Laboratory Assignment 4

Dealing with Deadlock in Java

Module: Concurrency (CS-210)
Academic year: 2019-20

Allocated marks: This assignment accounts for 2.5% of the total module marks.

Objectives

The learning objectives of this assignment are as follows.

- To analyse given code and identify deadlock conditions.
- To apply a Coffman condition breaking strategy to deal with deadlock.
- To use Multiverse [1] library and apply a Software Transactional Memory (STM) solution towards breaking deadlock.

Resources:

- For the STM part you will need to download `multiverse-core-0.7.0.jar` and `multiverse-core-0.7.0-javadoc.jar` files from the following website:

  `https://search.maven.org/classic/#search%7Cgav%7C1%7Cg%3A%22org.multiverse%22%20AND%20a%3A%22multiverse-core%22`

- Apart from the lecture notes, you may find the following tutorial useful for the STM part.

  `https://www.baeldung.com/java-multiverse-stm`

Tasks

Consider the following scenario.

> A library has a couple of instances of `Shelf`, and they can hold `Book` objects. Multiple `Swapper` threads can use these shelves and `swap` books from one to another concurrently.

We have provided you with an implementation of the scenario with this document; please download the code and use your favourite Java development environment for the following tasks. For completeness, we have provided the code here in this document (see Appendix A).

[1] `https://github.com/pveentjer/Multiverse`

Task 1. Identifying deadlock.

Considering the Coffman conditions, analyse the given code, and determine whether the implementation may lead to deadlock. Provide justifications to your assesor.

> Hint: The Coffman conditions for deadlock are as follows:
>
> **Mutual Exclusion.** Shared resources requiring exclusive access.
>
> **Hold and wait.** A process may hold a resource while waiting for further resources to become available.
>
> **No pre-emption.** A process holding a resource is solely responsible to letting it go: no one can force the process to let go.
>
> **Circular wait.** There exists a set of processes $\{P_1, \ldots, P_n\}$ such that P_1 waits on P_2, then P_2 waits on P_3, and so on. The cycle is completed when P_n is waiting on P_1.

Task 2: Breaking one of the Coffman conditions.

Based on your analysis, come up with a strategy to break the deadlock and implement it in Java.

> Hint: In lecture 12, we discussed a range of technique to deal with deadlock. Is any of these suitable for this case?

Task 3. Implement the code with Multiverse library.

Your next task is to make the code work with Multiverse library that implements software transactional memory (STM) for Java.

> Hint: In lecture 14, we discussed software transactional memory (STM). You may find the lecture notes useful here.

Task 4: List benefits and shortcomings of STM.

Provide *two* benefits and *two* shortcomings of using STM in dealing with concurrency.

> Once you have completed all the tasks, please make sure that you have been signed off.

Appendix A: Provided Code

— Book.java —

```
package libraryshelves;
public class Book {
    private String name;
    private boolean isDummy;
    Book(String name){
        this.name = name;
        if (name!="") isDummy = false;
        else isDummy = true;
    }
    Book(){
        this.name = "";
        this.isDummy= true;
    }
    public String getName() {
        return name;
    }
}
```

— Shelf.java —

```
package libraryshelves;
public class Shelf {
    private Book[] bookArray;
    private int capacity;
    private int id;
    private boolean isTaken;
    Shelf(int capacity, int id){
        this.id = id;
        this.capacity = capacity;
        bookArray = new Book[capacity];
        Book book = bookArray[0];
        for(int i=0; i<capacity; i++)
            bookArray[i] = new Book();
        this.isTaken = false;
    }
    public int getCapacity(){
        return capacity;
    }
    public int getId(){
        return id;
    }
    public synchronized void acquire()
            throws InterruptedException{
```

```
        while (isTaken) wait();
        isTaken = true;
        notifyAll();
    }
    public synchronized void release(){
        isTaken = false;
        notifyAll();
    }
    public Book getBookAtIndex(int index){
        return bookArray[index];
    }
    public void setBookAtIndex(int index, Book book){
        bookArray[index] = book;
    }
    public synchronized void swap(Shelf other, int originIndex,
       int destIndex){
        Book origin = getBookAtIndex(originIndex);
        Book dest = other.getBookAtIndex(destIndex);
        setBookAtIndex(originIndex, dest);
        other.setBookAtIndex(destIndex, origin);
    }
}
```

— Swapper.java —

```
package libraryshelves;
import java.util.Random;
public class Swapper implements Runnable{
    private Shelf shelfA;
    private Shelf shelfB;
    private String name;
    Swapper(String name, Shelf shelfA, Shelf shelfB){
        this.name = name;
        this.shelfA = shelfA;
        this.shelfB = shelfB;
    }
    @Override
    public void run() {
        Random random = new Random();
        int randomInt = 0;
        int randomIndA = 0;
        int randomIndB = 0;
        while(true){
            try {
                randomInt = random.nextInt(1000); // upto 1 sec
                Thread.sleep(randomInt);
```

```
                randomIndA =
                   random.nextInt(shelfA.getCapacity());
                randomIndB =
                   random.nextInt(shelfB.getCapacity());
                System.out.println(name + " is trying to acquire
                   " + shelfA.getId());
                shelfA.acquire();
                System.out.println(name + " is trying to acquire
                   " + shelfB.getId());
                shelfB.acquire();
                shelfA.swap(shelfB, randomIndA, randomIndB);
                System.out.println(name + " completed swap.");
                shelfA.release();
                shelfB.release();
                System.out.println(name + " has released the
                   shelves.");
            } catch (InterruptedException ex) {
                System.out.println("Thread Interrupted.");
                break;
            }
        }
    }
}
```

— LibraryShelves.java —

```
package libraryshelves;
public class LibraryShelves {
    public static void main(String[] args)
            throws InterruptedException {
        Shelf shelfA= new Shelf(5, 98);
        Shelf shelfB= new Shelf(5, 54);
        Swapper sw1 = new Swapper("sw1", shelfA, shelfB);
        Swapper sw2 = new Swapper("sw2", shelfB, shelfA);
        Thread t1 = new Thread(sw1);
        Thread t2 = new Thread(sw2);
        t1.start();
        t2.start();
        Thread.sleep(10000);
        t1.interrupt();
        t2.interrupt();
        t1.join();
        t2.join();
    }
}
```