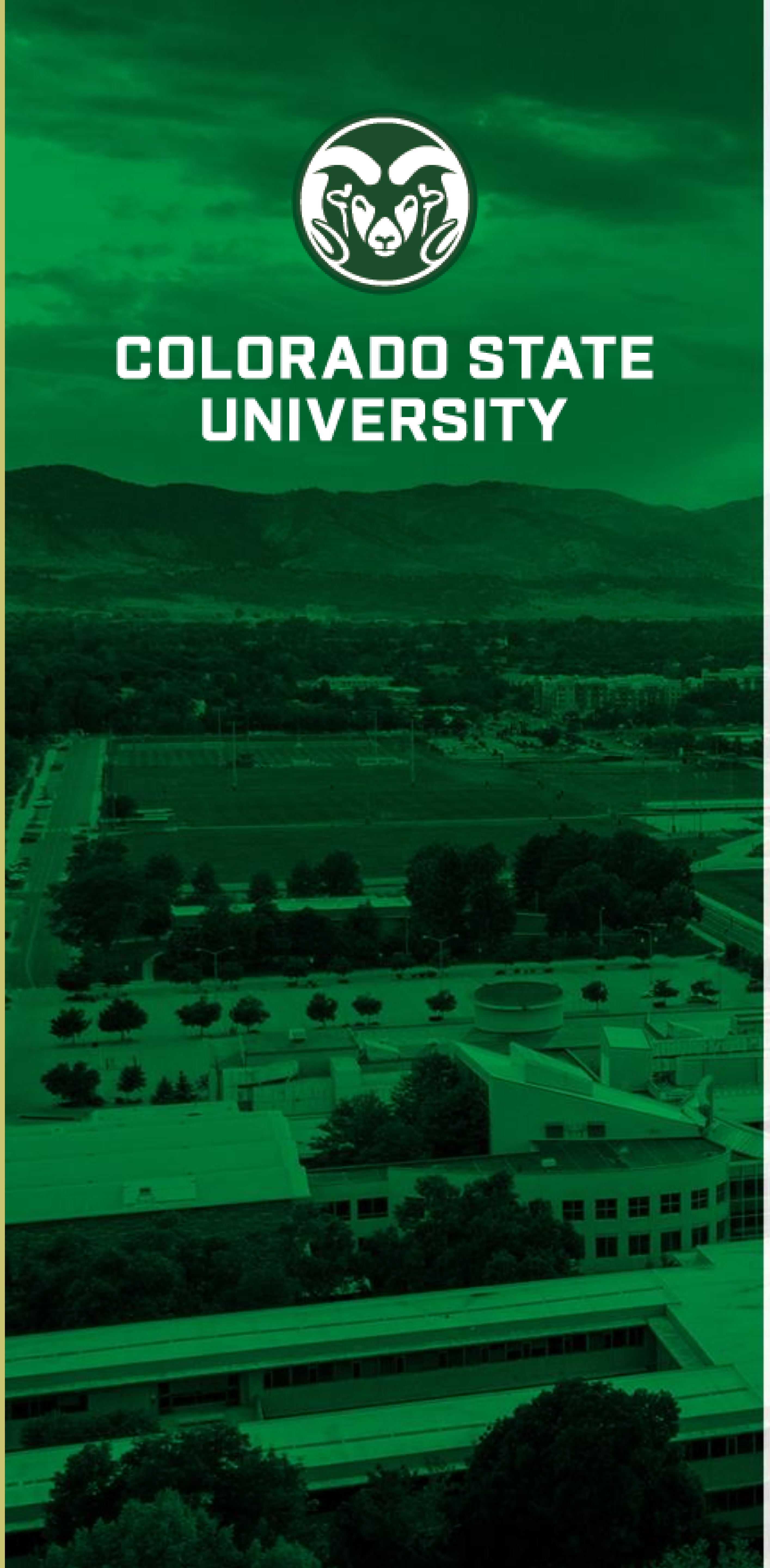


COLORADO STATE UNIVERSITY

HW 5 Overview

Course : Operating System

Topic: Synchronization of Producer and Consumer Threads

- Goal: Learn how threads share a buffer safely in Java.
  Students will:
- Implement a bounded circular buffer
- Use wait() and notify() for synchronization
- Coordinate multiple producers and consumers
- Verify correct ordering and matching of data

Topic: Synchronization of producer and Consumer Threads

Think of:

- 👦🍳 producers = Chefs producing dishes
- 🍽️ Buffer = Serving table with limited plates
- 🧍 Consumers = Waiters picking up dishes

Rules:

- Chefs wait if table is full.
- Waiters wait if table is empty.
- Everyone shares the same table safely.

What You'll Build

Four Java classes:

Buffer.java – Shared circular FIFO storage

Producer.java – Producer threads (add letters)

Consumer.java – Consumer threads (remove letters)

Coordinator.java – Main program (start threads & check results)

Flowchart

Coordinator

↓

Creates Buffer + Threads

↓

Producers produce → Buffer insert (wait if full)

↓

Consumers consume ← Buffer remove (wait if empty)

↓

notifyAll() → wake waiting threads

↓

Coordinator verifies consonant counts

↓

End

Buffer (Shared Area)

Holds fixed # of items (10 – 15).

FIFO → first inserted = first removed.

Uses synchronized blocks.

wait() → pause if buffer full/empty.

notifyAll() → wake threads after insert/delete.

Guarantees no data loss & no simultaneous access.

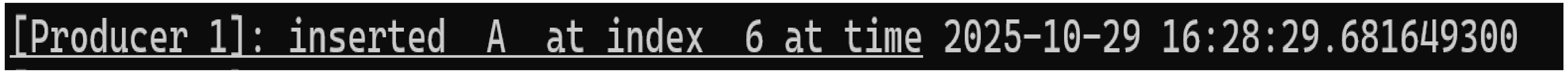
Producer

Produces random capital letters: (char)('A'+rand.nextInt(26))

Uses alphabetSeed for repeatable sequence.

Inserts letters into buffer; waits if full.

Prints:



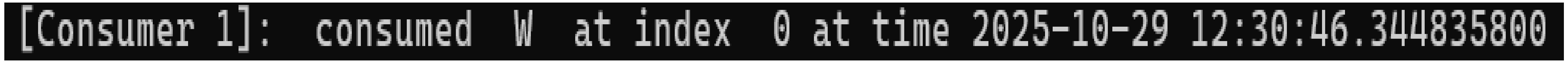
```
[Producer 1]: inserted  A  at index  6 at time 2025-10-29 16:28:29.681649300
```

Counts consonants generated.

Consumer

Removes letters from buffer; waits if empty.

Prints:



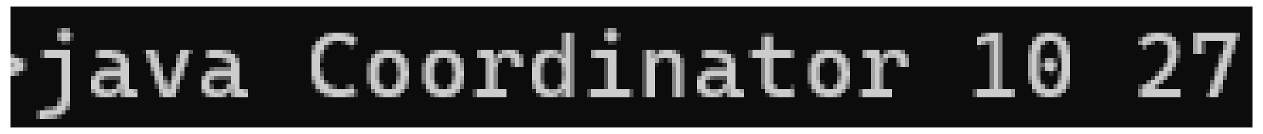
```
[Consumer 1]:  consumed  W  at index  0 at time 2025-10-29 12:30:46.344835800
```

Counts consonants consumed

Uses notifyAll() so producers can resume.

Coordinator (Main)

Run as:



```
java Coordinator 10 27
```

Seed decides buffer size & # threads.

alphabetSeed decides letter sequence.

Random ranges:

Buffer size 10–15

Total items 20–40

3–7 producers & consumers

Starts threads → joins them → checks if

# consonants produced = # consumed.

Seeds Example

Command:

```
java Coordinator 2022 370
```

→ Seed = 2022 sets sizes and counts.

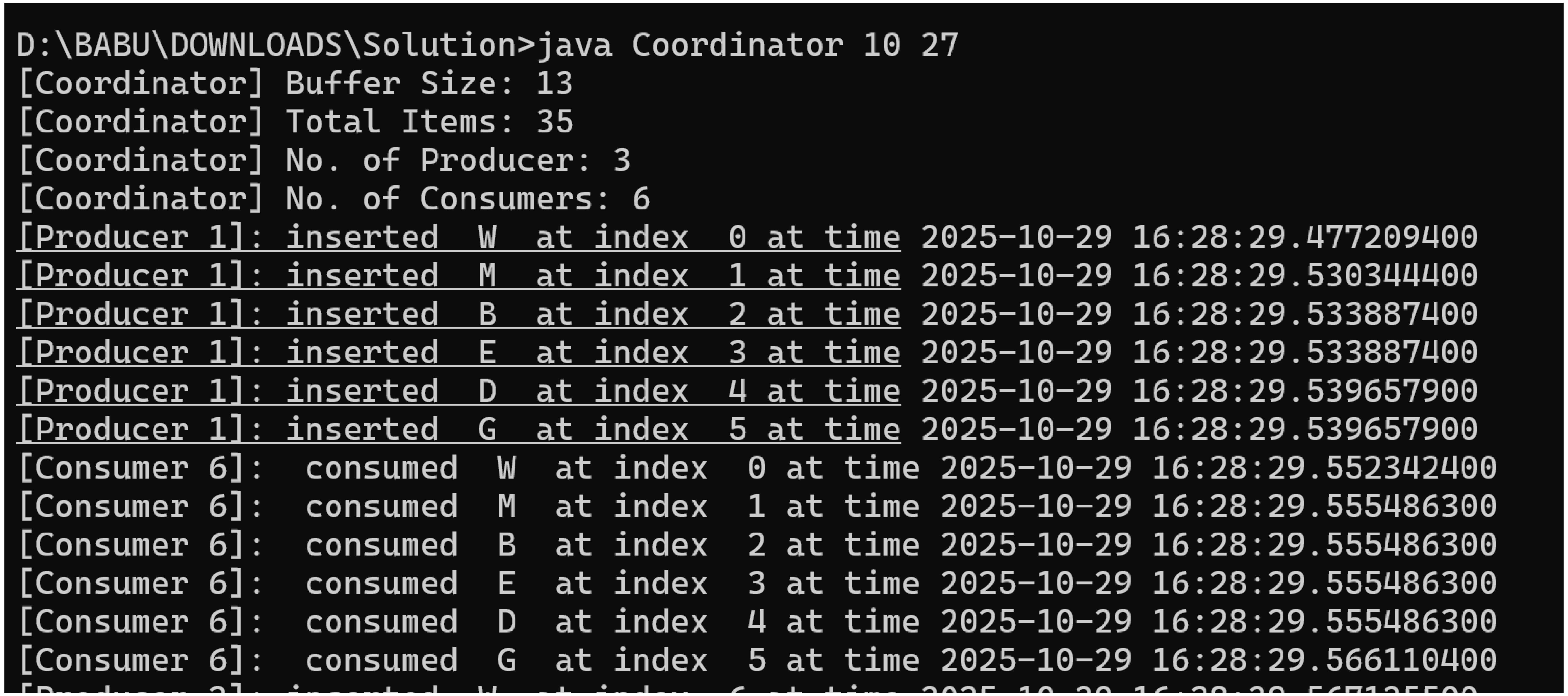
→ alphabetSeed = 370 sets letter pattern.

Same inputs = same output each run.

Correctness Requirements

- ✅ Consume each item exactly once.
- ✅ FIFO order preserved.
- ✅ Producers wait when full; consumers wait when empty.
- ✅ Threads terminate cleanly (no deadlocks).
- ✅ Consonant counts match.

Example Output



```
D:\BABU\DOWNLOADS\Solution>java Coordinator 10 27
[Coordinator] Buffer Size: 13
[Coordinator] Total Items: 35
[Coordinator] No. of Producer: 3
[Coordinator] No. of Consumers: 6
[Producer 1]: inserted  W  at index  0 at time 2025-10-29 16:28:29.477209400
[Producer 1]: inserted  M  at index  1 at time 2025-10-29 16:28:29.530344400
[Producer 1]: inserted  B  at index  2 at time 2025-10-29 16:28:29.533887400
[Producer 1]: inserted  E  at index  3 at time 2025-10-29 16:28:29.533887400
[Producer 1]: inserted  D  at index  4 at time 2025-10-29 16:28:29.539657900
[Producer 1]: inserted  G  at index  5 at time 2025-10-29 16:28:29.539657900
[Consumer 6]:  consumed  W  at index  0 at time 2025-10-29 16:28:29.552342400
[Consumer 6]:  consumed  M  at index  1 at time 2025-10-29 16:28:29.555486300
[Consumer 6]:  consumed  B  at index  2 at time 2025-10-29 16:28:29.555486300
[Consumer 6]:  consumed  E  at index  3 at time 2025-10-29 16:28:29.555486300
[Consumer 6]:  consumed  D  at index  4 at time 2025-10-29 16:28:29.555486300
[Consumer 6]:  consumed  G  at index  5 at time 2025-10-29 16:28:29.566110400
```

Submission Checklist

Submit one .zip or .tar file containing:

Coordinator.java

Producer.java

Consumer.java

Buffer.java

Makefile with build, run, clean

Name file like: FirstName-LastName-HW5.zip

Grading

Major deductions :

Unbounded buffer(-20)

Using Thread.sleep() for sync(-25)

Using advanced Java sync classes(-80)

Using flag variables instead of wait()/notify()(-80)

Key Takeaways

Threads must coordinate shared resources carefully.

wait() + notifyAll() solve producer–consumer conflicts.

Seeds make random behavior reproducible.

Deadlock-free, deterministic synchronization

COLORADO STATE UNIVERSITY

Thank you

Masfiq Reza

masfiq.reza@colostate.edu