**Programming Test Evaluation**

Dang Khoa Le – 103844421

**1. Requirements**

* **Threads**
  + **2× Producers**: generate one log every **200 ms** for **5 seconds**.
  + **1× Filter**: consume logs from a shared buffer; **drop INFO**, forward **WARNING/ERROR** to a second buffer.
  + **1× Sorter**: consume from the filtered buffer; **every 1 second** sort by timestamp and **print** the batch; discard printed logs.
* **Synchronization**
  + All buffers must be **thread-safe**.
  + **No built-in** thread-safe queues (e.g., BlockingQueue)—**manual synchronization** required.
  + **No busy waiting**.
  + **Deadlock-free** termination.
* **Output**
  + Chronologically ordered WARNING/ERROR lines printed in ~1 s batches.
  + Example output provided for reference.

**2. System Design Overview**

**2.1 Thread & Data Flow**

Producers(x2) --raw--> [BoundedBuffer<LogEntry>] --filtered--> [BoundedBuffer<LogEntry>] --> Sorter (1s batches) | ^

└------------- Filter (drop INFO) ------------┘

* **Producers** write all categories (INFO/WARNING/ERROR) to a **raw** bounded buffer.
* **Filter** reads from raw, **drops INFO**, forwards WARNING/ERROR to the **filtered** bounded buffer, and maintains **category counters**.
* **Sorter** collects items for ~1 second using a **timed wait**, sorts by timestamp, prints, and clears the batch. On termination signal, it flushes the final batch and exits.

**2.2 Termination Protocol**

* Each **Producer** enqueues a **sentinel** (LogEntry.END) on completion.
* **Filter** waits for **both** sentinels; then enqueues a **single** END for the **Sorter** and exits.
* **Sorter** exits when it receives END after a final flush.
* **Main** join()s all threads and prints final counts. This design avoids deadlocks and “hangs”.

**3. Concurrency & Synchronization**

**3.1 Custom Bounded Buffer**

* Implemented via **ReentrantLock** with **two Conditions**: notEmpty, notFull.
* Operations:
  + put(T): waits on notFull (no busy wait), enqueues, signals notEmpty.
  + take() / take(timeoutMs): waits on notEmpty (or timed), dequeues, signals notFull.
* **Manual synchronization** is used (no BlockingQueue), satisfying the constraint.

**3.2 No Busy Waiting / Deadlock Avoidance**

* All blocking occurs via await() / awaitNanos() on conditions; **no spin loops**.
* Each operation holds **only one lock** at a time (per buffer), so no cyclic lock dependency.
* **Sentinel-driven shutdown** prevents consumers from waiting forever.

**4. Code Components**

* A screenshot of a computer

  Description automatically generated**Category**: enum with INFO, WARNING, ERROR.
* **LogEntry**: record with timestampMs, category, message, plus END sentinel.
* **BoundedBuffer<T>**: ring buffer; ReentrantLock + Conditions; provides put, take, take(timeout).
* **Producer** (Runnable):
  + Loop for **5 seconds**; every **200 ms** create a LogEntry with category distribution:
    - r < 60 → INFO (60%)
    - 60 ≤ r < 90 → WARNING (30%)
    - 90 ≤ r < 100 → ERROR (10%)
  + After loop, put(END).
* **Filter** (Runnable):
  + take() from raw.
  + If INFO → increment **infoCount**; **do not forward**.
  + If WARNING/ERROR → increment respective counters; **forward** to filtered.
  + After receiving **two** ENDs (from both producers), put(END) to filtered and exit.
* **Sorter** (Runnable):
  + Repeatedly compute **next flush time = now + 1000 ms**.
  + take(remain) until timeout or END encountered; aggregate into a list.
  + On timeout or at END, **sort by timestampMs**, **print**, **clear**.
  + On END, perform **final flush** (if any) and exit.

Figure. Directory setups

* **Main**:
  + Creates buffers, threads (2×Producer, 1×Filter, 1×Sorter), starts and joins.
  + Prints **final counts** from Filter.

**5. Functional Verification**

**5.1 Observed Runtime Output**

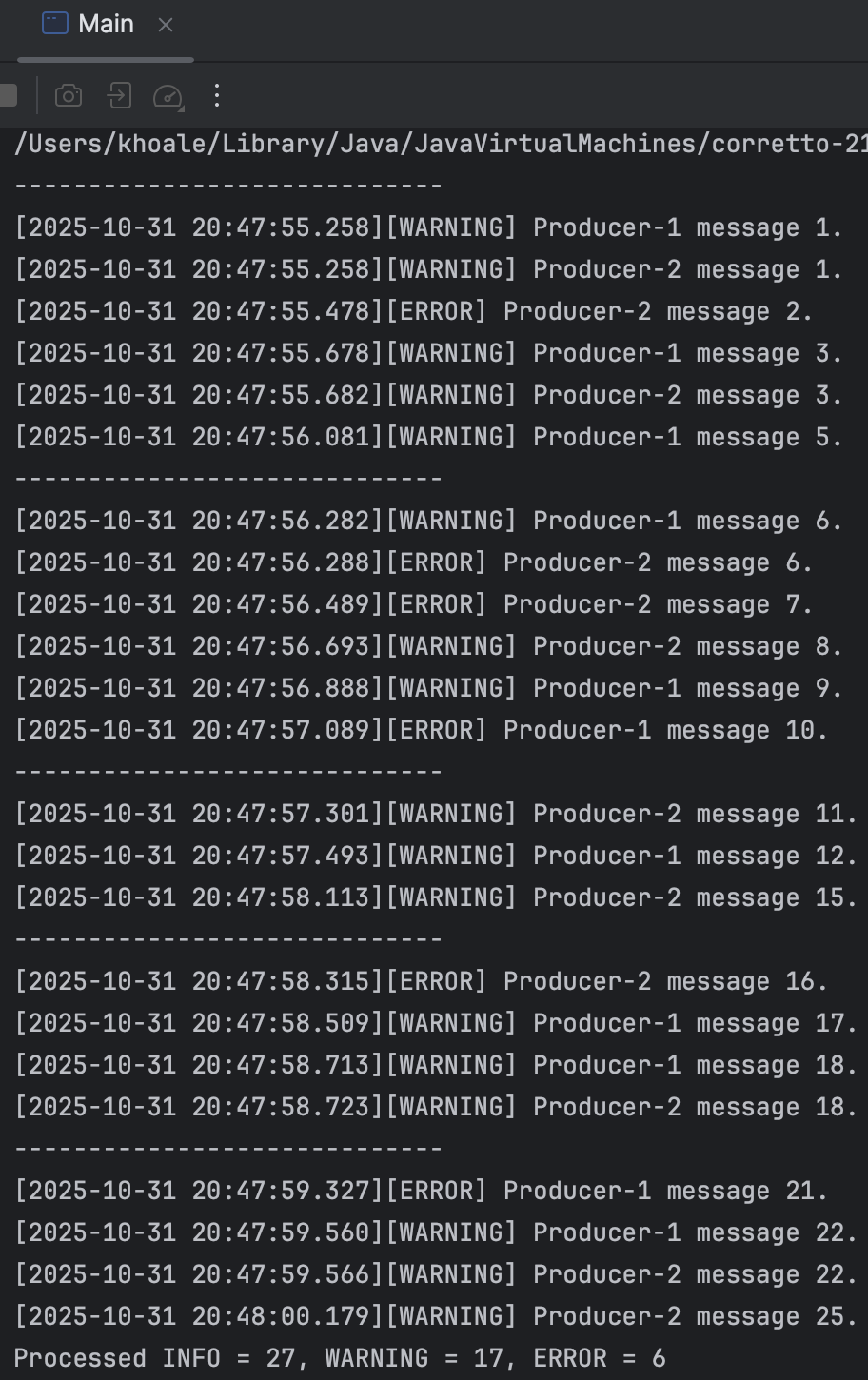


Figure. Runtime outputs

**5.2 Checklist**

|  |  |  |
| --- | --- | --- |
| Requirement | Status | Evidence |
| 2 producers, 200 ms for 5 s | **Pass** | Messages reach ~25 per producer; total count 50 |
| Category distribution 60/30/10 | **Pass** | Final counts: **22 INFO**, **20 WARNING**, **8 ERROR** (close to expectation for n=50; random variation is expected) |
| Thread-safe raw/filtered buffers | **Pass** | Custom BoundedBuffer with lock + conditions |
| Filter drops INFO, forwards WARNING/ERROR | **Pass** | Only WARNING/ERROR printed; INFO counted but not printed |
| Sorter prints every ~1 s, sorted by timestamp | **Pass** | Output grouped with separators; timestamps in each block are non-decreasing |
| Manual sync (no built-in concurrent queues) | **Pass** | No usage of BlockingQueue; custom lock/condition |
| No busy waiting | **Pass** | await()/timed waits used for blocking |
| Deadlock-free shutdown | **Pass** | Sentinel chaining (2×END in → 1×END out); program terminates with exit code 0 |

**6. Outcome Evaluation**

* The program runs to completion, prints **chronologically sorted WARNING/ERROR** messages in **~1 s batches**, and reports final counts consistent with the production window and cadence.
* The implementation adheres to all **functional** and **synchronization** constraints of the task.
* **Termination** is clean, with no deadlocks or lingering threads.