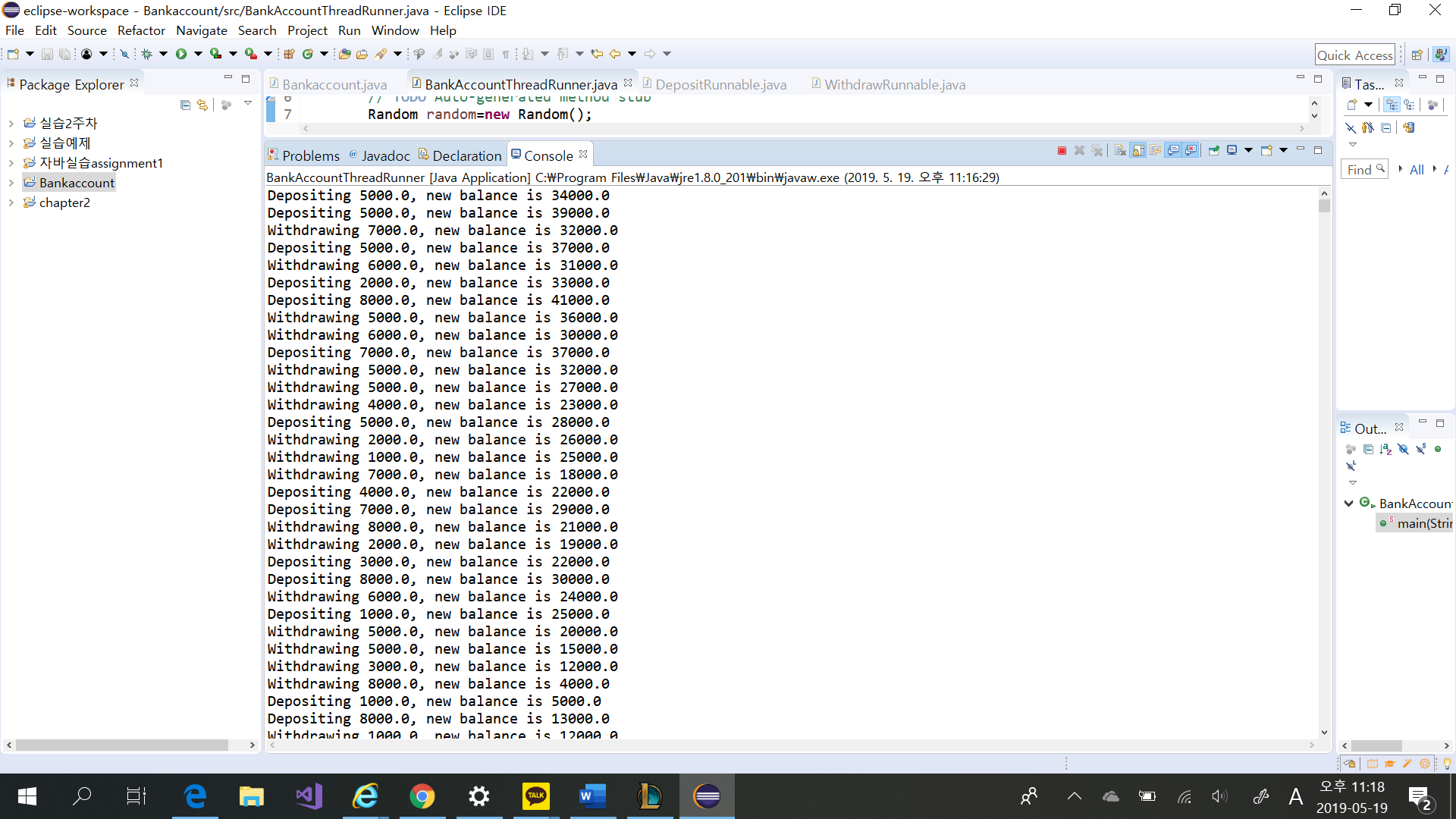
Difference between Synchronized and Lock

자바의 synchronized 블록은 쓰레드들의 진입 순서에 대한 어떠한 보장도 하지 않는다. 때문에 다수의 쓰레드들이 동일한 synchronized 블록에의 접근을 계속 시도한다면, 하나 이상의 쓰레드가 영영 접근 권한을 부여 받지 못하게 될 위험이 있다. 하지만 lock은 모든 메소드가 명시적으로 표현되어 있어 쓰레드를 선택하여 접근할 수 있게 할 수 있다.



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

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

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

**public** **class** Bankaccount {

**private** **double** balance;

**private** Lock balanceChangeLock;

**private** Condition sufficientFundsCondition;

**public** Bankaccount()

{

balance = 0;

balanceChangeLock = **new** ReentrantLock();

sufficientFundsCondition = balanceChangeLock.newCondition();

}

**public** **void** deposit(**double** amount)

{

balanceChangeLock.lock();

**try**

{

System.***out***.print("Depositing " + amount);

**double** newBalance = balance + amount;

System.***out***.println(", new balance is " + newBalance);

balance = newBalance;

sufficientFundsCondition.signalAll();

}

**finally**

{

balanceChangeLock.unlock();

}

}

**public** **void** withdraw(**double** amount)

**throws** InterruptedException

{

balanceChangeLock.lock();

**try**

{

**while** (balance < amount)

sufficientFundsCondition.await();

System.***out***.print("Withdrawing " + amount);

**double** newBalance = balance - amount;

System.***out***.println(", new balance is " + newBalance);

balance = newBalance;

}

**finally**

{

balanceChangeLock.unlock();

}

}

**public** **double** getBalance()

{

**return** balance;

}

}

**import** java.util.Random;

**public** **class** BankAccountThreadRunner {

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

// **TODO** Auto-generated method stub

Random random=**new** Random();

Bankaccount account = **new** Bankaccount();

**final** **int** REPETITIONS = 100;

**final** **int** THREADS = 100;

**for** (**int** i = 1; i <= THREADS; i++)

{ **double** AMOUNT1 = 1000\*(random.nextInt(8)+1);

DepositRunnable d = **new** DepositRunnable(

account, AMOUNT1, REPETITIONS);

**double** AMOUNT2 = 1000\*(random.nextInt(8)+1);

WithdrawRunnable w = **new** WithdrawRunnable(

account, AMOUNT2, REPETITIONS);

Thread dt = **new** Thread(d);

Thread wt = **new** Thread(w);

dt.start();

wt.start();

}

}

}

**public** **class** DepositRunnable **implements** Runnable

{

**private** **static** **final** **int** ***DELAY*** = 1;

**private** Bankaccount account;

**private** **double** amount;

**private** **int** count;

**public** DepositRunnable(Bankaccount anAccount, **double** anAmount,

**int** aCount)

{

account = anAccount;

amount = anAmount;

count = aCount;

}

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

{

**try**

{

**for** (**int** i = 1; i <= count; i++)

{

account.deposit(amount);

Thread.*sleep*(***DELAY***);

}

}

**catch** (InterruptedException exception) {}

}

}

**public** **class** WithdrawRunnable **implements** Runnable{

**private** **static** **final** **int** ***DELAY*** = 1;

**private** Bankaccount account;

**private** **double** amount;

**private** **int** count;

**public** WithdrawRunnable(Bankaccount anAccount, **double** anAmount,

**int** aCount)

{

account = anAccount;

amount = anAmount;

count = aCount;

}

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

{

**try**

{

**for** (**int** i = 1; i <= count; i++)

{

account.withdraw(amount);

Thread.*sleep*(***DELAY***);

}

}

**catch** (InterruptedException exception) {}

}

}