**Faculty of Computing**

**SE-314: Software Construction**

**Name: Malik Shahzaib Khan**

**CMS ID: 406702**

**Class: BESE 13A**

# Lab 04: tesT first programming - i

**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: 30th Sep 2024**

**Time: 10:00 AM** **- 12:50 PM   
 02:30 PM – 04:50 PM**

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

# Lab 04: Test- First Programming: Tweet Tweet

## 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** and **Testing**

## Lab Tasks:

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

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

If you want to see your code work on a live sample of tweets, you can run **Main.java** . ( Main.java will not be used in grading, and you are free to edit it as you wish.)

### Hints:

* Note that we use the class **Instant** to represent the date and time of tweets. You can check **this article on Java 8 dates and times** to learn how to use **Instant** .
* You may wonder what to do about lowercase and uppercase in the return value of **getMentionedUsers()** . This spec has an underdetermined postcondition, so read the spec carefully and think about what that means for your implementation and your test cases.
* **getTimespan()** *also* has an underdetermined postcondition in some circumstances, which gives the implementor (you) more freedom and the client (also you, when you’re writing tests) less certainty about what it will return.
* Read the spec for the **Timespan** class carefully, because it may answer many of the questions you have about **getTimespan()** .

## Solution:

**Extract.java:**

**package** twitter;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** java.util.regex.Matcher;

**import** java.util.regex.Pattern;

**import** java.time.Instant;

**public** **class** Extract {

**public** **static** Timespan getTimespan(List<Tweet> tweets) {

**if** (tweets.isEmpty()) {

**return** **new** Timespan(Instant.*now*(), Instant.*now*());

}

Instant start = tweets.get(0).getTimestamp();

Instant end = tweets.get(0).getTimestamp();

**for** (Tweet tweet : tweets) {

Instant tweetTime = tweet.getTimestamp();

**if** (tweetTime.isBefore(start)) {

start = tweetTime;

}

**if** (tweetTime.isAfter(end)) {

end = tweetTime;

}

}

**return** **new** Timespan(start, end);

}

**public** **static** Set<String> getMentionedUsers(List<Tweet> tweets) {

Set<String> mentionedUsers = **new** HashSet<>();

**for** (Tweet tweet : tweets) {

String text = tweet.getText();

String regex = "(?<![\\w.-])@([\\w\_]+)(?![\\w.-])";

Matcher matcher = Pattern.*compile*(regex).matcher(text);

**while** (matcher.find()) {

String username = matcher.group(1).toLowerCase();

mentionedUsers.add(username);

}

}

**return** mentionedUsers;

}

}

**ExtractTest.java:**

**package** twitter;

**import** **static** org.junit.Assert.\*;

**import** java.time.Instant;

**import** java.util.Arrays;

**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**;

}

@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** testGetTimespanSingleTweet() {

Timespan timespan = Extract.*getTimespan*(Arrays.*asList*(***tweet1***));

*assertEquals*("expected start", ***d1***, timespan.getStart());

*assertEquals*("expected end", ***d1***, timespan.getEnd());

}

@Test

**public** **void** testGetTimespanEmptyList() {

Timespan timespan = Extract.*getTimespan*(Arrays.*asList*());

// For simplicity, we expect the start and end to be the current time

Instant now = Instant.*now*();

*assertEquals*("expected start", now.truncatedTo(java.time.temporal.ChronoUnit.***SECONDS***),

timespan.getStart().truncatedTo(java.time.temporal.ChronoUnit.***SECONDS***));

*assertEquals*("expected end", now.truncatedTo(java.time.temporal.ChronoUnit.***SECONDS***),

timespan.getEnd().truncatedTo(java.time.temporal.ChronoUnit.***SECONDS***));

}

@Test

**public** **void** testGetMentionedUsersSingleMention() {

Tweet tweetWithMention = **new** Tweet(3, "user1", "Hello @user2, how are you?", ***d1***);

Set<String> mentionedUsers = Extract.*getMentionedUsers*(Arrays.*asList*(tweetWithMention));

*assertTrue*("expected mentioned users to contain user2", mentionedUsers.contains("user2"));

}

@Test

**public** **void** testGetMentionedUsersMultipleMentions() {

Tweet tweetWithMultipleMentions = **new** Tweet(4, "user3", "Hey @user4 and @User2!", ***d2***);

Set<String> mentionedUsers = Extract.*getMentionedUsers*(Arrays.*asList*(tweetWithMultipleMentions));

*assertTrue*("expected mentioned users to contain user2", mentionedUsers.contains("user2"));

*assertTrue*("expected mentioned users to contain user4", mentionedUsers.contains("user4"));

*assertEquals*("expected size to be 2", 2, mentionedUsers.size());

}

@Test

**public** **void** testGetMentionedUsersInvalidMention() {

Tweet tweetWithInvalidMention = **new** Tweet(5, "user5", "Contact me at @invalid-email@example.com", ***d1***);

Set<String> mentionedUsers = Extract.*getMentionedUsers*(Arrays.*asList*(tweetWithInvalidMention));

*assertTrue*("expected empty set for invalid mentions", mentionedUsers.isEmpty());

}

@Test

**public** **void** testGetMentionedUsersCaseInsensitivity() {

Tweet tweetWithMention = **new** Tweet(6, "user6", "Hey @User7!", ***d1***);

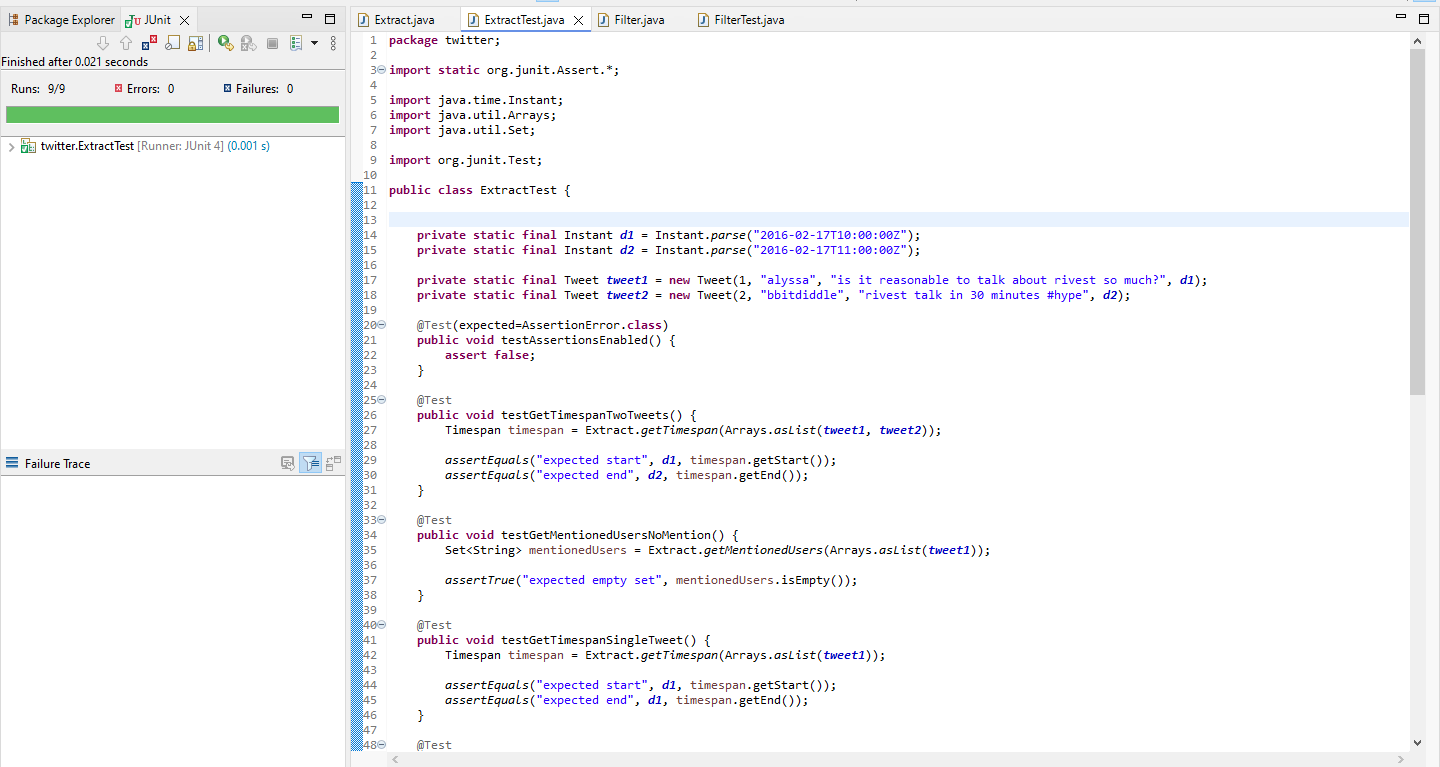
Set<String> mentionedUsers = Extract.*getMentionedUsers*(Arrays.*asList*(tweetWithMention));

*assertTrue*("expected mentioned users to contain user7", mentionedUsers.contains("user7"));

}

}

**Screenshot:**

****

## Task2: Filtering lists of Tweets

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

1. Devise, document, and implement test cases for **writtenBy()** , **inTimespan()** , and **containing()** , and put them in **FilterTest.java** .
2. Implement **writtenBy()** , **inTimespan()** , and **containing()** , and make sure your 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:**

**package** twitter;

**import** java.time.Instant;

**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) {

Instant tweetTime = tweet.getTimestamp();

**if** (!tweetTime.isBefore(timespan.getStart()) && !tweetTime.isAfter(timespan.getEnd())) {

result.add(tweet);

}

}

**return** result;

}

**public** **static** List<Tweet> containing(List<Tweet> tweets, List<String> words) {

List<Tweet> result = **new** ArrayList<>();

**for** (Tweet tweet : tweets) {

String text = tweet.getText();

**for** (String word : words) {

**if** (text.toLowerCase().contains(word.toLowerCase())) {

result.add(tweet);

**break**; // No need to check other words if one match is found

}

}

}

**return** result;

}

}

**FilterTest.java:**

**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**;

}

@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** testWrittenByCaseInsensitivity() {

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** testInTimespanNoResults() {

Instant testStart = Instant.*parse*("2016-02-17T12:00:00Z");

Instant testEnd = Instant.*parse*("2016-02-17T13:00:00Z");

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

*assertTrue*("expected empty list", inTimespan.isEmpty());

}

@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** testContainingNoMatches() {

List<Tweet> containing = Filter.*containing*(Arrays.*asList*(***tweet1***, ***tweet2***), Arrays.*asList*("nonexistent"));

*assertTrue*("expected empty list", containing.isEmpty());

}

@Test

**public** **void** testContainingCaseInsensitivity() {

List<Tweet> containing = Filter.*containing*(Arrays.*asList*(***tweet1***, ***tweet2***), Arrays.*asList*("RIVEST"));

*assertFalse*("expected non-empty list", containing.isEmpty());

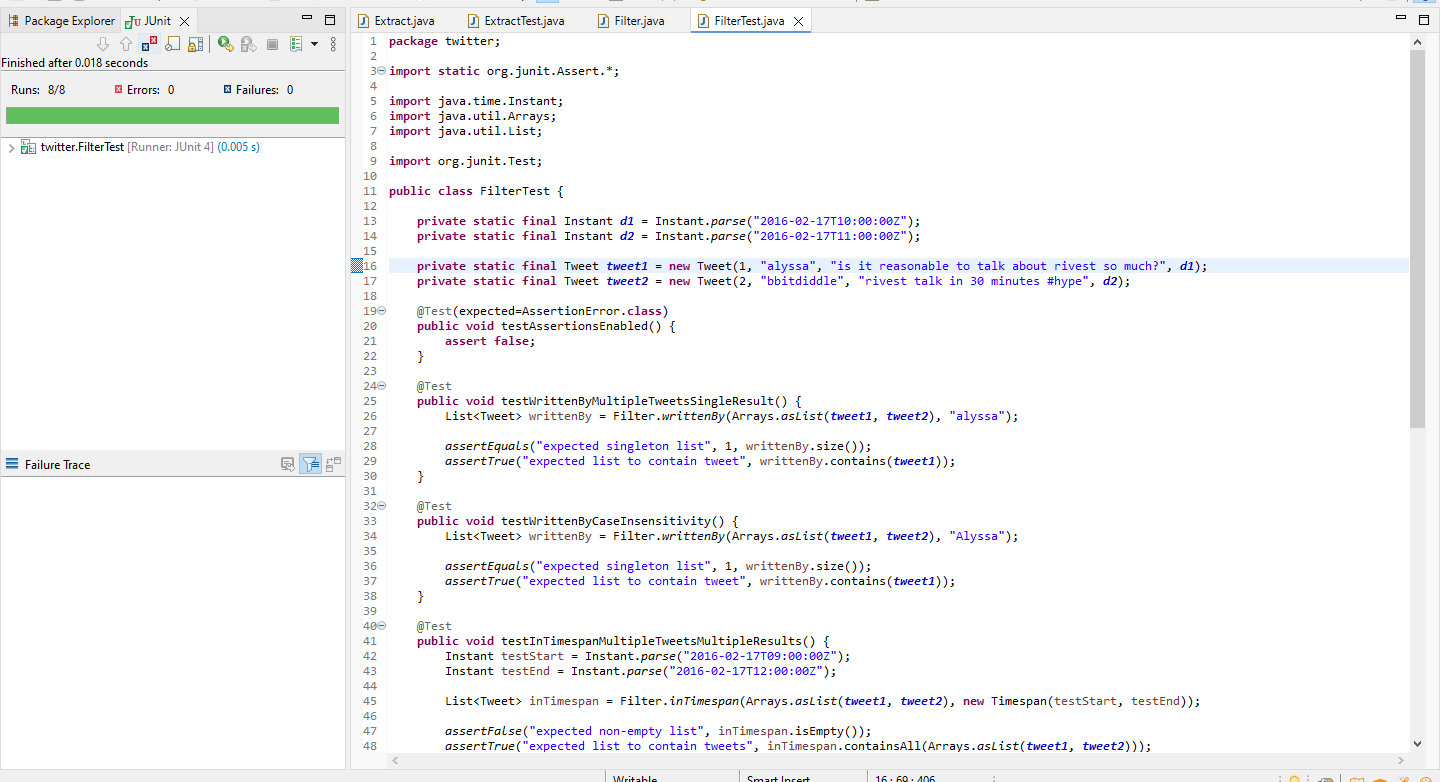
*assertTrue*("expected list to contain tweets", containing.contains(***tweet1***));

*assertTrue*("expected list to contain tweets", containing.contains(***tweet2***));

}

}

**Screenshot:**

****

## Deliverables:

Compile a single word document by filling in the solution part and submit this Word file on LMS. 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.)