**Centennial College**

**COMP 228: Java Programming**

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

**Student:** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

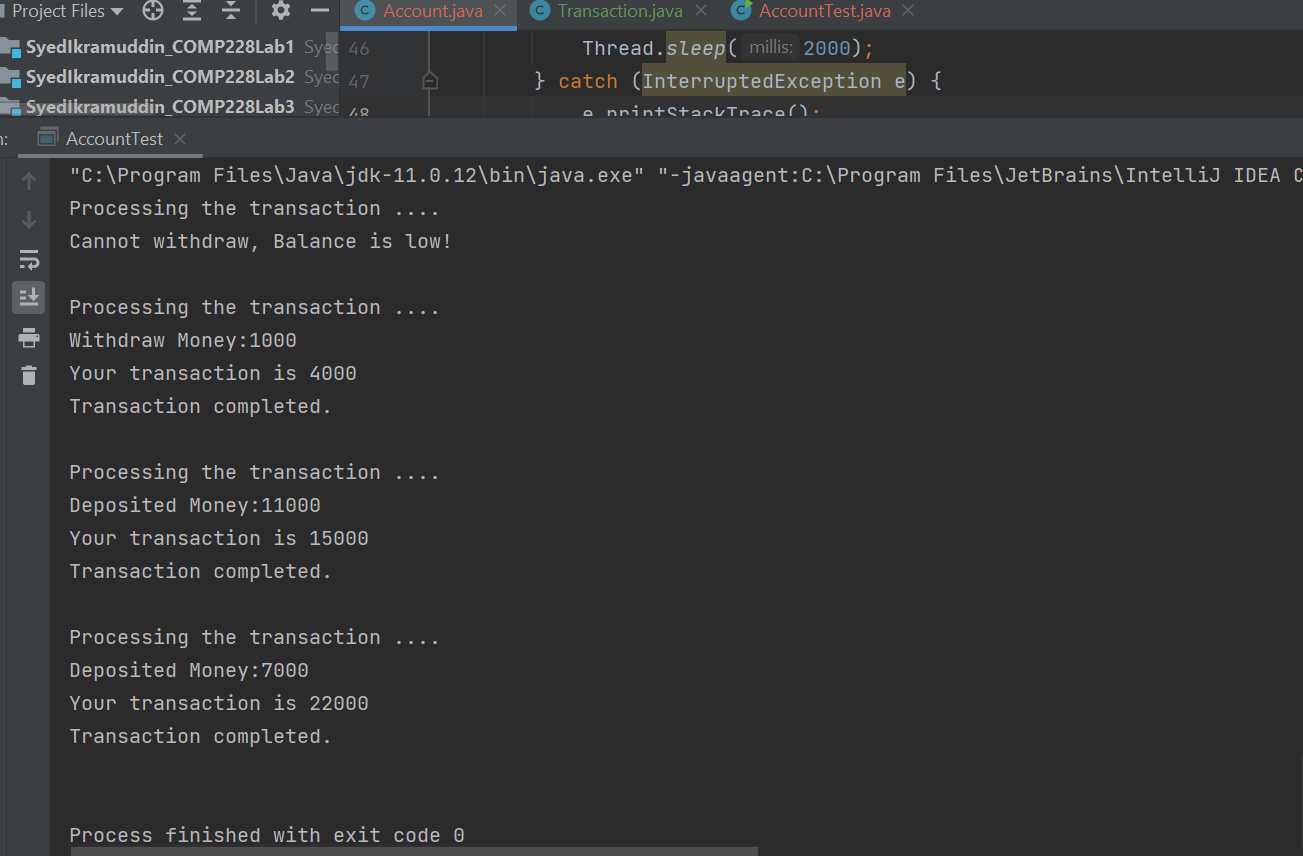
Create an **AccountTest** class to test multiple transactions (threads). Use an ArrayList to create a list of three or more Transaction objects. Use method **execute** of ExecutorService to execute the threads. Display the results.

(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% |

**Output:**



**Code:**

**Account.java**

package com.exercise1;

public class Account {

int balance;

String name;

final int accountNumber;

public Account(int accountNumber,String name, int initialBalance) {

this.accountNumber = accountNumber;

this.balance = initialBalance;

this.name = name;

}

public synchronized void deposit(int money){

System.out.println("Processing the transaction ....");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

this.balance += money;

System.out.println("Deposited Money:"+ money);

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Your transaction is "+balance);

System.out.println("Transaction completed.\n");

}

public synchronized void withdraw(int money){

System.out.println("Processing the transaction ....");

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

if(money > this.balance){

System.out.println("Cannot withdraw, Balance is low!\n");

return;

}

this.balance -= money;

System.out.println("Withdraw Money:"+money);

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Your transaction is "+balance);

System.out.println("Transaction completed.\n");

}

public int getBalance() {

return balance;

}

public void setBalance(int balance) {

this.balance = balance;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public int getAccountNumber() {

return accountNumber;

}

}

**Transaction.java**

package com.exercise1;

public class Transaction implements Runnable {

Account account;

String typeOfTransaction;

int money;

public Transaction(Account account, String typeOfTransaction, int money) {

this.account = account;

this.typeOfTransaction = typeOfTransaction;

this.money = money;

}

@Override

public void run() {

if(typeOfTransaction.equals("withdraw")){

this.account.withdraw(money);

}

else if (typeOfTransaction.equals("deposit")){

this.account.deposit(money);

}

else{

this.account.getBalance();

}

}

}

**AccountTest.java**

package com.exercise1;

import java.util.ArrayList;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

public class AccountTest {

public static void main(String[] args){

Account myAccount = new Account(11090717, "Syed",5000);

//Transactions

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

transactions.add(new Transaction(myAccount,"withdraw",10000));

transactions.add(new Transaction(myAccount,"deposit",7000));

transactions.add(new Transaction(myAccount,"withdraw",1000));

transactions.add(new Transaction(myAccount,"deposit",11000));

//Executor Service

ExecutorService executorService = Executors.newCachedThreadPool();

for(int i = 0; i < transactions.size(); i++){

executorService.execute(transactions.get(i));

}

executorService.shutdown();

}

}