COMPSCI 532: Systems for Data Science
Homework 4 - Flink - Turn-In Report

Submitted by:
Abhilasha Senapati
Avinash Nandyala

Deliverables:

Question 1: *How do instantiations of the Alert class reach the log4j logging mechanism? Please list the classes that the Alerts are passed through. A place to start is by looking at which files import the Alert class and which files import the log4j logging library (org.sl4j).*

Ans.
The Alert instances flow through the following classes to reach log4j logging:

1. FRAUDDETECTOR (FraudDetector.java)
   - Creates Alert instances using "new Alert()"
   - Sets alert ID with "alert.setId(transaction.getAccountId())"
   - Emits alerts via "collector.collect(alert)"

2. FRAUDDETECTIONJOB (FraudDetectionJob.java)
   - Receives DataStream<Alert> from FraudDetector
   - Passes alerts to AlertSink via "alerts.addSink(new AlertSink())"

3. ALERTSINK (org.apache.flink.walkthrough.common.sink.AlertSink)
   - This is from the Flink walkthrough common library
   - Receives Alert instances from the DataStream
   - Uses slf4j logging internally

4. LOG4J2 CONFIGURATION (log4j2.properties)
   - Defines logger for AlertSink: "logger.sink.name = org.apache.flink.walkthrough.common.sink.AlertSink"
   - Sets logging level to INFO
   - Outputs to console via ConsoleAppender

LOGGING MECHANISM:
- slf4j as logging facade (via log4j-slf4j-impl dependency)
- Log4j2 as actual logging implementation
- AlertSink uses slf4j internally to log Alert instances

CLASSES THAT IMPORT ALERT:

- FraudDetector.java - imports "org.apache.flink.walkthrough.common.entity.Alert"
- FraudDetectionJob.java - imports "org.apache.flink.walkthrough.common.entity.Alert"

Question 2: *Why do you think that the next sequence of transactions should not generate an alert?*
Transaction 1: (Account 1, Timestamp 1.005 s, $2, Zip 01003)
Transaction 2: (Account 2, Timestamp 20 s, $501, Zip 02115)
Transaction 3: (Account 1, Timestamp 55 s, $1000, Zip 78712)

Ans.
This sequence should NOT generate an alert because:

1. DIFFERENT ACCOUNTS: Transaction 2 is for Account 2, not Account 1
   - The fraud detector looks for patterns within the same account
   - Transaction 2 ($501) is from Account 2, so it doesn't relate to Account 1's transactions

2. DIFFERENT ZIP CODES: Transaction 1 (01003) and Transaction 3 (78712) have same account but different zip codes
   - The enhanced fraud detector requires both small and large transactions to be in the SAME zip code
   - Transaction 1: Zip 01003
   - Transaction 3: Zip 78712
   - These are different locations, so no fraud pattern is detected

3. NO VALID FRAUD PATTERN: There's no small transaction followed by a large transaction from the same account in the same zip code
   - Transaction 1: Account 1, $2 (small), Zip 01003
   - Transaction 2: Account 2, $501 (large), Zip 02115 - Different account, different zip
   - Transaction 3: Account 1, $1000 (large), Zip 78712 - Same account as Transaction 1, but different zip

Since the zip codes are different, no alert is generated.

Question 3: *In this homework we focused on a credit fraud use case, can you think of any other real life situations where a stream processing approach might be useful? Why do you think so?*

Ans.
Yes! Stream processing is highly useful because it enables real-time analysis, allowing systems to generate results instantly instead of waiting for hours or days. It is designed to be scalable, capable of handling millions of events per second efficiently by processing data as it arrives rather than storing it first. This makes systems more responsive, enabling immediate action when problems or anomalies are detected. Additionally, stream processing is cost-effective, as it

reduces the need for large-scale data storage and delayed batch analysis. Here are some examples:

1. E-COMMERCE FRAUD DETECTION
   - Why: Online shopping generates millions of transactions per second
   - Use: Detect unusual buying patterns, stolen credit cards, fake accounts
   - Example: Someone buying expensive items from different locations in short time

2. CYBERSECURITY MONITORING
   - Why: Hackers attack systems constantly, need immediate response
   - Use: Detect suspicious login attempts, malware, data breaches
   - Example: Multiple failed logins from different countries in 5 minutes

3. STOCK MARKET TRADING
   - Why: Stock prices change every millisecond, need real-time decisions
   - Use: Detect insider trading, market manipulation, high-frequency trading
   - Example: Unusual trading patterns before major announcements

4. SOCIAL MEDIA MONITORING
   - Why: Billions of posts, comments, and messages every day
   - Use: Detect fake news, hate speech, trending topics
   - Example: Sudden spike in mentions of a company's stock

The key is that these situations involve:
- LARGE AMOUNTS of data
- NEED FOR IMMEDIATE ACTION
- CONTINUOUS DATA STREAMS
- PATTERN DETECTION
- REAL-TIME DECISIONS

Stream processing is perfect for these scenarios because it can handle the volume and speed while providing immediate insights and responses.