

Problem 2: Threads synchronization and communication

$$A1 = 500 * (500 + 1)/2 = 125250$$

$$B1 = 250 * (250 + 1)/2 = 31375$$

$$B2 = A1 + [200 * (200 + 1)/2] = 145350$$

$$A2 = B2 + [300 * (300 + 1)/2] = 190500$$

$$B3 = A2 + [400 * (400 + 1)/2] = 270700$$

$$A3 = B3 + [400 * (400 + 1)/2] = 350900$$

In the code, `join()` are used for both threads to allow them to wait for each other to execute. This is used within the for loop, as given:

```
public static void main(String[] args) throws InterruptedException {
    int testSize = 5;

    Data mySample = new Data();

    for (int i = 0; i < testSize; i++) {
        System.out.println("\n\nThis is iteration #" + (i+1) + "\n");

        mySample.goFunA2 = false;
        mySample.goFunB2 = false;
        mySample.goFunA3 = false;
        mySample.goFunB3 = false;

        ThreadA ta = new ThreadA(mySample);
        ThreadB tb = new ThreadB(mySample);

        ta.start();
        tb.start();

        ta.join();
        tb.join();
    }
}
```

In the code of the threads themselves, a while(goFun__ == false) block is used to allow the synchronized portion of the thread to thread.wait() for a notification from the other thread, as given:

```
synchronized(sample) {  
    try {  
        while (sample.goFunA2 == false) {  
            sample.wait();  
            System.out.println("Thread A2 waiting ");  
        }  
  
        int n = 300;  
        sample.A2 = sample.B2 + n*(n+1)/2;  
        System.out.println("A2 finished " + sample.A2);  
        sample.goFunB3 = true;  
        sample.notify();  
  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
}
```

This avoids using thread.sleep().

The resulting code, when done in 5 iterations to verify how correct the implementation is, gives the following results:

| | |
|---|--|
| <pre>This is iteration #1 A1 finished 125250 B1 finished 31375 B2 finished 145350 Thread A2 waiting A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> | <pre>This is iteration #6 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> |
| <pre>This is iteration #2 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> | <pre>This is iteration #7 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> |
| <pre>This is iteration #3 A1 finished 125250 B1 finished 31375 B2 finished 145350 Thread A2 waiting A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> | <pre>This is iteration #8 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> |
| <pre>This is iteration #4 A1 finished 125250 B1 finished 31375 B2 finished 145350 Thread A2 waiting A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> | <pre>This is iteration #9 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> |
| <pre>This is iteration #5 A1 finished 125250 B1 finished 31375 B2 finished 145350 Thread A2 waiting A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> | <pre>This is iteration #10 A1 finished 125250 B1 finished 31375 B2 finished 145350 A2 finished 190500 Thread B3 waiting B3 finished 270700 Thread A3 waiting A3 finished 350900</pre> |

In all of the iterations, all values match the calculated values in the correct orders.