DEADLOCK.

The following classes are implemented to demonstrate deadlock and how to fix it.

I have an account class to represent the account details

```
package ca.mcgill.ecse420.a1;

class Account {

    private int balance = 10000;

    public void deposit(int amount) {
        balance += amount;
    }

    public void withdraw(int amount) {
        balance -= amount;
    }

    public int getBalance() {
        return balance;
    }

    public static void transfer(Account acc1, Account acc2, int
amount) {
        acc1.withdraw(amount);
        acc2.deposit(amount);
    }
}
```

I have 4 public methods in the account class:
Deposit: deposits certain amount to an account
Withdraw: Withdraws a particular amount from an account.
getBalance: returns the balance associated with an account.
Transfer: which takes two parameter as an input and transfers from one account to another.

```
package ca.mcgill.ecse420.a1;

import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

@SuppressWarnings("InfiniteLoopStatement")
public class Runner {
    private Account acc1 = new Account();
    private Account acc2 = new Account();
    private Lock lock1 = new ReentrantLock();
    private Lock lock2 = new ReentrantLock();

    //firstThread runs
    public void firstThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
                Account.transfer(acc1, acc2,
random.nextInt(100));
        }
    }

    //SecondThread runs
    public void secondThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
                Account.transfer(acc2, acc1,
random.nextInt(100));
        }
    }

    //When both threads finish execution, finished runs
    public void finished() {
        System.out.println("Account 1 balance: " +
acc1.getBalance());
        System.out.println("Account 2 balance: " +
acc2.getBalance());
        System.out.println("Total balance: " +
(acc1.getBalance() + acc2.getBalance()));
    }
}
```

The class above is to demonstrate Deadlock. It’s running two threads and transferring money from one account to another. The total balance after the loop terminates should be 2000 but due to threading issues it returns a completely different amount.

One possible way to solve this problem is to lock the threads as showed below.

```
package ca.mcgill.ecse420.a1;

import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

@SuppressWarnings("InfiniteLoopStatement")
public class Runner {

    private Account acc1 = new Account();
    private Account acc2 = new Account();
    private Lock lock1 = new ReentrantLock();
    private Lock lock2 = new ReentrantLock();

    //firstThread runs
    public void firstThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
          lock1.lock();
          lock2.lock();
          try {
               Account.transfer(acc1, acc2,
random.nextInt(100));
          } finally {
               lock1.unlock();
               lock2.unlock();
          }
        }
    }

    //SecondThread runs
    public void secondThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
          lock1.lock();
          lock2.lock();
          try {
               Account.transfer(acc2, acc1,
random.nextInt(100));
          } finally {
               lock1.unlock();
               lock2.unlock();
          }
        }
    }
    //When both threads finish execution, finished runs
    public void finished() {
        System.out.println("Account 1 balance: " +
acc1.getBalance());
        System.out.println("Account 2 balance: " +
acc2.getBalance());
        System.out.println("Total balance: " +
(acc1.getBalance() + acc2.getBalance()));
    }
}
```

The above code will solve the issue and at the end balance will be 2000.

What happened if we change the order of lock and lock account2 first and then account1 next since we are transferring from account2 to account1 as showed below. Since we are doing the locks in different orders this will lead to deadlocks. The first thread try to acquire lock2 but it can't because the second thread acquire locks2 and vice versa for thread lock1. Another possible case of deadlock is with the case of synchronized blocks.

```
package ca.mcgill.ecse420.a1;

import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

@SuppressWarnings("InfiniteLoopStatement")
public class Runner {

    private Account acc1 = new Account();
    private Account acc2 = new Account();
    private Lock lock1 = new ReentrantLock();
    private Lock lock2 = new ReentrantLock();

    //firstThread runs
    public void firstThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
          lock1.lock();
          lock2.lock();
          try {
               Account.transfer(acc1, acc2,
random.nextInt(100));
```

```
            } finally {
                 lock1.unlock();
                 lock2.unlock();
            }
          }
      }

      //SecondThread runs
      public void secondThread() throws InterruptedException {
          Random random = new Random();
          for (int i = 0; i < 10000; i++) {
            lock2.lock();
            lock1.lock();
            try {
                 Account.transfer(acc2, acc1,
random.nextInt(100));
            } finally {
                 lock1.unlock();
                 lock2.unlock();
            }
          }
      }

      //When both threads finish execution, finished runs
      public void finished() {
          System.out.println("Account 1 balance: " +
acc1.getBalance());
          System.out.println("Account 2 balance: " +
acc2.getBalance());
          System.out.println("Total balance: " +
(acc1.getBalance() + acc2.getBalance()));
      }
}
```

A solution to this problem is to write a method that you can lock it in any order and still avoid deadlock. I will write a method to acquire the locks safely to avoid deadlocks as showed below.

```
package ca.mcgill.ecse420.a1;
 import java.util.Random;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;
 @SuppressWarnings("InfiniteLoopStatement")
public class Runner {
     private Account acc1 = new Account();
```

```
    private Account acc2 = new Account();
    private Lock lock1 = new ReentrantLock();
    private Lock lock2 = new ReentrantLock();
     //don't hold several locks at once. If you do, always
acquire the locks in the same order
    //try to get the both locks
    private void acquireLocks(Lock firstLock, Lock secondLock)
throws InterruptedException {
        while (true) {
            // Acquire locks
            boolean gotFirstLock = false;
            boolean gotSecondLock = false;
            try {
                /**
                 * tryLock() which will only acquire a lock if
it’s available
                 * and not already acquired by another thread
and tryLock(long
                 * time,TimeUnit unit), which will try to
acquire a lock and, if
                 * it's unavailable wait for the specified timer
to expire
                 * before giving up
                 */
                gotFirstLock = firstLock.tryLock();
                gotSecondLock = secondLock.tryLock();
            } finally {
                if (gotFirstLock && gotSecondLock) return;
                else if (gotFirstLock) firstLock.unlock();
                else if (gotSecondLock) secondLock.unlock();
            }
            // Locks not acquired
            Thread.sleep(1);
        }
    }
     //firstThread runs
    public void firstThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
            acquireLocks(lock1, lock2);
            try {
                Account.transfer(acc1, acc2,
random.nextInt(100));
            } finally {
                lock1.unlock();
                lock2.unlock();
            }
        }
    }
     //SecondThread runs
    public void secondThread() throws InterruptedException {
        Random random = new Random();
        for (int i = 0; i < 10000; i++) {
            acquireLocks(lock2, lock1);
            try {
                Account.transfer(acc2, acc1,
random.nextInt(100));
            } finally {
                lock1.unlock();
                lock2.unlock();
            }
        }
    }
     //When both threads finish execution, finished runs
    public void finished() {
        System.out.println("Account 1 balance: " +
acc1.getBalance());
        System.out.println("Account 2 balance: " +
acc2.getBalance());
        System.out.println("Total balance: " +
(acc1.getBalance() + acc2.getBalance()));
    }
}
```