# Assignment 1 - Q2

| | |
|---|---|
| Date | @September 26, 2025 |
| Label | Assignment |

## 2. Deadlocks

Below is a simple program that showcases a deadlock.

```
package examples;

public class Deadlock {

   private static class DeadlockThread1 extends Thread {

      public void run() {
         synchronized (lock1) {
            System.out.println("Thread 1 holding lock 1...");

            try {
               Thread.sleep(1000); // sleep to increase chances of deadlock
            } catch (InterruptedException e) {
               // ignore exception
            }

            System.out.println("Thread 1 waiting for lock 2...");

            synchronized (lock2) {
               System.out.println("Thread 1 holding locks 1 and 2...");
            }
         }
      }
   }

   private static class DeadlockThread2 extends Thread {

      public void run() {
```

```
            synchronized (lock2) {
                System.out.println("Thread 2 holding lock 2...");

                try {
                    Thread.sleep(1000); // sleep to increase chances of deadlock
                } catch (InterruptedException e) {
                    // ignore exception
                }

                System.out.println("Thread 2 waiting for lock 1...");

                synchronized (lock1) {
                    System.out.println("Thread 2 holding locks 1 and 2...");
                }
            }
        }
    }

    public static Object lock1 = new Object();
    public static Object lock2 = new Object();

    public static void main(String[] args) {
        DeadlockThread1 t1 = new DeadlockThread1();
        DeadlockThread2 t2 = new DeadlockThread2();

        t1.start();
        t2.start();
    }
}
```

## 2.1 How to Cause Deadlocks

We will give a brief overview of how the above program demonstrates a deadlock. Suppose there are two shared resources: `A` & `B`. Naturally, we would implement locks to ensure proper synchronization and safe access to these shared resources (in this example, there are no resources, but we have 2 locks). Suppose now we have two threads: `t1` & `t2`. `t1` tries to access `A`, then `B`. `t2` on the other hand tries to access `B`, then `A`. Given this, there is an interleaving where `t1` acquires `A`, `t2` acquires `B`, and both are stuck waiting for each other to relinquish the second resource they both need. This interleaving leads to a deadlock — both threads are stuck waiting and nothing happens!

## 2.2 How to Avoid Deadlocks

In general, we can modify the program using some design patterns to avoid deadlocks. The simplest fix would be to implement resource ordering. This means that both threads would try acquiring the locks in the same order. Thus, if a thread can't obtain the first lock, it blocks which removes the risk of contention with other resources downstream. Another solution would be to implement a non-blocking acquisition. For example, a thread would try to obtain a lock with a defined timeout interval. If the timeout expires, then the thread backs of and backs off by relinquishing its locks. Yet another solution would be to reduce the scope of the lock. By minimizing the critical section, we reduce time holding the lock and reduce the chances of lock contention.

By far the easiest solution is to implement resource ordering. You may find below the previous program modified with resource ordering to prevent deadlocks from occurring.

```
public class ResourceOrdering {

    private static class ResourceOrderingThread extends Thread {

        private Integer threadNum;
```

```
    public ResourceOrderingThread(Integer threadNum) {
      this.threadNum = threadNum;
    }

    public void run() {
      synchronized (lock1) {
        System.out.println(
          String.format("Thread %d holding lock 1...", threadNum)
        );

        try {
          Thread.sleep(1000);
        } catch (InterruptedException e) {
          // ignore exception
        }

        System.out.println(
          String.format("Thread %d waiting for lock 2...", threadNum)
        );

        synchronized (lock2) {
          System.out.println(
            String.format(
              "Thread %d holding locks 1 and 2...",
              threadNum
            )
          );
        }
      }
    }
  }

  public static Object lock1 = new Object();
  public static Object lock2 = new Object();

  public static void main(String[] args) {
    ResourceOrderingThread t1 = new ResourceOrderingThread(1);
    ResourceOrderingThread t2 = new ResourceOrderingThread(2);

    t1.start();
    t2.start();
  }
}
```