<https://stackoverflow.com/questions/62786471/writing-a-test-case-for-multi-threaded-fizzbuzz>

# [Writing a test case for Multi-threaded FizzBuzz](https://stackoverflow.com/questions/62786471/writing-a-test-case-for-multi-threaded-fizzbuzz)

Q:

I am solving a LeetCode problem from this link: <https://leetcode.com/problems/fizz-buzz-multithreaded/>.

Basically, I am writing a FizzBuzz class that does the standard fizzbuzz, but with 4 threads running at the same time, each calling different methods. (one thread is calling fizzbuzz, one calling fizz, one calling buzz, and one calling number). Here is my solution which is accepted:

class FizzBuzz {

private int n;

int num = 1; // FizzBuzz starts at 1.

public FizzBuzz(int n) {

this.n = n;

}

// printFizz.run() outputs "fizz".

public synchronized void fizz(Runnable printFizz) throws InterruptedException {

while(num <= n) {

if(num % 3 == 0 && num % 5 != 0) {

printFizz.run();

num++;

notifyAll();

} else {

wait();

}

}

}

// printBuzz.run() outputs "buzz".

public synchronized void buzz(Runnable printBuzz) throws InterruptedException {

while(num <= n) {

if(num % 3 != 0 && num % 5 == 0) {

printBuzz.run();

num++;

notifyAll();

} else {

wait();

}

}

}

// printFizzBuzz.run() outputs "fizzbuzz".

public synchronized void fizzbuzz(Runnable printFizzBuzz) throws InterruptedException {

while(num <= n) {

if(num % 15 == 0) {

printFizzBuzz.run();

num++;

notifyAll();

} else {

wait();

}

}

}

// printNumber.accept(x) outputs "x", where x is an integer.

public synchronized void number(IntConsumer printNumber) throws InterruptedException {

while(num <= n) {

if(num % 3 != 0 && num % 5 != 0) {

printNumber.accept(num);

num++;

notifyAll();

} else {

wait();

}

}

}

}

The problem is I am having difficulty simulating the situation in my IDE.

This is my main method:

public class FizzBuzzMain {

public static void main(String[] args) {

FizzBuzz fizzBuzz = new FizzBuzz(15);

Runnable printFizzBuzz = new PrintFizzBuzz();

Runnable printFizz = new PrintFizz();

Runnable printBuzz = new PrintBuzz();

Thread t1 = new Thread(printFizzBuzz);

Thread t2 = new Thread(printFizz);

Thread t3 = new Thread(printBuzz);

IntConsumer printNumber = new PrintNumber();

t1.start();

t2.start();

t3.start();

try {

fizzBuzz.number(printNumber);

fizzBuzz.fizz(printFizz);

fizzBuzz.buzz(printBuzz);

fizzBuzz.buzz(printFizzBuzz);

} catch(Exception e) {

e.printStackTrace();

}

}

}

I'd like to initialize fourth thread with IntConsumer (this is given from the question), so that I can execute t4 as well, but I am not sure how to do that. Obviously, my main method stops running at some point since all threads go to wait state and no one is waking them up (t4 is missing here).

A1:

You can do it like below:

public static void main(String[] args) {

FizzBuzz fizzBuzz = new FizzBuzz(15);

Runnable printFizz = () -> System.out.println("fizz");

Runnable printBuzz = () -> System.out.println("buzz");

Runnable printFizzBuzz = () -> System.out.println("fizzbuzz");

IntConsumer printNumber = number -> System.out.println(number);

Thread threadA = new Thread(() -> {

try {

fizzBuzz.fizz(printFizz);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

Thread threadB = new Thread(() -> {

try {

fizzBuzz.buzz(printBuzz);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

Thread threadC = new Thread(() -> {

try {

fizzBuzz.fizzbuzz(printFizzBuzz);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

Thread threadD = new Thread(() -> {

try {

fizzBuzz.number(printNumber);

} catch (InterruptedException e) {

e.printStackTrace();

}

});

threadA.start();

threadB.start();

threadC.start();

threadD.start();

}

And the solution could use Semaphore and AtomicInteger to achieve this concurrency as below:

public class FizzBuzz {

private int n;

private Semaphore lock;

private AtomicInteger counter;

public FizzBuzz(int n) {

this.n = n;

this.lock = new Semaphore(1);

this.counter = new AtomicInteger(1);

}

// printFizz.run() outputs "fizz".

public void fizz(Runnable printFizz) throws InterruptedException {

int step = n/3 - n/15;

int i = 0;

while (i < step) {

lock.acquire();

if (counter.get() % 3 == 0 && counter.get() % 15 != 0) {

printFizz.run();

counter.incrementAndGet();

i++;

}

lock.release();

}

}

// printBuzz.run() outputs "buzz".

public void buzz(Runnable printBuzz) throws InterruptedException {

int step = n/5 - n/15;

int i = 0;

while (i < step) {

lock.acquire();

if (counter.get() % 5 == 0 && counter.get() % 15 != 0) {

printBuzz.run();

counter.incrementAndGet();

i++;

}

lock.release();

}

}

// printFizzBuzz.run() outputs "fizzbuzz".

public void fizzbuzz(Runnable printFizzBuzz) throws InterruptedException {

int step = n/15;

int i = 0;

while (i < step) {

lock.acquire();

if (counter.get() % 15 == 0) {

printFizzBuzz.run();

counter.incrementAndGet();

i++;

}

lock.release();

}

}

// printNumber.accept(x) outputs "x", where x is an integer.

public void number(IntConsumer printNumber) throws InterruptedException {

int step = n - n/3 - n/5 + n/15;

int i = 0;

while (i < step) {

lock.acquire();

if (counter.get() % 3 != 0 && counter.get() % 5 != 0) {

printNumber.accept(counter.get());

counter.incrementAndGet();

i++;

}

lock.release();

}

}

}

A2:

I fixed your classes so that you can run it in your IDE and then implemented the solution provided by [Emma](https://stackoverflow.com/users/6553328/emma):

abstract class FizzBuzzRunner implements Runnable {

protected FizzBuzz fizzBuzz;

public FizzBuzzRunner(FizzBuzz fizzBuzz) {

this.fizzBuzz = fizzBuzz;

}

abstract protected void print();

}

class PrintFizz extends FizzBuzzRunner {

public PrintFizz(FizzBuzz fizzBuzz) {

super(fizzBuzz);

}

@Override

public void run() {

try {

this.fizzBuzz.fizz(this);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

protected void print() {

System.out.println("fizz");

}

}

class PrintBuzz extends FizzBuzzRunner {

public PrintBuzz(FizzBuzz fizzBuzz) {

super(fizzBuzz);

}

@Override

public void run() {

try {

this.fizzBuzz.buzz(this);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

protected void print() {

System.out.println("buzz");

}

}

class PrintFizzBuzz extends FizzBuzzRunner {

public PrintFizzBuzz(FizzBuzz fizzBuzz) {

super(fizzBuzz);

}

@Override

public void run() {

try {

this.fizzBuzz.fizzbuzz(this);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

@Override

protected void print() {

System.out.println("fizzbuzz");

}

}

class IntConsumer extends PrintFizzBuzz {

public IntConsumer(FizzBuzz fizzBuzz) {

super(fizzBuzz);

}

@Override

public void run() {

try {

this.fizzBuzz.number(this);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public void accept(int n) {

System.out.println(n);

}

}

A small change to the main:

public class Main {

public static void main(String[] args) {

FizzBuzz fizzBuzz = new FizzBuzz(15);

Runnable printFizzBuzz = new PrintFizzBuzz(fizzBuzz);

Runnable printFizz = new PrintFizz(fizzBuzz);

Runnable printBuzz = new PrintBuzz(fizzBuzz);

Runnable printNumber = new IntConsumer(fizzBuzz);

Thread t1 = new Thread(printFizzBuzz);

Thread t2 = new Thread(printFizz);

Thread t3 = new Thread(printBuzz);

Thread t4 = new Thread(printNumber);

t1.start();

t2.start();

t3.start();

t4.start();

}

}

And finally the FizzBuzz class:

class FizzBuzz {

private int n;

private Semaphore semNumber;

private Semaphore semFizz;

private Semaphore semBuzz;

private Semaphore semFizzBuzz;

public FizzBuzz(int n) {

this.n = n;

semNumber = new Semaphore(1);

semFizz = new Semaphore(0);

semBuzz = new Semaphore(0);

semFizzBuzz = new Semaphore(0);

}

// printFizz.print() outputs "fizz".

public void fizz(FizzBuzzRunner printFizz) throws InterruptedException {

for (int counter = 3; counter <= n; counter += 3) {

semFizz.acquire();

printFizz.print();

if ((counter + 3) % 5 == 0) {

counter += 3;

}

semNumber.release();

}

}

// printBuzz.print() outputs "buzz".

public void buzz(FizzBuzzRunner printBuzz) throws InterruptedException {

for (int counter = 5; counter <= n; counter += 5) {

semBuzz.acquire();

printBuzz.print();

if ((counter + 5) % 3 == 0) {

counter += 5;

}

semNumber.release();

}

}

// printFizzBuzz.print() outputs "fizzbuzz".

public void fizzbuzz(FizzBuzzRunner printFizzBuzz) throws InterruptedException {

for (int counter = 15; counter <= n; counter += 15) {

semFizzBuzz.acquire();

printFizzBuzz.print();

semNumber.release();

}

}

// printNumber.accept(x) outputs "x", where x is an integer.

public void number(IntConsumer printNumber) throws InterruptedException {

for (int counter = 1; counter <= n; counter++) {

semNumber.acquire();

if (counter % 15 == 0) {

semFizzBuzz.release();

} else if (counter % 5 == 0) {

semBuzz.release();

} else if (counter % 3 == 0) {

semFizz.release();

} else {

printNumber.accept(counter);

semNumber.release();

}

}

}

}