**Centennial College**

**COMP 228: Java Programming**

**Lab Assignment 6: Developing Multithreaded Applications using Java Multithreading API and Collections API.**

**Student:** Aryan Patel

Due Date: End of Week 13.

Purpose: The purpose of this Lab assignment is to:

1. Practice Multithreaded application development.
2. Develop a Multithreaded GUI Java application using Collection API.

References: Read the textbook, ppt slides, and consult references (if any).

This material provides the necessary information you need to complete the exercises.

Be sure to read the following general instructions carefully:

This lab should be completed individually by all the students.

YOU NEED TO SUBMIT THE FOLLOWING 2 DOCUMENTS IN THE DROPBOX TITLED LAB6:

1. THE FIRST ONE IS A WORD DOCUMENT. USE THIS DOCUMENT AND ADD SCREEN SHOTS OF THE RUNNING STATE OF EACH EXERCISE (If there are more than 1 exercise). DO NOT DELETE THE QUESTIONS. THE SCREEN SHOTS SHOULD FOLLOW EACH QUESTION AND COVER ALL THE ASPECTS/FUNCTIONALITIES OF EACH EXERCISE. AFTER THE SCREEN SHOTS PLEASE COPY THE CODE FROM THE CODE WINDOW AND PASTE THE COMPLETE CODE. DO NOT GIVE ME SCREEN SHOTS OF THE CODE. DO NOT ZIP THIS FILE AND KEEP IT SEPARATE FROM YOUR ZIPPED PROGAM FILE.

2. SUBMIT ALSO ONE ZIPPED PROJECT FILE THAT CONTAINS ALL THE EXERISES SEPARATELY INTO THE SAME DROP BOX.

This material provides the necessary information you need to complete the exercises.

You must name your Eclipse project according to the following rule:

**YourFullName\_COMP228Labnumber**

Example: **JohSmith\_COMP228Lab6**

Each exercise should be placed in a separate package named *exercise1*, *exercise2*, etc.

Submit your assignment in a **zip file** that is named according to the following rule:

**YourLastName\_COMP228Labnumber.zip**

Example: **JohSmith\_COMP228Lab6.zip**

Apply the naming conventions for variables, methods, classes, and packages:

- *variable names* start with a *lowercase* character

- *classes* start with an *uppercase* character

- **packages** use only *lowercase* characters

- *methods* start with a *lowercase* character

### **Exercise 1:**

This exercise is similar to PrintTask example from Week 12.

Write a Java application that handles multiple ATM transactions (withdraw, deposit) at the same time. Create an **Account** class and implement both **deposit** and **withdraw** operations. Synchronize the operations to allow thread synchronization. Use Java Runnable interface to implement a **Transaction** class. Perform **withdraw** and deposit **operations** in **run** method.

Create an **AccountTest** class to test multiple transactions (threads). Use an ArrayList to create a list of three or more Transaction objects. Use method **execute** of ExecutorService to execute the threads. Display the results.

Account Test class

package exercise1;

import java.util.ArrayList;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

public class AccountTest {

public static void main(String[] args) throws InterruptedException {

// Initial Balance in the Account

int initialBalanceAmount = 4500;

System.out.println("------------- Account Transactions---------------" + "\n");

System.out.println("Initial balance in the account:" + initialBalanceAmount + "\n");

// Account object

Account account = new Account(initialBalanceAmount);

// Transactions ArrayList

ArrayList<Transaction> listOfTransactions = new ArrayList<Transaction>();

// Transactions in account

Transaction firstTransaction = new Transaction(account, 500, true);

Transaction secondTransaction = new Transaction(account, 60, false);

Transaction thirdTransaction = new Transaction(account, 500, true);

// transactions added to the list

listOfTransactions.add(firstTransaction);

listOfTransactions.add(secondTransaction);

listOfTransactions.add(thirdTransaction);

// ExecutorService to manage threads

ExecutorService executorService = Executors.newCachedThreadPool();

// transaction loop

listOfTransactions.forEach((n) -> executorService.execute(n));

// shut down ExecutorService

executorService.shutdown();

try {

boolean transactionComplete = executorService.awaitTermination(40, TimeUnit.SECONDS);

if (transactionComplete) {

System.out.println("Final " + account.toString());

} else {

System.out.println("Time up");

}

} catch (InterruptedException exception) {

exception.printStackTrace();

}

}

}

Account class

**package** exercise1;

**public** **class** Account **extends** Thread {

// Balance in account

**private** **double** balanceAmount;

// Constructor

**public** Account(**double** initialDepositAmount) {

balanceAmount = initialDepositAmount;

}

**public** **synchronized** **double** getBalance() {

**return** balanceAmount;

}

// synchronized function to deposit amount in a/c

**public** **synchronized** **void** deposit(**double** amountdeposited) {

**try** {

Thread.*sleep*(1000);

balanceAmount += amountdeposited;

} **catch** (InterruptedException exception) {

exception.printStackTrace();

}

System.***out***.printf("%s deposited %.2f in the account.\n", Thread.*currentThread*().getName(), amountdeposited);

System.***out***.println(toString() + "\n");

}

// synchronized function to withdraw amount from a/c

**public** **synchronized** **void** withdraw(**double** amountwithdrawn) {

**try** {

Thread.*sleep*(1000);

**if** (balanceAmount >= amountwithdrawn) {

balanceAmount -= amountwithdrawn;

}

} **catch** (InterruptedException exception) {

exception.printStackTrace();

}

System.***out***.printf("%s withdrawn %.2f from the account.\n", Thread.*currentThread*().getName(), amountwithdrawn);

System.***out***.println(toString() + "\n");

}

@Override

**public** String toString() {

**return** "Account Balance is:" + balanceAmount;

}

}

Transection class

**public** **class** Transaction **implements** Runnable {

// field for transaction

**private** **final** Account account;

**private** **final** **int** amount;

**private** **boolean** deposit = **true**;

// constructor

**public** Transaction(Account account, **int** amount, **boolean** deposit) {

**super**();

**this**.account = account;

**this**.amount = amount;

**this**.deposit = deposit;

}

// method run contains the code that a thread will execute

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

**if** (deposit) {

account.deposit(amount);

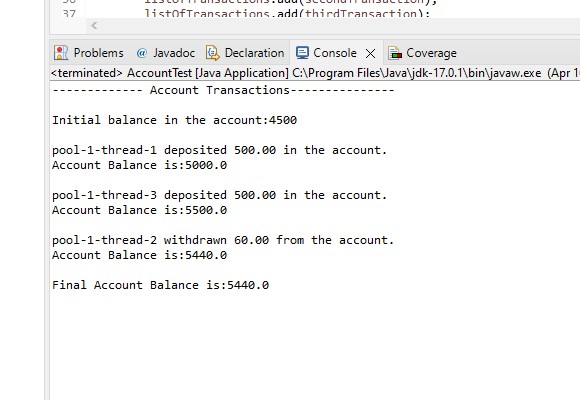
} **else**{

account.withdraw(amount);

}

}

}



(10 marks)

**Evaluation:**

|  |  |
| --- | --- |
| **Functionality** |  |
| Correct implementation of Multithreading | 50% |
| Correct implementation of Collections API | 30% |
| Comments, correct naming of variables, methods, classes, etc. | 5% |
| **Friendly input/output** | 15% |
| **Total** | 100% |