**Java Threads Transactions**

In this challenge, simulate a banking system. Create the *Account* and *Transaction* classes.

1. The *Account* class has a data member *int balance*, initially assigned to zero. The class should implement the following three methods:
   1. *String deposit(int money)* to add *money* to the *balance*. This method should return a string that describes the deposit transaction, i.e., *"Depositing $money"*.
   2. *String withdraw(int money)* to subtract *money* from the *balance*. This method should return a string that describes the withdraw transaction, i.e., *"Withdrawing $money"*. Note that, if there is insufficient balance to successfully withdraw the desired amount, then the *balance* should not be adjusted, and the returned string should be *"Withdrawing $money (Insufficient Balance)"*.
   3. *int getBalance()* to return the account balance.
2. The Transaction class has two data members *Account account* and *List transactions*. The class should implement the following three methods:
   1. *void deposit(int money)* to invoke the deposit method in the *Account* class. This should add the transaction message to the transactions list.
   2. *void withdraw(int money)* to invoke the withdraw method in the *Account* class. This should add the transaction message to the transactions list.
   3. *List getTransaction()* to return the transactions.

**Evaluation**  
The locked stub code in the editor validates the correctness of the Account and Transaction class implementations by making deposit and withdrawal transactions using threads. The locked stub code prints each transaction followed by the account balance. The output of the execution is non-deterministic, so the checker performs each of the transactions in the provided order. If all the transactions are executed correctly given a starting balance of $0, then the checker considers such transactions valid. For example, the following list of transactions is valid:

Depositing $59  
Withdrawing $2  
Depositing $62  
Depositing $16

But the following list of transactions is not:

Withdrawing $59  
Withdrawing $2  
Depositing $62  
Depositing $16

The first two withdrawals should not have occurred. Rather, they should have returned:

Withdrawing $59 (Insufficient Balance)  
Withdrawing $2 (Insufficient Balance)

**Constraints**  
1 ≤ threadsCount ≤ 10  
1 ≤ money ≤ 100  
Each thread makes no more than 104 transactions.

**Input Format For Custom Testing**  
The first line contains the value of threadsCount describing the total number of threads.

Each of the next threadsCount lines contains an integer transactionsCount, the total number of transactions performed by each of the threads.

**Sample Case**  
Sample Input

2  
3  
2

Sample Output

Depositing $59  
Withdrawing $2  
Depositing $62  
Depositing $16  
Withdrawing $52  
Balance $83

Explanation  
Note that this execution is never deterministic. Each execution with the same input could generate different output. For example, the following is also a valid output:

Withdrawing $67 (Insufficient Balance)  
Depositing $35  
Depositing $7  
Depositing $80  
Depositing $45  
Balance $167

Solution: <https://gist.github.com/andersonmo/887e91e8caa74877ff7c3a3e2f4c573e>

|  |
| --- |
| import java.security.SecureRandom; |
|  | import java.util.List; |
|  | import java.util.Scanner; |
|  | //---- My code ---- |
|  | import java.util.ArrayList; |
|  | /\* |
|  | \* Create the Account and Transaction classes here. |
|  | \*/ |
|  | class Account { |
|  | int balance=0; |
|  |  |
|  | public String deposit(int money){ |
|  | balance += money; |
|  | return "Depositing $"+money; |
|  | } |
|  |  |
|  | public String withdraw(int money){ |
|  | if(balance<money){ |
|  | return "Withdraw $"+money+" (Insufficient Balance)"; |
|  | } else{ |
|  | balance -= money; |
|  | return "Withdraw $"+money; |
|  | } |
|  | } |
|  |  |
|  | public int getBalance(){ |
|  | return balance; |
|  | } |
|  | } |
|  |  |
|  | class Transaction { |
|  | Account account = new Account(); |
|  | List<String> transactions = new ArrayList<>(); |
|  |  |
|  | public Transaction(Account account){ |
|  | this.account = account; |
|  | } |
|  |  |
|  | public void deposit(int money){ |
|  | transactions.add(account.deposit(money)); |
|  | } |
|  |  |
|  | public void withdraw(int money){ |
|  | transactions.add(account.withdraw(money)); |
|  | } |
|  |  |
|  | public List<String> getTransaction(){ |
|  | return transactions; |
|  | } |
|  | } |
|  | //---- End my code ---- |
|  |  |
|  | class TransactionRunnable implements Runnable { |
|  | private static final SecureRandom RANDOM\_GENERATOR = new SecureRandom(); |
|  | private final Transaction transaction; |
|  | private final int transactionsCount; |
|  |  |
|  | public TransactionRunnable(Transaction transaction, int transactionsCount) { |
|  | this.transaction = transaction; |
|  | this.transactionsCount = transactionsCount; |
|  | } |
|  |  |
|  | @Override |
|  | public void run() { |
|  | for (int i = 0; i < this.transactionsCount; i++) { |
|  | int transactionType = RANDOM\_GENERATOR.nextInt() % 2; |
|  | int money = RANDOM\_GENERATOR.nextInt(100) + 1; |
|  |  |
|  | if (transactionType == 0) { |
|  | this.transaction.deposit(money); |
|  | } else { |
|  | this.transaction.withdraw(money); |
|  | } |
|  | } |
|  | } |
|  | } |
|  |  |
|  | public class Solution { |
|  | private static final Scanner SCANNER = new Scanner(System.in); |
|  | private static final Account ACCOUNT = new Account(); |
|  | private static final Transaction TRANSACTION = new Transaction(ACCOUNT); |
|  |  |
|  | public static void main(String[] args) throws InterruptedException { |
|  | int threadsCount = Integer.parseInt(SCANNER.nextLine()); |
|  | Thread[] threads = new Thread[threadsCount]; |
|  |  |
|  | int expectedTransactionsCount = 0; |
|  | for (int i = 0; i < threadsCount; i++) { |
|  | int transactionsCount = Integer.parseInt(SCANNER.nextLine()); |
|  | expectedTransactionsCount += transactionsCount; |
|  |  |
|  | threads[i] = new Thread(new TransactionRunnable(TRANSACTION, transactionsCount)); |
|  | } |
|  |  |
|  | for (int i = 0; i < threadsCount; i++) { |
|  | threads[i].start(); |
|  | } |
|  |  |
|  | for (int i = 0; i < threadsCount; i++) { |
|  | threads[i].join(); |
|  | } |
|  |  |
|  | List<String> transactions = TRANSACTION.getTransaction(); |
|  |  |
|  | if (transactions.size() != expectedTransactionsCount) { |
|  | System.out.println("Wrong Answer"); |
|  | } else { |
|  | boolean correct = true; |
|  | for (String transaction: transactions) { |
|  | if (transaction == null) { |
|  | correct = false; |
|  |  |
|  | break; |
|  | } |
|  | } |
|  |  |
|  | if (!correct) { |
|  | System.out.println("Wrong Answer"); |
|  | } else { |
|  | int balance = ACCOUNT.getBalance(); |
|  |  |
|  | if (balance < 0) { |
|  | System.out.println("Wrong Answer"); |
|  | } else { |
|  | for (String transaction: transactions) { |
|  | System.out.println(transaction); |
|  | } |
|  |  |
|  | System.out.println("Balance $" + balance); |
|  | } |
|  | } |
|  | } |
|  | } |
|  | } |