Phase 1:

- Explanation: In this phase, the application needs to ensure that there's at least a one-minute gap between two consecutive log statements. Additionally, a specific condition should be tested every minute.

- Example: Suppose we have a function "test\_condition()" that needs to be tested every minute, and we want to log its result. The application should log the result of "test\_condition()" every minute and ensure that there's at least a one-minute gap between each log statement.

My Take :   
1. MainApplication.java to run the sping    
2. LogTimer.java this component is used to evaluate the provided conditions    
3. Used @Scheduled(fixedRate = 60000), with this spring functionality I was able to easily print “Condition met” after every minute.   
4. First if condition checks if conditionMet() method return true, then print “Condition met.”

5. Second if condition checks if previousLog and latestLog have one minute gap in between. If so, then it goes on to print “Logging statement...” and also set previousLog to latestLog(because the latest log will become last log in future).

6. after writing the last line of 5th point, i realised i should update the variable name of “previousLog” to “lastLog”  and also update print statement for this condition to “Last Log = Lastest Log”. Preventing code smells as i go along.

Phase 2:

- Explanation: This phase involves implementing logging functionality while also storing the timestamp when each log statement is printed.

- Example: When a log statement is printed, it should include the current timestamp. For instance, a log statement might look like: "2024-04-04 14:30:00 - Condition tested: True." 

My Take:

I can implement DateTimeFormatter object here to get YY:MM:DD HH:MM:SS

Let's call this “formatter “. i will also need to update the print statements to adapt this formatting. I can simply call format method on our formatter and assign latestLog to it and then concatenate it with “condition met.” and “Last log = Latest Log”;

While setting format i learnt about the “.ofPattern” method of DateTimeFormatter object.   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
   
 

Phase 3:

- Explanation: The application should verify the frequency condition (e.g., every 30 seconds) based on the timestamp stored during the last run.

- Example: If the last log statement was printed at 14:30:00, the application should ensure that the next log statement is printed no earlier than 14:30:30.

My Take :   
I can use Duration object here; this allows me to store number of seconds and milliseconds. Duration.ofSeconds(30) should work just fine.

Rather than having two if statements I can use one?

Yes, it works.

Makes my code more efficient? I think it removes code smell.

Made formatter final.

Output :

2024-04-10 12:49:29 Condition met.

2024-04-10 12:49:29 Last log  = Latest Log

2024-04-10 12:49:59 Condition met.

2024-04-10 12:49:59 Last log  = Latest Log

2024-04-10 12:50:30 Condition met.

2024-04-10 12:50:30 Last log  = Latest Log

Phase 4:

- Explanation: This phase requires storing the timestamp of each log in a database table as well as locally.

- Example: Each time a log statement is printed, the timestamp along with the log message should be stored in a database table named "log\_table". Additionally, the timestamp can be stored locally in a variable for easy reference.

My Take:

I want to be sure about the fundamentals of DB and connectivity of DB with this project before moving further.

I am using pgAdmin for postgreSQL. I already have a server in which I have created a new database “TimeStamps” and a table in that database called “TimeStamp”.

I have set two columns in timeStamp table, lastLog and latestLog, and set their DataTypes to “time with time zone”, hopefully this should be just.

Configured application.properties to established db connection

Added postgreSQL dependency to pom.xml   
   
   
   
   
 

To make LogTimer.java add timeStamp to the database I will first add autowired functionality. This repo will handle DB operations.

need to make an Entity to represent timestamp in DB, I will call it TimeStampEntity 

I have deleted the table I built earlier in pgAdmin IDE, I am now using Entity to create table and columns.    
   
   
   
   
   
   
   
   
   
   
   
   
 

Used cmd + n in intellij to create getters and setter for these

I have also created Interface called TimestampRepository : 

public interface TimestampRepository extends JpaRepository<TimestampEntity, Long> {   
}

Now i will move back to the LogTimer and see how to add the timestamp in the database. I believe the most appropriate place to add this functionality is within the if statement of our checkCondition().   
   
Realised that I need a service class too, @Service annotation in spring will help mark this class as a spring service and then it will be automatically detected and intantiateed by the spring container   
   
i will also add @Autowired to automatically inject bean when service is instantiated

Asked ChatGPT for saveTimestamp() method and adjusted it to my need.

   
   
   
   
Now, let’s call this method in checkCondition() in LogTime.java    
   
   
   
   
   
   
   
   
 

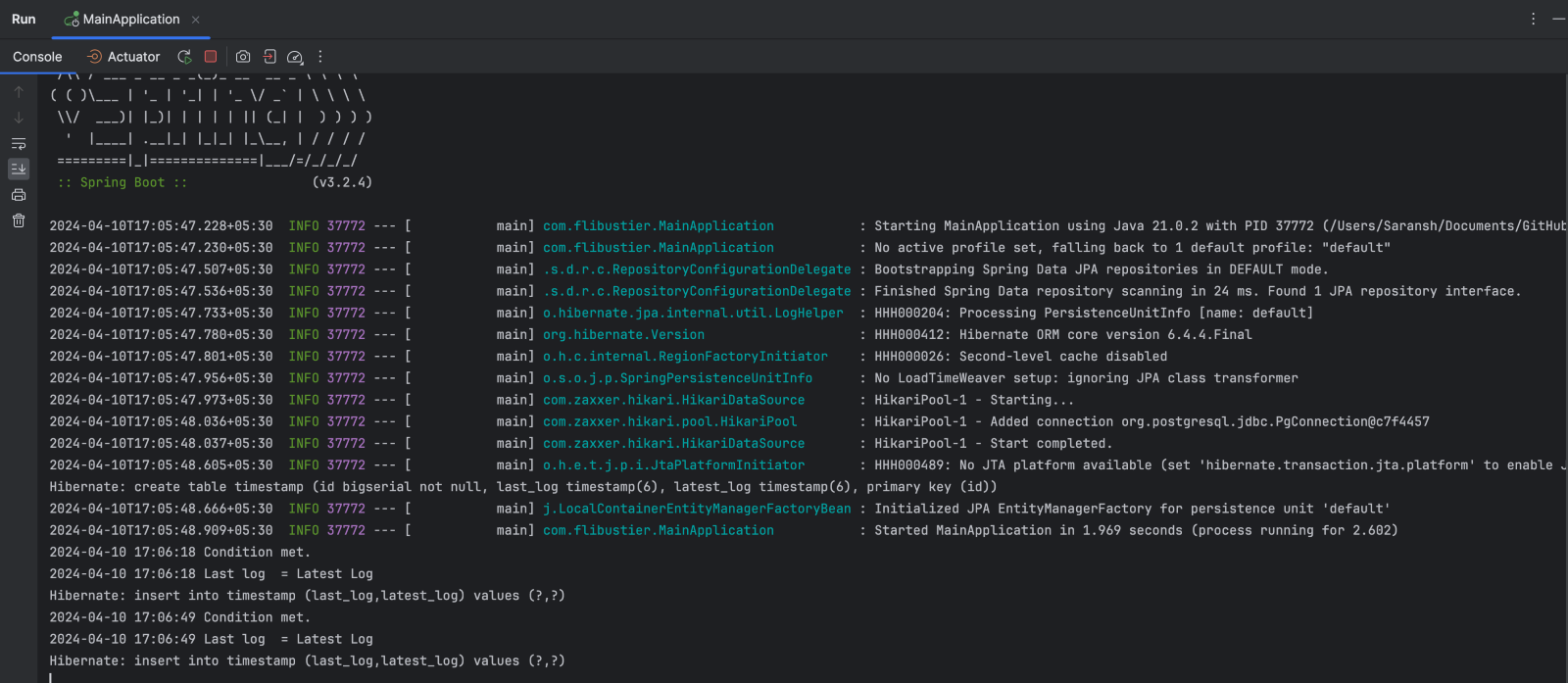


Table “timestamp” was created, and I am also getting the 30 sec intervals. 

I need help in understanding pgAdmin’s side of the story, i do see “timestamp” table there, I know this was generated by my code because I earlier deleted the table I created manually and here I also see some acitvities but don’t know what is going on.

A screenshot of a computer

Description automatically generated

I was right about this; I just checked the db table “timestamp” and found data in it:

IMPORTANT:

Implement this before moving any further:

1. Database connection through connection URL
2. Show database queries fired in console using configuration
3. Write test for the code implemented so far

My take:

First, let’s write junit test cases, I am going to inspect the code and find out where to start from.

I am practicing junit basics before moving any further. 

TimestampEntityTest.java:

Starting with @Test annotation to indicate that testSetAndGetLastLog is a test case inside which i have first created an instance of “TimestampEntity” then “setLastLog” method is called on the instance, and finally I have used assertion to very that the behaviour is as expected. 

Did the same thing for latestLog, with @Test annotation i have marked the the “testSetAndGetLatestLog” as test and in this test i have created instance of “TimestampEntity”, then called “setLatestLog” method on the instance of “Timestamp” and finally assertion is made to verify that bahaviour is as expected.

All the imports were automatic, I just had to add junit to classpath(via suggested actions).

TimestampServiceTest.java:

I will be using mockito for this, i will come back with more on this.

First, I am going to import org.mockito .Mockito.\*

Creating a Mock instance called “TimestampRepository” using “@Mock” annotation

Then, i will inject(provide mockito dependencies) to “TimestampService” instance

Now, i need a setUp method with “@BeforeEach” annotation, this method initializes the mocks using “MockitoAnnotations.openMocks(this)”.

Finally, the test case “testSaveTimestamp()” with “@Test” annotation,

I will mock the “TimestampEntity” here and the call method “saveTimestamp” on it, and then assert if the save method is called correctly by the service class.

All three tests were successful.

LogTimerTest.java:

This test will follow the same testing and mocking principles as TimestampServiceTest.java

First i will mock the “timestampService” using Mockito(@Mock), then inject dependencies to “LogTimer” instance using “@InjectMocks”and finally using “@BeforeEach” to run setUp() method. With these three we have successfully established an appropriate environment for testing “LogTimer.java”.

Now, finally using “@Test” on “ testCheckCondition()” (I have used such method names because this is a dummy project and will be later changed anyway.) under this method we declare “LocalDateTime” object called “currentTime”.

I will mock the behaviour of “conditionMet()” method using “when” and “then” from mockito library, this will the “LogTimer” object “logtimer” that when it’s “conditionMet()” method is called, it should return ‘true’.

Call the checkCondition on logTimer instance, and finally verify that timestampService saves the timestamp properly and is called exactly one time.

Next :    
Database connection through connection URL

using JDBC to achieve this, I have added this line of code to application.properties.

“spring.datasource.url=jdbc:postgresql://localhost:5432/TimeStamps”

This enables me to run PostgreSQL database locally on port 5432; also learnt how I can change it to 8008 or any other port available in my database server.

Username and password were also added.

Next :   
Show database queries fired in console using configuration

Added “spring.jpa.show-sql=true” to application.properties to show SQL logs

My approach is going to be TDD from this point forward.

Phase 4.5:

Update timestamp storage to maintain only one row in the database, optimizing data management.

Also add argument captor. 

Transactions in hibernate and JPA.

My Take : 

I will update the ‘testSaveTimestamp’ test in ‘TimestampServiceTest.java’, I will first rename the test to ‘testSaveTimestamp\_UpdatesExistingRow’ in this:

1. TimestampEntity  object called existingEntity is created. This is a mock timestampEntity that represents an existing row in DB.

2.  we will assume the existing entity ID to be 1.

3. setting the lastLog in this mock entity

Going to set id = 1L in TimestampEntity class, this will help me track that only one row is being used to store and update the timestamps in database.

   
well, just by adding @Transactional annotation I was able to make this saveTime method to behave as a transaction and it helps because I can manage saveTime method transaction declaratively.

With an if statement I check if the timestamp is empty or not, if it is not then timestampEntity timstamp at the first index.

Else it creates a new timestampEntity

Found a bug and fixed it, earlier i was setting the latest log before the last log which created discrepancy, both the logs were exact same. Now i have simply moved the setLastLog one line above the setLatestLog.   
   
this phase is done, I see row being updated and even after deleting all the data from the timstamp table in DB a new row was automatically created.

What is persistence?   
Prsistence is like saving a game.

When you save a game, your progress is stored even after you turn off the console. Similarly in software presistence means storing data so that it stays even when the program stops running. Its like saving files on a computer or keeping information in a database.

It includes :

Data storage, data retrival, data manipulation, data consistancy and data persistence layer.

Phase 5:

- Explanation: Upon printing a log statement, the application should retain its value along with the timestamp in a local variable.

- Example: After printing a log statement such as "Condition tested: True" at 14:30:00, the application should store this log message and its corresponding timestamp (14:30:00) in a local variable for further processing.

y Take :

In logTimer:

 declared a new instance variable called logInterval or type Duration, this shows the interval between log prints.

Creating a public method called updateLogInterval(Duration newInterval) to update log interval. Udating the checkCondition() method to use logInterval variable for comparision.

In TimestampService:

UpdatingveTimestamp() method to remove the unnecessary retrieval of all timestamps from database. Removed unnecessary check for empty list of timestamp.

Updating the logic to directly set the lastLog to the current latestLog  before saving to database.   
   
in TimestampServiceTest:   
added argumentCaptor to capture the timestampEntity argument passed to the save method of TimestampRepository. Updating the testSaveTimestamp() method to verify that the current argument is passed to the save method by capturing it and printing its value for inspection

Phase 6:

- Explanation: After successfully printing a log statement, the application should update the database with the timestamp of the successful print.

- Example: If the log statement "Condition tested: True" is successfully printed at 14:30:00, the application should update the database with the timestamp 14:30:00 indicating the success of this print.

This Phase appears to be completed. This phase’s objective is to update the database with the timestamp of the successful print after printing a log statement, this step has already been taken care of in the previous Phases 3 and 4.    
   
my “saveTimestamp” method in “TimestampService” class takes care of this; it updates  the DB with the latest timestamp, while @Transactional annotation also indicated that it runs within a transactional context.

this “saveTimestamp” method  retrieves the existing “TimestampEntity” from the DB or creates a new one if there is no data to fetch, then it updates the “lasstLog” then “LatestLog”and finally saves the entity to DB using “timestampRepository.save(timestampEntity)”.

Phase 7:

- Explanation: Retrieve information from the database and adjust log printing based on the success status.

- Example: If the last attempt to print the log statement was successful, no further action is needed. If it was unsuccessful, the application should retry printing the log statement within a specific timeframe.

Phase 8:

- Explanation: Compare the timestamp retrieved from the database with the current timestamp to ensure accurate logging.

- Example: When the application retrieves the timestamp from the database (e.g., 14:30:00) and compares it with the current timestamp (e.g., 14:32:00), it can determine the elapsed time since the last log statement was printed.

Phase 9:

- Explanation: Print the log statement within a 5-minute window only if the last attempt was unsuccessful; update the database accordingly.

- Example: If the last attempt to print the log statement failed at 14:30:00, the application should attempt to print it again within the next 5 minutes (until 14:35:00). If successful, update the database with a true value and the current timestamp.