Project #3

Multicore Computing

Problem 2

Date	Jun 13, 2020
Instructor	Bongsoo Sohn
Std. Name	Junhyuck Woo
Std. ID	20145337

INDEX

ENVIRONMENT

Hardware

- MacBook Pro (15-inch, 2017)
- Processor: 2.8 GHz Quad-Core Intel Core i7
- Memory: 16GB 2133 MHz LPDDR3

Operating System

- macOS Catalina, ver: 10.15.4

IDE (Integrated Development Environment)

- Visual Studio Code 1. 45.1
- gcc version 8.4.0 (Homebrew GCC 8.4.0_1)

Testing Environment

- iTerm2
- Build 3.3.9
- openjdk 14.0.1 2020-04-14
  OpenJDK Runtime Environment (build 14.0.1+7)
  OpenJDK 64-Bit Server VM (build 14.0.1+7, mixed mode, sharing)

SUBPROBLEM #1

(i)-a: Explain the interface/class BlockingQueue and ArrayBlockingQueue

BlockingQueue

- As the interface designed in JAVA, basic data structure is the queue. This data structure controls the data flow by block the insertion when the queue is full. This makes achieve to thread-safe. Also this is primarily used to make producer-consumer pattern.

ArrayBlockingQueue

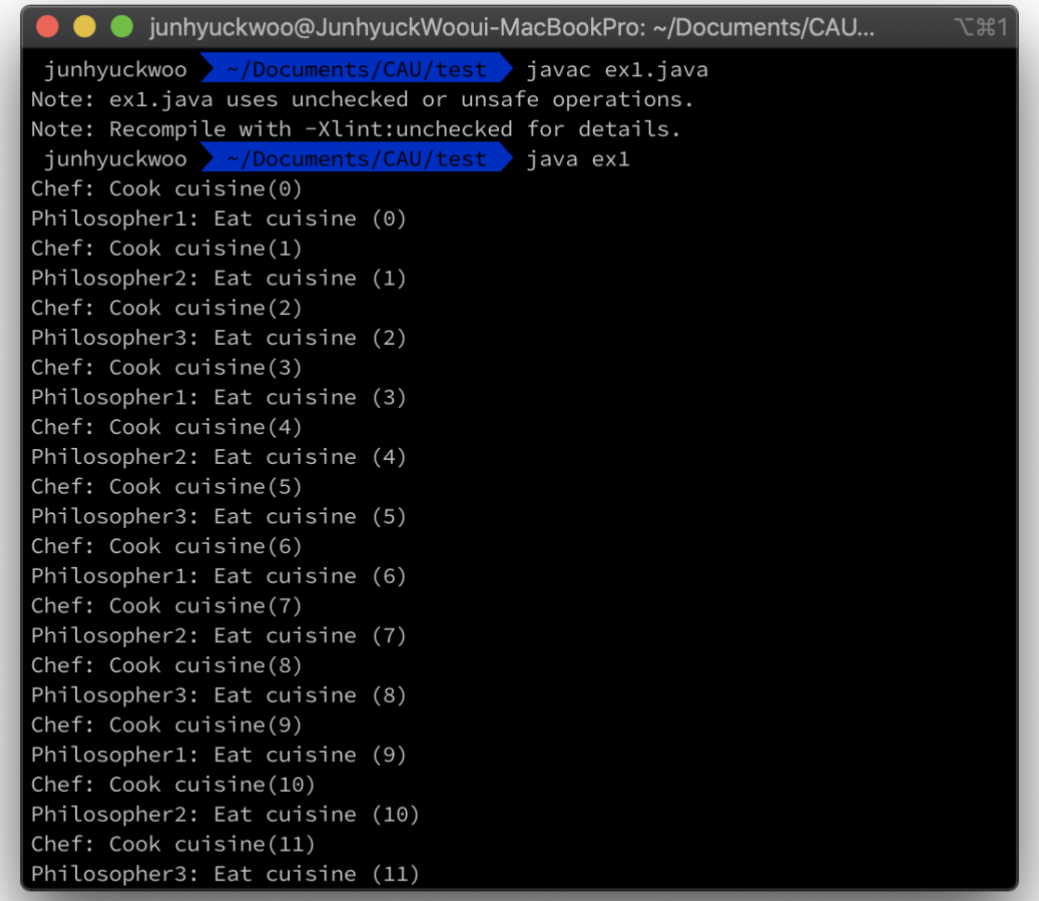
- It is a one of the implemntation of BlockingQueue. As following the name, the ArrayBlockingQueue has a characteristic of array. When it uses, user needs to declare the size of queue and no one can change it after declaration. The other things are same as described in the Blocking Queue.

(i)-b: Example of multithreaded JAVA code

- Source Code

```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June 13, 2020
// Project #3
//  - problem 1: BlockingQueue
import java.util.concurrent.ArrayBlockingQueue;
import java.util.concurrent.BlockingQueue;

public class ex1 {

    public static void main(String[] args) {
        BlockingQueue table = new
ArrayBlockingQueue<String>(3);
        Cook cook = new Cook(table);
        Philosopher p1 = new Philosopher(table,
"1");
        Philosopher p2 = new Philosopher(table,
"2");
        Philosopher p3 = new Philosopher(table,
"3");

        cook.start();
        p1.start();
        p2.start();
        p3.start();
    }
}

class Cook extends Thread {
    private BlockingQueue dish;
    public Cook(BlockingQueue table) { dish =
table; }
    public void run() {
        for (int i=0; i < 12; i++) {
            System.out.println("Chef: Cook cuisine(" +
i + ")");
            try {
                dish.put("cuisine ("+i+")");
Thread.sleep((int)(Math.random()*2000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

class Philosopher extends Thread {
    private BlockingQueue dish;
    private String id;
    public Philosopher(BlockingQueue table, String
num) { dish = table; id = num; }
    public void run() {
        String str;
        for (int i=0; i<4; i++) {
            try {
                str=(String)dish.take();
                System.out.println("Philosopher" + id +
": Eat " + str);

Thread.sleep((int)(Math.random()*2000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- Output


junhyuckwoo@JunhyuckWooui-MacBookPro: ~/Documents/CAU...
⌥⌘1
junhyuckwoo ~/Documents/CAU/test javac ex1.java
Note: ex1.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
junhyuckwoo ~/Documents/CAU/test java ex1
Chef: Cook cuisine(0)
Philosopher1: Eat cuisine (0)
Chef: Cook cuisine(1)
Philosopher2: Eat cuisine (1)
Chef: Cook cuisine(2)
Philosopher3: Eat cuisine (2)
Chef: Cook cuisine(3)
Philosopher1: Eat cuisine (3)
Chef: Cook cuisine(4)
Philosopher2: Eat cuisine (4)
Chef: Cook cuisine(5)
Philosopher3: Eat cuisine (5)
Chef: Cook cuisine(6)
Philosopher1: Eat cuisine (6)
Chef: Cook cuisine(7)
Philosopher2: Eat cuisine (7)
Chef: Cook cuisine(8)
Philosopher3: Eat cuisine (8)
Chef: Cook cuisine(9)
Philosopher1: Eat cuisine (9)
Chef: Cook cuisine(10)
Philosopher2: Eat cuisine (10)
Chef: Cook cuisine(11)
Philosopher3: Eat cuisine (11)

SUBPROBLEM #2

(i)-a: Explain the class Semaphore

Semaphore

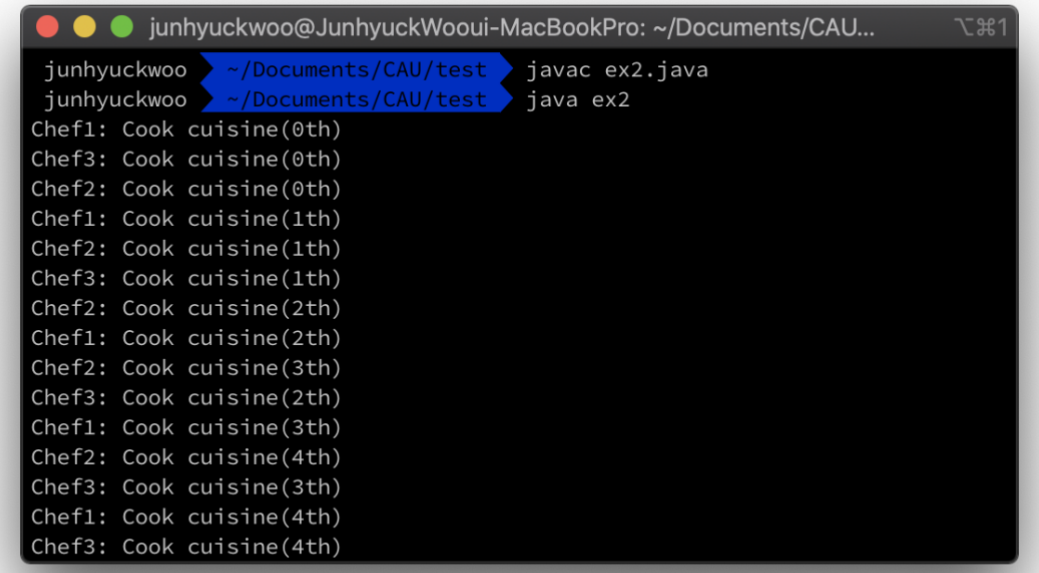
- As the class designed in JAVA, it is used to control the resource occupation. For example, when the process enters the critical section, process occupied the lock and the other process could not enter the critical section.

(i)-b: Example of multithreaded JAVA code

- Source Code

```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June 13, 2020
// Project #3
//  - problem 2: Semaphore
import java.util.concurrent.Semaphore;

public class ex2 {

    public static void main(String[] args) {
        Semaphore kitchen = new Semaphore(1);
        Chef cook1 = new Chef(kitchen, "1");
        Chef cook2 = new Chef(kitchen, "2");
        Chef cook3 = new Chef(kitchen, "3");

        cook1.start();
        cook2.start();
        cook3.start();
    }
}

class Chef extends Thread {
    private Semaphore fire;
    private String id;
    public Chef(Semaphore kitchen, String num) { fire = kitchen; id = num; }
    public void run() {
        for (int i=0; i < 5; i++) {
            try {
                fire.acquire();
                System.out.println("Chef" + id + ": Cook cuisine(" + i + "th)");
                Thread.sleep((int)(Math.random()*2000));
                fire.release();
                Thread.sleep((int)(Math.random()*2000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- Output


junhyuckwoo@JunhyuckWooui-MacBookPro: ~/Documents/CAU...
⌥⌘1
junhyuckwoo ~/Documents/CAU/test javac ex2.java
junhyuckwoo ~/Documents/CAU/test java ex2
Chef1: Cook cuisine(0th)
Chef3: Cook cuisine(0th)
Chef2: Cook cuisine(0th)
Chef1: Cook cuisine(1th)
Chef2: Cook cuisine(1th)
Chef3: Cook cuisine(1th)
Chef2: Cook cuisine(2th)
Chef1: Cook cuisine(2th)
Chef2: Cook cuisine(3th)
Chef3: Cook cuisine(2th)
Chef1: Cook cuisine(3th)
Chef2: Cook cuisine(4th)
Chef3: Cook cuisine(3th)
Chef1: Cook cuisine(4th)
Chef3: Cook cuisine(4th)

SUBPROBLEM #3

(i)-a: Explain the class ReadWriteLock

ReadWriteLock

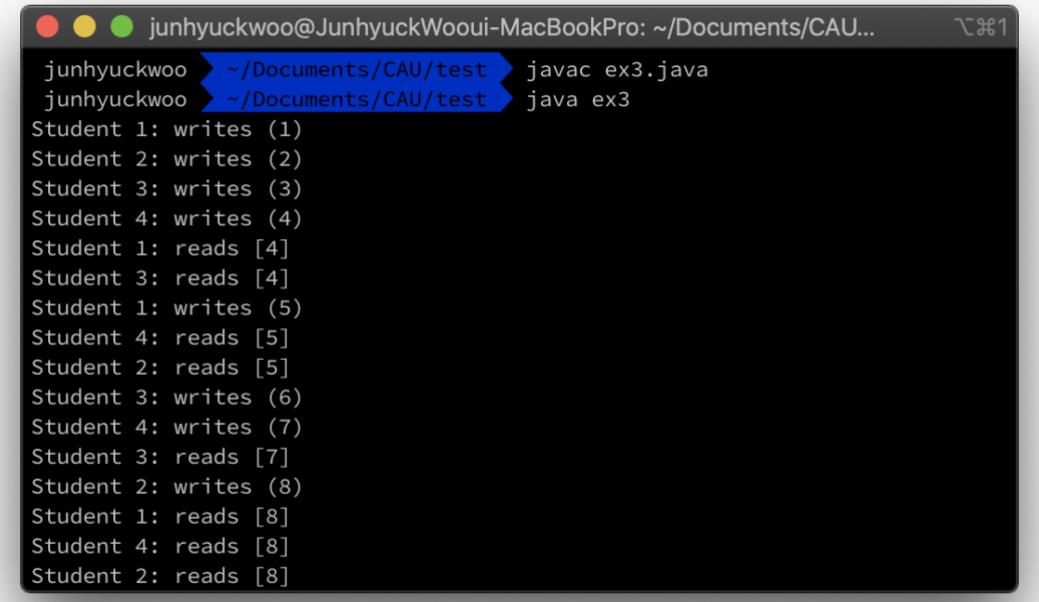
- ReadWriteLock is devided as ReadLock and WriteLock. ReadLock is used for control flow the read, and WriteLock is used for control flow the write. The writing action can bring a race condition, so writelock allows to enter critical section only one users, however reading action allows to enter multiple users.

(i)-b: Example of multithreaded JAVA code

- Source Code

```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June 13, 2020
// Project #3
//  - problem 3: ReadWriteLock
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReadWriteLock;
import
java.util.concurrent.locks.ReentrantReadWriteLock;

public class ex3 {

    public static void main(String[] args) {
        int[] book = new int[1];
        book[0] = 0;
        ReadWriteLock lock = new
ReentrantReadWriteLock();
        Lock writeLock = lock.writeLock();
        Lock readLock = lock.readLock();

        Student std1 = new Student(writeLock,
readLock,"1", book);
        Student std2 = new Student(writeLock,
readLock, "2", book);
        Student std3 = new Student(writeLock,
readLock, "3", book);
        Student std4 = new Student(writeLock,
readLock, "4", book);

        std1.start();
        std2.start();
        std3.start();
        std4.start();
    }
}

class Student extends Thread {
    private Lock writing;
    private Lock reading;
    private String id;
    private int[] page;
    public Student(Lock writelock, Lock readlock,
String num, int[] book) {
writing = writelock;
reading = readlock ;
        id = num;
        page = book;
    }

    public void run() {
        for (int i=1; i < 3; i++) {

            // Write the pages
            writing.lock();
            try {
                page[0] += 1;
                System.out.println("Student " + id + ":
writes (" + page[0] + ")");
            } finally {
                writing.unlock();
            }

            try {

Thread.sleep((int)(Math.random()*2000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            // Read pages
            reading.lock();
            try {
                System.out.println("Student " + id + ":
reads [" + page[0] + "]");
            }  finally {
                reading.unlock();
            }

            try {

Thread.sleep((int)(Math.random()*2000));
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

- Output


junhyuckwoo@JunhyuckWooui-MacBookPro: ~/Documents/CAU...
⌥⌘1
junhyuckwoo ~/Documents/CAU/test javac ex3.java
junhyuckwoo ~/Documents/CAU/test java ex3
Student 1: writes (1)
Student 2: writes (2)
Student 3: writes (3)
Student 4: writes (4)
Student 1: reads [4]
Student 3: reads [4]
Student 1: writes (5)
Student 4: reads [5]
Student 2: reads [5]
Student 3: writes (6)
Student 4: writes (7)
Student 3: reads [7]
Student 2: writes (8)
Student 1: reads [8]
Student 4: reads [8]
Student 2: reads [8]

SUBPROBLEM #4

(i)-a: Explain the class AtomicInteger

AtomicInteger

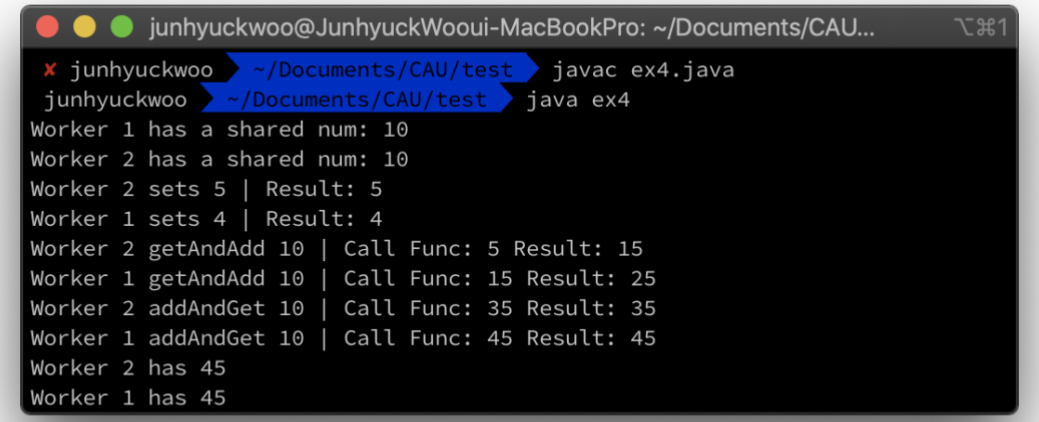
- It is the class with has the integer data structure. It covers the variable size of simultaneity. To do that, class offers various type of method. In the example code, get(), set(), addAndGet(), and getAndAdd() are covered.

(i)-b: Example of multithreaded JAVA code

- Source Code

```
// Writer: Junhyuck Woo
// Lecture: Multicore Computing
// Organization: Chung-Ang University
// Deadline: June 13, 2020
// Project #3
//  - problem 4: AtomicInteger
import java.util.concurrent.atomic.AtomicInteger;

public class ex4 {
    public static void main(String[] args) {
        AtomicInteger init = new AtomicInteger(10);
        Worker c1 = new Worker(init, "1", 4);
        Worker c2 = new Worker(init, "2", 5);

        c1.start();
        c2.start();
    }
}

class Worker extends Thread {
    private AtomicInteger num;
    private String thread_id;
    private int buf;
    public Worker(AtomicInteger init, String id, int tmp) { num = init; thread_id=id; buf = tmp; }

    public void run() {
        // Get
        System.out.println("Worker " + thread_id + " has a shared num: " + num.get());

        // Set
        num.set(buf);
        System.out.println("Worker " + thread_id + " sets " + buf + " | Result: " + num.get());

        // getAndAdd
        System.out.println("Worker " + thread_id + " getAndAdd " + 10 + " | Call Func: " +
num.getAndAdd(10) + " Result: " + num.get());

        // AddAndGet
        System.out.println("Worker " + thread_id + " addAndGet " + 10 + " | Call Func: " +
num.addAndGet(10) + " Result: " + num.get());

        // Get
        System.out.println("Worker " + thread_id + " has " + num.get());
    }
}
```

- Output


junhyuckwoo@JunhyuckWooui-MacBookPro: ~/Documents/CAU...
⌥⌘1
junhyuckwoo ~/Documents/CAU/test javac ex4.java
junhyuckwoo ~/Documents/CAU/test java ex4
Worker 1 has a shared num: 10
Worker 2 has a shared num: 10
Worker 2 sets 5 | Result: 5
Worker 1 sets 4 | Result: 4
Worker 2 getAndAdd 10 | Call Func: 5 Result: 15
Worker 1 getAndAdd 10 | Call Func: 15 Result: 25
Worker 2 addAndGet 10 | Call Func: 35 Result: 35
Worker 1 addAndGet 10 | Call Func: 45 Result: 45
Worker 2 has 45
Worker 1 has 45