

HW5: Programming Assignment v10.14.2021.5:00PM

SYNCHRONIZATION of PRODUCER and CONSUMER Threads

This assignment requires creation of multiple producer 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: Thursday, October 28, 2021, 11:00PM

Extended Due Date with 20% penalty: Friday, October 29, 2021, 11:00PM

Optional Extra Credit version: You can try it if you have time, and you would like to try it. Same Due Date. No late submissions permitted. See the addition at the bottom.

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 (`Bdbuffer.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 p producers, each producer is required to produce n/p items. The item that the producer adds to the buffer is a random character (with values between 'a' and 'z', both inclusive).

When a 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 2 inserted v at index 1 at time 2021-10-12 14:28:37.392914
```

When the random character is generated, you must also add it to a String so that it contains all the elements that the Producer generates. `getProducedStr()` method is used to return this generated string back to the `Invoker.java` file.

3. Consumers (`Consumer.java`): The consumers are responsible for consuming elements, generated 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 2 consumed q at index 6 at time 2021-10-12 14:28:37.394665
```

When a consumer successfully consumes an item from the buffer it should also add it to the string. This class should have a function named `getConsumedStr()`, which returns the string consumed 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 Producers and other Consumers,
- ii. Number of elements that producer should generate,
- 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 seed which is used by the random number generator to generate the random numbers to be inserted.

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 of 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 (`Invoker.java`): Your main program should accept the seed command line argument. 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 5 and 10) | i.e. [5,10] |
| ii. | Number of items to be produced and consumed | (between 10 and 20) | i.e. [10,20] |
| iii. | Number of consumers | (between 2 and 5) | i.e. [2,5] |
| iv. | Number of producers | (between 2 and 5) | i.e. [2,5] |

Each producer thread terminates when the specified number of items has been produced. After this, you need to use the methods, `getProducedStr()` and `getConsumedStr()`, for each of the Producers and Consumers to get the full Produced/Consumed string.

Since the strings at the consumer end are not always in order, the main/calling function must sort both the strings obtained. Finally, if the sorted strings are the same, the `Invoker.java` program prints the following:

The sorted Produced and Consumed strings are the same as shown: `hhmmmmmqgssuuvv`

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 produced. However, if the consumption is very quick, within the smallest time resolution, production/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 the FIFO Circular Buffer and ensure that the buffer can hold the right number items at a time, and the access to it is synchronized.
2. The number of items to be consumed should be equally distributed among the consumer threads (whenever possible). Verify that the number of elements can be perfectly divided among the consumers with no fractions involved, if not arrange for one of the consumer threads to handle the difference. Also, a seed is to be passed to the producer.
3. A producer should wait if the buffer is full.
4. A consumer should wait if the buffer is empty.
5. Make sure that the printing requirements are met, specifically the underlined statements.
6. Your solution must satisfy the correctness constraint (i.e.: you consume each item exactly once, in the order that it was produced, and demonstrate this by printing out the items produced and consumed, along with the location and the timestamp with microsecond resolution). The code to get the timestamp with microsecond resolution is provided to you, in `Invoker.java`. The produced string of all Producer threads should be obtained after the join of each Producer object, by calling `getProducedStr()` in the Producer class. The consumed string of each Consumer thread is obtained after each Consumer thread join, by calling `getConsumedStr()` in the Consumer class. The sorted string of all the Consumers should be the same as the sorted string of all the Producers.
7. There should be no deadlock. Your program will be executed multiple times, and it should run to completion every time without a deadlock.
8. Your program should work for any combination of the number of consumers, producers, number of elements, and buffer size.

3. Files Provided

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

Please refer to the `README.txt` file inside the package for the questions. You need to answer the questions in the README file.

4. Example Outputs:

1. Example 1: Set Seed as 13

```

~$ java Invoker 13
[Invoker] Buffer Size: 9
[Invoker] Total Items: 12
[Invoker] No. of Producers: 2
[Invoker] No. of Consumers: 3
Producer 1 inserted m at index 0 at time 2021-10-13 19:54:15.605070
Consumer 3 consumed m at index 0 at time 2021-10-13 19:54:15.612287
Producer 2 inserted m at index 1 at time 2021-10-13 19:54:15.612916
Consumer 2 consumed m at index 1 at time 2021-10-13 19:54:15.613265
Producer 1 inserted a at index 2 at time 2021-10-13 19:54:15.627417
Consumer 1 consumed a at index 2 at time 2021-10-13 19:54:15.627819
Producer 1 inserted j at index 3 at time 2021-10-13 19:54:15.628321
Consumer 3 consumed j at index 3 at time 2021-10-13 19:54:15.628638
Producer 2 inserted a at index 4 at time 2021-10-13 19:54:15.628959
Consumer 2 consumed a at index 4 at time 2021-10-13 19:54:15.629272
Producer 2 inserted j at index 5 at time 2021-10-13 19:54:15.629606
Consumer 3 consumed j at index 5 at time 2021-10-13 19:54:15.629908
Producer 1 inserted o at index 6 at time 2021-10-13 19:54:15.630230
Consumer 1 consumed o at index 6 at time 2021-10-13 19:54:15.630563
Producer 1 inserted r at index 7 at time 2021-10-13 19:54:15.631084
Consumer 3 consumed r at index 7 at time 2021-10-13 19:54:15.631408
Producer 2 inserted o at index 8 at time 2021-10-13 19:54:15.631711
Consumer 2 consumed o at index 8 at time 2021-10-13 19:54:15.632059
Producer 2 inserted r at index 0 at time 2021-10-13 19:54:15.632335
Producer 1 inserted q at index 1 at time 2021-10-13 19:54:15.632617
Consumer 1 consumed r at index 0 at time 2021-10-13 19:54:15.632919
Producer 2 inserted q at index 2 at time 2021-10-13 19:54:15.633344
Consumer 2 consumed q at index 1 at time 2021-10-13 19:54:15.633653
Consumer 1 consumed q at index 2 at time 2021-10-13 19:54:15.633935
The sorted Produced and Consumed strings are the same as shown: aaajmmooqrrr

```

2. **Example 2:** Set Seed as 6

```

~$ java Invoker 6
[Invoker] Buffer Size: 6
[Invoker] Total Items: 18
[Invoker] No. of Producers: 2
[Invoker] No. of Consumers: 4
Producer 1 inserted r at index 0 at time 2021-10-13 19:55:55.096322
Consumer 4 consumed r at index 0 at time 2021-10-13 19:55:55.103695
Producer 2 inserted r at index 1 at time 2021-10-13 19:55:55.104359
Consumer 3 consumed r at index 1 at time 2021-10-13 19:55:55.104701
Producer 2 inserted g at index 2 at time 2021-10-13 19:55:55.118666
Consumer 1 consumed g at index 2 at time 2021-10-13 19:55:55.119071
Producer 2 inserted s at index 3 at time 2021-10-13 19:55:55.119400
Consumer 2 consumed s at index 3 at time 2021-10-13 19:55:55.119904
Producer 1 inserted g at index 4 at time 2021-10-13 19:55:55.120379
Consumer 3 consumed g at index 4 at time 2021-10-13 19:55:55.120696
Producer 1 inserted s at index 5 at time 2021-10-13 19:55:55.121028
Consumer 2 consumed s at index 5 at time 2021-10-13 19:55:55.121325
Producer 2 inserted w at index 0 at time 2021-10-13 19:55:55.121846
Consumer 1 consumed w at index 0 at time 2021-10-13 19:55:55.122183
Producer 2 inserted j at index 1 at time 2021-10-13 19:55:55.122520
Producer 2 inserted l at index 2 at time 2021-10-13 19:55:55.122819
Producer 2 inserted h at index 3 at time 2021-10-13 19:55:55.123121
Consumer 4 consumed j at index 1 at time 2021-10-13 19:55:55.123408
Consumer 2 consumed l at index 2 at time 2021-10-13 19:55:55.123724
Consumer 2 consumed h at index 3 at time 2021-10-13 19:55:55.124163
Producer 1 inserted w at index 4 at time 2021-10-13 19:55:55.124674
Consumer 3 consumed w at index 4 at time 2021-10-13 19:55:55.125012
Producer 1 inserted j at index 5 at time 2021-10-13 19:55:55.125315
Consumer 4 consumed j at index 5 at time 2021-10-13 19:55:55.125593
Producer 2 inserted g at index 0 at time 2021-10-13 19:55:55.125852
Consumer 1 consumed g at index 0 at time 2021-10-13 19:55:55.126110
Producer 2 inserted s at index 1 at time 2021-10-13 19:55:55.126374
Consumer 4 consumed s at index 1 at time 2021-10-13 19:55:55.126612
Producer 1 inserted l at index 2 at time 2021-10-13 19:55:55.126908
Consumer 3 consumed l at index 2 at time 2021-10-13 19:55:55.127186
Producer 1 inserted h at index 3 at time 2021-10-13 19:55:55.127470
Consumer 4 consumed h at index 3 at time 2021-10-13 19:55:55.127757
Producer 1 inserted g at index 4 at time 2021-10-13 19:55:55.128007
Consumer 1 consumed g at index 4 at time 2021-10-13 19:55:55.128223
Producer 1 inserted s at index 5 at time 2021-10-13 19:55:55.128546
Consumer 4 consumed s at index 5 at time 2021-10-13 19:55:55.128789
The sorted Produced and Consumed strings are the same as shown: gggghhjllrrssssww

```

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,
 - Invoker.java
 - Producer.java
 - Consumer.java
 - Bdbuffer.java
- a Makefile that performs both a *make build* as well as a *make clean*,
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program, and answers for the 4 questions

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

Optional Extra Credit version: The archive file must be named as: <FirstName>-<LastName>-HW5_EC.<tar/zip>. E.g. if you are John Doe and submitting for assignment 1, then the tar file should be named John-Doe-HW5_EC.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, 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
Correctly performing Tasks 1-8 (10 points each)	80 points
Providing a working Makefile.	5 points
Questions in the README file	5 points
Text alignment and underlining	5 points
Descriptive comments	5 points

Questions: (see README.txt file in the skeleton provided)

Restriction and Deductions:

[R1]. There is a 100-point deduction if you use an unbounded buffer for this assignment.

[R2]. There is a 100-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 100-point deduction for using any classes other than the following:

- | | |
|-----------------------|------------------------|
| 1. java.util.Random | 2. java.lang.Exception |
| 3. java.time.Instant | 4. java.time.Clock |
| 5. java.time.Duration | 6. java.util.Formatter |
| 7. java.util.Arrays | |

There is a 100-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.
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.
3. Use %3d for the formatter to display the ID aligned next to the Consumer and next to the Producer. In other words, outputs from both, Consumer and Producer, should be aligned.
4. 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.
5. 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.
6. 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](#).

Revisions: Any revisions in the assignment will be noted below.

- Oct 21, 2021: Optional Extra Credit version added below.
- Oct 25, 2021: Note that ***All intervals are inclusive***
 - Number of elements in buffer/buffer size (between 5 and 10) i.e. [5,10]
 - Number of items to be produced and consumed (between 10 and 20) i.e. [10,20]
 - Number of consumers (between 2 and 5) i.e. [2,5]
 - Number of producers (between 2 and 5) i.e. [2,5]

Optional Extra Credit version

IMPORTANT: Please note that you should implement the regular assignment and turn it in before attempting the extra credit. This should be a modification of your original submission.

For extra credit, you may choose to submit a second version of the program in addition to your original submission. The new program will re-arrange the string to its original order as opposed to sorting it. The input is no longer randomly generated, but instead, it's in a file, which contains any number of characters, new lines, and spaces.

Requirements

All other requirements from the main assignment which are not explicitly over-written stay in place.

1. The Invoker takes a seed and an input file containing a string that may have any combination of characters, including spaces and newlines.
2. The Invoker splits the file into chunks to feed to each producer.
 - a. At this point generate for each character in the input file a <index, char>
 - b. "Hello" - > (0, H) (1, e) (2, l) ... (4, o)
3. The Bdbuffer no longer holds just characters. It must hold tuples.
 - a. Each tuple is (index, character). Note that these do not have to be actual tuples. This just means that each element of the Bdbuffer is a key-value pair. It can be in any data structure you design or use from the JAVA collections (list, array, tuple, map, etc..)
4. The producers are responsible for **inserting** tuples items in the Bdbuffer.
 - a. **Not producing them!**
 - b. Each producer gets an equal number of tuples. Any extras go to the last producer.
5. When a tuple is inserted in the Bdbuffer, you must add it to a data structure internal to each producer.
 - a. You will use this data structure to reconstruct the string from all of the producers.
 - b. The producer class should have a function named `getProducedTuples()`, which returns the tuples inserted by the Producer.
6. The consumers are responsible for consuming elements, inserted by the producer, from the Bdbuffer.
7. When a consumer successfully consumes an item from the Bdbuffer it should also add it to a data structure in the same way each producer does when inserting. This class should have a function named `getConsumedTuples()`, which returns the tuples consumed by the Consumer.
8. The Invoker reconstructs the string twice. Once from all the characters the producers inserted. And a second time from the characters that the consumers consumed.
 - a. Use the methods, `getProducedTuples()` and `getConsumedTuples()`, for each of the Producers and Consumers to get the full Produced/Consumed data structures.
9. The main/calling function must sort both the structures based on the key(index).
 - a. So that the string is back to its original order.
10. Finally, if the sorted strings are the same, the Invoker.java program prints the following:

Producer's output:


```
HelloWorld HereWe Are Again !@
Consumer's output:
HelloWorld HereWe Are Again !@
```

The producer class object should take these arguments:

- i. Copy of the instance of the Bdbuffer it will access in common with other Producers and other Consumers,
- ii. A data structure containing each tuple to be inserted.
- 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.)

The consumer class objects should take these arguments:

- i. Copy of the instance of the Bdbuffer it will access in common with the Producers and other Consumers,
- ii. The number of elements that this thread of consumers 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 (Invoker.java): Your main program should accept:

- i. Seed
- ii. Filename

Using this seed, the following elements must be randomly initialized.

- | | | |
|---|--------------------|-------------|
| i. Number of elements in Bdbuffer/Bdbuffer size | (between 5 and 10) | i.e. [5,10] |
| ii. Number of consumers | (between 2 and 5) | i.e. [2,5] |
| iii. Number of producers | (between 2 and 5) | i.e. [2,5] |

Using the file name:

- i. Open the file and read all the contents

Example Outputs:

1. **Example 1:** input.txt contains "HelloWorld HereWe Are Again !@"

```
java Invoker 10 input.txt
[Invoker] Bdbuffer Size: 8
[Invoker] Total Items: 30
[Invoker] No. of Producers 3 No. of Consumers: 3
Producer 1 inserted H at index 0 at time 2021-10-14 12:47:27.084819
Consumer 3 consumed H at index 0 at time 2021-10-14 12:47:27.088406
Producer 3 inserted e at index 1 at time 2021-10-14 12:47:27.088822
Producer 3 inserted at index 2 at time 2021-10-14 12:47:27.089253
Producer 2 inserted at index 3 at time 2021-10-14 12:47:27.089591
Producer 3 inserted A at index 4 at time 2021-10-14 12:47:27.089953
Consumer 1 consumed e at index 1 at time 2021-10-14 12:47:27.090394
Consumer 2 consumed at index 2 at time 2021-10-14 12:47:27.090754
Consumer 2 consumed at index 3 at time 2021-10-14 12:47:27.091220
Consumer 3 consumed A at index 4 at time 2021-10-14 12:47:27.091552
Producer 1 inserted e at index 5 at time 2021-10-14 12:47:27.091904
Consumer 3 consumed e at index 5 at time 2021-10-14 12:47:27.092199
Producer 1 inserted l at index 6 at time 2021-10-14 12:47:27.092535
Consumer 2 consumed l at index 6 at time 2021-10-14 12:47:27.092836
Producer 3 inserted g at index 7 at time 2021-10-14 12:47:27.093177
Producer 2 inserted H at index 0 at time 2021-10-14 12:47:27.093500
Producer 3 inserted a at index 1 at time 2021-10-14 12:47:27.093813
Consumer 1 consumed g at index 7 at time 2021-10-14 12:47:27.094143
Consumer 2 consumed H at index 0 at time 2021-10-14 12:47:27.094471
Producer 1 inserted l at index 2 at time 2021-10-14 12:47:27.094785
```

```
Consumer 3 consumed a at index 1 at time 2021-10-14 12:47:27.095083
Producer 1 inserted o at index 3 at time 2021-10-14 12:47:27.095368
Consumer 2 consumed l at index 2 at time 2021-10-14 12:47:27.095648
Consumer 1 consumed o at index 3 at time 2021-10-14 12:47:27.095927
Producer 3 inserted i at index 4 at time 2021-10-14 12:47:27.096207
Producer 2 inserted e at index 5 at time 2021-10-14 12:47:27.096495
Producer 3 inserted n at index 6 at time 2021-10-14 12:47:27.096827
Producer 3 inserted at index 7 at time 2021-10-14 12:47:27.097085
Consumer 1 consumed i at index 4 at time 2021-10-14 12:47:27.097348
Consumer 2 consumed e at index 5 at time 2021-10-14 12:47:27.097624
Producer 1 inserted W at index 0 at time 2021-10-14 12:47:27.097894
Consumer 3 consumed n at index 6 at time 2021-10-14 12:47:27.098168
Producer 1 inserted o at index 1 at time 2021-10-14 12:47:27.098481
Consumer 2 consumed at index 7 at time 2021-10-14 12:47:27.098734
Consumer 1 consumed W at index 0 at time 2021-10-14 12:47:27.098959
Producer 3 inserted ! at index 2 at time 2021-10-14 12:47:27.099189
Producer 2 inserted r at index 3 at time 2021-10-14 12:47:27.099453
Producer 3 inserted @ at index 4 at time 2021-10-14 12:47:27.099668
Consumer 1 consumed o at index 1 at time 2021-10-14 12:47:27.099921
Consumer 1 consumed ! at index 2 at time 2021-10-14 12:47:27.100330
Consumer 1 consumed r at index 3 at time 2021-10-14 12:47:27.100543
Consumer 1 consumed @ at index 4 at time 2021-10-14 12:47:27.100735
Producer 1 inserted r at index 5 at time 2021-10-14 12:47:27.100926
Consumer 3 consumed r at index 5 at time 2021-10-14 12:47:27.101164
Producer 1 inserted l at index 6 at time 2021-10-14 12:47:27.101369
Consumer 1 consumed l at index 6 at time 2021-10-14 12:47:27.101576
Producer 2 inserted e at index 7 at time 2021-10-14 12:47:27.101809
Consumer 2 consumed e at index 7 at time 2021-10-14 12:47:27.102045
Producer 2 inserted W at index 0 at time 2021-10-14 12:47:27.102256
Producer 1 inserted d at index 1 at time 2021-10-14 12:47:27.102461
Consumer 3 consumed W at index 0 at time 2021-10-14 12:47:27.102661
Producer 2 inserted e at index 2 at time 2021-10-14 12:47:27.102879
Producer 2 inserted at index 3 at time 2021-10-14 12:47:27.103073
Consumer 2 consumed d at index 1 at time 2021-10-14 12:47:27.103258
Producer 2 inserted A at index 4 at time 2021-10-14 12:47:27.103442
Consumer 3 consumed e at index 2 at time 2021-10-14 12:47:27.103620
Producer 2 inserted r at index 5 at time 2021-10-14 12:47:27.103858
Consumer 2 consumed at index 3 at time 2021-10-14 12:47:27.104087
Consumer 3 consumed A at index 4 at time 2021-10-14 12:47:27.104495
Consumer 3 consumed r at index 5 at time 2021-10-14 12:47:27.104805
Producer's output:
HelloWorld HereWe Are Again !@
Consumer's output:
HelloWorld HereWe Are Again !@
```