IT2030 – Object Oriented Programming   **Semester 1, 2018**
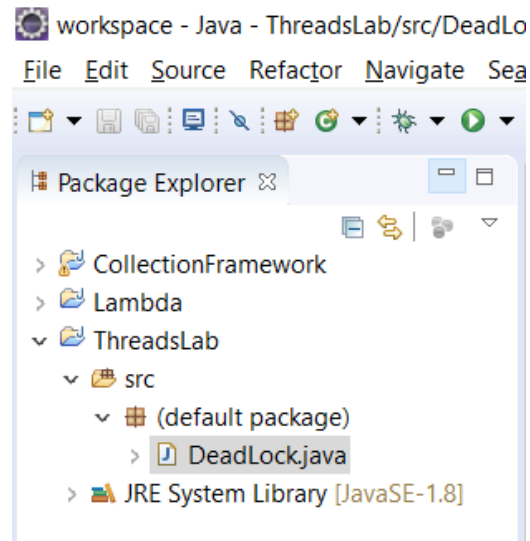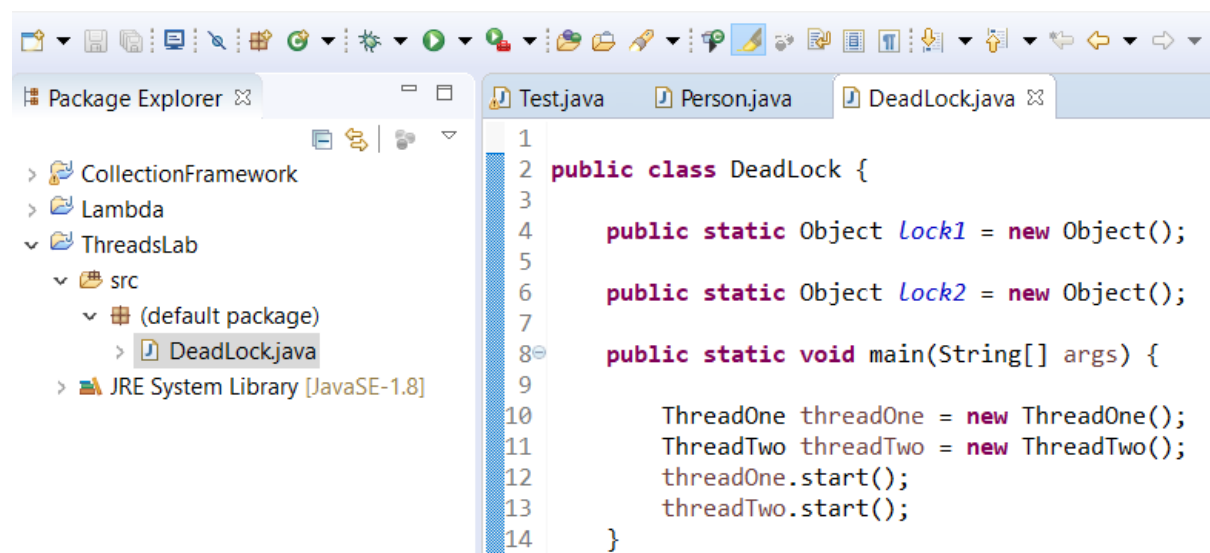
Exercise 01 (20 minutes)

How to create Deadlock using Two Threads. Create Two Threads and two locks. First create a project using eclipse. Then implement **DeadLock.java** class within the project.

Then you can implement DeadLock.java class as follows.

Now you should create two locks for both threads respectively (**lock1, lock2**)

Then, create both **ThreadOne** and **ThreadTwo** classes as **static inner classes** with in the **DeadLock.java**.

IT2030 – Object Oriented Programming   **Semester 1, 2018**

```java
/**
 * Thread one with locks
 * @author udara.s
 *
 */
static class ThreadOne extends Thread {

    @Override
    public void run() {

        System.out.println("Started Executing Thread 1");

        synchronized (lock1) {

            System.out.println("Thread 1 holding lock 1....");

            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("Thread 1 is awaiting for the lock2");

            synchronized (lock2) {

                System.out.println("Thread 1 is holding lock 1 & lock 2");

            }
        }
    }
}
```

You can create static inner class and override the run() method with two synchronized blocks as above. To create the deadlock two locks needs to be interchanged in both Threads.

Now create another ThreadTwo class similar to the ThreadOne class and interchange two locks as below (lock1 and lock2)

ThreadOne => synchronized (**lock1**){ synchronized (**lock2**){ //resource }}

ThreadTwo => synchronized (**lock2**){ synchronized (**lock1**){ //resource }}

Lab Exercise - Threads

IT2030 – Object Oriented Programming   Semester 1, 2018

```java
/**
 * Thread 2 with two locks
 * @author udara.s
 *
 */
static class ThreadTwo extends Thread {

    @Override
    public void run() {

        System.out.println("Started Executing Thread 2");

        synchronized (lock2) {

            System.out.println("Thread 2 holding lock 2....");

            try {
                Thread.sleep(10);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }

            System.out.println("Thread 2 is awaiting for the lock1");

            synchronized (lock1) {

                System.out.println("Thread 2 is holding lock 1 & lock 2");

            }
        }
    }
}
```
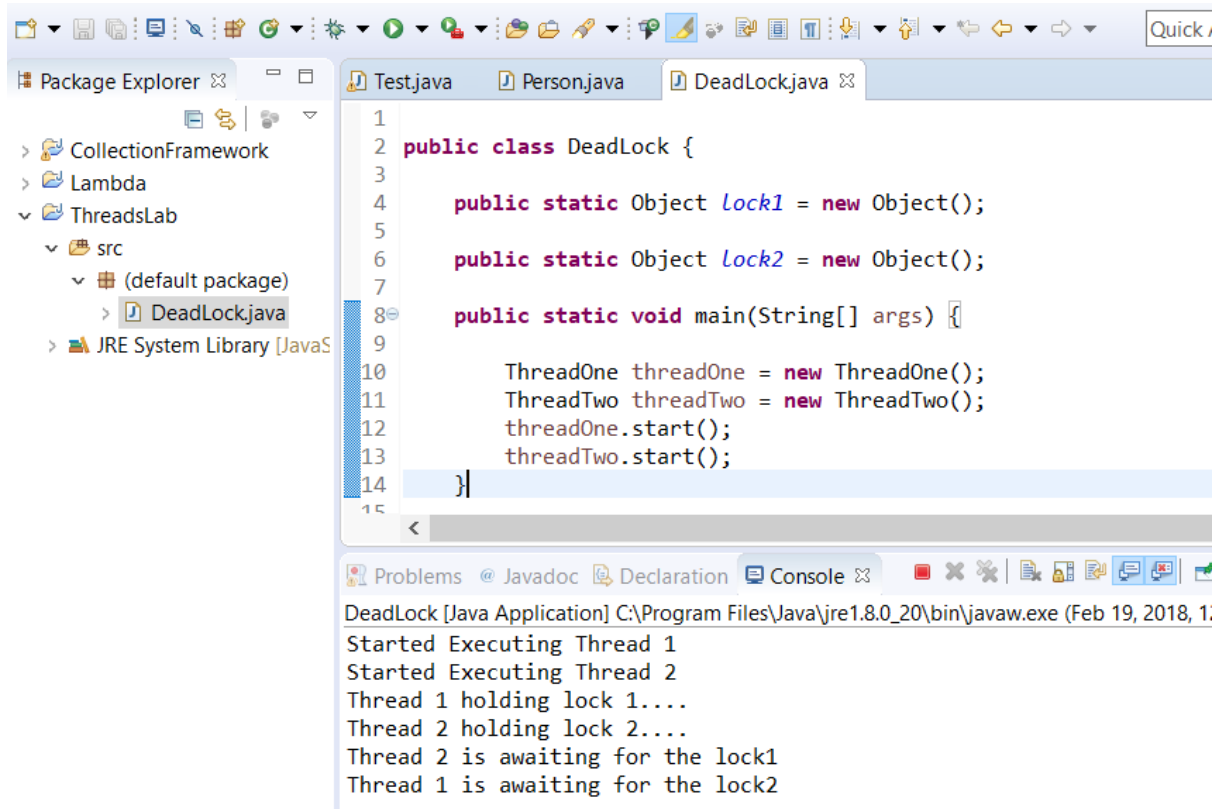
Now Run your Java application and check the outputs. In console Program is in the running state.

ThreadOne does not proceed up the line **"Thread 1 is holding lock 1 & 2"** and **ThreadTwo** also not execute lines **"Thread 2 is holding lock 1 & 2".**

Both Threads are awaiting to each other until releases their locks to access resources.

```
public class DeadLock {

    public static Object lock1 = new Object();

    public static Object lock2 = new Object();

    public static void main(String[] args) {

        ThreadOne threadOne = new ThreadOne();
        ThreadTwo threadTwo = new ThreadTwo();
        threadOne.start();
        threadTwo.start();
    }
```

```
Started Executing Thread 1
Started Executing Thread 2
Thread 1 holding lock 1....
Thread 2 holding lock 2....
Thread 2 is awaiting for the lock1
Thread 1 is awaiting for the lock2
```
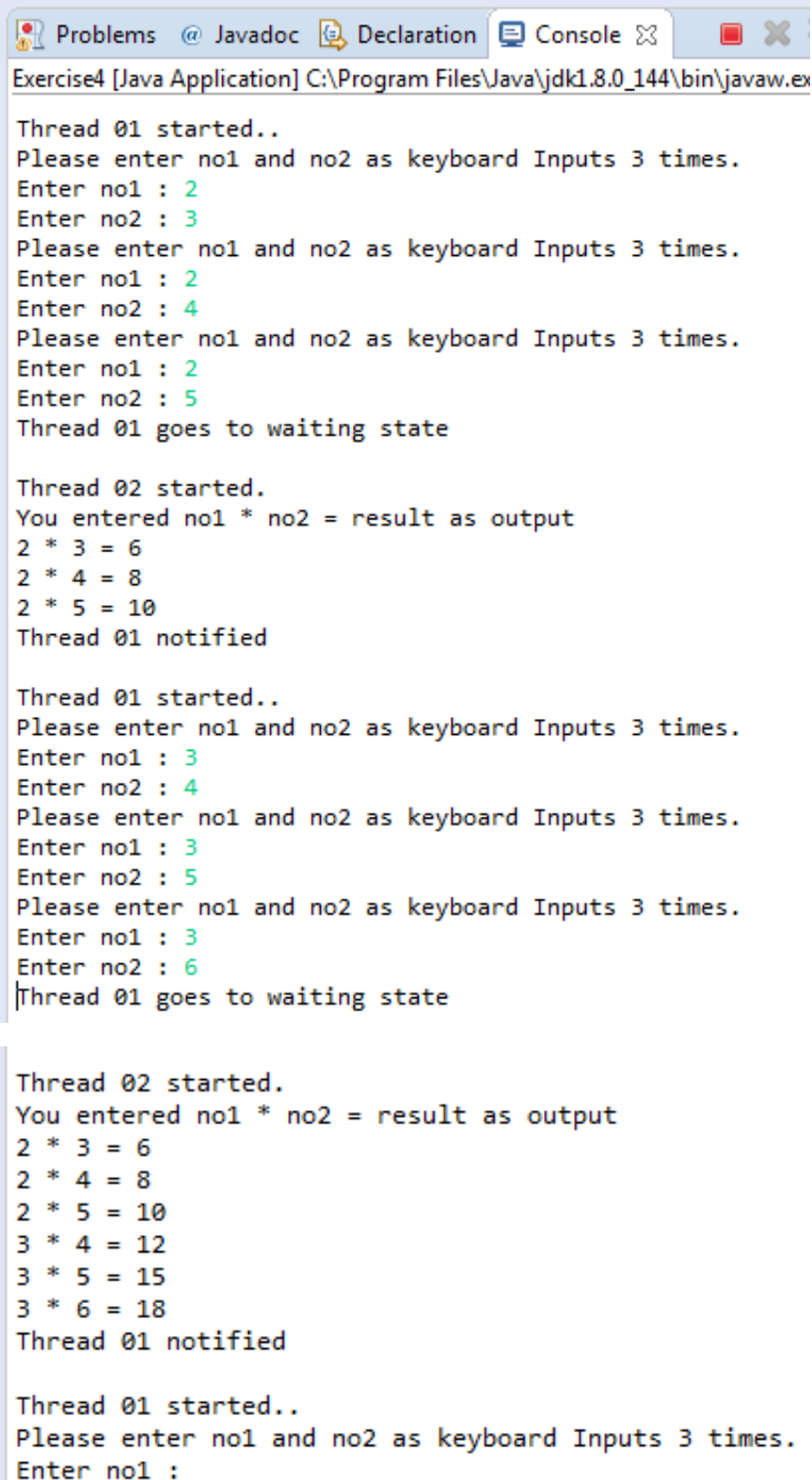
Exercise 02 (20 minutes)

Now modify the above program and remove all static inner classes (ThreadOne and ThreadTwo) and instead implement two separate concrete classes (Thread1 and Thread2). Then you should recreate the same above scenario using that approach. You should display the same outcome as above with this latest modification.

Exercise 03 (10 minutes)

Modify your program again to resolve the deadlock. Once you resolve the deadlock execute and display the outcome.

Exercise 04 (1 hour) Use wait() and notify()

Create a program to display following output. **Refer the mentioned steps below for this implementation**.

```
 Problems   @ Javadoc   Declaration   Console ⊠        ■  ✗

Exercise4 [Java Application] C:\Program Files\Java\jdk1.8.0_144\bin\javaw.ex

Thread 01 started..
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 2
Enter no2 : 3
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 2
Enter no2 : 4
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 2
Enter no2 : 5
Thread 01 goes to waiting state

Thread 02 started.
You entered no1 * no2 = result as output
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
Thread 01 notified

Thread 01 started..
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 3
Enter no2 : 4
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 3
Enter no2 : 5
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 : 3
Enter no2 : 6
Thread 01 goes to waiting state

Thread 02 started.
You entered no1 * no2 = result as output
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
Thread 01 notified

Thread 01 started..
Please enter no1 and no2 as keyboard Inputs 3 times.
Enter no1 :
```

IT2030 – Object Oriented Programming   **Semester 1, 2018**

Follow the steps below.

1) Create two Threads (T1) and (T2) and use wait() method in one thread and notify() method in other Thread

2) You should request key board inputs 3 times for no1 and no2 and store them in a map
   Hint: Map<String, Integer> *number1* **= new LinkedHashMap<String, Integer>();**

3) T1, Thread read keyboard inputs and store them in a map and T2 Thread should display content in the Map.

4) T1 and T2 **Threads should continuously run** and both should switch for keyboard inputs and display() map content of each keyboard inputs. (T1 → T2 → T1…)
   Hint: Use wait( ) and notify() methods

5) Keyboard input is taken as key for the map (no1 * no2) and multiplied output will be stored as relevant value of the map.
   Hint: map.put(no1 + " * " + no2 + " = ", no1 * no2);