CSCE 314 Programming Languages, Spring 2024 – Hyunyoung Lee – Document for Exercise 12

First, watch Video 12.1 to get a general idea on Java concurrency, and then after/while watching Videos 12.2 and 12.3, the questions below can be answered. Studying the BankAccount example programs (posted on Canvas Modules Weeks 12-13) will be helpful. To receive the grade, you need to ***submit your answers on the Exercise 12 link (quiz style) on Canvas by: 11:59 p.m. Wednesday, April 17, 2024***.

Consider the following partial code for banking with deposit and withdraw functions. Study the explanations on the right-side column together with the code.

```
public class DepositRunnable implements Runnable
{ // try to deposit
  public void run() {
    try { account.deposit(amount); }
    catch (InterruptedException exception){}}
  private BankAccount account;
  private double amount;
}
public class WithdrawRunnable implements Runnable
{ // try to withdraw
  public void run() {
    try { account.withdraw(amount); }
    catch (InterruptedException exception){}}
  private BankAccount account;
  private double amount;
}
public class BankAccount
{ private double balance;
  private Lock balanceChangeLock; // (1)
  private Condition sufficientFundsCondition; //(2)
  public BankAccount() {
    balance = 0;
    balanceChangeLock = new ReentrantLock();
    sufficientFundsCondition =
      balanceChangeLock.newCondition(); // (3)
  }
  public void deposit(double amount)
  { balanceChangeLock.lock(); // (4)
    try {
      double nb = balance + amount; // (5)
      balance = nb;  // (6)
      sufficientFundsCondition.signalAll(); // (7)
    } finally { balanceChangeLock.unlock(); } //(8)
  }
  public void withdraw(double amount)
              throws InterruptedException
  { balanceChangeLock.lock(); // (9)
    try {
      while (balance < amount)  // (10)
         sufficientFundsCondition.await(); // (11)
      double nb = balance - amount; // (12)
      balance = nb; } // (13)
    finally { balanceChangeLock.unlock(); } // (14)
  }
}
```

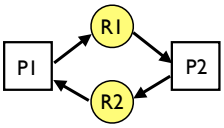
Notice that the code uses a lock object and a condition object (lines (1) and (2)). Suppose that two threads $P1$ and $P2$ are concurrently (almost at the same time) accessing one account with different functions, such that $P1$ is performing a deposit and $P2$ performing a withdrawal. The following statements explain each step-by-step of the development of the code.

Explain why the lock is needed. In other words, what happens if the lock was not used?

If the lock is not used, for example, thread $P1$ executing lines (5) and (6) and thread $P2$ executing lines (12) and (13) may interleave and consequently (Q1) _____ may occur.

Explain why the condition object is needed. In other words, what happens if only the lock is used without the condition object?

The given code without the condition object can lead to (Q2) _____. Consider the following execution scenario: Upon acquiring the lock $P2$ enters the try block. Suppose the condition (balance < amount) is true at that point in time, then $P2$ goes into a busy loop checking whether the condition has become false (so it could withdraw the amount of money, exit the try block, and release the lock). The condition becomes false only if $P1$ deposits money to the account, which requires acquiring the lock. However, the lock is held by $P2$, which would not give up the lock until after it executes the withdrawal, thus reaching a (Q3) _____ and none can make any progress. In the following wait-for graph,



an arrow from Pn to Rn, $n \in \{1, 2\}$, means thread Pn is waiting to acquire resource (or condition) Rn to make progress, and an arrow from Rn to Pm, $m, n \in \{1, 2\}$, means resource Rn is held by (or bound to) thread Pn.

In the above scenario, $R1$ is (Q4) _____ that $P1$ needs to acquire in order to enter the try-block and deposit money but is bound to $P2$, and $R2$ is the condition (Q5) _____, the condition that $P2$ can exit the `while` loop and release the lock but is bound to $P1$'s entering the critical section and depositing money.

Choices: exception, deadlock, race hazard, synchronization `balanceChangeLock`, `sufficientFundsCondition`, `(balance < amount)`, `(balance >= amount)`

```
public class DepositRunnable implements Runnable
{ // . . .
  public void run() {
    try { account.deposit(amount); }
    catch (InterruptedException exception){}}
  private BankAccount account;
  private double amount;
}
public class WithdrawRunnable implements Runnable
{ // . . .
  public void run() {
    try { account.withdraw(amount); }
    catch (InterruptedException exception){}}
  private BankAccount account;
  private double amount;
}
public class BankAccount
{ private double balance;
  private Lock balanceChangeLock; // (1)
  private Condition sufficientFundsCondition; //(2)
  public BankAccount() {
    balance = 0;
    balanceChangeLock = new ReentrantLock();
    sufficientFundsCondition =
      balanceChangeLock.newCondition(); // (3)
  }
  public void deposit(double amount)
  { balanceChangeLock.lock(); // (4)
    try {
      double nb = balance + amount; // (5)
      balance = nb;  // (6)
      sufficientFundsCondition.signalAll(); // (7)
    } finally { balanceChangeLock.unlock(); } //(8)
  }
  public void withdraw(double amount)
              throws InterruptedException
  { balanceChangeLock.lock(); // (9)
    try {
      while (balance < amount)   // (10)
          sufficientFundsCondition.await(); // (11)
      double nb = balance - amount; // (12)
      balance = nb; } // (13)
    finally { balanceChangeLock.unlock(); } // (14)
  }
}
```

Now, consider the following scenario with the given code as it is, with the lock and condition objects:

Suppose that two threads $P1$ and $P2$ are concurrently (almost at the same time) accessing one account with different functions such that $P1$ is performing deposit and $P2$ performing withdrawal. Also suppose that $P2$ acquired the lock and reached line (10), and the condition `(balance < amount)` is true.

The following steps explain what happens in terms of the actions taken by the two threads. Fill in the blanks with either $P1$ or $P2$.

1. Since the condition `(balance < amount)` is true, (Q6) _____ temporarily releases the lock and waits (line (11)) on the signal that will be issued in line (7) by (Q7) _____ in the future.

2. Thread (Q8) _____ acquires the lock in line (4), deposits money in lines (5) and (6), issues the signal (line (7)) indicating that money has been deposited into the account, and finally releases the lock (line (8)).

3. Upon receiving the signal, thread (Q9) _____ can now acquire the lock back, checks whether `(balance < amount)` is false (line (10)), and if so, proceed to withdraw money in lines (12) and (13), and finally releases the lock (line (14)). If `(balance < amount)` is still true, then repeat the steps starting from 1.