# Faculty of Computing

**SE-314: Software Construction**

**Class: BESE 13AB**

# Lab 05: Code Review

# CLO-02: Apply patterns, frameworks, and techniques for Software Construction. CLO-04: Use modern tools such as Eclipse, NetBeans etc. for software construction.

**Date: 07th 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 |
| **SHARED WITH (FELLOW)** | AWAIS NAZIR |

# Instructor: Dr. Mehvish Rashid

# Lab Engineer: Mr. Aftab Farooq

# Lab 05: Tweet Tweet (Code Review)

Students will have hands-on experience of Code review. Given a set of specifications and unit tests, you will check for compliance with the specifications, and then provide their code review.

Material:

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

Swap previous lab codes with each other and mention them in the report.

Study about designing Specifications and testing strategy

|  |
| --- |
| **FELLOW TEST CASES** |
| FilterTest.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.List;  import org.junit.Test;  public class FilterTest {      //      // Testing strategy:      //      // Partition for writtenBy(tweets, username) -> result:      //      //   tweets.size: 0, >0      //   result.size: 0, 1, >1      //   the usernames are exactly the same or differ in cases      //      // Partition for inTimespan(tweets, timespan) -> result:      //      //   tweets.size: 0, >0      //   result.size: 0, 1, >1      //   the start of timespan equals its end or doesn't      //      // Partition for containing(tweets, words) -> result:      //      //   tweets.size: 0, >0      //   words.size: 0, 1, >1      //   result.size: 0, 1, >1      //   tweets contains exactly one or multiple of the words      //   the words are exactly the same or differ in cases      //        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);      private static final Tweet tweet3 = new Tweet(3, "Alyssa", "A portrait in the sum room.", d2);        @Test(expected=AssertionError.class)      public void testAssertionsEnabled() {          assert false; // make sure assertions are enabled with VM argument: -ea      }      @Test      public void testWrittenByNoTweets() {          List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(), "alyssa");            assertTrue("expected empty list", writtenBy.isEmpty());      }      @Test      public void testWrittenByMultipleTweetsNoResults() {          List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1, tweet2), "mit");            assertTrue("expected empty list", writtenBy.isEmpty());      }      @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 testWrittenByMultipleTweetsMultipleResults() {          List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1, tweet2, tweet3), "alyssa");            assertFalse("expected non-empty list", writtenBy.isEmpty());          assertTrue("expected list to contain tweets", writtenBy.containsAll(Arrays.asList(tweet1, tweet3)));          assertEquals("expected same order", 0, writtenBy.indexOf(tweet1));      }      @Test      public void testInTimespanNoTweets() {          List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(), new Timespan(Instant.MIN, Instant.MAX));            assertTrue("expected empty list", inTimespan.isEmpty());      }      @Test      public void testInTimespanMultipleTweetsNoResults() {          Instant testStart = Instant.parse("2016-02-17T08:00:00Z");          Instant testEnd = Instant.parse("2016-02-17T09:00:00Z");          List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1, tweet2), new Timespan(testStart, testEnd));            assertTrue("expected empty list", inTimespan.isEmpty());      }      @Test      public void testInTimespanMultipleTweetsSingleResult() {          Instant testStart = Instant.parse("2016-02-17T10:00:00Z");          Instant testEnd = Instant.parse("2016-02-17T10:00:00Z");            List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1, tweet2), new Timespan(testStart, testEnd));            assertEquals("expected singleton list", 1, inTimespan.size());          assertTrue("expected list to contain tweet", inTimespan.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 testContainingNoTweets() {          List<Tweet> containing = Filter.containing(Arrays.asList(), Arrays.asList("talk"));            assertTrue("expected empty list", containing.isEmpty());      }      @Test      public void testContainingNoWords() {          List<Tweet> containing = Filter.containing(Arrays.asList(tweet1), Arrays.asList());            assertTrue("expected empty list", containing.isEmpty());      }      @Test      public void testContainingMultipleTweetsNoResults() {          List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("reason"));            assertTrue("expected empty list", containing.isEmpty());      }      @Test      public void testContainingMultipleTweetsSingleResult() {          List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("reasonable"));            assertEquals("expected singleton list", 1, containing.size());          assertTrue("expected list to contain tweet", containing.contains(tweet1));      }      @Test      public void testContainingMultipleTweetsSingleWord() {          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 testContainingMultipleTweetsMultipleWords() {          List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("rivest", "Hype"));            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));      }      /\*       \* Warning: all the tests you write here must be runnable against any Filter       \* class that follows the spec. It will be run against several staff       \* implementations of Filter, which will be done by overwriting       \* (temporarily) your version of Filter with the staff's version.       \* DO NOT strengthen the spec of Filter or its methods.       \*       \* In particular, your test cases must not call helper methods of your own       \* that you have put in Filter, because that means you're testing a stronger       \* spec than Filter says. If you need such helper methods, define them in a       \* different class. If you only need them in this test class, then keep them       \* in this test class.       \*/  }  ExtractTest.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.Set;  import java.util.HashSet;  import org.junit.Test;  public class ExtractTest {      //      // Testing strategy:      //      // Partition for getTimespan(tweets) -> result:      //      //   tweets.size: 0, 1, >1      //   tweets contains repeated timestamps or doesn't      //   tweets comes in time order or doesn't      //      // Partition for getMentionedUsers(tweets) -> result:      //      //   tweets contains username-mentions or doesn't      //   tweets contains username-mentions preceded by valid characters or doesn't      //   tweets contains repeated username-mentions or doesn't      //   the username-mentions comes in the beginning, the end or the middle      //        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 Instant d3 = Instant.parse("2016-02-17T10:30:00Z");        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);      private static final Tweet tweet3 = new Tweet(3, "MITOCW", "@MITopenlearning component, free lecture notes, exams, and videos from @MIT.", d1);      private static final Tweet tweet4 = new Tweet(4, "mitopenlearning", "@mit component, transforming teaching and learning at @mit, home of @mitocw", d2);      private static final Tweet tweet5 = new Tweet(5, "mit6005", "an email address like bitdiddle@mit.edu does NOT contain a mention", d3);        @Test(expected=AssertionError.class)      public void testAssertionsEnabled() {          assert false; // make sure assertions are enabled with VM argument: -ea      }      @Test      public void testGetTimespanNoTweets() {          Timespan timespan = Extract.getTimespan(Arrays.asList());          // any results without exceptions are acceptable      }      @Test      public void testGetTimespanOneTweet() {          Timespan timespan = Extract.getTimespan(Arrays.asList(tweet1));          assertEquals("expected start", d1, timespan.getStart());          assertEquals("expected end", d1, timespan.getEnd());      }      @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 testGetTimespanRepeated() {          Timespan timespan = Extract.getTimespan(Arrays.asList(tweet1, tweet2, tweet4));            assertEquals("expected start", d1, timespan.getStart());          assertEquals("expected end", d2, timespan.getEnd());      }      @Test      public void testGetTimespanOutOfOrder() {          Timespan timespan = Extract.getTimespan(Arrays.asList(tweet2, tweet1, tweet5));            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 testGetMentionedUsersPrecedingValid() {          Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet5));            assertTrue("expected empty set", mentionedUsers.isEmpty());      }      /\*\*       \* Transform the strings in the set into lower cases.       \*       \* @param strings       \*            set of strings, not modified by this method.       \* @return a set of strings transformed into lower cases.       \*/      private static Set<String> toLowerCase(Set<String> strings) {          Set<String> result = new HashSet<>();          for (String string : strings) {              result.add(string.toLowerCase());          }          return result;      }      @Test      public void testGetMentionedUsersNoRepeated() {          Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet3));          Set<String> expected = Set.of("mit", "mitopenlearning");            assertTrue("expected results", expected.equals(toLowerCase(mentionedUsers)));      }      @Test      public void testGetMentionedUsersRepeated() {          Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet4));          Set<String> expected = Set.of("mit", "mitocw");          assertTrue("expected results", expected.equals(toLowerCase(mentionedUsers)));      }      @Test      public void testGetMentionedUsersCaseInsensitive() {          Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet3, tweet4));          Set<String> expected = Set.of("mit", "mitopenlearning", "mitocw");          assertTrue("expected results", expected.equals(toLowerCase(mentionedUsers)));      }        @Test      public void testExtractInstantiation() {          // Just to instantiate the Extract class and check no exceptions occur          Extract extract = new Extract();          assertNotNull("expected non-null instance of Extract", extract);      }      /\*       \* Warning: all the tests you write here must be runnable against any       \* Extract class that follows the spec. It will be run against several staff       \* implementations of Extract, which will be done by overwriting       \* (temporarily) your version of Extract with the staff's version.       \* DO NOT strengthen the spec of Extract or its methods.       \*       \* In particular, your test cases must not call helper methods of your own       \* that you have put in Extract, because that means you're testing a       \* stronger spec than Extract says. If you need such helper methods, define       \* them in a different class. If you only need them in this test class, then       \* keep them in this test class.       \*/  } |

**Lab Tasks :**

* **Are test cases chosen using the input/output-space partitioning approach.** This approach is explained in the reading about testing.

|  |
| --- |
| **OBSERVATIONS** |
| The test cases for both the Filter and Extract classes are indeed chosen using the input/output-space partitioning approach. This method involves dividing the input space into distinct categories or partitions, each of which is expected to yield different outcomes. |
| **IMPROVEMENTS** |
| Not applicable, Good Enough already. |

* **Comment about the *testing strategy*** – how is the input/output space of each method partitioned.

|  |
| --- |
| **COMMENTS** |
| The class fellow’s original testing strategy primarily utilized input/output space partitioning, focusing on various scenarios for each method in the Filter and Extract classes. This approach involved dividing the input data into distinct categories, such as tweet counts, username variations, and time ranges, to systematically test different conditions. Each test aimed to verify that the methods correctly handled these partitions, ensuring functionality across a range of expected inputs. Additionally, the tests checked for both expected outcomes and edge cases, aiming to validate the behavior of the methods under varying circumstances. |

* **Are test cases small and well-chosen.** Usage of large set of tweets from Twitter for each test is not recommended. Have they created own artificial tweets carefully chosen to test the partition they’re trying to test.

|  |
| --- |
| **OBSERVATIONS** |
| The test cases you've provided are generally well-chosen and utilize small, artificial datasets that effectively target specific partitions for testing the functionality of the Filter and Extract classes. However, there are some improvements that can be made to ensure the test cases remain small and focused while still covering a broad range of scenarios. |
| **IMPROVEMENTS** |
| 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", "First tweet content.", d1);  private static final Tweet tweet2 = new Tweet(2, "bbitdiddle", "Second tweet content.", d2);  private static final Tweet tweet3 = new Tweet(3, "Alyssa", "Another tweet content.", d2);  @Test(expected=AssertionError.class)  public void testAssertionsEnabled() {  assert false; // make sure assertions are enabled with VM argument: -ea  }  // Test for no tweets  @Test  public void testWrittenByNoTweets() {  List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(), "alyssa");  assertTrue("expected empty list", writtenBy.isEmpty());  }  // Test for no matching tweets  @Test  public void testWrittenByNoResults() {  List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1, tweet2), "mit");  assertTrue("expected empty list", writtenBy.isEmpty());  }  // Test for single match  @Test  public void testWrittenBySingleResult() {  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 for multiple matches  @Test  public void testWrittenByMultipleResults() {  List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1, tweet2, tweet3), "alyssa");  assertEquals("expected two tweets from Alyssa", 2, writtenBy.size());  assertTrue("expected list to contain tweet1", writtenBy.contains(tweet1));  assertTrue("expected list to contain tweet3", writtenBy.contains(tweet3));  }  // Test for no tweets in timespan  @Test  public void testInTimespanNoTweets() {  List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(), new Timespan(Instant.MIN, Instant.MAX));  assertTrue("expected empty list", inTimespan.isEmpty());  }  // Test for timespan with no results  @Test  public void testInTimespanNoResults() {  List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1, tweet2), new Timespan(Instant.parse("2016-02-17T09:00:00Z"), Instant.parse("2016-02-17T09:59:59Z")));  assertTrue("expected empty list", inTimespan.isEmpty());  }  // Test for timespan with one result  @Test  public void testInTimespanSingleResult() {  List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1, tweet2), new Timespan(d1, d1));  assertEquals("expected singleton list", 1, inTimespan.size());  assertTrue("expected list to contain tweet", inTimespan.contains(tweet1));  }  // Test for timespan with multiple results  @Test  public void testInTimespanMultipleResults() {  List<Tweet> inTimespan = Filter.inTimespan(Arrays.asList(tweet1, tweet2), new Timespan(d1.minusSeconds(60), d2.plusSeconds(60)));  assertFalse("expected non-empty list", inTimespan.isEmpty());  assertTrue("expected list to contain tweets", inTimespan.containsAll(Arrays.asList(tweet1, tweet2)));  }  // Test for no tweets containing words  @Test  public void testContainingNoTweets() {  List<Tweet> containing = Filter.containing(Arrays.asList(), Arrays.asList("talk"));  assertTrue("expected empty list", containing.isEmpty());  }  // Test for no words  @Test  public void testContainingNoWords() {  List<Tweet> containing = Filter.containing(Arrays.asList(tweet1), Arrays.asList());  assertTrue("expected empty list", containing.isEmpty());  }  // Test for no tweets containing specified word  @Test  public void testContainingNoResults() {  List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("reason"));  assertTrue("expected empty list", containing.isEmpty());  }  // Test for containing one specific word  @Test  public void testContainingSingleWord() {  List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("first"));  assertEquals("expected singleton list", 1, containing.size());  assertTrue("expected list to contain tweet", containing.contains(tweet1));  }  // Test for containing multiple words  @Test  public void testContainingMultipleWords() {  List<Tweet> containing = Filter.containing(Arrays.asList(tweet1, tweet2), Arrays.asList("first", "second", "talk"));  assertFalse("expected non-empty list", containing.isEmpty());  assertTrue("expected list to contain tweets", containing.containsAll(Arrays.asList(tweet1, tweet2)));  }  }  ExtractTest.java  package twitter;  import static org.junit.Assert.\*;  import java.time.Instant;  import java.util.Arrays;  import java.util.Set;  import java.util.HashSet;  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", "No mentions here!", d1);  private static final Tweet tweet2 = new Tweet(2, "bbitdiddle", "Check out @MIT and @mitocw!", d2);  private static final Tweet tweet3 = new Tweet(3, "MITOCW", "@MITopenlearning free lectures!", d1);  private static final Tweet tweet4 = new Tweet(4, "mitopenlearning", "@mit component, transforming learning.", d2);  private static final Tweet tweet5 = new Tweet(5, "mit6005", "This is not a mention: bitdiddle@mit.edu", d1);  @Test(expected=AssertionError.class)  public void testAssertionsEnabled() {  assert false; // make sure assertions are enabled with VM argument: -ea  }  // Test for no tweets for timespan  @Test  public void testGetTimespanNoTweets() {  Timespan timespan = Extract.getTimespan(Arrays.asList());  assertEquals("expected start", Instant.MIN, timespan.getStart());  assertEquals("expected end", Instant.MIN, timespan.getEnd());  }  // Test for timespan with one tweet  @Test  public void testGetTimespanOneTweet() {  Timespan timespan = Extract.getTimespan(Arrays.asList(tweet1));  assertEquals("expected start", d1, timespan.getStart());  assertEquals("expected end", d1, timespan.getEnd());  }  // Test for timespan with two tweets  @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 for getting mentioned users with no mentions  @Test  public void testGetMentionedUsersNoMention() {  Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet1));  assertTrue("expected empty set", mentionedUsers.isEmpty());  }  // Test for getting mentioned users with valid mentions  @Test  public void testGetMentionedUsersValidMentions() {  Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet2, tweet3));  Set<String> expected = Set.of("mit", "mitocw", "mitopenlearning");  assertEquals("expected mentioned users", expected, new HashSet<>(mentionedUsers));  }  // Test for handling case insensitivity in mentions  @Test  public void testGetMentionedUsersCaseInsensitive() {  Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet3, tweet4));  Set<String> expected = Set.of("mit", "mitopenlearning");  assertEquals("expected mentioned users", expected, new HashSet<>(mentionedUsers));  }  // Test for invalid email as mention  @Test  public void testGetMentionedUsersInvalidEmail() {  Set<String> mentionedUsers = Extract.getMentionedUsers(Arrays.asList(tweet5));  assertTrue("expected empty set", mentionedUsers.isEmpty());  }  } |

* **Tests should find bugs.** Provide **one** buggy implementation and see if assigned tests catch the bugs. So, consider ways an implementation might inadvertently fail to meet the spec, and choose tests that will expose those bugs.

|  |
| --- |
| **BUGGY IMPLEMENTATION** |
| public static List<Tweet> writtenBy(List<Tweet> tweets, String username) {  List<Tweet> result = new ArrayList<>();  for (Tweet tweet : tweets) {  String author = tweet.getAuthor();  // Bug: Incorrect comparison, only checks for exact match  if (author.equals(username)) {  result.add(tweet);  }  }  return result;  }  In this buggy implementation, the method only checks for an exact match of the username. It fails to account for case insensitivity, which contradicts the specification that states usernames should be compared in a case-insensitive manner. |
| **TEST TO EXPOSE BUG** |
| @Test  public void testWrittenByMultipleTweetsMultipleResults() {  List<Tweet> writtenBy = Filter.writtenBy(Arrays.asList(tweet1, tweet2, tweet3), "alyssa");    assertFalse("expected non-empty list", writtenBy.isEmpty());  assertTrue("expected list to contain tweets", writtenBy.containsAll(Arrays.asList(tweet1, tweet3)));  assertEquals("expected same order", 0, writtenBy.indexOf(tweet1));  } |

* **Tests must be legal clients of the spec.** Provide **three** legal, variant implementation that still strictly satisfy the specs. Then, run assigned test cases against these specs, and verify if test cases pass these good implementations. That means that test cases can’t make extra assumptions that are only true for their own implementation.

|  |
| --- |
| **VARIANTS** |
| Variant#1 Filter.java (Using Streams)  package twitter;  import java.time.Instant;  import java.util.List;  import java.util.stream.Collectors;  public class Filter {  public static List<Tweet> writtenBy(List<Tweet> tweets, String username) {  return tweets.stream()  .filter(tweet -> tweet.getAuthor().equalsIgnoreCase(username))  .collect(Collectors.toList());  }  public static List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {  Instant start = timespan.getStart();  Instant end = timespan.getEnd();  return tweets.stream()  .filter(tweet -> {  Instant timestamp = tweet.getTimestamp();  return !timestamp.isBefore(start) && !timestamp.isAfter(end);  })  .collect(Collectors.toList());  }  public static List<Tweet> containing(List<Tweet> tweets, List<String> words) {  return tweets.stream()  .filter(tweet -> {  String text = tweet.getText();  return words.stream().anyMatch(word -> text.toLowerCase().contains(word.toLowerCase()));  })  .collect(Collectors.toList());  }  }  Variant#2 Filter.java (Using for-each with explicit conditions)  package twitter;  import java.time.Instant;  import java.util.List;  import java.util.ArrayList;  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 timestamp = tweet.getTimestamp();  if (!timestamp.isBefore(timespan.getStart()) && !timestamp.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) {  for (String word : words) {  if (tweet.getText().toLowerCase().contains(word.toLowerCase())) {  result.add(tweet);  break; // Break once we found a matching word  }  }  }  return result;  }  }  Variant#3 Filter.java (Using Indexing)  package twitter;  import java.time.Instant;  import java.util.List;  import java.util.ArrayList;  public class Filter {  public static List<Tweet> writtenBy(List<Tweet> tweets, String username) {  List<Tweet> result = new ArrayList<>();  for (int i = 0; i < tweets.size(); i++) {  if (tweets.get(i).getAuthor().equalsIgnoreCase(username)) {  result.add(tweets.get(i));  }  }  return result;  }  public static List<Tweet> inTimespan(List<Tweet> tweets, Timespan timespan) {  List<Tweet> result = new ArrayList<>();  Instant start = timespan.getStart();  Instant end = timespan.getEnd();  for (int i = 0; i < tweets.size(); i++) {  Instant timestamp = tweets.get(i).getTimestamp();  if (!(timestamp.isBefore(start) || timestamp.isAfter(end))) {  result.add(tweets.get(i));  }  }  return result;  }  public static List<Tweet> containing(List<Tweet> tweets, List<String> words) {  List<Tweet> result = new ArrayList<>();  for (int i = 0; i < tweets.size(); i++) {  for (String word : words) {  if (tweets.get(i).getText().toLowerCase().contains(word.toLowerCase())) {  result.add(tweets.get(i));  break;  }  }  }  return result;  }  } |
| **RESULTS** |
| Variant#1:    Variant#2:    Variant#3 |

**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.)