Interview Round 1: Giridhara (Platform Engineering Intern... @ Wed Jan 17, 2024 2:45pm - 3:45pm (GMT+5:30) (chikkagiriprasad@gmail.com)

Problem Statement

Problem Statement: Transaction Log Analyzer

Background:

In a fintech application, transactions are logged with various details. These logs are crucial for monitoring and analyzing the flow of transactions. However, the logs can be extensive and not always straightforward to interpret.

Task:

You are given a list of transaction logs, each log entry containing the transaction ID, timestamp, and status (such as "SUCCESS", "FAILED", "PENDING"). Your task is to write a function that processes these logs to provide a summary of transactions.

Specific Requirements:

Count Transactions: Calculate the total number of transactions for each status category.
Identify Peak Time: Determine the time interval (e.g., hour) with the highest number of transactions.
Error Analysis: If there are any failed transactions, list the transaction IDs of the failed transactions.

Tests:

Write a test class with sample dataset to ensure the implementation is correct

Example string:

“TXN_1, 2023-11-22 01:00:00, SUCCESS”
“TXN_2, 2023-11-22 01:10:00, FAILED”

Here's a breakdown of how values are passed and returned:

1. Transaction Class:

- Represents a single transaction log entry.
- Values (id, timestamp, status) are passed to the constructor when creating a new Transaction object.

2. TransactionProcessor Class:

- Represents the main processing and analysis class for transaction logs.

- The constructor takes a List<Transaction> as an argument, initializing the list of transaction logs.

3. countTransactionsByStatus Method:

- Counts transactions for each status category and returns a Map<String, Integer> where keys are status categories and values are transaction counts.

4. identifyPeakTime Method:

- Determines the time interval with the highest number of transactions and returns a formatted string representing the peak time interval.

5. getFailedTransactionIds Method:

- Lists the transaction IDs of failed transactions and returns a List<String> containing these IDs.

6. TestTransactionProcessor Class:

- Provides a sample dataset and tests the functionality of the TransactionProcessor class.
- Displays the results of the transaction analysis in the console.

Values are passed as arguments to methods, and results are returned accordingly. The main method in TestTransactionProcessor creates an instance of TransactionProcessor with sample data and calls the methods to display the results.

Here provide detailed descriptions for each part of the code:

1. Transaction Class:

- **Purpose:** Represents a single transaction log entry.
- **Fields:**
  ~ id: Unique identifier for the transaction.
  ~ timestamp: Date and time when the transaction occurred.
  ~ status: The status of the transaction (e.g., "SUCCESS", "FAILED", "PENDING").
- **Constructor:**
- Initializes a Transaction object with values for id, timestamp, and status.
-

2. TransactionProcessor Class:

- **Purpose:** Processes and analyzes a list of transaction logs.
- **Fields:**
  ~ **transactionLogs:** List of Transaction objects representing the transaction logs.
- **Constructor:**Initializes a TransactionProcessor object with a list of transaction logs.
- **Methods:**
  ~ **countTransactionsByStatus():** Counts transactions for each status category and returns a map with status categories as keys and transaction counts as values.
  ~ **identifyPeakTime():** Determines the time interval with the highest number of transactions and returns a formatted string representing the peak time interval.
  ~ **getFailedTransactionIds():** Lists the transaction IDs of failed transactions and returns a list of these IDs.

3. countTransactionsByStatus Method:

- **Purpose:** Counts transactions for each status category.
- **Algorithm:**
  ~ Iterates through the list of transaction logs.
  ~ Updates a map (statusCount) with counts for each transaction status category.
  ~ Returns the map containing status categories and their respective transaction counts.
- **Return Type:** Map<String, Integer>.

4. identifyPeakTime Method:

- **Purpose:** Determines the time interval with the highest number of transactions.
- **Algorithm:**
  ~ Iterates through the list of transaction logs.
  ~ Updates a map (hourCount) with counts for each hour of the day.
  ~ Finds the hour with the maximum transactions.
  ~ Returns a formatted string representing the peak time interval.
- **Return Type:** String.

5. getFailedTransactionIds Method:

- **Purpose:** Lists the transaction IDs of failed transactions.
- **Algorithm:**
  ~ Iterates through the list of transaction logs.
  ~ Adds the IDs of transactions with a status of "FAILED" to a list.
  ~ Returns the list containing IDs of failed transactions.
- **Return Type:** List<String>.

6. TestTransactionProcessor Class:

- **Purpose:** Provides a sample dataset and tests the functionality of the TransactionProcessor class.
- **Methods:**
  ~ main(String[] args): Entry point for running the test.
- **Functionality:**
  ~ Creates a sample list of transactions.
  ~ Instantiates a TransactionProcessor object with the sample data.
  ~ Calls various methods of TransactionProcessor to display the results of transaction analysis in the console.

These detailed descriptions provide a clear understanding of the purpose, structure, and functionality of each component in the Java code.