Homework 6 - Texting

Introduction

In this assignment you will practice

- Using Predicate (https://docs.oracle.com/javase/9/docs/api/java/util/function/Predicate.html)
- Using LocalDateTime (https://docs.oracle.com/javase/9/docs/api/java/time/LocalDateTime.html)
- Using Optional (https://docs.oracle.com/javase/9/docs/api/java/util/Optional.html)
- Using List (https://docs.oracle.com/javase/9/docs/api/java/util/List.html)s and Map (https://docs.oracle.com/javase/9/docs/api/java/util/Map.html)s
- Using Lambda expressions, Method references, Anonymous inner classes, and Inner classes

Problem Description

In this homework, you will be creating a database that can store text messages and filter those messages based on certain constraints!

- `Message`
  - A representaton of a text message.
- `Database`
  - A database of past `Message` s that have been sent.

Solution Description

Classes:

- `Message`
  - Fields:
    - `Optional<String> contactName`
    - `String to`
    - `String from`
    - `String body`
    - `LocalDateTime date`
    - `boolean isImportant`
  - Methods:
    - Getter methods for all 6 fields.
  - Constructor with the following signature.
    - `public Message(Optional<String> contactName, String to, String from, String body, LocalDateTime date, boolean isImportant)`
- `Database`
  - Fields:
    - `List<Message> messages`
  - Methods:
    - `Message getMessage(int n)` - Returns the nth message where 0 <= n < size of list.
    - `List<Message> getMessages()` - Returns all the messages stored in the database.
    - `List<Message> filter(Predicate<Message> filter)` - Returns the list of messages filtered by the predicate. Must not change `messages` field.
    - `List<Message> getMessagesBetween(LocalDateTime start, LocalDateTime end)` - Returns a list of messages if the time that the message is sent is between the start LocalDateTime and the end LocalDateTime (inclusive). Must not change `messages` field. Must call `filter` with a **lambda expression**.
    - `Map<String, List<Message>> sortMessagesByContact()` - Returns a Map that stores a contact name as a key and a list of messages from that contact as a value. If a message has no associated contact, it should not appear in the Map. Must not change `messages` field. Must call `filter` with an **anonymous inner class** in the method body.
    - `List<Message> getMessagesWithKeyword(String keyword)` - Returns a list of messages if the body of the message contains the keyword. **Note:** The messages don't need to match the case of the keyword (ex. "LOL" would match "lol"). Must not change `messages` field. Must call `filter` with an instance of an **inner class**. The inner class must not be anonymous.
    - `List<Message> getMessagesWithPriority()` - Returns a list of messages if the message is marked as important. Must not change `messages` field. Must call `filter` with a **method reference**.
  - Constructor with the following signature.
    - `public Database(List<Message> messages)`

Important: `getMessagesBetween(LocalDateTime start, LocalDateTime end)` , `sortMessagesByContact()` , `getMessageWithKeyword(String keyword)` , and `getMessagesWithPriority()` must use `filter` with a lambda expression, an anonymous inner class, an inner class, and a method reference respectively.

Make the visibility of all the methods and fields as you see best fits good programming style.

Tips and Considerations

If anything seems confusing, read through the entire description and instructions again. As always, feel free to contact your TAs, post on Piazza, or come in for TA office hours. In addition, here are some tips specific to this homework:

Import `java.util.function.Predicate`, `java.util.Optional`, `java.time.LocalDateTime`, `java.util.List`, `java.util.Map`, and any class that implements List or Map. Do not import anything that oversimplifies the assignment.

If there are any classes that you are unfamiliar with (e.g. Predicate, LocalDateTime, Optional), look them up in the Java API (https://docs.oracle.com/javase/9/docs/api/index.html?overview-summary.html).

Test things out in JShell!

Grading

Submit code that compiles!!!

Non compiling code will receive an automatic zero.

- Message (15 points)
  - [5] Each field is correctly declared and assigned
  - [5] Getter methods
  - [5] Correctly implements constructor
- Database (85 points)
  - [5] Correctly implements constructor
  - [5] `messages` is correctly declared and assigned
  - [5] `getMessage(int n)`
  - [5] `getMessages()`
  - [13] `getMessagesBetween(LocalDateTime start, LocalDateTime end)`
  - [13] `sortMessagesByContact()`
  - [13] `getMessageWithKeyword(String keyword)`
  - [13] `getMessagesWithPriority()`
  - [13] `filter(Predicate<Message> filter)`

Checkstyle

For each of your homework assignments we will run checkstyle and deduct one point for every checkstyle error.

For this homework the **checkstyle cap is 100**, meaning you can lose up to 100 points on this assignment due to style errors.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on the course website. To check the style of your code run `java -jar <path-to-checkstyle.jar> -a *.java`.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips (http://cs1331.gatech.edu/customization-tips.html) page for more information.

Collaboration

When completing homeworks for CS 1331 you may talk with other students about:

- What general strategies or algorithms you used to solve problems in the homeworks
- Parts of the homework specification you are unsure of and need more explanation
- Online resources that helped you find a solution
- Key course concepts and Java language features used in your solution
- **You may not discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.**

Examples of approved/disapproved collaboration:

OKAY: "Hey, I'm really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?"

BY NO MEANS OKAY: "Hey… the homework is due in like 20 minutes… Can I see your code? I *promise* won't copy it directly!"

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Submission

Submit each of your Java source files necessary for Database to compile on Canvas as separate attachments. When you're ready, double-check that you have submitted and not just saved a draft. **Download each file and compile them to assure that nothing went wrong with the submission process.**

Files needed:

- Database.java
- Message.java

Verify the Success of Your Submission to Canvas

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that your program runs with no syntax or runtime errors. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

- After uploading the files to Canvas you should receive an email from Canvas listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
- After submitting the files to Canvas, return to the Assignment menu option and this homework. It should show the submitted files.
- Download copies of your submitted files from the Canvas Assignment page placing them in a new folder.
- Re-run and test the files you downloaded from Canvas to make sure it's what you expect.
- This procedure helps guard against a few things.
    - It helps ensure that you turn in the correct files.
    - It helps you realize if you omit a file or files. Missing files will not be given any credit, and non-compiling/non-running homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. Do not wait until the last minute! (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
    - Helps find syntax errors or runtime errors that you may have added after you last tested your code.