HW5: Programming Assignment version_10/29/2025_9:00PM
SYNCHRONIZATION of Producer and CONSUMER Threads

This assignment requires the creation of multiple producers and consumer threads that access a buffer using synchronization. You will implement a solution for the bounded-buffer producer-consumer problem using threads in Java.

Due Date: November 13, 2025

Extended Due Date with 10% penalty: November 15, 2025

1. Description of Task

You are required to implement this assignment in Java. It assumes that n items are to be produced and consumed.

1. The Bounded Buffer (`Buffer.java`): This buffer can hold a fixed number of items. This buffer needs to be a first-in first-out (FIFO) buffer. You should implement this as a Circular Buffer that satisfies the FIFO. There should be exactly one instance of the buffer. The producer and the consumers must reference the same buffer.

2. Producers (`Producer.java`): The producers are responsible for producing data items to be added to the buffer. If the buffer is full, a producer must wait for a consumer to consume at least one item before it can add a new item to the buffer. If there are *g* producers and the total number of items *n*, each producer is required to produce *n/g* items. If the *n* isn't perfectly dividable by the *g*, then the last producer will take care of the remains. For example, if we have 10 items to produce and 3 producers, then the first 2 producers will produce 3 items, and the last producer will make 4 items since 10 = 3 * 3 + 1 = 3 * 2 + 4 * 1. **Taking the remains by the last one is also applied to consumers as well (Using '/' and '%' will be helpful).** The item that the producer adds to the buffer is a random **large** alphabet character.

When a producer adds a random alphabet to the buffer, the alphabet is chosen by these rules:

1. Take an alphabetSeed as an argument for Producer constructor.
2. Save the Random object with the alphabetSeed into your member variable *rand* in the constructor.
   - Impot java.util.Random
3. When the random alphabet required to add, generate the random character as:
   - (char) ('A' + *rand*.nextInt(26))

When the producer successfully inserts an item in the buffer it should print the location of insertion and time when insertion occurs with microsecond resolution, using this format:

```
[Producer 1]: inserted R at index 0 at time 2022-10-11 07:33:01.049
```

After the producer inserts the random alphabet character, the random character should be checked whether it is consonant or not. If the character is consonant, then add 1 to *numOfConsonants* which is a member variable of Producer. The *numOfConsonants* is for checking the num of consonants which are generated from the instance, and *getNumOfGeneratedConsonants()* needs to return the *numOfConsonants* as a member method.

3. Consumers (`Consumer.java`): The consumers are responsible for consuming elements, produced by the producer, from the buffer. If the buffer is empty, the consumers must wait for the producer to add an item to the buffer. There may be one or more consumer threads running at the same time.

When a consumer successfully removes an item from the buffer it should print the location of removal and time when removal occurs with microsecond resolution, using this format:

```
[Consumer 3]:  consumed R at index 0 at time 2022-10-11 07:33:01.090
```

When a consumer successfully consumes an item from the buffer, it should also check whether the consumed character is consonant or not. If the consumed character is consonant, add 1 to its member variable *numOfConsonants*. This class should have a function named *`getNumOfConsumedConsonants()`*, which returns an integer which is the num of consonants by the Consumer.

Mind that the statement of Producers producing an item should be underlined, so that it is easy to distinguish a statement from Producer and Consumer.

You will use `wait()` and `notify()` as the primitives to synchronize access to the buffer.

The producer class object should take these arguments:

i. Copy of the instance of the buffer it will access in common with other Producer and other Consumers,
ii. Number of elements that producer should produce,
iii. The ID, which is the number of the producer thread (i.e.: the first producer thread should be 1, the second should be 2, etc.)
iv. The *alphabetSeed* which is used by the random number producer to generate the random large alphabet character in the range [A, Z].

The consumer class objects should take these arguments:

i. Copy of the instance of the buffer it will access in common with the Producers and other Consumers,
ii. Number of elements that this thread consumer should consume. This is the total number of elements to be consumed divided by the total number of consumers (if it is evenly matched),
iii. The ID, which is the number of the consumer thread (i.e.: the first consumer thread should be 1, the second should be 2, etc.)

4. Main / Calling program (`Coordinator.java`): Your main program should accept the *Seed* and *alphabetSeed* as command line arguments. Using this *Seed*, the following elements must be randomly initialized.

Note that all four intervals are inclusive, i.e. end points are included.

i.	Number of elements in buffer/buffer size	(between 10 and 15)	i.e. [10,15]
ii.	Number of items to be produced and consumed	(between 20 and 40)	i.e. [20,40]
iii.	Number of consumers	(between 3 and 7)	i.e. [3,7]
iv.	Number of producers	(between 3 and 7)	i.e. [3,7]

Each producer thread terminates when the specified number of items has been produced. After this, you need to use the methods, getNumOfGeneratedConsonants() and getNumOfConsumedConsonants(), for each of the Producers and Consumers to get the **total** number of Generated/Consumed consonants.

If the number of generated consonants is the same as the number of consumed consonants, the Coordinator.java program prints the following:

```
The generated & consumed nums of consonants are the same as shown: 3
```

Correctness Verification:

- The items produced should match the items consumed.
- The circular buffer should work as intended. Only one thread should be able to access the buffer at a time.
- An item can be consumed only after it has been generated. However, if the consumption is very quick, within the smallest time resolution, generation/consumption may appear to happen at the same time, and the reports may get printed in wrong order, if the consumer printing occurs first. To avoid this use `System.out.flush()`

2. Task Requirements

1. Implement a FIFO Circular Buffer that can hold a fixed number of items at a time. The buffer must follow the First-In, First-Out (FIFO) rule and be shared among all Generator and Consumer threads. Only one thread should access or modify the buffer at a time using proper synchronization.
2. The number of items to be produced and consumed should be distributed equally among the Generator and Consumer threads whenever possible. If the total number of items is not perfectly divisible, the last thread (Producer or Consumer) should handle the remaining items. An alphabetSeed is passed to each Producer to create consistent random alphabets.
3. A Producer thread must wait if the buffer is full, and a Consumer thread must wait if the buffer is empty. Use wait() and notifyAll() methods for synchronization. Do not use Thread.sleep() for coordination.
4. Each Producer produces random uppercase alphabets (A–Z) and inserts them into the buffer. Each Consumer removes those alphabets for processing.

   Both Producer and Consumer threads should print their activity in an easy-to-read format. Example:

   [Producer 1]: inserted R at index 0
   [Consumer 2]: consumed R at index 0
   (strict spacing alignment are not required.)

5. After all threads complete, verify that:
   The total number of generated consonants equals the total number of consumed consonants.
   Each item inserted by a Producer has been consumed exactly once.

6. The program should be free of deadlocks and run to completion for all valid combinations of:

   Buffer size
   Number of items
   Number of Producers and Consumers

7. The solution must work correctly when executed multiple times with different input seeds:
   java Coordinator <Seed> <alphabetSeed>

3. Files Provided

Files provided for this assignment include: the description file (this file) and skeleton Coordinator.java and Producer.java files. This can be downloaded as a zip file from the course Canvas assignment page.

4. Example Outputs:

1. **Example 1**: Set *Seed* as 10 and *alphabetSeed* as 27

```
$ 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 2022-10-11 09:22:51.509
[Producer 1]: inserted  M  at index  1 at time 2022-10-11 09:22:51.547
[Producer 1]: inserted  B  at index  2 at time 2022-10-11 09:22:51.548
[Producer 1]: inserted  E  at index  3 at time 2022-10-11 09:22:51.548
[Producer 1]: inserted  D  at index  4 at time 2022-10-11 09:22:51.548
[Producer 1]: inserted  G  at index  5 at time 2022-10-11 09:22:51.548
[Producer 1]: inserted  A  at index  6 at time 2022-10-11 09:22:51.549
[Producer 1]: inserted  S  at index  7 at time 2022-10-11 09:22:51.549
[Producer 1]: inserted  A  at index  8 at time 2022-10-11 09:22:51.549
[Producer 1]: inserted  B  at index  9 at time 2022-10-11 09:22:51.549
[Producer 1]: inserted  R  at index 10 at time 2022-10-11 09:22:51.550
[Consumer 6]:  consumed  W  at index  0 at time 2022-10-11 09:22:51.550
[Consumer 6]:  consumed  M  at index  1 at time 2022-10-11 09:22:51.550
[Consumer 6]:  consumed  B  at index  2 at time 2022-10-11 09:22:51.551
[Consumer 6]:  consumed  E  at index  3 at time 2022-10-11 09:22:51.551
[Consumer 6]:  consumed  D  at index  4 at time 2022-10-11 09:22:51.551
[Consumer 6]:  consumed  G  at index  5 at time 2022-10-11 09:22:51.551
[Consumer 6]:  consumed  A  at index  6 at time 2022-10-11 09:22:51.551
[Consumer 6]:  consumed  S  at index  7 at time 2022-10-11 09:22:51.552
[Consumer 6]:  consumed  A  at index  8 at time 2022-10-11 09:22:51.552
[Consumer 6]:  consumed  B  at index  9 at time 2022-10-11 09:22:51.552
[Consumer 5]:  consumed  R  at index 10 at time 2022-10-11 09:22:51.552
[Producer 3]: inserted  W  at index 11 at time 2022-10-11 09:22:51.552
[Producer 3]: inserted  M  at index 12 at time 2022-10-11 09:22:51.553
[Producer 2]: inserted  W  at index  0 at time 2022-10-11 09:22:51.553
[Producer 3]: inserted  B  at index  1 at time 2022-10-11 09:22:51.553
[Producer 3]: inserted  E  at index  2 at time 2022-10-11 09:22:51.553
[Consumer 1]:  consumed  W  at index 11 at time 2022-10-11 09:22:51.554
[Consumer 1]:  consumed  M  at index 12 at time 2022-10-11 09:22:51.554
[Consumer 1]:  consumed  W  at index  0 at time 2022-10-11 09:22:51.554
[Consumer 1]:  consumed  B  at index  1 at time 2022-10-11 09:22:51.554
[Consumer 1]:  consumed  E  at index  2 at time 2022-10-11 09:22:51.554
[Producer 3]: inserted  D  at index  3 at time 2022-10-11 09:22:51.555
[Producer 3]: inserted  G  at index  4 at time 2022-10-11 09:22:51.555
[Producer 3]: inserted  A  at index  5 at time 2022-10-11 09:22:51.555
[Producer 3]: inserted  S  at index  6 at time 2022-10-11 09:22:51.555
[Producer 2]: inserted  M  at index  7 at time 2022-10-11 09:22:51.555
[Producer 2]: inserted  B  at index  8 at time 2022-10-11 09:22:51.555
[Producer 2]: inserted  E  at index  9 at time 2022-10-11 09:22:51.555
[Producer 2]: inserted  D  at index 10 at time 2022-10-11 09:22:51.556
[Producer 2]: inserted  G  at index 11 at time 2022-10-11 09:22:51.556
[Producer 2]: inserted  A  at index 12 at time 2022-10-11 09:22:51.556
[Producer 2]: inserted  S  at index  0 at time 2022-10-11 09:22:51.556
[Producer 2]: inserted  A  at index  1 at time 2022-10-11 09:22:51.556
[Producer 2]: inserted  B  at index  2 at time 2022-10-11 09:22:51.556
[Consumer 5]:  consumed  D  at index  3 at time 2022-10-11 09:22:51.556
[Consumer 5]:  consumed  G  at index  4 at time 2022-10-11 09:22:51.556
[Consumer 5]:  consumed  A  at index  5 at time 2022-10-11 09:22:51.556
```

```
[Consumer 5]:  consumed  S  at index  6 at time 2022-10-11 09:22:51.557
[Consumer 4]:  consumed  M  at index  7 at time 2022-10-11 09:22:51.557
[Consumer 4]:  consumed  B  at index  8 at time 2022-10-11 09:22:51.557
[Consumer 4]:  consumed  E  at index  9 at time 2022-10-11 09:22:51.557
[Consumer 4]:  consumed  D  at index 10 at time 2022-10-11 09:22:51.557
[Consumer 4]:  consumed  G  at index 11 at time 2022-10-11 09:22:51.557
[Consumer 3]:  consumed  A  at index 12 at time 2022-10-11 09:22:51.557
[Consumer 3]:  consumed  S  at index  0 at time 2022-10-11 09:22:51.557
[Consumer 3]:  consumed  A  at index  1 at time 2022-10-11 09:22:51.557
[Consumer 3]:  consumed  B  at index  2 at time 2022-10-11 09:22:51.557
[Producer 3]: inserted  A  at index  3 at time 2022-10-11 09:22:51.558
[Producer 3]: inserted  B  at index  4 at time 2022-10-11 09:22:51.558
[Producer 3]: inserted  R  at index  5 at time 2022-10-11 09:22:51.558
[Producer 2]: inserted  R  at index  6 at time 2022-10-11 09:22:51.558
[Producer 3]: inserted  D  at index  7 at time 2022-10-11 09:22:51.558
[Producer 3]: inserted  I  at index  8 at time 2022-10-11 09:22:51.558
[Consumer 2]:  consumed  A  at index  3 at time 2022-10-11 09:22:51.558
[Consumer 2]:  consumed  B  at index  4 at time 2022-10-11 09:22:51.558
[Consumer 2]:  consumed  R  at index  5 at time 2022-10-11 09:22:51.559
[Consumer 2]:  consumed  R  at index  6 at time 2022-10-11 09:22:51.559
[Consumer 2]:  consumed  D  at index  7 at time 2022-10-11 09:22:51.559
[Consumer 3]:  consumed  I  at index  8 at time 2022-10-11 09:22:51.559
The generated & consumed numbers of consonants are the same as shown: 25
```

2. **Example 2**: Set *Seed* as 2022 and *alphabetSeed* 370

```
$ java Coordinator 2022 370
[Coordinator] Buffer Size: 13
[Coordinator] Total Items: 24
[Coordinator] No. of Producer: 5
[Coordinator] No. of Consumers: 6
[Producer 1]: inserted  U  at index  0 at time 2022-10-11 09:33:41.220
[Producer 1]: inserted  B  at index  1 at time 2022-10-11 09:33:41.257
[Producer 1]: inserted  X  at index  2 at time 2022-10-11 09:33:41.257
[Producer 1]: inserted  A  at index  3 at time 2022-10-11 09:33:41.257
[Consumer 6]:  consumed  U  at index  0 at time 2022-10-11 09:33:41.258
[Consumer 6]:  consumed  B  at index  1 at time 2022-10-11 09:33:41.258
[Consumer 6]:  consumed  X  at index  2 at time 2022-10-11 09:33:41.258
[Consumer 6]:  consumed  A  at index  3 at time 2022-10-11 09:33:41.258
[Producer 5]: inserted  U  at index  4 at time 2022-10-11 09:33:41.259
[Producer 5]: inserted  B  at index  5 at time 2022-10-11 09:33:41.259
[Producer 5]: inserted  X  at index  6 at time 2022-10-11 09:33:41.260
[Producer 5]: inserted  A  at index  7 at time 2022-10-11 09:33:41.260
[Producer 5]: inserted  L  at index  8 at time 2022-10-11 09:33:41.260
[Producer 4]: inserted  U  at index  9 at time 2022-10-11 09:33:41.260
[Producer 4]: inserted  B  at index 10 at time 2022-10-11 09:33:41.261
[Producer 3]: inserted  U  at index 11 at time 2022-10-11 09:33:41.261
[Producer 3]: inserted  B  at index 12 at time 2022-10-11 09:33:41.261
[Producer 3]: inserted  X  at index  0 at time 2022-10-11 09:33:41.261
[Producer 3]: inserted  A  at index  1 at time 2022-10-11 09:33:41.261
[Producer 2]: inserted  U  at index  2 at time 2022-10-11 09:33:41.262
[Producer 4]: inserted  X  at index  3 at time 2022-10-11 09:33:41.262
[Consumer 1]:  consumed  U  at index  4 at time 2022-10-11 09:33:41.262
[Consumer 1]:  consumed  B  at index  5 at time 2022-10-11 09:33:41.263
[Consumer 1]:  consumed  X  at index  6 at time 2022-10-11 09:33:41.263
[Consumer 1]:  consumed  A  at index  7 at time 2022-10-11 09:33:41.263
[Consumer 2]:  consumed  L  at index  8 at time 2022-10-11 09:33:41.264
[Consumer 2]:  consumed  U  at index  9 at time 2022-10-11 09:33:41.264
[Consumer 2]:  consumed  B  at index 10 at time 2022-10-11 09:33:41.264
[Consumer 2]:  consumed  U  at index 11 at time 2022-10-11 09:33:41.264
[Consumer 3]:  consumed  B  at index 12 at time 2022-10-11 09:33:41.265
[Consumer 3]:  consumed  X  at index  0 at time 2022-10-11 09:33:41.265
[Consumer 3]:  consumed  A  at index  1 at time 2022-10-11 09:33:41.265
[Consumer 3]:  consumed  U  at index  2 at time 2022-10-11 09:33:41.265
[Consumer 4]:  consumed  X  at index  3 at time 2022-10-11 09:33:41.266
[Producer 5]: inserted  E  at index  4 at time 2022-10-11 09:33:41.266
[Producer 4]: inserted  A  at index  5 at time 2022-10-11 09:33:41.266
[Producer 2]: inserted  B  at index  6 at time 2022-10-11 09:33:41.266
[Producer 5]: inserted  C  at index  7 at time 2022-10-11 09:33:41.266
[Producer 5]: inserted  X  at index  8 at time 2022-10-11 09:33:41.266
[Consumer 5]:  consumed  E  at index  4 at time 2022-10-11 09:33:41.267
[Consumer 5]:  consumed  A  at index  5 at time 2022-10-11 09:33:41.267
[Consumer 5]:  consumed  B  at index  6 at time 2022-10-11 09:33:41.267
[Consumer 5]:  consumed  C  at index  7 at time 2022-10-11 09:33:41.267
[Consumer 4]:  consumed  X  at index  8 at time 2022-10-11 09:33:41.267
[Producer 2]: inserted  X  at index  9 at time 2022-10-11 09:33:41.267
[Consumer 4]:  consumed  X  at index  9 at time 2022-10-11 09:33:41.268
[Producer 2]: inserted  A  at index 10 at time 2022-10-11 09:33:41.268
[Consumer 4]:  consumed  A  at index 10 at time 2022-10-11 09:33:41.268
The generated & consumed numbers of consonants are the same as shown: 13
```

5. What to Submit

Use the CS370 *Canvas* to submit a single .zip or .tar file that contains:

- All .java files listed below and descriptive comments within,
  - `Coordinator.java`
  - `Producer.java`
  - `Consumer.java`
  - `Buffer.java`
- a Makefile that performs *make build*, *make clean,* and *make tar*

For this and all other assignments, ensure that you have submitted a valid .zip/.tar file. After submitting your file, you can download it and examine it to make sure it is indeed a valid zip/tar file, by trying to extract it.

Filename Convention: The archive file must be named as: <FirstName>-<LastName>-HW5.<tar/zip>. E.g. if you are John Doe and submitting for assignment 1, then the tar file should be named John-Doe-HW5.tar

6. Grading

The assignments must compile and function correctly on machines in the CSB-120 Lab. Assignments that work on your laptop on your particular flavor of Linux/Mac OS X/Windows, but not on the Lab machines, are considered unacceptable.

The grading will also be done on a 100-point scale. The points are broken up as follows:

Objective	**Points**
Implements a circular FIFO buffer correctly	20 points
Correctly suspends Producers when buffer is full and Consumers when buffer is empty; uses wait() and notifyAll() appropriately (no Thread.sleep() for sync)	25 points
All threads start, run, and finish without deadlock, join() used correctly in Coordinator, program terminates cleanly every time	15 points
Each item is produced and consumed exactly once; total consonants generated = total consonants consumed	15 points
Output clearly shows generator and consumer activity in readable form, Underlining Producer output is recommended, but not required. But timestamps must be included	15 points
Makefile is working correctly	10 points
Total	100 points

Restriction and Deductions:

[R1]. There is a 20-point deduction if you use an unbounded buffer for this assignment.

[R2]. There is a 25-point deduction if you use Thread.sleep() to synchronize access to the buffer. You can only use wait() and notify() as the primitives to synchronize access to the buffer. Thread.sleep() may be used for inserting random delays.

[R3]. Java has advanced classes for synchronization. These cannot be used for this assignment. Hence, there is a 80-point deduction for using any classes other than the following:

1. java.util.Random
2. java.lang. Arrays
3. java.time.Instant
4. java.time.Clock
5. java.time.Duration
6. java.util.Formatter

There is a 80-point deduction for using any external library.

[R4]. There is an 80-point deduction for using a Boolean flag or any variable that toggles in values so that your producer and consumer take turns adding to or consuming from the buffer. The solution must be based entirely on the use of wait() and notify().

You are required to **work alone** on this assignment.

Notes:

1. The output should have Producer statements underlined. You can use "`\033[0;4m`" to start underline formatting and "`\033[0;0m`" to stop the underline formatting in your print statement. This is recommended, not a requirement.
2. Look at the spacing for the word 'at' in Producer and the word 'from' in Consumer. Maintain the same spacing as only then the time stamps line up correctly and can be viewed easily. Again, there is no deduction for wrong formatting; it just makes the output look easier.
3. Use `%2C` for the formatter to display the character aligned next to the Consumer and next to the Producer. In other words, outputs from both Consumer and Producer should be aligned.
4. Use `%2d` for the same reason as #3.
5. The number of elements to be consumed might be a multiple of the number of consumers (when divided evenly), otherwise one of the consumers might have to take on a few more elements.
6. The number of elements to be produced might be a multiple of the number of producers (when divided evenly), otherwise one of the producers might have to take on a few more elements.
7. Do not define a package inside of your programs which includes all your programs, as this will raise an issue when the programs are run on terminals using command line.

7. Late Policy

Click here for the class policy on submitting late assignments.

8. Revisions

Any revisions in this assignment will be noted below.