Goal: In this problem, we will learn how to use Threads, Exception Handling and inter-thread communication.

Problem Description:

In this problem, you will be dealing with three types of threads. Please read the following descriptions carefully.

- Thread1 (***AccontGenerationThread***): This thread will be responsible for creating new accounts for a bank. AccountGenerationThread will create a total of 30 accounts. Each account is associated with three pieces of information - *AccountHolderName* (a string), *AccountNumber* (a 12-digit string containing two small English letters at prefix and the remaining 10 digits), and a *MaximumTransactionLimit* (a random integer between 1-1000 BDT) for the account holder. *AccountGenerationThread* should randomly create such accounts. After creating each account, it would sleep for 1000 milliseconds before initiating the creation of another new account.
- Thread2 (***DepositGenerationThread***): This thread will be responsible for creating transactions associated with deposits. It will randomly create 500 transactions, wherein for each transaction, you will randomly choose an *AccountNumber* from the generated accounts in the previous thread and generate *DepositAmount* (a random value between 1-50000 BDT) to put in a shared array *DepositTransactions* for processing. After each generation, this thread should sleep for 1000 milliseconds before another generation.
- Thread3 (***WithdrawGenerationThread***): This thread will be responsible for creating transactions associated with withdraw. It will randomly create 500 transactions, wherein for each transaction, you will randomly choose an *AccountNumber* from the generated accounts in the previous thread and generate *WithdrawAmount* (a random value between 1-100000 BDT) to put in a shared array *WithdrawTransactions* for processing. After each generation, this thread should sleep for 1000 milliseconds before another generation.
- Thread4 (***DepositProcessingThread1***) and Thread5 (***DepositProcessingThread2***): These two threads will take the transactions from the shared Array *DepositTransactions* and process them to update the corresponding account's *balance*. If the *DepositAmount* stored in the transaction violates the corresponding account's *MaximumTransactionLimit*, they should throw an exception as "*Maximum DepositTransaction Limit Violated*" and continue processing further transactions. After each processing, *DepositProcessingThread1* should wait for 1000 milliseconds and *DepositProcessingThread2 should wait for 800 milliseconds*. *DepositProcessingThread1* and *DepositProcessingThread2* can be considered worker threads both serving similar purposes and brought to make the processing faster with different processing limits.
- Thread6 (***WithdrawProcessingThread1***) and Thread7 (***WithdrawProcessingThread2***): These two threads will take the transactions from the shared Array *WithdrawTransactions* and process them to update the corresponding account's *balance*. If the *WithdrawAmount* stored in the transaction violates the corresponding account's *MaximumTransactionLimit*, they should throw an exception as "*Maximum WithdrawTransaction Limit Violated*" and continue processing further transactions. After each processing, *WithdrawProcessingThread1* should wait for 1000 milliseconds and

WithdrawProcessingThread2 should wait for 800 milliseconds. *WithdrawProcessingThread1* and *WithdrawProcessingThread2* can be considered worker threads both serving similar purposes and brought to make the processing faster with different processing limits.

Submission Format:

I would highly encourage you to write the codes using as many files as possible for the clarity of the coding. But, while submitting the solution, please compile all the code in one file, so that, I can run on my machine through a single command (Javac Roll_9.java, Java Roll_9). Also, please name the main class (Public class) as public class Roll_x, where x denotes your actual roll.

Penalty/Reward:

Apart from the quality of the solutions, other important points would carry some numerical penalties,

- For any sort of unexpected plagiarism, you will be heavily penalized
- The assignment wants to evaluate your learning over the thread, exception handling, and inter-thread communication. So, please integrate those principles here.
- Late submissions or submissions not maintaining the proper submission format will incur a penalty.
- You are encouraged to make the number of worker threads randomized between 2-5.