:30:00Z"));
        Set<String> mentionedUsers =
Extract.getMentionedUsers(Collections.singletonList(tweet));
        Set<String> expectedMentions = new HashSet<>(Arrays.asList("nigger"));
        assertEquals("mentioned user should be in the set", expectedMentions,
mentionedUsers);
    }

    @Test
    public void getMutlipleMentionedUsers() {
        // This is the test when different tweets mention different users in them
        Tweet tweet1 = new Tweet(4, "nigger", "Hi @nigger!", Instant.parse("2024-
02-25T12:30:00Z"));
        Tweet tweet2 = new Tweet(5, "bigger", "Hello @bigger!",
Instant.parse("2024-02-25T12:30:00Z"));
        Set<String> my_mentionedUsers =
Extract.getMentionedUsers(Arrays.asList(tweet1, tweet2));
        Set<String> my_expectedMentions = new HashSet<>(Arrays.asList("nigger",
"bigger"));
        assertEquals("mentioned users in the set", my_expectedMentions,
my_mentionedUsers);
    }
}
```

Test cases successful



```
ExtractTest
Runs: 6/6        Errors: 0        Failures: 0

twitter.ExtractTest [Runner: JUnit 4] (0.015 s)
    getSingleMentionedUser (0.015 s)
    testGetMentionedUsersNoMention (0.000 s)
    testGetTimespanTwoTweets (0.000 s)
    testAssertionsEnabled (0.001 s)
    getMutlipleMentionedUsers (0.000 s)
    getSingleTweetTimeSpan (0.000 s)

Failure Trace
```

Task2: Filtering lists of Tweets

In this problem, you will test and implement the methods in **Filter.java** .

a Devise, document, and implement test cases for **writtenBy()** , **inTimespan()** ,and **containing()** , and put them in **FilterTest.java** .

b Implement **writtenBy()** , **inTimespan()** , and **containing()** , and make sureyour tests pass.

Hints:

- For questions about lowercase/uppercase and how to interpret timespans, reread the hints in the previous question.
- For all problems on this problem set, you are free to rewrite or replace the provided example tests and their assertions.

Filter.java file

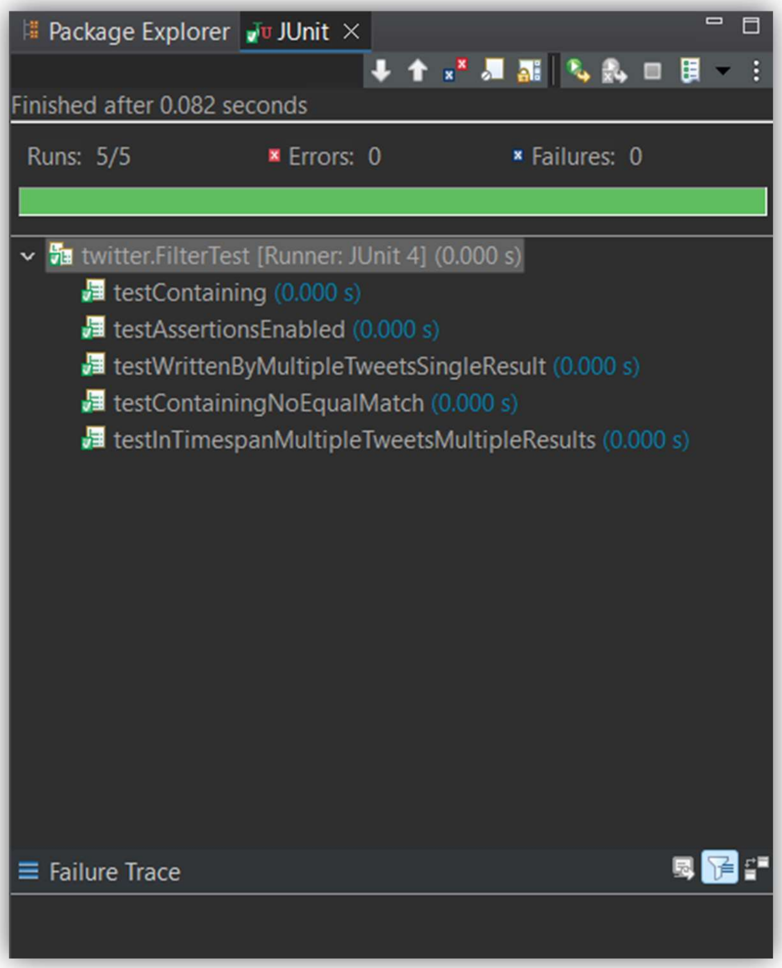
```java
/* Copyright (c) 2007-2016 MIT 6.005 course staff, all rights reserved.
 * Redistribution of original or derived work requires permission of course staff.
 */
package twitter;

import java.util.ArrayList;
import java.util.List;

public class Filter {

    public static List<Tweet> writtenBy(List<Tweet> tweets, String username) {
        List<Tweet> result = new ArrayList<>();
        for (Tweet tweet : tweets) {
            if (tweet.getAuthor().equalsIgnoreCase(username)) {
                result.add(tweet);
            }
        }
        return result;
    }

    public static List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {
         List<Tweet> result = new ArrayList<>();
         for (Tweet tweet : tweets) {
             if (timespan.getStart().isBefore(tweet.getTimestamp()) &&
timespan.getEnd().isAfter(tweet.getTimestamp())) {
                 result.add(tweet);
             }
         }
         return result;
    }


    public static List<Tweet> containing(List<Tweet> tweets, List<String> words) {
        List<Tweet> expected_filtered_tweets = new ArrayList<>();
        for (Tweet singleTweet : tweets) {
            String text = singleTweet.getText().toLowerCase();

            for (String word : words) {
                if (text.contains(word.toLowerCase())) {
                    expected_filtered_tweets.add(singleTweet);
                    break;
                }
            }
        }

        return expected_filtered_tweets;
    }

}
```

FilterTest.java file

```
    package twitter;

    import static org.junit.Assert.*;

    import java.time.Instant;
    import java.util.Arrays;
    import java.util.List;

    import org.junit.Test;

    public class FilterTest {
        private static final Instant d1 = Instant.parse("2016-02-
    17T10:00:00Z");
        private static final Instant d2 = Instant.parse("2016-02-
    17T11:00:00Z");

        private static final Tweet tweet1 = new Tweet(1, "alyssa", "is it
    reasonable to talk about rivest so much?", d1);
    private static final Tweet tweet2 = new Tweet(2, "bbitdiddle", "rivest
talk in 30 minutes #hype", d2);

    @Test(expected=AssertionError.class)
    public void testAssertionsEnabled() {
        assert false; // make sure assertions are enabled with VM argument:
-ea
    }

    @Test
    public void testWrittenByMultipleTweetsSingleResult() {
        List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1,
tweet2), "alyssa");

        assertEquals("expected singleton list", 1, writtenBy.size());
        assertTrue("expected list to contain tweet",
writtenBy.contains(tweet1));
    }

    @Test
    public void testInTimespanMultipleTweetsMultipleResults() {
        Instant testStart = Instant.parse("2016-02-17T09:00:00Z");
        Instant testEnd = Instant.parse("2016-02-17T12:00:00Z");

        List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1,
tweet2), new Timespan(testStart, testEnd));

        assertFalse("expected non-empty list", inTimespan.isEmpty());
        assertTrue("expected list to contain tweets",
inTimespan.containsAll(Arrays.asList(tweet1, tweet2)));
        assertEquals("expected same order", 0, inTimespan.indexOf(tweet1));
    }

    @Test
    public void testContaining() {
        List<Tweet> containing = Filter.containing(Arrays.asList(tweet1,
tweet2), Arrays.asList("talk"));

        assertFalse("expected non-empty list", containing.isEmpty());
        assertTrue("expected list to contain tweets",
containing.containsAll(Arrays.asList(tweet1, tweet2)));
        assertEquals("expected same order", 0, containing.indexOf(tweet1));
    }
```

```
        @Test
        public void testContainingNoEqualMatch() {
            List<Tweet> containing = Filter.containing(Arrays.asList(tweet1,
    tweet2), Arrays.asList("apple"));
            assertTrue("expected empty list", containing.isEmpty());
        }

    }
```

Test cases successful



```
Finished after 0.082 seconds

Runs:  5/5          Errors:  0          Failures:  0

twitter.FilterTest [Runner: JUnit 4] (0.000 s)
    testContaining (0.000 s)
    testAssertionsEnabled (0.000 s)
    testWrittenByMultipleTweetsSingleResult (0.000 s)
    testContainingNoEqualMatch (0.000 s)
    testInTimespanMultipleTweetsMultipleResults (0.000 s)
```