**CS1083**

**Assignment #10**

**Daniyal Khan**

**3765942**

**Account.java:**  
  
import java.text.NumberFormat;

public class Account implements Comparable<Account>{

private String name;

private double balance;

private double interestRate;

private int term;

private final int ACC\_NUM;

private static int id = 8000;

public Account (String name, double initialBalance) {

this.name = name;

this.balance = initialBalance;

this.ACC\_NUM = id;

id++;

}

public void setInterestRate (double interestRate) throws PositiveInterestException {

if (interestRate < 0) {

throw new PositiveInterestException("Postive Interest Exception: Interest rate must be positive");

}

this.interestRate = interestRate;

}

public void applyInterest () {

if (this.term > 0) {

double interest = (this.interestRate/100) \* this.balance;

this.balance += interest;

}

}

public void withdraw(double amount) throws InsufficientFundsException {

double diff = this.balance - amount;

double penalty = 0.05 \* this.balance;

if (this.term > 0) { // penalty is only for accounts that are in a middle of a term

diff = this.balance - penalty - amount;

if (diff < 0) {

throw new InsufficientFundsException("InsufficientFundsException: Not enough funds to complete transaction");

}

this.balance = diff;

} else {

if (diff < 0) {

throw new InsufficientFundsException("InsufficientFundsException: Not enough funds to complete transaction");

}

this.balance = diff;

}

}

public void setTerm (int year) {

this.term = year;

}

public void reduceTerm() {

if (this.term - 1 >= 0) {

this.term--;

}

}

public double getBalance () {

return balance;

}

public void transferBalance(Account other, double amount) throws InsufficientFundsException {

double penalty = 0.025 \* amount;

// System.out.println(penalty);

if (this.balance < amount) {

throw new InsufficientFundsException("InsufficientFundsException: Not enough funds to complete transaction");

}

other.balance += amount;

this.balance = this.balance - amount- penalty;

}

public void closeAccTransfer (Account other) {

other.balance += this.balance;

}

public void deposit(double amount) {

this.balance += amount;

}

public int getAccountNum() {

return ACC\_NUM;

}

public String toString() {

NumberFormat format = NumberFormat.getCurrencyInstance();

String toReturn = "";

if (this.name.equals("CHEQUE")) {

toReturn = "ID: " + ACC\_NUM + "\t" +

name + " " + format.format(balance) + "\t";

} else {

toReturn = "ID: " + ACC\_NUM + "\t" +

name + " " + format.format(balance) + "\t" +

"Interest Rate: " + interestRate + "%" + "\t" +

"(Term: " + term + " Years)";

}

return toReturn;

}

public int compareTo (Account other) {

if (this.ACC\_NUM == other.ACC\_NUM) {

return 0;

} else if (this.ACC\_NUM > other.ACC\_NUM) {

return 1;

} else {

return -1;

}

}

public int compareTo (int acc\_num) {

if (this.ACC\_NUM == acc\_num) {

return 0;

} else if (this.ACC\_NUM > acc\_num) {

return 1;

} else {

return -1;

}

}

}

**LinkyList.java:**

import *java*.*util*.*ArrayList*;

import *java*.*util*.*List*;

*public* *class* LinkyList {

*public* AccountNode head;

*public* AccountNode tail;

*public* int size;

*public* *LinkyList* () {

head = null;

tail = null;

size = 0;

}

*public* Account *findNode* (int acc\_num) {

AccountNode current = head;

*while* (current != null && current.*data*.*getAccountNum*() != acc\_num) {

current = current.*next*;

}

*if* (current == null) {

*return* null;

}

*return* current.*data*;

}

*public* void *insertNode* (Account toAdd) {

AccountNode newNode = *new* *AccountNode*(toAdd);

size++;

*if* (head == null && tail == null) {

head = newNode;

tail = newNode;

} *else* *if* (head.*data*.*compareTo*(toAdd) > 0) {

newNode.*next* = head;

head.*prev* = newNode;

head = newNode;

} *else* *if* (tail.*data*.*compareTo*(toAdd) < 0) {

newNode.*prev* = tail;

tail.*next* = newNode;

tail = newNode;

} *else* {

AccountNode current = *new* *AccountNode*(toAdd);

*while* (current.*data*.*compareTo*(toAdd) > 0) {

current = current.*next*;

}

current.*prev*.*next* = newNode;

newNode.*prev* = current.*prev*;

newNode.*next* = current;

current.*prev* = newNode;

}

}

*public* boolean *removeNode* (int acc\_num) {

*if* (head == null && tail == null) {

*return* false;

}

*else* *if* (head.*data*.*compareTo*(acc\_num) == 0) {

head = head.*next*;

*if* (head == null) {

tail = null;

} *else* {

head.*prev* = null;

}

size--;

*return* true;

}

*else* *if* (tail.*data*.*compareTo*(acc\_num) == 0) {

tail = tail.*prev*;

*if* (tail == null) {

head = null;

} *else* {

tail.*next* = null;

}

size--;

*return* true;

}

*else* {

AccountNode current = head;

*while* (current.*data*.*compareTo*(acc\_num) < 0) {

current = current.*next*;

}

*if* (current == null) {

*return* false;

}

*else* *if* (current.*data*.*compareTo*(acc\_num) != 0) {

*return* false;

}

*else* {

current.*prev*.*next* = current.*next*;

current.*next*.*prev* = current.*prev*;

size--;

*return* true;

}

}

}

*public* String *toString*() {

AccountNode current = head;

String toReturn = "";

*while* (current != null) {

toReturn += current.*data*.*toString*() + "\n";

current = current.*next*;

}

*return* toReturn;

}

*public* ArrayList<Account> *getAllAccounts*() {

ArrayList<Account> accountList = *new* ArrayList<>();

AccountNode current = head;

*while* (current != null) {

accountList.*add*(current.*data*);

current = current.*next*;

}

*return* accountList;

}

*private* *class* AccountNode {

*public* Account data;

*public* AccountNode next;

*public* AccountNode prev;

*public* *AccountNode* (Account data) {

this.*data* = data;

this.*next* = null;

this.*prev* = null;

}

}

}

**AccountDriver.java:**

import *java*.*util*.*ArrayList*;

import *java*.*util*.*Scanner*;

import *java*.*io*.*\**;

*public* *class* AccountDriver {

*static* LinkyList accounts = *new* *LinkyList*();

*static* int requestsProcessed = 0;

*static* int lineNumber = 0;

*public* *static* void *main* (String args[]) {

*try* {

Scanner scan = *new* *Scanner*(*new* *File*("AccountData.txt"));

*while* (scan.*hasNextLine*()) {

String command = scan.*nextLine*();

String data = scan.*nextLine*();

String[] dataArr = data.*split*(",");

lineNumber += 2;

*if* (command.*equals*("OPEN")) {

requestsProcessed++;

*openAccount*(dataArr);

} *else* *if* (command.*equals*("CLOSE")) {

requestsProcessed++;

*closeAccount*(dataArr);

} *else* *if* (command.*equals*("DEPOSIT")) {

requestsProcessed++;

*depostAccount*(dataArr);

} *else* *if* (command.*equals*("WITHDRAW")) {

*try* {

requestsProcessed++;

*withdrawAccount*(dataArr);

} *catch* (InsufficientFundsException infe) {

System.*out*.*println*(infe.*getMessage*());

}

} *else* *if* (command.*equals*("TRANSFER")) {

*try* {

requestsProcessed++;

*transferAccount*(dataArr);

} *catch* (InsufficientFundsException ife) {

System.*out*.*println*(ife.*getMessage*());

}

} *else* *if* (command.*equals*("COMPLETE")) {

requestsProcessed++;

*completeAccount*(dataArr);

}

*if* (requestsProcessed%4 == 0) { // *printing the accounts every 4 requests*

System.*out*.*println*(accounts);

}

}

} *catch* (FileNotFoundException fnfe) {

System.*out*.*println*(fnfe.*getMessage*());

}

}

*public* *static* void *openAccount*(String dataArr[]) {

*try* {

*if* (dataArr.*length* == 2) {

Account acc = *new* *Account*(dataArr[0], Double.*parseDouble*(dataArr[1]));

accounts.*insertNode*(acc);

} *else* *if* (dataArr.*length* == 4) {

Account acc = *new* *Account*(dataArr[0], Double.*parseDouble*(dataArr[1]));

acc.*setInterestRate*(Double.*parseDouble*(dataArr[2]));

acc.*setTerm*(Integer.*parseInt*(dataArr[3]));

accounts.*insertNode*(acc);

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

} *catch* (PositiveInterestException pie) {

System.*out*.*println*(pie.*getMessage*());

}

}

*public* *static* void *closeAccount*(String dataArr[]) {

*if* (dataArr.*length* == 2) {

Account acc1 = accounts.*findNode*(Integer.*parseInt*(dataArr[0]));

Account acc2 = accounts.*findNode*(Integer.*parseInt*(dataArr[1]));

*if* (acc1 == null || acc2 == null) {

System.*out*.*println*("Account not found");

} *else* {

acc1.*closeAccTransfer*(acc2);

accounts.*removeNode*(Integer.*parseInt*(dataArr[0]));

}

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

}

*public* *static* void *depostAccount*(String dataArr[]) {

*if* (dataArr.*length* == 2) {

Account acc = accounts.*findNode*(Integer.*parseInt*(dataArr[0]));

*if* (acc == null) {

System.*out*.*println*("Account ID does note exist for request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

} *else* {

acc.*deposit*(Double.*parseDouble*(dataArr[1]));

}

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

}

*public* *static* void *withdrawAccount*(String dataArr[]) *throws* InsufficientFundsException {

*if* (dataArr.*length* == 2) {

Account acc = accounts.*findNode*(Integer.*parseInt*(dataArr[0]));

*if* (acc == null) {

System.*out*.*println*("Account ID does note exist for request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

} *else* {

acc.*withdraw*(Double.*parseDouble*(dataArr[1]));

}

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

}

*public* *static* void *transferAccount*(String dataArr[]) *throws* InsufficientFundsException {

*if* (dataArr.*length* == 3) {

Account acc1 = accounts.*findNode*(Integer.*parseInt*(dataArr[0]));

Account acc2 = accounts.*findNode*(Integer.*parseInt*(dataArr[1]));

*if* (acc1 == null || acc2 == null) {

System.*out*.*println*("Account ID does note exist for request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

} *else* {

acc1.*transferBalance*(acc2, Double.*parseDouble*(dataArr[2]));

}

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

}

*public* *static* void *completeAccount*(String dataArr[]) {

*if* (dataArr.*length* == 1) {

ArrayList<Account> accountsArr = accounts.*getAllAccounts*();

*for* (Account acc *:* accountsArr) {

acc.*applyInterest*();

acc.*reduceTerm*();

}

} *else* {

System.*out*.*println*("Error on reading in data from request " + requestsProcessed +

" (on lines: " + (lineNumber-1) + " or " + lineNumber + ")");

}

}

}

PositiveInterestException.java:

*public* *class* PositiveInterestException *extends* Exception {

*public* *PositiveInterestException* (String msg) {

super(msg);

}

}

InsufficientFundsException.java:

*public* *class* InsufficientFundsException *extends* Exception {

*public* *InsufficientFundsException* (String msg) {

super(msg);

}

}