**Faculty of Computing**

**SE-314: Software Construction**

**Class: BESE 13AB**

# 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()** .

## 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.

**Source Code: Zip your source code and upload one file per group on LMS as well.**

**Solution**

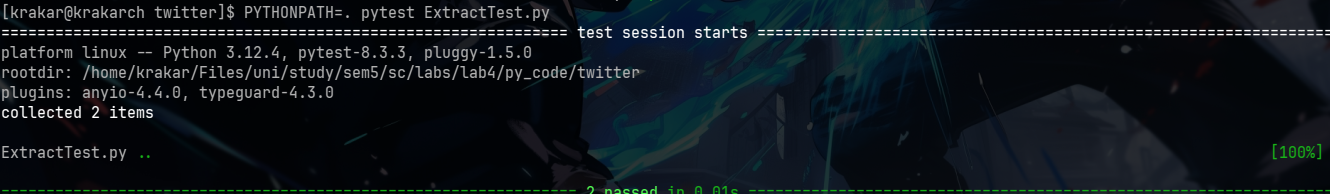
\

**Task 1:**

Failed TestCase:



## Passed TestCases:



Code:

*from Timespan import Timespan*

*from Tweet import Tweet*

*from datetime import datetime*

*from typing import List, Set*

*class Extract:*

*@staticmethod*

*def get\_timespan(tweets: List[Tweet]) -> Timespan:*

*if not tweets:*

*raise ValueError("The list of tweets cannot be empty.")*

*# Get the minimum and maximum timestamps*

*min\_timestamp = min(tweet.timestamp for tweet in tweets)*

*max\_timestamp = max(tweet.timestamp for tweet in tweets)*

*# Return a Timespan object*

*return Timespan(min\_timestamp, max\_timestamp)*

*@staticmethod*

*def get\_mentioned\_users(tweets: List[Tweet]) -> Set[str]:*

*mentioned\_users = set()*

*for tweet in tweets:*

*text = tweet.text*

*# Find all mentions in the tweet text*

*words = text.split()*

*for word in words:*

*if word.startswith('@') and len(word) > 1:*

*username = word[1:] # Remove the '@' symbol*

*if username.isalnum() or '\_' in username or '-' in username:*

*mentioned\_users.add(username.lower()) # Add in lowercase for case insensitivity*

*return mentioned\_users*

Testcases:

*import pytest*

*from datetime import datetime*

*from Extract import Extract*

*from Tweet import Tweet*

*from Timespan import Timespan*

*def test\_get\_timespan():*

*# Create sample tweets*

*tweet1 = Tweet(1, "user1", "Hello world", datetime(2024, 1, 1, 10, 0, 0))*

*tweet2 = Tweet(2, "user2", "Goodbye world", datetime(2024, 1, 2, 10, 0, 0))*

*tweet3 = Tweet(3, "user3", "Hello again", datetime(2024, 1, 1, 12, 0, 0))*

*# Test case 1: Minimum time span covering all tweets*

*tweets = [tweet1, tweet2, tweet3]*

*expected\_timespan = Timespan(datetime(2024, 1, 1, 10, 0, 0), datetime(2024, 1, 2, 10, 0, 0))*

*assert Extract.get\_timespan(tweets) == expected\_timespan*

*# Test case 2: Single tweet*

*tweet4 = Tweet(4, "user4", "Just one tweet", datetime(2024, 1, 1, 15, 0, 0))*

*tweets = [tweet4]*

*expected\_timespan = Timespan(datetime(2024, 1, 1, 15, 0, 0), datetime(2024, 1, 1, 15, 0, 0))*

*assert Extract.get\_timespan(tweets) == expected\_timespan*

*# Test case 3: No tweets*

*tweets = []*

*with pytest.raises(ValueError):*

*Extract.get\_timespan(tweets)*

*def test\_get\_mentioned\_users():*

*# Create sample tweets*

*tweet1 = Tweet(1, "user1", "Hello @user2 and @user3", datetime(2024, 1, 1, 10, 0, 0))*

*tweet2 = Tweet(2, "user2", "Goodbye @user1", datetime(2024, 1, 2, 10, 0, 0))*

*tweet3 = Tweet(3, "user3", "No mentions here", datetime(2024, 1, 1, 12, 0, 0))*

*# Test case 1: Extracted mentioned users*

*tweets = [tweet1, tweet2, tweet3]*

*expected\_users = {"user1", "user2", "user3"}*

*assert Extract.get\_mentioned\_users(tweets) == expected\_users*

*# Test case 2: No mentions*

*tweet4 = Tweet(4, "user4", "No mentions here", datetime(2024, 1, 1, 15, 0, 0))*

*tweets = [tweet4]*

*expected\_users = set()*

*assert Extract.get\_mentioned\_users(tweets) == expected\_users*

*# Test case 3: Duplicate mentions*

*tweet5 = Tweet(5, "user5", "Mentioning @user2 again", datetime(2024, 1, 1, 16, 0, 0))*

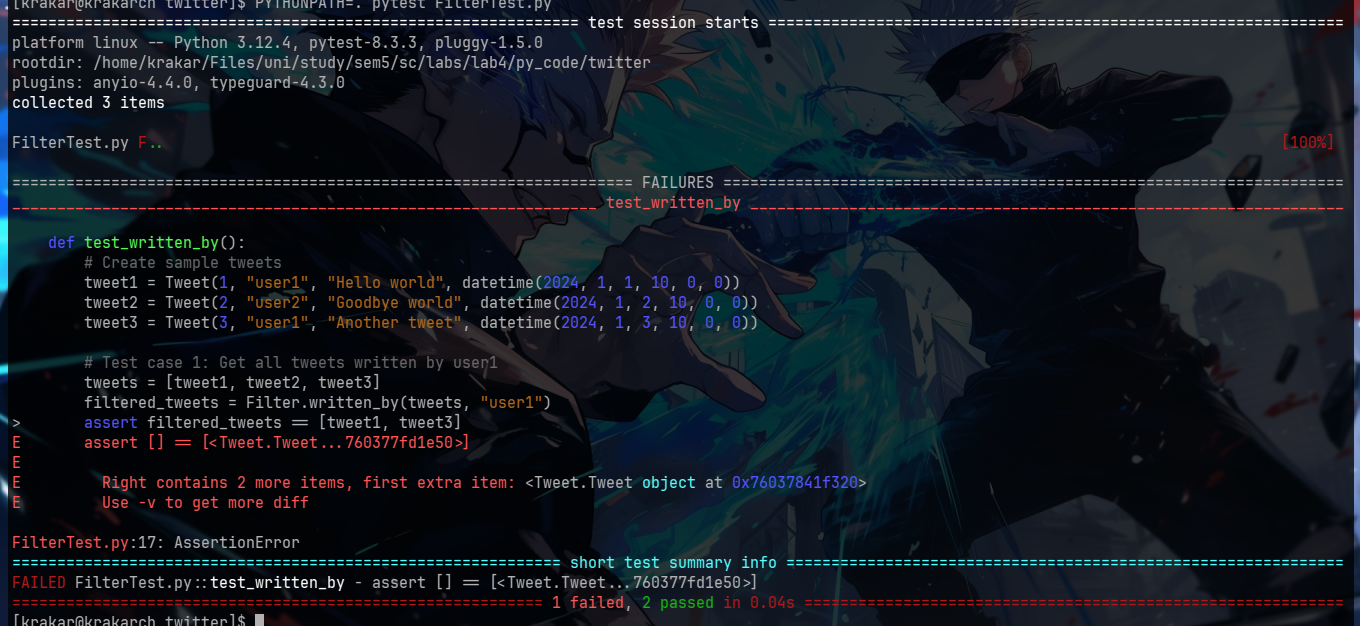
*tweets = [tweet1, tweet5]*

*expected\_users = {"user2", "user3"}*

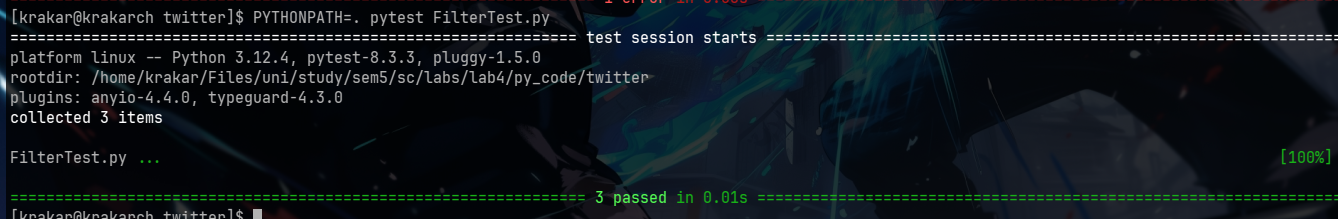
*assert Extract.get\_mentioned\_users(tweets) == expected\_users*

## Task 2:

Failed Testcases:



Passed Testcases:



Code:

*from typing import List*

*from Tweet import Tweet*

*from Timespan import Timespan*

*class Filter:*

*@staticmethod*

*def written\_by(tweets: List[Tweet], username: str) -> List[Tweet]:*

*"""*

*Find tweets written by a particular user.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*username: the username to filter tweets by (case-insensitive).*

*Returns:*

*A list of tweets written by the specified user, in the same order as in the input list.*

*"""*

*return [tweet for tweet in tweets if tweet.author.lower() == username.lower()]*

*@staticmethod*

*def in\_timespan(tweets: List[Tweet], timespan: Timespan) -> List[Tweet]:*

*"""*

*Find tweets that were sent during a particular timespan.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*timespan: a Timespan object representing the desired time interval.*

*Returns:*

*A list of tweets that were sent during the timespan, in the same order as in the input list.*

*"""*

*return [tweet for tweet in tweets if timespan.start <= tweet.timestamp <= timespan.end]*

*@staticmethod*

*def containing(tweets: List[Tweet], words: List[str]) -> List[Tweet]:*

*"""*

*Find tweets that contain certain words.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*words: a list of words to search for in the tweets (case-insensitive).*

*Returns:*

*A list of tweets that contain at least one of the words, in the same order as in the input list.*

*"""*

*words\_lower = {word.lower() for word in words}*

*return [tweet for tweet in tweets if any(word in tweet.text.lower().split() for word in words\_lower)]*

Testcases:

*from typing import List*

*from Tweet import Tweet*

*from Timespan import Timespan*

*class Filter:*

*@staticmethod*

*def written\_by(tweets: List[Tweet], username: str) -> List[Tweet]:*

*"""*

*Find tweets written by a particular user.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*username: the username to filter tweets by (case-insensitive).*

*Returns:*

*A list of tweets written by the specified user, in the same order as in the input list.*

*"""*

*return [tweet for tweet in tweets if tweet.author.lower() == username.lower()]*

*@staticmethod*

*def in\_timespan(tweets: List[Tweet], timespan: Timespan) -> List[Tweet]:*

*"""*

*Find tweets that were sent during a particular timespan.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*timespan: a Timespan object representing the desired time interval.*

*Returns:*

*A list of tweets that were sent during the timespan, in the same order as in the input list.*

*"""*

*return [tweet for tweet in tweets if timespan.start <= tweet.timestamp <= timespan.end]*

*@staticmethod*

*def containing(tweets: List[Tweet], words: List[str]) -> List[Tweet]:*

*"""*

*Find tweets that contain certain words.*

*Args:*

*tweets: a list of Tweet objects with distinct ids.*

*words: a list of words to search for in the tweets (case-insensitive).*

*Returns:*

*A list of tweets that contain at least one of the words, in the same order as in the input list.*

*"""*

*words\_lower = {word.lower() for word in words}*

*return [tweet for tweet in tweets if any(word in tweet.text.lower().split() for word in words\_lower)]*

## 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.)