Parallel and MultiThreaded Programming

CSYE 7215

Homework 11

Due: December 14, 2020

Put all your Java, compiled class files and documentation into zip file named Homework11.zip and submit it via the dropbox on canvas before the END of due date. Put your name on all files. There will be a short quiz on this assignment.

1. **In article “Concurrency and Parallelism in Real World”:**
2. **What are three examples of real world, Explain each**

🡪 In given article author gave three real-world example to explain parallelism and concurrency.

1. **Olympic games**

In the Olympic games example we know that there are many people participate in each games but How many people can run at the same time we can consider that as Parallelism on the other end How many Runners are competing in that event is considered as concurrency.

**Parallelism** 🡪 How many people can run at the same time

**Concurrency** 🡪 How many Runners are competing in that event

1. **Bankers**

In Bankers example Number of bankers working at bank is considered as parallelism while How many task each banker perform at given time is considered as concurrency

**Parallelism** 🡪 Number of bankers working at bank

**Concurrency** 🡪 How many task each banker perform at given time.

1. **Food booth**

In the food booth example we know that in food festival there are many jobs to be done Usually one person handles one job. They communicate and pass work around. People wait in queues to order, then they wait in another queue to get their food. Sometimes, all of the cooks do all jobs.

**Parallelism** 🡪 Number of cooks working at food festival

**Concurrency** 🡪 How many task each cook perform at given time.

**B) In Banking example how do you make customers to get efficient service and to make all tellers busy?**

🡪 To get the efficient banking service in banking example we can use the Queues .

🡪 Queues are a concurrency mechanism that “store” work to be done later.

🡪 Work is **pulled off of the queue by the banker, processed, and usually** put onto another queue. At any given time, most of the work is waiting.

**2. The following code segments are test strategies that test the internal behavior of threads. These code segments are described in chapter12 of the book “Java Concurrency in Practice”. The code segments as it is may not work, you are to correct these code segments to the intended goals with new Java libraries if it doesn’t work. Some solutions may require research.**

**A) Describe each code segment and the intended test strategy**

**B) Compile the code and run. Correct the code if it doesn’t work**

**C) Explain the intended results outputs**

Text

Description automatically generated

🡪 In 12.1 we are creating the BoundedBuffer using semaphore and implements required bounding and blocking

🡪 BoundedBuffer implements a fixedlength arraybased queue with blocking **put** and **take** **methods controlled by a pair of counting semaphores.**

🡪 The **availableItems** semaphore represents the number of elements that can be removed from the buffer, and is initially zero (since the buffer is initially empty).

🡪while **availableSpaces** represents how many items can be inserted into the buffer, and is initialized to the size of the buffer.

🡪 A **take** operation **first requires that a permit be obtained from availableItems**.

🡪 conversely, so that on exit from either the put or take methods, the sum of the counts of both semaphores always equals the bound.

Text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

🡪 In 12.2 we have wrote most basic unit tests for BoundedBuffer are similar to what we'd use in a sequential context create a bounded buffer, call its methods, and assert post conditions and invariants.

🡪 In **testEmptyWhenConstructed** method we quickly checking that a freshly created buffer should identify itself as empty, and also as not full.

🡪 In **testFullAfterPut** method we are checking safety . we are inserting N elements into a buffer with capacity N (which should succeed without blocking), and test that the buffer recognizes that it is full (and not empty).

Text

Description automatically generated

Text

Description automatically generated

🡪 In 12.3 we are **testing blocking operations**. It creates **a "taker" thread** that attempts to take an **element from an empty buffer**.

🡪 **If take succeeds, it registers failure.** The test runner thread **starts the taker thread, waits a long time, and then interrupts it. If the taker thread has correctly blocked in the take operation, it will throw InterruptedException,** and **the catch block for this exception treats this as success and lets the thread exit**.

🡪 The main test runner thread then attempts **to join with the taker thread and verifies that the join returned successfully by calling Thread.isAlive**.

**Output:**

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

🡪 In 12.4 we have create a function to generate random numbers using xor functionality and make sure that we are generating enough randomness to ensure the numbers change from run to run.

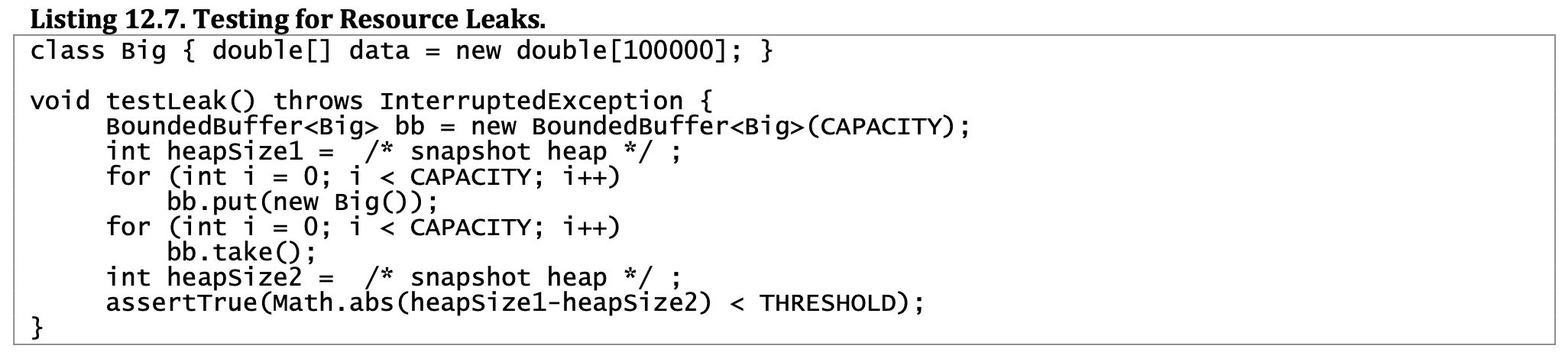
🡪 In 12.5 and 12.6 we starts with **N producer threads** that **generate elements and enqueue them**, and **N consumer threads** that **dequeue them**.

🡪 Each thread **updates the checksum of the elements as they go in or out**, using a per thread checksum that **is combined at the end of the test run so as to add no more synchronization or contention than required to test the buffer**.

**Output:**

Text

Description automatically generated



🡪 In 12.7 we are testing that resource is leaking or not in our test cases first we are checking the initial size of the heap then putting and taking N numbers of object and then verifying the memory leakage.

**Output**

Text

Description automatically generated