Object Oriented Programming Lab - M2023

Assignment 8

Date: 02-11-2023

Problem 1

In commercial data processing, it's common to have several files in each application system. In an accounts receivable system, for example, there's generally a master file containing detailed information about each customer, such as the customer's name, address, telephone number, outstanding balance, credit limit, discount terms, contract arrangements and possibly a condensed history of recent purchases and cash payments. As transactions occur (i.e., sales are made and payments arrive in the mail), information about them is entered into a file. At the end of each business period (a month for some companies, a week for others, and a day in some cases), the file of transactions (called "trans.txt") is applied to the master file (called "oldmast.txt") to update each account's purchase and payment record. During an update, the master file is rewritten as the file "newmast.txt", which is then used at the end of the next business period to begin the updating process again.

File-matching programs must deal with certain problems that do not arise in single-file programs. For example, a match does not always occur. If a customer on the master file has not made any purchases or cash payments in the current business period, no record for this customer will appear on the transaction file. Similarly, a customer who did make some purchases or cash payments could have just moved to this community, and if so, the company may not have had a chance to create a master record for this customer. Write a complete file-matching accounts receivable program. Use the account number on each file as the record key for matching purposes. Assume that each file is a sequential text file with records stored in increasing account-number order.

a) Define class **TransactionRecord**. Objects of this class contain an account number and amount for the transaction. Provide methods to modify and retrieve these values.

b) Define class **AccountRecord** with members to account number, first name, last name and balance. Provide methods to set and retrieve these values and include another method combine, which takes a TransactionRecord object and combines the balance of the AccountRecord object and the amount value of the TransactionRecord object.

c) Write a program to create data for testing the program. Use the sample account data in below figure. Run the program to create the files trans.txt and oldmast.txt to be used by your file-matching program.

Sample data for master file

Master file account number	Name	Balance
100	Alan Jones	348.17
300	Mary Smith	27.19
500	Sam Sharp	0.00
700	Suzy Green	-14.22

Sample data for transaction file

Transaction file account number	Transaction amount
100	27.14
300	62.11
400	100.56
900	82.17

d) Create class FileMatch to perform the file-matching functionality. The class should contain methods that read oldmast.txt and trans.txt. When a match occurs (i.e., records with the same account number appear in both the master file and the transaction file), add the dollar amount in the transaction record to the current balance in the master record, and write the "newmast.txt" record. (Assume that purchases are indicated by positive amounts in the transaction file and payments by negative amounts.) When there's a master record for a particular account, but no corresponding transaction record, merely write the master record to "newmast.txt". When there's a transaction record, but no corresponding master record, print to a log file the message "Unmatched transaction record for account number…" (fill in the account number from the transaction record). The log file should be a text file named "log.txt".

Hint: Class FileOutputStream provides methods for writing byte arrays and individual bytes to a file, but we wish to write objects to a file. For this reason, we wrap a FileOutputStream in an ObjectOutputStream by passing the new FileOutputStream object to the ObjectOutputStream's constructor.

```
E.g.    ObjectOutputStream output;
        output = new ObjectOutputStream(new FileOutputStream( "trans.txt" ) );
        output.writeObject(record)      //to write the corresponding object into the file
```

Data must be read from the file in the same forma in which it was written. Therefore, we use an ObjectInputStream wrapped around a FileInputStream in this progr

```
E.g.    ObjectInputStream input;
        input = new ObjectInputStream(new FileInputStream( " trans.txt " ) );
        record = input.readObject()    // to read data from the file as object
```