HW5: Programming Assignment v03.23.2022.03:00PM
Synchronization of Generator and Consumer Threads

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

Due Date: Thursday, March 31, 2022, 11:00PM

Extended Due Date with 20% penalty: Friday, April 1, 2022, 11:00PM

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 generator and the consumers must reference the same buffer.

2. Generators (`Generator.java`): The generators are responsible for producing data items to be added to the buffer. If the buffer is full, a generator 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 generators and the total number of items n, each generator is required to produce n/g items. If the n isn't perfectly dividable by the g, then the last generator will take care about the remains. For example, if we have 10 items to produce and 3 generators, then the first 2 generators will produce 3 items and the last generator 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 generator adds to the buffer is a random prime number (The details of the prime number is described below).

When a generator adds a random prime number to the buffer, the prime number is chosen by these rules:

1. Get a random number *N* from a range [3, 31] every time.
2. Calculate the *N*-th prime number
3. Use the *N*-th number of primes to add into the buffer.

- E.g.) If *N* is 1, then the *N*-th prime number is 2. If *N* is 3, then the *N*-th number of primes is 5.

When the generator 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:

```
[Generator 1]: inserted  47 at index 0 at time 2022-03-09 23:48:09.093064
```

After the generator inserted the prime number, the prime number should be added to *sumOfPrimes* which is a member variable of Generator. The *sumOfPrimes* is for checking the sum of prime numbers that are generated from the instance, and *getSumOfGeneratedPrimes()* needs to return the *sumOfPrimes* as a member method.

3. Consumers (`Consumer.java`): The consumers are responsible for consuming elements, produced by the generator, from the buffer. If the buffer is empty, the consumers must wait for the generator 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  47 at index 0 at time 2022-03-09 23:48:09.101682
```

When a consumer successfully consumes an item from the buffer it should also add it to its member variable *sumOfPrimes*. This class should have a function named *`getSumOfConsumedPrimes()`*, which returns an integer which is the sum of consumed primes by the Consumer.

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

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

The generator class object should take these arguments:

i. Copy of the instance of the buffer it will access in common with other Generator and other Consumers,
ii. Number of elements that generator should produce,
iii. The ID, which is the number of the generator thread (i.e.: the first generator thread should be 1, the second should be 2, etc.)
iv. The *PrimeSeed* which is used by the random number generator to generate the random numbers in the range [3, 31] to calculate a prime number.

The consumer class objects should take these arguments:

i. Copy of the instance of the buffer it will access in common with the Generators 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 (`Coordinator.java`): Your main program should accept the *Seed* and *PrimeSeed* 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 5 and 10)	i.e. [5,10]
ii.	Number of items to be produced and consumed	(between 10 and 30)	i.e. [10,30]
iii.	Number of consumers	(between 2 and 5)	i.e. [2,5]
iv.	Number of generators	(between 2 and 5)	i.e. [2,5]

Each generator thread terminates when the specified number of items has been produced. After this, you need to use the methods, getSumOfGeneratedPrimes() and getSumOfConsumedPrimes(), for each of the Generators and Consumers to get the **total** sum of Generated/Consumed primes.

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

```
The generated & consumed sums of primes are the same as shown: 1225
```

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 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 *PrimeSeed* is to be passed to the generator.
3. A generator 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 generated 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 Coordinator.java. The sum of primes of all Generator threads should be obtained after the join of each Generator object, by calling `getSumOfGeneratedPrimes()` in the Generator class. The sum of consumed primes of each Consumer thread is obtained after each Consumer thread join, by calling `getSumOfConsumedPrimes()` in the Consumer class. The sum of generated primes should be the same as the sum of consumed primes.
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, generators, number of elements, and buffer size.

3. Files Provided

Files provided for this assignment include: the description file (this file), a README, and skeleton Coordinator.java and Generator.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 3 and *PrimeSeed* as 31

```
$java Coordinator 3 31
[Coordinator] Buffer Size: 7
[Coordinator] Total Items: 15
[Coordinator] No. of Generator: 5
[Coordinator] No. of Consumers: 2
[Generator 1]: inserted  47 at index 0 at time 2022-03-09 23:48:09.093064
[Generator 1]: inserted  71 at index 1 at time 2022-03-09 23:48:09.100991
[Consumer 2]:  consumed  47 at index 0 at time 2022-03-09 23:48:09.101682
[Consumer 2]:  consumed  71 at index 1 at time 2022-03-09 23:48:09.102048
[Generator 5]: inserted  47 at index 2 at time 2022-03-09 23:48:09.102790
[Generator 5]: inserted  71 at index 3 at time 2022-03-09 23:48:09.103193
[Generator 4]: inserted  47 at index 4 at time 2022-03-09 23:48:09.103845
[Generator 3]: inserted  47 at index 5 at time 2022-03-09 23:48:09.104547
[Generator 2]: inserted  47 at index 6 at time 2022-03-09 23:48:09.105268
[Generator 3]: inserted  71 at index 0 at time 2022-03-09 23:48:09.105583
[Generator 3]: inserted 127 at index 1 at time 2022-03-09 23:48:09.105879
[Consumer 1]:  consumed  47 at index 2 at time 2022-03-09 23:48:09.106720
[Consumer 1]:  consumed  71 at index 3 at time 2022-03-09 23:48:09.107128
[Consumer 1]:  consumed  47 at index 4 at time 2022-03-09 23:48:09.107510
[Consumer 1]:  consumed  47 at index 5 at time 2022-03-09 23:48:09.107764
[Consumer 1]:  consumed  47 at index 6 at time 2022-03-09 23:48:09.108003
[Consumer 1]:  consumed  71 at index 0 at time 2022-03-09 23:48:09.108246
[Consumer 1]:  consumed 127 at index 1 at time 2022-03-09 23:48:09.108471
[Generator 1]: inserted 127 at index 2 at time 2022-03-09 23:48:09.108867
[Consumer 2]:  consumed 127 at index 2 at time 2022-03-09 23:48:09.109176
[Generator 5]: inserted 127 at index 3 at time 2022-03-09 23:48:09.109540
[Generator 4]: inserted  71 at index 4 at time 2022-03-09 23:48:09.109858
[Generator 2]: inserted  71 at index 5 at time 2022-03-09 23:48:09.110226
[Generator 4]: inserted 127 at index 6 at time 2022-03-09 23:48:09.110502
[Consumer 2]:  consumed 127 at index 3 at time 2022-03-09 23:48:09.110819
[Consumer 2]:  consumed  71 at index 4 at time 2022-03-09 23:48:09.111127
[Consumer 2]:  consumed  71 at index 5 at time 2022-03-09 23:48:09.111398
[Consumer 2]:  consumed 127 at index 6 at time 2022-03-09 23:48:09.111624
[Generator 2]: inserted 127 at index 0 at time 2022-03-09 23:48:09.111849
[Consumer 2]:  consumed 127 at index 0 at time 2022-03-09 23:48:09.112149
The generated & consumed sums of primes are the same as shown: 1225
```

2. **Example 2**: Set *Seed* as 4 and *PrimeSeed* 1

```
$ java Coordinator 4 1
[Coordinator] Buffer Size: 7
[Coordinator] Total Items: 26
[Coordinator] No. of Generator: 5
[Coordinator] No. of Consumers: 5
[Generator 1]: inserted 101 at index 0 at time 2022-03-09 23:48:33.508129
[Consumer 5]:  consumed 101 at index 0 at time 2022-03-09 23:48:33.515885
[Generator 2]: inserted 101 at index 1 at time 2022-03-09 23:48:33.516872
[Consumer 1]:  consumed 101 at index 1 at time 2022-03-09 23:48:33.517730
[Generator 5]: inserted 101 at index 2 at time 2022-03-09 23:48:33.518542
[Generator 5]: inserted  23 at index 3 at time 2022-03-09 23:48:33.518919
[Generator 4]: inserted 101 at index 4 at time 2022-03-09 23:48:33.519620
[Generator 4]: inserted  23 at index 5 at time 2022-03-09 23:48:33.519971
[Generator 3]: inserted 101 at index 6 at time 2022-03-09 23:48:33.520702
[Generator 4]: inserted  29 at index 0 at time 2022-03-09 23:48:33.521031
[Generator 4]: inserted  13 at index 1 at time 2022-03-09 23:48:33.521289
[Consumer 1]:  consumed 101 at index 2 at time 2022-03-09 23:48:33.521617
[Consumer 1]:  consumed  23 at index 3 at time 2022-03-09 23:48:33.521930
[Consumer 1]:  consumed 101 at index 4 at time 2022-03-09 23:48:33.522274
[Consumer 1]:  consumed  23 at index 5 at time 2022-03-09 23:48:33.522558
[Generator 2]: inserted  23 at index 2 at time 2022-03-09 23:48:33.522832
[Generator 2]: inserted  29 at index 3 at time 2022-03-09 23:48:33.523222
[Consumer 2]:  consumed 101 at index 6 at time 2022-03-09 23:48:33.523907
[Consumer 2]:  consumed  29 at index 0 at time 2022-03-09 23:48:33.524218
[Consumer 2]:  consumed  13 at index 1 at time 2022-03-09 23:48:33.524488
[Consumer 2]:  consumed  23 at index 2 at time 2022-03-09 23:48:33.524764
[Consumer 2]:  consumed  29 at index 3 at time 2022-03-09 23:48:33.524986
[Generator 1]: inserted  23 at index 4 at time 2022-03-09 23:48:33.525376
[Consumer 3]:  consumed  23 at index 4 at time 2022-03-09 23:48:33.526117
[Generator 1]: inserted  29 at index 5 at time 2022-03-09 23:48:33.526429
[Consumer 5]:  consumed  29 at index 5 at time 2022-03-09 23:48:33.526686
[Generator 2]: inserted  13 at index 6 at time 2022-03-09 23:48:33.527084
[Generator 2]: inserted   5 at index 0 at time 2022-03-09 23:48:33.527381
[Generator 4]: inserted   5 at index 1 at time 2022-03-09 23:48:33.527646
[Generator 5]: inserted  29 at index 2 at time 2022-03-09 23:48:33.528030
[Generator 5]: inserted  13 at index 3 at time 2022-03-09 23:48:33.528380
[Generator 5]: inserted   5 at index 4 at time 2022-03-09 23:48:33.528721
[Generator 3]: inserted  23 at index 5 at time 2022-03-09 23:48:33.529044
[Consumer 4]:  consumed  13 at index 6 at time 2022-03-09 23:48:33.529711
[Consumer 4]:  consumed   5 at index 0 at time 2022-03-09 23:48:33.529949
[Consumer 4]:  consumed   5 at index 1 at time 2022-03-09 23:48:33.530132
[Consumer 4]:  consumed  29 at index 2 at time 2022-03-09 23:48:33.530320
[Consumer 4]:  consumed  13 at index 3 at time 2022-03-09 23:48:33.530482
[Consumer 5]:  consumed   5 at index 4 at time 2022-03-09 23:48:33.530706
[Consumer 5]:  consumed  23 at index 5 at time 2022-03-09 23:48:33.531020
[Generator 1]: inserted  13 at index 6 at time 2022-03-09 23:48:33.531208
[Generator 1]: inserted   5 at index 0 at time 2022-03-09 23:48:33.531449
[Consumer 3]:  consumed  13 at index 6 at time 2022-03-09 23:48:33.531655
```

```
[Consumer 3]:  consumed   5 at index 0 at time 2022-03-09 23:48:33.531886
[Generator 3]: inserted  29 at index 1 at time 2022-03-09 23:48:33.532142
[Generator 5]: inserted  67 at index 2 at time 2022-03-09 23:48:33.532366
[Generator 3]: inserted  13 at index 3 at time 2022-03-09 23:48:33.532514
[Generator 3]: inserted   5 at index 4 at time 2022-03-09 23:48:33.532751
[Consumer 5]:  consumed  29 at index 1 at time 2022-03-09 23:48:33.532882
[Consumer 5]:  consumed  67 at index 2 at time 2022-03-09 23:48:33.533045
[Consumer 3]:  consumed  13 at index 3 at time 2022-03-09 23:48:33.533184
[Consumer 3]:  consumed   5 at index 4 at time 2022-03-09 23:48:33.533355
The generated & consumed sums of primes are the same as shown: 922
```

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`
  - `Generator.java`
  - `Consumer.java`
  - `Buffer.java`
- a Makefile that performs *make build*, *make clean,* and *make tar*
- 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

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. Arrays
3. java.time.Instant
4. java.time.Clock
5. java.time.Duration
6. java.util.Formatter

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 generator 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 Generator 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 Generator 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 `%1d` for the formatter to display the ID aligned next to the Consumer and next to the Generator. In other words, outputs from both, Consumer and Generator, 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 generators (when divided evenly), otherwise one of the generators 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.

Several parts of the description are updated (3/22/2022 01:00 AM) and the details are below.

1. 'Producer' is changed to 'Generator' at the first paragraph of general description, Description of Task, and Notes.
2. 'Bdbuffer.java' is changed to 'Buffer.java' at the Description of Task.
3. More descriptions are added in the Generators part of the Description of Task.

'java.util.Exception' is allowed for synchronization.