W4: Testing of Multi-threaded applications

.NET / Java

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Dep. PXL-IT – Elfde-Liniestraat 26 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

Java concurrency

Topics of today:

- Blocking queue
- Test its blocking behavior
- Test its performance under stress test conditions
- Available frameworks for unit testing of multi-threaded classes.

PXL IT

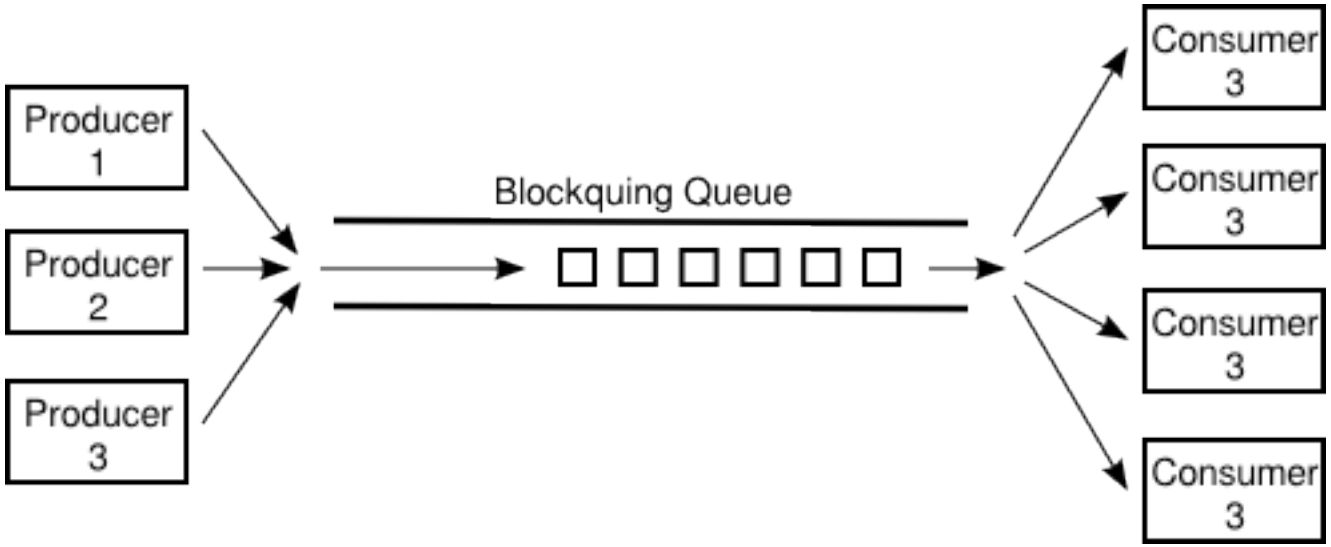
A SimpleBlockingQueue

- Use package *java.util.concurrent*
- As data structure use *java.util.LinkedList* (not synchronized)
- Add synchronization and blocking behavior (wait/notify)
- Requirement: make the *queue generic!*

PXL IT

Testing blocking operations

- wasInterrupted
- reachedAfterGet
- throwableThrown



PXL IT

Testing blocking operations

```
@Test
public void testPutOnEmptyQueueBlocks() throws InterruptedException {
        final SimpleBlockingQueue queue = new SimpleBlockingQueue();
        BlockingThread blockingThread = new BlockingThread(queue);
        blockingThread.start();
        Thread.sleep(5000);
        assertThat(blockingThread.isReachedAfterGet(), is(false));
        assertThat(blockingThread.isWasInterrupted(), is(false));
        assertThat(blockingThread.isThrowableThrown(), is(false));
        queue.put(new Object());
        Thread.sleep(1000);
        assertThat(blockingThread.isReachedAfterGet(), is(true));
        assertThat(blockingThread.isWasInterrupted(), is(false));
        assertThat(blockingThread.isThrowableThrown(), is(false));
        blockingThread.join();
```

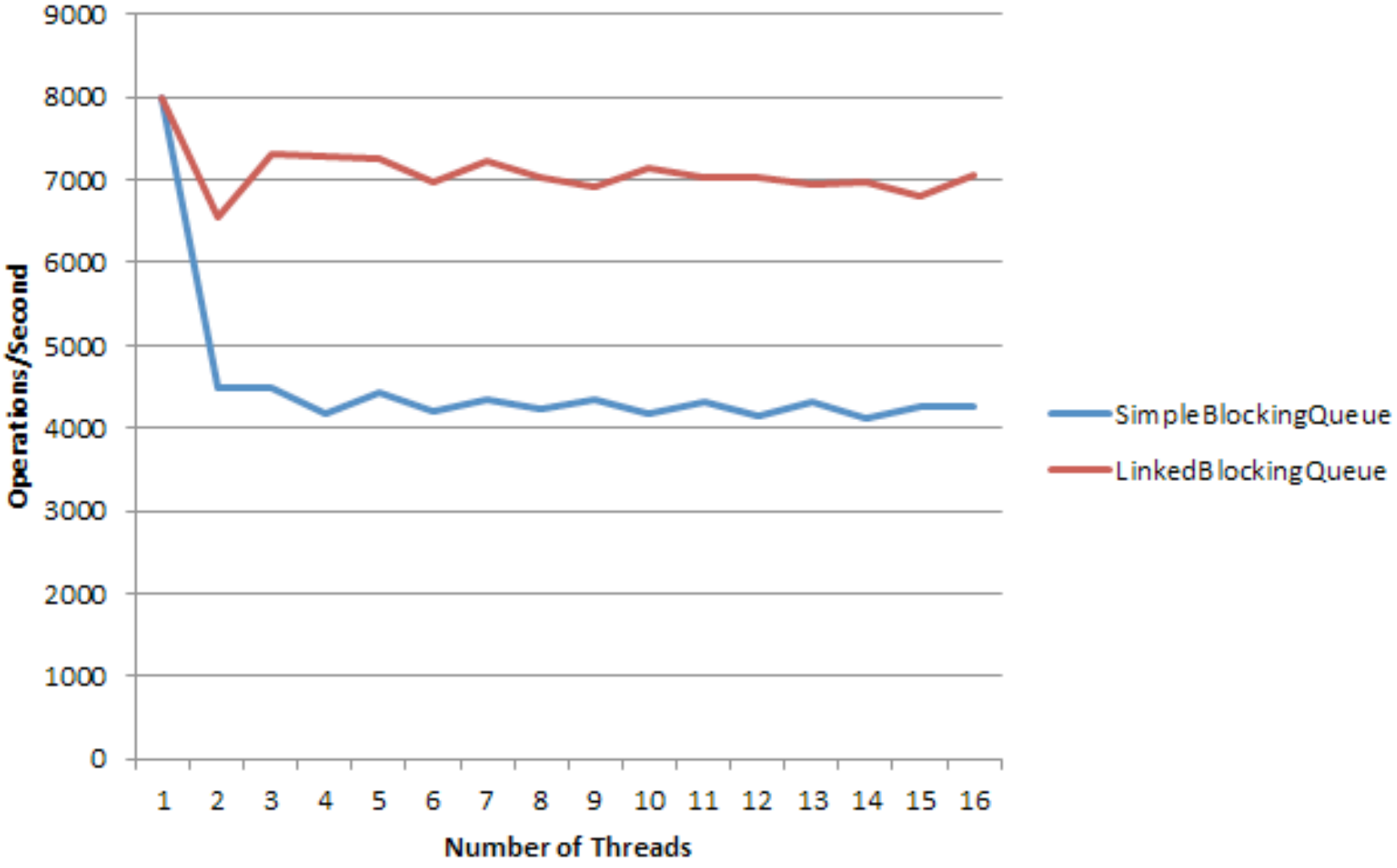
Testing for correctness

Write a new JUnit test to test that Producer out == Consumer in

- Integer values as queue elements
- Integer values from thread local random generator
- Producer thread computes the sum over the elements inserted.
- Sum over all producer threads is compared to the sum of all consumer threads

PXL IT

Testing performance



Testing frameworks

- JMock
- Grobo Utils

JMock Blitzer

```
@Test
public void stressTest() throws InterruptedException {
    final SimpleBlockingQueue<Integer> queue = new SimpleBlockingQueue<Integer>();
    blitzer.blitz(new Runnable() {
        public void run() {
            try {
                queue.put(42);
                queue.get();
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    });
    assertThat(queue.getSize(), is(0));
}
```

Grobo Utils

```
public class SimpleBlockingQueueGroboUtilTest {

	private static class MyTestRunnable extends TestRunnable {
		private SimpleBlockingQueue<Integer> queue;

		public MyTestRunnable(SimpleBlockingQueue<Integer> queue) {
			this.queue = queue;
		}
		@Override
		public void runTest() throws Throwable {
			for (int i = 0; i < 1000000; i++) {
				this.queue.put(42);
				this.queue.get();
			}
		}
	}
	@Test
	public void stressTest() throws Throwable {
		SimpleBlockingQueue<Integer> queue = new SimpleBlockingQueue<Integer>();
		TestRunnable[] testRunnables = new TestRunnable[6];
		for (int i = 0; i < testRunnables.length; i++) {
			testRunnables[i] = new MyTestRunnable(queue);
		}
		MultiThreadedTestRunner mttr = new MultiThreadedTestRunner(testRunnables);
		mttr.runTestRunnables(2 * 60 * 1000);
		assertThat(queue.getSize(), is(0));
	}
}
```