Advanced Java: Multi-threading Part 11 - Deadlock

https://www.youtube.com/watch?v=ghCUBi41bGA&index=11&list=PLBB24CFB073F1048E

**3rd class – Account.java:**

**public** **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);

}

}

2nd class – Runner.java:

**import** java.util.Random;

**import** java.util.concurrent.locks.Lock;

**import** java.util.concurrent.locks.ReentrantLock;

**public** **class** Runner {

**private** Account acc1 = **new** Account();

**private** Account acc2 = **new** Account();

**private** Lock lock1 = **new** ReentrantLock();

**private** Lock lock2 = **new** ReentrantLock();

**private** **void** acquireLocks(Lock firstLock, Lock secondLock) **throws** InterruptedException {

**while**(**true**){

// Acquire locks

**boolean** gotFirstLock = **false**;

**boolean** gotSecondLock = **false**;

**try** {

gotFirstLock = firstLock.tryLock();

gotSecondLock = secondLock.tryLock();

} **finally** {

**if**(gotFirstLock && gotSecondLock){

**return**;

}

**if**(gotFirstLock){

firstLock.unlock();

}

**if**(gotSecondLock){

secondLock.unlock();

}

}

//Locks not acquired

Thread.*sleep*(1);

}

}

**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();

}

}

}

**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();

}

}

}

**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()));

}

}

**1st class – App.java:**

**public** **class** App {

**public** **static** **void** main(String args[]) **throws** InterruptedException{

**final** Runner runner = **new** Runner();

Thread t1 = **new** Thread(**new** Runnable(){

**public** **void** run() {

**try** {

runner.firstThread();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

});

Thread t2 = **new** Thread(**new** Runnable(){

**public** **void** run() {

**try** {

runner.secondThread();

} **catch** (InterruptedException e) {

e.printStackTrace();

}

}

});

t1.start();

t2.start();

t1.join();

t2.join();

runner.finished();

}

}

**Results:**

Account 1 balance: 12678

Account 2 balance: 7322

Total balance: 20000

**Important notes:**

-Deadlock can occur when two or more threads start waiting for each other, resulting in malfunction of the threads.

-The programmer gave 2 ways to avoid the deadlock in this situation.

(1) Put both locks in the correct order in both threads. Like below:

Lock1.lock();

Lock2.lock();

(2) Make an acquireLocks method in the code.