Laboratory Assignment 5

# Software Transactional Memory with `multiverse` Library

Module: Concurrency (CS-210)
Academic year: 2020-21

**Allocated marks: This assignment accounts for 2% of the total module marks.**

---

## Objectives

The learning objectives of this assignment are as follows.

- To apply data parallelism (software transactional memory) to deal with a concurrent problem.

## 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 in our Github repository; please download the code from Github, and use your favourite Java development environment for the following tasks.

> Link to the Github repository:
> `https://github.com/AlmaRahat/CS-210-Concurrency/tree/main/java-code/libraryshelves`
> Documentation for the `multiverse` library can be found in: `https://github.com/AlmaRahat/CS-210-Concurrency/tree/main/java-code/multiverse`
> The bank account example and the required `jar` files can be found in: `https://github.com/AlmaRahat/CS-210-Concurrency/tree/main/java-code/AccountSTM`

For completeness, we have provided the code for the library scenario in this document (see Appendix A).

### Task 1. 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 lectures 18 and 19, we discussed data parallelism usin software transactional memory (STM). You may find the lecture notes useful here.

**Task 2: 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();
    }
}
```