

## CA670 Concurrent Programming

Name	Salman Khaleel Sab
Student Number	18210266
Programme	MCM (Cloud)
Module Code	CA670
Assignment Title	Assignment One - Java Threads
Submission date	19-March-2019
Module coordinator	David Sinclair

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offenses in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged, and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the referencing guidelines found recommended in the assignment guidelines.

Name : Salman Khaleel Sab  
Date : 19-March-2019

# Assignment One - Java Threads

## Problem Statement:

We need to develop a multi-thread based Java program that should be capable of executing on multiple cores for the below scenario.

A simple Banking application developed on Java which allows ten different departments to perform the following operations on its fifty internal accounts.

- Deposit an amount into a named account.
- Withdraw an amount from a named account.
- Transfer an amount between named accounts.

The application must be capable of handling concurrent transfer between different pairs of account when possible. The application **must** be:

- correct,
- fair,
- have no deadlock, and
- not have any individual thread "starved".

Application should be able to handle at least 10,000 transactions efficiently.

## Design of the application:

I developed a multi-threading Java application which is capable of running on multiple core system. This application is capable of handling more than 10,000 concurrent transactions between it's fifty internal accounts of ten departments. It is also possible to increase the number of departments/accounts.

## Technologies Used:

1. Java for Multithreading
2. MySQL for Data storage.

## MySQL Tables:

1. Accounts table: Fifty internal accounts information is stored in this table with two columns Account number and Balance.
2. Departments table: Ten departments information is stored in this table with two columns DepartmentId and DepartmentName

SQL structure file is submitted along with this report in a file named "queries.sql".

Java application consists of four separate packages and seven different classes. Below table Represents the structure of the application.

Packages	Class name
assignmentone.databaseconnection	DBConnection.java
assignmentone.databasefunctions	Account.java
	AccountGetter.java
	DepartmentGetter.java
assignmentone.main	AssignmentOneMain.java

assignmentone.transactions	BankTransactions.java
	Get_Dep_Acc_TransactionType

A brief description of each class is given below:

1. DBConnection: This class is written explicitly to establish the connection from Java application to MySQL using jdbc drivers. Here a static method named “getConnection()” is written which creates connection to MySQL and returns a Connection object. Method is made static because it can be called without creating the instance of the class where it resides.
2. Account: This class will initialize 51 accounts
3. AccountGetter: This class will fetch fifty account numbers from Accounts table and return a ArrayList consisting of only account numbers.
4. DepartmentGetter: This class will fetch ten department Ids from Department table and return a ArrayList consisting of only department Ids. This class is also creates tasks based on the inputs sent via constructor. It implements Runnable interface.
5. AssignmentOneMain: This is the main class of the application which gets the number of cores of running system. It gets random Department, two random Account Numbers, random Transaction type(Deposit,Withdraw,Transfer), random Amount. All of these will be passed to DepartmentGetter constructor along with the list of accounts.
6. BankTransactions: This class is implements the following methods: deposit(), withdraw(), transfer(), getBalance(), setBalance().
7. Get\_Dep\_Acc\_TransactionType: This class takes the input from AccountGetter, DepartmentGetter classes and returns randomly selected department id, accountnumber, amount and transaction type (Deposit,withdraw,transfer).

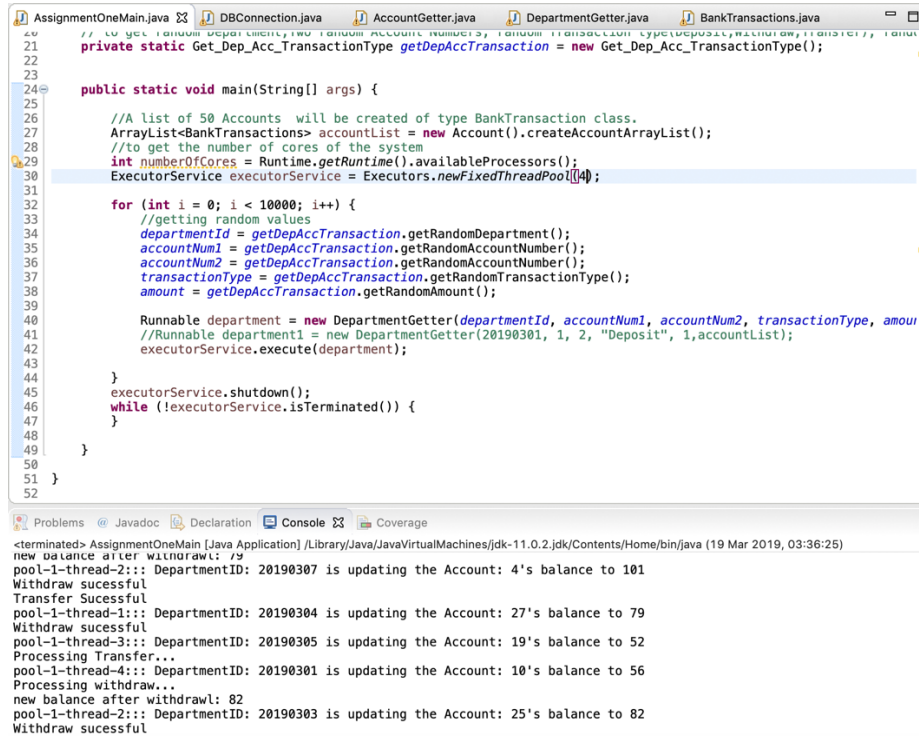
### Application design highlights [1] [2] [3] [4]:

1. **Random Input:** I used Random class of Util package to generate random inputs. Department Id, Account numbers, Amount and Transaction types are randomly selected for each iteration.
2. **Persistent Data:** MySQL is used to store the data.
3. **Absence of Deadlock and Thread starvation:** Synchronized methods are used to prevent deadlocks. At any point of time only one thread can access a synchronized method. After each transaction is performed, balance is updated in the database and the updated result will be visible to all other threads.
4. **Correctness:** Input is validated before performing any transactions. One such example is if we need to perform a transfer between to accounts, we need two account numbers. In this application as explained above account numbers are generated randomly, in a worst case scenario we might get same account numbers. Two account numbers are validated first to see if they are same. If so, account number will be regenerated until both are different. Many other screenshots which confirms the correctness of the application are submitted along with this report.

### Test Cases:

1. Number of cores: 4, Workload: 10,000 shown in the Image 1 - **PASS**.
2. Number of cores: 8, Workload: 15,000 shown in the Image 2- **PASS**.
3. Number of cores: 4, Workload: 10,000, Transaction type: Deposit screenshot submitted in the folder named “screenshots”- **PASS**.

4. Number of cores: 4, Workload: 10,000, Transaction type: Deposit, Withdraw and Transfer output of command line interface submitted in the folder named “output”- **PASS**.
5. Number of cores: 8, Workload: 10,000, Transaction type: Deposit output of command line interface submitted in the folder named “output”- **PASS**.



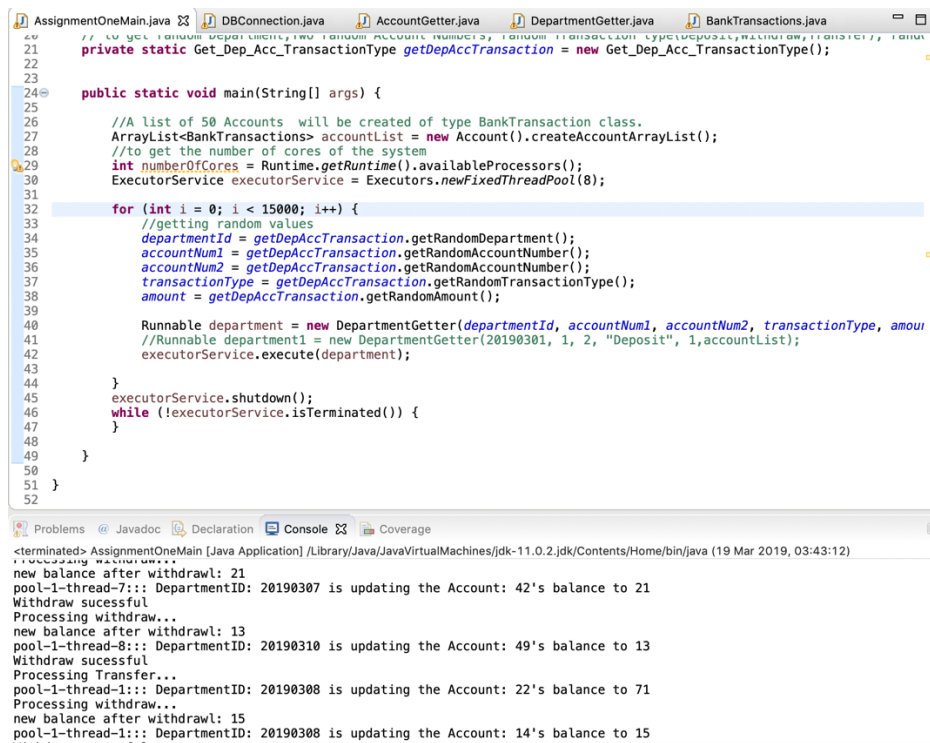
```
AssignmentOneMain.java DBConnection.java AccountGetter.java DepartmentGetter.java BankTransactions.java
20 // to get random department, two random account numbers, random transaction type(deposit, withdraw, transfer), random
21 private static Get_Dep_Acc_TransactionType getDepAccTransaction = new Get_Dep_Acc_TransactionType();
22
23
24 public static void main(String[] args) {
25
26     //A list of 50 Accounts will be created of type BankTransaction class.
27     ArrayList<BankTransactions> accountList = new Account().createAccountArrayList();
28     //to get the number of cores of the system
29     int numberOfCores = Runtime.getRuntime().availableProcessors();
30     ExecutorService executorService = Executors.newFixedThreadPool(4);
31
32     for (int i = 0; i < 10000; i++) {
33         //getting random values
34         departmentId = getDepAccTransaction.getRandomDepartment();
35         accountNum1 = getDepAccTransaction.getRandomAccountNumber();
36         accountNum2 = getDepAccTransaction.getRandomAccountNumber();
37         transactionType = getDepAccTransaction.getRandomTransactionType();
38         amount = getDepAccTransaction.getRandomAmount();
39
40         Runnable department = new DepartmentGetter(departmentId, accountNum1, accountNum2, transactionType, amount);
41         //Runnable department1 = new DepartmentGetter(20190301, 1, 2, "Deposit", 1, accountList);
42         executorService.execute(department);
43     }
44     executorService.shutdown();
45     while (!executorService.isTerminated()) {
46     }
47 }
48
49 }
50
51 }
52 }
```

Problems Javadoc Declaration Console Coverage

<terminated> AssignmentOneMain [Java Application] [Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java (19 Mar 2019, 03:36:25)]

new balance after withdrawl: 79  
pool-1-thread-2::: DepartmentID: 20190307 is updating the Account: 4's balance to 101  
Withdraw successful  
Transfer Successful  
pool-1-thread-1::: DepartmentID: 20190304 is updating the Account: 27's balance to 79  
Withdraw successful  
pool-1-thread-3::: DepartmentID: 20190305 is updating the Account: 19's balance to 52  
Processing Transfer...  
pool-1-thread-4::: DepartmentID: 20190301 is updating the Account: 10's balance to 56  
Processing withdrawl...  
new balance after withdrawl: 82  
pool-1-thread-2::: DepartmentID: 20190303 is updating the Account: 25's balance to 82  
Withdraw successful

Image 1



```
AssignmentOneMain.java DBConnection.java AccountGetter.java DepartmentGetter.java BankTransactions.java
20 // to get random department, two random account numbers, random transaction type(deposit, withdraw, transfer), random
21 private static Get_Dep_Acc_TransactionType getDepAccTransaction = new Get_Dep_Acc_TransactionType();
22
23
24 public static void main(String[] args) {
25
26     //A list of 50 Accounts will be created of type BankTransaction class.
27     ArrayList<BankTransactions> accountList = new Account().createAccountArrayList();
28     //to get the number of cores of the system
29     int numberOfCores = Runtime.getRuntime().availableProcessors();
30     ExecutorService executorService = Executors.newFixedThreadPool(8);
31
32     for (int i = 0; i < 15000; i++) {
33         //getting random values
34         departmentId = getDepAccTransaction.getRandomDepartment();
35         accountNum1 = getDepAccTransaction.getRandomAccountNumber();
36         accountNum2 = getDepAccTransaction.getRandomAccountNumber();
37         transactionType = getDepAccTransaction.getRandomTransactionType();
38         amount = getDepAccTransaction.getRandomAmount();
39
40         Runnable department = new DepartmentGetter(departmentId, accountNum1, accountNum2, transactionType, amount);
41         //Runnable department1 = new DepartmentGetter(20190301, 1, 2, "Deposit", 1, accountList);
42         executorService.execute(department);
43     }
44     executorService.shutdown();
45     while (!executorService.isTerminated()) {
46     }
47 }
48
49 }
50
51 }
52 }
```

Problems Javadoc Declaration Console Coverage

<terminated> AssignmentOneMain [Java Application] [Library/Java/JavaVirtualMachines/jdk-11.0.2.jdk/Contents/Home/bin/java (19 Mar 2019, 03:43:12)]

new balance after withdrawl: 21  
pool-1-thread-7::: DepartmentID: 20190307 is updating the Account: 42's balance to 21  
Withdraw successful  
Processing withdrawl...  
new balance after withdrawl: 13  
pool-1-thread-8::: DepartmentID: 20190310 is updating the Account: 49's balance to 13  
Withdraw successful  
Processing Transfer...  
pool-1-thread-1::: DepartmentID: 20190308 is updating the Account: 22's balance to 71  
Processing withdrawl...  
new balance after withdrawl: 15  
pool-1-thread-1::: DepartmentID: 20190308 is updating the Account: 14's balance to 15  
Withdraw successful

Image 2

## REFERENCES

- [1] "Concurrency-Oracle," [Online]. Available: <https://docs.oracle.com/javase/tutorial/essential/concurrency/>. [Accessed 18 03 2019].
- [2] D. Sinclair, "CA670-Java Threads," [Online]. Available: [https://www.computing.dcu.ie/~davids/courses/CA670/CA670\\_Java\\_Threads\\_2p.pdf](https://www.computing.dcu.ie/~davids/courses/CA670/CA670_Java_Threads_2p.pdf). [Accessed 17 03 2019].
- [3] "StackOverflow," [Online]. Available: <https://stackoverflow.com/questions/29364771/how-to-handle-multithreading-in-simple-cash-deposit-withdraw-program>. [Accessed 17 03 2019].
- [4] "Geeksforgeeks," [Online]. Available: <https://www.geeksforgeeks.org/multithreading-in-java/>. [Accessed 18 03 2019].