CS 1120 – Computer Science II (with Java)

Instructor: <Bob Hardin>, Western Michigan University

Lab TA: <Nishant Gupta>

<Sam Stanchak, Blake Wrege>

Repository Name: pp4

SOFTWARE LIFE CYCLE REPORT – FOR LAB ASSIGNMENT <7>

**PHASE 1: SPECIFICATION (“What do we build?”)**

You are to write a program in Java to implement this Sunset Banking System. You are given the data of all checking accounts and all the transactions. You job is to process all the transactions and have them reflected in the balance of the accounts. For example if there is a deposit transaction to the account 100010 with the amount of 100 dollars, then the balance of account 100010 should be increased by 100. It works the other way around for withdraw transactions. You are given three input files.

**PHASE 2: DESIGN**

**Basic Structure:**

This program is split into seven classes:

1. Account - Scans the account array for information.

2. Deposit - Extends Transaction and stores the deposit data

3. FileInput - Reads the text files: withdraw.txt, deposit.txt, and accounts.txt then passes them to TransactionsList.

4. Test - Runs the program transactions and displays the result.

5. Transaction - Changes the balance and calls the time

6. TransactionList - Processes the Withdraws and/or Deposits to the account.

7. Withdraw - Extends Transaction and stores the Withdraw data

**Pseudocode:**

**2.**

Account:

-This class splits the line into account number, first name, last name, phone number, and balance using a scanner.

- Sets the return statements for the Test Class

Deposit:

-Adds deposits to the account balance

FileInput:

- Scans withdraw.txt into a string array for the TransactionList<withdraw>

- Scans deposit.txt into a string array for the TransactionList<deposit>

- Scans Account.txt into a string array for the Test class

Test:

-Uses time of day to determine if the sun has set.

- Displays the arrays of account numbers, first names, last names, and phone numbers

Transaction:

-Changes the balance depending on deposits and withdrawals to the account

TransactionList:

-Implements the process for the Account array

Withdraw:

- Subtracts the withdrawals from the account balance

**3.**

Account Class--

Fields: *accountNumber; holderFirstName; holderLastName; holderPhoneNumber; balance;*

Constructors: Initialize fields *accountNumber; holderFirstName; holderLastName;* holderPhoneNumber; balance;

Methods:

Method Account() This class splits the line of the text file into account number, first name, last name, phone number, and balance using a scanner.

Method get() Sets the fields from account into return statements for the Test Class

Deposit Class--

Fields: *accountNumber; amount; balanceChange;*

Constructors: Initialize fields *accountNumber; amount; balanceChange;*

Method Process() Gets the values deposited to the bank and adds them to the correct bank accounts.

FileInput Class--

Fields: *count; countArray; Line; account; accountArray*

Constructors: Initialize fields *count; countArray; Line; account; accountArray*

Methods:

Method TransactionList<withdraw>Scans withdraw.txt into a string array called list

Method TransactionList<deposit>Scans deposit.txt into a string array called list

Method Account[] Scans Account.txt line by line into a string array for the Test class

Test Class--

Fields: *date; sunset; accounts*

Constructors: Initialize fields *date; sunset; accounts*

Methods:

Method Sun() Uses time of day to determine if the sun has set.

Method Display() Displays the arrays of account numbers, first names, last names, and phone numbers

Method hasSet() Displays "waiting" if the sun has not set and displays the new balance if it has.

Transaction Class--

Fields: *accountNumber; amount; balanceChange;*

Constructors: Initialize fields *accountNumber; amount; balanceChange;*

Method Process() Changes the balance depending on deposits and withdrawals to the account

TransactionList Class--

Fields: *list; i*

Constructors: Initialize fields *list; i*

Methods

Method Process() Implements the process for the array of Accounts

Method Add() Adds each transaction to the linked list of transactions

Withdraw Class--

Fields: *accountNumber; amount; balanceChange;*

Constructors: Initialize fields *accountNumber; amount; balanceChange;*

Method Process() Gets the values withdrawn from the bank and subtracts them to from the correct bank accounts.

**PHASE 3: RISK ANALYSIS (“What can go wrong, and how bad can it be?”)**

There is no security on this program. If sunset banking was a real bank it would have serious issues maintaining the privacy of its clients.

**PHASE 4: VERIFICATION (“Are the algorithms correct?”)**

The program has one sequential logic path and it has been verified by us.

**PHASE 5: CODING**

**Account.java:**

package lab7;

import java.util.Scanner;

public class Account {

private int accountNumber;

private String holderFirstName;

private String holderLastName;

private String holderPhoneNumber;

private double balance;

/\*\*

\* This constructor splits the line into account number, first name, last name,

\* phone number, and balance using a scanner.

\* @param line A line of the file account.txt.

\*/

public Account(String line) {

Scanner scan = new Scanner(line);

accountNumber = scan.nextInt();

holderFirstName = scan.next();

holderLastName = scan.next();

holderPhoneNumber = scan.next();

balance = scan.nextDouble();

}

/\*\*

\* Get method which returns the account number.

\* @return Account Number

\*/

public int getAccountNumber() {

return accountNumber;

}

public String getHolderFirstName() {

return holderFirstName;

}

public String getHolderLastName() {

return holderLastName;

}

public String getHolderPhoneNumber() {

return holderPhoneNumber;

}

public void setBalance(double balance) {

this.balance = balance;

}

public double getBalance() {

return balance;

}

}

----------------------------------------------------------------------------------------------------------------------------

**Deposit.java:**

package lab7;

import java.util.Date;

import java.util.GregorianCalendar;

public class Deposit extends Transaction {

int accountNumber;

double amount;

double balanceChange;

GregorianCalendar calendar = new GregorianCalendar();

Date date = calendar.getTime();

public Deposit(int accountNumber, double amount) {

this.accountNumber = accountNumber;

this.amount = amount;

}

void process(Account[] accounts) {

for(int i = 0; i < accounts.length; i++) {

if(accounts[i].getAccountNumber() == this.accountNumber) {

double newBalance = accounts[i].getBalance() + this.amount;

accounts[i].setBalance(newBalance);

}

}

}

}

----------------------------------------------------------------------------------------------------------------------------

**FileInput.java:**

package lab7;

import java.io.FileReader;

import java.io.IOException;

import java.util.Scanner;

/\*\*

\* This class reads the files for accounts, withdraw, and deposit.

\* @author sam, blake

\*

\*/

public class FileInput {

/\*\*

\* This method reads the file with the account information.

\* @param fileName The file name with the account information.

\* @return Returns Account object array.

\* @throws IOException

\*/

public Account[] loadAccounts(String fileName) throws IOException {

int count = 0; // counter for first loop

int countArray = 0; // counter for second loop

String line; // string for nextLine

Account account; // account object

Account[] accountArray; // account object array

FileReader fileReader = new FileReader(fileName); // initialize FileReader for fileName

Scanner scan = new Scanner(fileReader);

// this loop finds out how many lines are in file and sets them to count

while (scan.hasNextLine()) {

scan.nextLine();

count++;

}

accountArray = new Account[count]; // initialize account object array with lines in file

fileReader.close(); // reset scanner and filereader

fileReader = new FileReader(fileName);

scan.close();

scan = new Scanner(fileReader);

// this loop scans nextLine for each line and sends it to account class and puts account object in account array

while (scan.hasNextLine()) {

line = scan.nextLine();

account = new Account(line); // account object accepts string

accountArray[countArray] = account; // puts account object in account array

countArray++;

}

scan.close(); // close scanners

return accountArray;

}

/\*\*

\* This method reads the withdraw file information.

\* @param fileName File name for withdraw file.

\* @return TransactionList of data type withdraw.

\* @throws IOException

\*/

public TransactionList<Withdraw> loadWithdrawTransactions(String fileName) throws IOException {

String line; // string for each line in file

String[] stringArray; // array to accept strings when splitting

int accountNumber;

double amount;

TransactionList<Withdraw> list = new TransactionList<Withdraw>(); // initialize transactionlist of type withdraw

FileReader fileReader = new FileReader(fileName); // initialize filereader

Scanner scan = new Scanner(fileReader);

// this loop reads the nextLine while there is one, splits it where there is whitespace, and parses the splits to accountNumber and amount

while (scan.hasNextLine()) {

line = scan.nextLine();

stringArray = line.split(" "); // split line by whitespace

accountNumber = Integer.parseInt(stringArray[0]);

amount = Double.parseDouble(stringArray[1]);

Withdraw withdraw = new Withdraw(accountNumber, amount); // initialize withdraw object

list.add(withdraw);

}

scan.close(); // close scanner

return list;

}

/\*\*

\* This method reads the deposit file information.

\* @param fileName File name for deposit file.

\* @return TransactionList of data type deposit.

\* @throws IOException

\*/

public TransactionList<Deposit> loadDepositTransaction(String fileName) throws IOException {

String line; // string for each line in file

String[] stringArray; // array to accept strings when splitting

int accountNumber;

double amount;

TransactionList<Deposit> list = new TransactionList<Deposit>(); // initialize transactionlist of type withdraw

FileReader fileReader = new FileReader(fileName); // initialize filereader

Scanner scan = new Scanner(fileReader);

// this loop reads the nextLine while there is one, splits it where there is whitespace, and parses the splits to accountNumber and amount

while (scan.hasNextLine()) {

line = scan.nextLine();

stringArray = line.split(" "); // split line by whitespace

accountNumber = Integer.parseInt(stringArray[0]);

amount = Double.parseDouble(stringArray[1]);

Deposit deposit = new Deposit(accountNumber, amount); // initialize deposit object

list.add(deposit);

}

scan.close(); // close scanner

return list;

}

}

----------------------------------------------------------------------------------------------------------------------------

**Test.java:**

**package** lab7;

**import** java.io.IOException;

**import** java.util.Date;

**import** java.util.GregorianCalendar;

**public** **class** Test {

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

GregorianCalendar calendar = **new** GregorianCalendar();

Date date = calendar.getTime();

**int** sunset = date.~~getHours~~();

FileInput fileReader = **new** FileInput();

Account[] accounts = fileReader.loadAccounts("account.txt");

TransactionList<Withdraw> withdraws = fileReader.loadWithdrawTransactions("withdraw.txt");

withdraws.process(accounts);

TransactionList<Deposit> deposits = fileReader.loadDepositTransaction("deposit.txt");

deposits.process(accounts);

**if** (sunset<20){

System.*out*.println("The sun sets at 8pm, it is before sunset: "+date);

}

**if** (sunset>19){

System.*out*.println("The sun sets at 8pm, it is after sunset: "+date);

}

**for** (**int** i = 0; i < accounts.length; i++) {

System.*out*.print(accounts[i].getAccountNumber() + " ");

System.*out*.print(accounts[i].getHolderFirstName() + " ");

System.*out*.print(accounts[i].getHolderLastName() + " ");

System.*out*.print(accounts[i].getHolderPhoneNumber() + " ");

System.*out*.print(accounts[i].getBalance());

System.*out*.println();

}

}

}

----------------------------------------------------------------------------------------------------------------------------

**Transaction.java:**

package lab7;

import java.util.Date;

import java.util.GregorianCalendar;

public class Transaction {

int accountNumber;

double amount;

double balanceChange;

GregorianCalendar calendar = new GregorianCalendar();

Date date = calendar.getTime();

void process(Account[] accounts) {}

}

----------------------------------------------------------------------------------------------------------------------------

**TransactionList.java:**

**package** lab7;

**import** java.util.ArrayList;

**public** **class** TransactionList<T **extends** Transaction> {

ArrayList<T> list = **new** ArrayList<T>();

**void** process(Account[] accounts) {

**int** i=0;

**for**(i=0;i<list.size();i++){

list.get(i).process(accounts);

}

}

**void** add(T transaction) {

list.add(transaction);

}

T get(**int** index) {

**return** list.get(index);

}

**int** size() {

**return** list.size();

}

}

----------------------------------------------------------------------------------------------------------------------------

**TransactionList.java:**

package lab7;

import java.util.Date;

import java.util.GregorianCalendar;

public class Withdraw extends Transaction {

int accountNumber;

double amount;

double balanceChange;

GregorianCalendar calendar = new GregorianCalendar();

Date date = calendar.getTime();

public Withdraw(int accountNumber, double amount) {

this.accountNumber = accountNumber;

this.amount = amount;

}

void process(Account[] accounts) {

for(int i = 0; i < accounts.length; i++) {

if(accounts[i].getAccountNumber() == this.accountNumber) {

double newBalance = accounts[i].getBalance() - this.amount;

accounts[i].setBalance(newBalance);

}

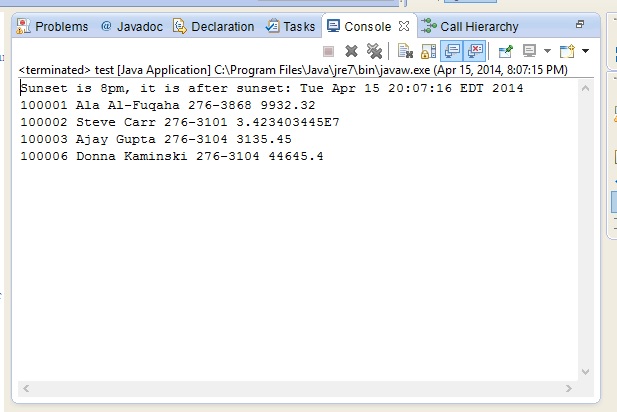
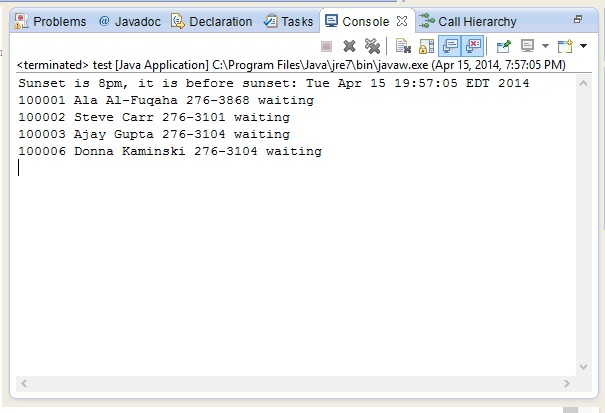
}

}

}

**PHASE 6: TESTING (“Did we build it correctly?”)**

**Example Output:**



**PHASE 7: REFINING THE PROGRAM (“Add bells and whistles to the program”)**

Features Added:

Added a feature where it tells the user whether the sun has set or not. It tells the user to wait if the sun has not set.

**PHASE 8: PRODUCTION**

We have prepared this SLC and the program itself for my TA for grading. Each team member’s individual contributions can be found in the readme file in our repository.

**PHASE 9: MAINTENANCE**

We shall use my TA’s feedback to perform maintenance if necessary.