2022.1 Multicore Computing, Project #2

Problem 3

Document

소프트웨어학부

20176342 송민준

(i)-a Explain the interface/class BlockingQueue and ArrayBlockingQueue with your own English sentences.

Blocking Queue is a type of queue that supports blocking. add() and offer() add an item to the queue, but add() creates an exception if the item fails to insert, and offer returns false without making an exception

Add() and offer() are as normal as any other data structure, but put will block the queue if there is not enough space and wait indefinitely.

Similarly, take() takes items out of the queue, but if there are no items, it blocks them indefinitely until they occur so that they can wait.

Therefore, it is recommended to use the blocking method in situations where resources compete with each other in a multi-threaded environment.

And the 'poll()' method can give you a timeout option, so you have the option of waiting for a few seconds even if you don't have any elements to import.

"Array Blocking Queue" is "Blocking Queue" and first-in first-out queue.. Initially, an array is created according to the size and the size cannot be modified. Because it is an array-based interface, it implements all optional methods of Collection and Iterator.

(i)-b. Create and include (in your document) your own example of multithreaded JAVA code (ex1.java) that is simple and executable. (DO NOT copy&paste) Your example code should use put() and take() methods. Also, include example execution results (i.e. output) in your document.

- **ex1.java**

source code

```
package proj2.prob3;

import java.util.ArrayList;
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;
```

```
public class ex1 {

    public static void main(String[] args) {

        BlockingQueue storage = new ArrayBlockingQueue<String>(50);

        Thread factory = new Thread(new FoodFactory(storage));

        factory.start();

        for(var i = 0; i<3; i++){
            Thread monkey = new Thread(new Monkey("Monkey"+i, storage));
            monkey.start();
        }

        for(var i = 0; i<3; i++){
            Thread human = new Thread(new Human("Human"+i, storage));
            human.start();
        }
    }
}

class FoodFactory implements Runnable{

    private BlockingQueue storage;
    private String[] foodList;

    public FoodFactory(BlockingQueue storage) {
        this.storage = storage;
        this.foodList = new String[]{"Banana","Rice"};
    }

    @Override
    public void run() {

        for (int i = 0; i < 1000; i++) {

            int k = (int)(Math.random()*2);
            System.out.println(k);
            try {
                 System.out.println("Factory produce : "+this.foodList[k]);
                 storage.put(this.foodList[k]);
                 System.out.println("Factory produce Done.
Current :"+storage);


                 Thread.sleep((int)(Math.random()*4000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Monkey implements Runnable{

    private BlockingQueue storage;
    String name;
    public Monkey(String name, BlockingQueue storage) {
        this.name = name;
        this.storage = storage;
    }

    @Override
    public void run() {
        String str;

        for (int i = 0; i < 1000; i++) {
            try {
                System.out.println(this.name+ " : Trying to take Banana");
                str=(String)storage.take();
                if(str.equals("Banana")){
                    System.out.println(this.name+" Takes Banana! Queue Size : 
"+storage.size());
                }
                else {
                    System.out.println(this.name+" Takes Rice. Put it back ");
                    storage.put(str);

                }
                Thread.sleep((int)(Math.random()*15000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Human implements Runnable{

    private BlockingQueue storage;
    String name;
    public Human(String name, BlockingQueue storage) {
        this.name = name;
        this.storage = storage;
    }
```

```java
    @Override
    public void run() {
        String str;

        for (int i = 0; i < 1000; i++) {
            try {
                System.out.println(this.name+ " : Trying to Rice");
                str=(String)storage.take();
                if(str.equals("Rice")){
                    System.out.println(this.name+" Takes Rice! Queue Size :
"+storage.size());
                }
                else {
                    System.out.println(this.name+" Takes Banana. Put it back
");

                    storage.put(str);

                }
                Thread.sleep((int)(Math.random()*15000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Execution Result of ex1.java

```
PS C:₩Users₩a₩multicore> java proj2/prob3/ex1

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Monkey2 : Trying to take Banana

Monkey2 Takes Rice. Put it back

Monkey1 : Trying to take Banana

Monkey0 : Trying to take Banana

Monkey1 Takes Rice. Put it back

Human0 : Trying to Rice
```

```
Human2 : Trying to Rice

Human1 : Trying to Rice

Human0 Takes Rice! Queue Size : 0

Human0 : Trying to Rice

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Human2 Takes Rice! Queue Size : 0

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Human1 Takes Rice! Queue Size : 0

Monkey2 : Trying to take Banana

Monkey1 : Trying to take Banana

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Monkey0 Takes Rice. Put it back

Human0 Takes Rice! Queue Size : 0

Monkey0 : Trying to take Banana

0

Factory produce : Banana

Factory produce Done. Current :[Banana]

Monkey2 Takes Banana! Queue Size : 0

Human0 : Trying to Rice

Human1 : Trying to Rice

1
```

```
Factory produce : Rice

Monkey1 Takes Rice. Put it back

Factory produce Done. Current :[Rice]

Monkey0 Takes Rice. Put it back

Human0 Takes Rice! Queue Size : 0

Human2 : Trying to Rice

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Human1 Takes Rice! Queue Size : 0

Monkey0 : Trying to take Banana

1

Factory produce : Rice

Factory produce Done. Current :[Rice]

Human2 Takes Rice! Queue Size : 0

Monkey2 : Trying to take Banana
```

Explanation of ex1.java

There is a food factory that produces bananas and rice.

And there are three monkeys that eat bananas and three humans who eat rice.

Monkeys compete with each other to eat bananas and humans compete with each other to eat rice.

Food produced in the food factory is stored in an array (Array Blocking Queue),

and if humans pick up bananas from the queue, they put them back in the queue.

And if the monkey picks up the rice from the cue, it puts it back in the cue.

(ii)-a. Do the things similar to (i)-a for the class ReadWriteLock.

'ReadWriteLock' is similar to 'ReentrantLock' used for mutual exclusion of threads and shared resources, but it is different.

'ReadWriteLock' allows multiple threads to read one shared resource at the same time because the read action does not matter if all threads do it at the same time.

This means that multiple threads can read in the critical section. However, 'write lock' can only write lock one thread at a time. It has a similar effect to mutex lock.

This means that only one thread can enter the critical section to write action.

(iii)-b. Do the things similar to (i)-b for lock(), unlock(), readLock() and writeLock() of ReadWriteLock. (ex2.java)

- **ex2.java**

source code

```
package proj2.prob3;

import java.util.ArrayList;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class ex2 {

    private static final ReentrantReadWriteLock lock
        = new ReentrantReadWriteLock(true);
    private static String message = "";

    public static void main(String[] args)
        throws InterruptedException
    {

        ArrayList<Thread> readThreads = new ArrayList<Thread>();

        for(var i = 0; i<3; i++){
            Thread t = new Thread(new Read("Read"+i));
            readThreads.add(t);
        }
        Thread t2 = new Thread(new Writer1());
        Thread t3 = new Thread(new Writer2());
        Thread t4 = new Thread(new Writer3());

        for(var i = 0; i<3; i++){
            readThreads.get(i).start();
        }
        t2.start();
        t3.start();
        t4.start();
        for(var i = 0; i<3; i++){
            readThreads.get(i).join();
        }
        t2.join();
        t3.join();
        t4.join();

    }

    static class Read implements Runnable {
        String name;
        Read(String name){
            this.name = name;
        }
        public void run()
        {

            for (int i = 0; i <= 10; i++) {
                lock.readLock().lock();

                System.out.println( this.name + " : Message = " + message);
                lock.readLock().unlock();
            }
        }
    }

    static class Writer1 implements Runnable {
        public void run()
        {

            for (int i = 0; i <= 10; i++) {
                try {
                    lock.writeLock().lock();
                    System.out.println( "Writer 1 Will concat 1 = " + message);
```

```java
                    message = message.concat("1");
                }
                finally {
                    lock.writeLock().unlock();
                }
            }
        }
    }

    static class Writer2 implements Runnable {
        public void run()
        {

            for (int i = 0; i <= 10; i++) {
                try {
                    lock.writeLock().lock();
                    System.out.println( "Writer 2 Will concat 2 " + message);

                    message = message.concat("2");
                }
                finally {
                    lock.writeLock().unlock();
                }
            }
        }
    }

    static class Writer3 implements Runnable {
        public void run()
        {

            for (int i = 0; i <= 10; i++) {
                try {
                    lock.writeLock().lock();
                    System.out.println( "Writer 3 Will concat 3 " + message);

                    message = message.concat("3");
                }
                finally {
                    lock.writeLock().unlock();
                }
            }
        }
    }

}
```

Result of ex2.java

```
PS C:₩Users₩a₩multicore> java proj2/prob3/ex2

Read1 : Message =

Read2 : Message =

Read0 : Message =

Writer 1 Will concat 1 =

Writer 3 Will concat 3 1

Writer 2 Will concat 2 13

Read1 : Message = 132

Read0 : Message = 132

Read2 : Message = 132

Writer 1 Will concat 1 = 132

Writer 3 Will concat 3 1321

Writer 2 Will concat 2 13213

Read1 : Message = 132132

Read2 : Message = 132132

Read0 : Message = 132132

Writer 1 Will concat 1 = 132132

Writer 3 Will concat 3 1321321

Writer 2 Will concat 2 13213213

Read1 : Message = 132132132

Read0 : Message = 132132132

Read2 : Message = 132132132

Writer 1 Will concat 1 = 132132132

Writer 3 Will concat 3 1321321321

Writer 2 Will concat 2 13213213213
```

```
Read1 : Message = 132132132132

Read0 : Message = 132132132132

Read2 : Message = 132132132132

Writer 1 Will concat 1 = 132132132132

Writer 3 Will concat 3 1321321321321

Writer 2 Will concat 2 13213213213213

Read1 : Message = 132132132132132

Read2 : Message = 132132132132132

Read0 : Message = 132132132132132

Writer 1 Will concat 1 = 132132132132132

Writer 3 Will concat 3 1321321321321321

Writer 2 Will concat 2 13213213213213213

Read1 : Message = 132132132132132132

Read0 : Message = 132132132132132132

Read2 : Message = 132132132132132132

Writer 1 Will concat 1 = 132132132132132132

Writer 3 Will concat 3 1321321321321321321

Writer 2 Will concat 2 13213213213213213213

Read1 : Message = 132132132132132132132

Read2 : Message = 132132132132132132132

Read0 : Message = 132132132132132132132

Writer 1 Will concat 1 = 132132132132132132132

Writer 3 Will concat 3 1321321321321321321321

Writer 2 Will concat 2 13213213213213213213213

Read2 : Message = 132132132132132132132132

Read0 : Message = 132132132132132132132132

Read1 : Message = 132132132132132132132132
```

```
Writer 1 Will concat 1 = 132132132132132132132

Writer 3 Will concat 3 1321321321321321321321

Writer 2 Will concat 2 13213213213213213213213

Read2 : Message = 132132132132132132132132

Read1 : Message = 132132132132132132132132

Read0 : Message = 132132132132132132132132

Writer 1 Will concat 1 = 132132132132132132132132

Writer 3 Will concat 3 1321321321321321321321321

Writer 2 Will concat 2 13213213213213213213213213

Read2 : Message = 132132132132132132132132132

Read0 : Message = 132132132132132132132132132

Read1 : Message = 132132132132132132132132132

Writer 1 Will concat 1 = 132132132132132132132132132

Writer 3 Will concat 3 1321321321321321321321321321

Writer 2 Will concat 2 13213213213213213213213213213
```

Explanation of ex2.java

ex2 has three read and three write threads. Each read thread will repeat 10 print operations, and each write thread will write the numbers 1, 2, and 3 10 times. Therefore, safe read and write can be performed through read lock, unlock, and write lock and unlock.

Initially, 'reader' reads an empty string, but each iteration reads one, three, two, or one folded string. The 'writer' acquires 'write lock' sequentially and connects the characters (numbers) that it is responsible for.

(iii)-a. Do the things similar to (i)-a for the class AtomicInteger

The 'AtomicInteger' class provides a class that can be read and written atomically. Provide concurrency (stability) using the 'Compare and swap' technique.

This technique compares the variable value of the currently operating thread with the variable value of the cpu cache and replaces it with a new value if it matches.

Compared to the synchronized area, the 'Atomic Integer' class provides much faster speeds.

The 'AtomicInteger' class provides the 'get()' , 'set()' , 'addAndGet()' , and 'getAndAdd()' methods. The 'get()' method allows values to be safely imported regardless of the thread, and the 'set()' method allows values to be set securely as well.

The 'addAndGet()' method is a method that first adds the value passed to the parameter and then returns the result value.

The 'getAndAdd()' method is a method that gets the value first and then adds the value passed to the parameter. You can see the difference between 'prefix' and 'postfix'.

(iii)-b. Do the things similar to (i)-b for get(), set(), getAndAdd(), and addAndGet() methods of AtomicInteger. (ex3.java)

- ex3.java

Source code

```java
package proj2.prob3;

import java.util.ArrayList;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class ex3 {

    public static AtomicInteger a = new AtomicInteger(1);

    public static void main(String[] args){
        primeCalculator.a = a;
        primeCalculator.max = 100000;
        primeCalculator.primeCount.set(0);

         ArrayList<Thread> th = new ArrayList<Thread>();
         for(var i = 0; i<10; i++){
             Thread t = new Thread(new primeCalculator("Th"+i));
             th.add(t);
         }

         for(var i = 0; i<10; i++){
            th.get(i).start();
        }

        for(var i = 0; i<10; i++){
            try {
                th.get(i).join();
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        System.out.println("prime count : "+primeCalculator.primeCount.get());
    }
}

class primeCalculator implements Runnable{
    String name;
    static AtomicInteger a;
    static AtomicInteger primeCount = new AtomicInteger(-1);
    static int max;

    primeCalculator(String name){
        this.name = name;
    }
    public void run(){
        while(true){
            int num = a.addAndGet(1);
            if(num < primeCalculator.max){
                if(isPrime(num)){
                    int count = primeCount.getAndAdd(1);
                }

            }
            else {
                break;
            }
        }

    }
    private boolean isPrime(int x){
        int i;
        if(x<=1) return false;
        for(i=2;i<x;i++){
          if(x%i == 0) return false;
        }
        return true;
      }
}
```

Result of ex3.java

```
PS C:₩Users₩a₩multicore> java proj2/prob3/ex3

prime count : 9592
```

Explanation of ex3.java

The result shows that the number of 'prime numbers' is obtained, as we did in the previous homework.

The total number of threads is 10 and synchronization is implemented using only "Atomic Integer" instead of creating a "synchronized area".

Each thread uses an integer called 'a' which is a shared resource to determine if a is a prime number. The values that each thread must determine are obtained using the 'addAndGet()' method.

Therefore, threads do not check the same value between themselves. If the number of integers determined by the thread is small, the primeCount value is raised using the 'getAndAdd()' method.

If the value each thread seeks is greater than primeCalculator.max (100000), it stops through break; after all threads stop working, the primeCalculator.primeCount.get() method outputs a small number of threads obtained by 10 threads.

(iv)-a. Do the things similar to (i)-a for the class CyclicBarrier.

If there are multiple threads, the cyclicBarrier blocks a certain number of threads until it reaches wait() for the thread. In cyclicbarrier, the timeout option allows a particular number of threads to pass even if wait() is not reached.

(iv)-b. Do the things similar to (i)-b for await() methods of CyclicBarrier. (ex4.java)

- ex4.java

Source code

```java
package proj2.prob3;

import java.util.ArrayList;
import java.util.Random;
import java.util.concurrent.BrokenBarrierException;
import java.util.concurrent.CyclicBarrier;
import java.util.concurrent.atomic.AtomicInteger;
import java.util.concurrent.locks.ReentrantReadWriteLock;

public class ex4 {
    private static CyclicBarrier  cyclicBarrier = new CyclicBarrier(10);


    public static void main(String[] args) {
        for(int i = 0; i < 10; ++i) {
            new Thread(new Student(i)).start();
        }
    }

    public static class Student implements Runnable {
        private int id = 0;
        private double successRate = 30;
        private int tryCount = 0;
        private static Random random = new Random(System.currentTimeMillis());

        public Student(int id) {
            this.id = id;
        }
        @Override
        public void run() {
            while(true){
                int sleep = random.nextInt(2000) + 1000;
                try {
                    Thread.sleep(sleep);
                    boolean temp = this.tryToSubmit();
                    System.out.println("Student(" + id + ")
Try("+(++tryCount)+") to submit homework = "+temp);
                    if(temp){
                        System.out.println("Student(" + id + ")
Try("+(++tryCount)+") submitted homework Successfully");

                        try {
                            cyclicBarrier.await();
                        } catch (BrokenBarrierException e) {
                            // TODO Auto-generated catch block
                            e.printStackTrace();
                        }
                        break;
                    }
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }


            }

            System.out.println("Profressor says All Student has submitted the
homework : Student(" + id + ") END!");
        }

        private boolean tryToSubmit(){
            int a = (int)(Math.random()*100);

            if(a<successRate){
                return true;
            }
            else {
                return false;
            }
        }

    }
}
```

Result of ex4.java

```
PS C:₩Users₩a₩multicore> java proj2/prob3/ex4

Student(4) Try(1) to submit homework = true

Student(4) Try(2) submitted homework Successfully

Student(8) Try(1) to submit homework = false

Student(5) Try(1) to submit homework = false

Student(9) Try(1) to submit homework = false

Student(7) Try(1) to submit homework = false

Student(6) Try(1) to submit homework = true

Student(6) Try(2) submitted homework Successfully

Student(1) Try(1) to submit homework = false

Student(3) Try(1) to submit homework = false

Student(0) Try(1) to submit homework = false

Student(2) Try(1) to submit homework = false

Student(8) Try(2) to submit homework = true

Student(8) Try(3) submitted homework Successfully

Student(1) Try(2) to submit homework = false

Student(7) Try(2) to submit homework = false

Student(5) Try(2) to submit homework = false

Student(3) Try(2) to submit homework = false

Student(2) Try(2) to submit homework = true

Student(2) Try(3) submitted homework Successfully

Student(1) Try(3) to submit homework = false

Student(9) Try(2) to submit homework = false

Student(0) Try(2) to submit homework = true

Student(0) Try(3) submitted homework Successfully

Student(7) Try(3) to submit homework = true
Student(7) Try(4) submitted homework Successfully
Student(9) Try(3) to submit homework = false
Student(3) Try(3) to submit homework = true
Student(3) Try(4) submitted homework Successfully
Student(5) Try(3) to submit homework = true
Student(5) Try(4) submitted homework Successfully
Student(1) Try(4) to submit homework = false
Student(9) Try(4) to submit homework = true
Student(9) Try(5) submitted homework Successfully
Student(1) Try(5) to submit homework = true
Student(1) Try(6) submitted homework Successfully
Profressor says All Student has submitted the homework : Student(0) END!
Profressor says All Student has submitted the homework : Student(8) END!
Profressor says All Student has submitted the homework : Student(3) END!
Profressor says All Student has submitted the homework : Student(5) END!
Profressor says All Student has submitted the homework : Student(1) END!
Profressor says All Student has submitted the homework : Student(9) END!
Profressor says All Student has submitted the homework : Student(6) END!
Profressor says All Student has submitted the homework : Student(4) END!
Profressor says All Student has submitted the homework : Student(2) END!
Profressor says All Student has submitted the homework : Student(7) END!
```

Explanation of ex4.java

In the ex4 example code above, there are 10 students and each student randomly attempts to submit a task every 2 to 3 seconds.

Each assignment is successful with a 30% chance of submitting, and if it fails, wait for 2 to 3 seconds again, and try until the resubmission is successful.

Use a 10-size cyclicBarrier to wait for all students to submit their assignments successfully, and at the end, the professor informs you that all students have submitted their assignments successfully.