**Java Programming**

**Final Project -Group E**

Arif Gencosmanoglu

Mingxuan Xie

Dharmaraj Paradkar

Bhuvanesh Arasu

Yan Zhu

Shivakumar K. P

**Case Study: A Credit-Inquiry Program**

**Introduction:**

The application allows a credit manager to obtain lists of customers with zero balances, customers with credit balances (customers whom has money to use) and customers with debit balances (customers who owes the company some money). Credit balances are stored in negative amounts, whereas debit balances are stored in positive amounts.

This application has two different options to retrieve the stored data;

First one is using the file system. Here the data will be stored in a text file with a specific format. Every time the application retrieves data, it will start from the beginning of the file and read it line by line until there is no more data to read. Therefore, each line will be written in a way that the program will understand which data represents what kind of variable.

The second way of retrieving data is reading the database that has a table that the data is stored. Because tables are formed by columns and rows, the application would easily find what it is looking by using a language named SQL. This language will let the application query the tables to retrieve the information needed.

**Explaining the program:**

// enum type for the credit-inquiry program's options.

public enum MenuOption

**{**

// declare contents of enum type

ZERO\_BALANCE**(**1**),**

CREDIT\_BALANCE**(**2**),**

DEBIT\_BALANCE**(**3**),**

END**(**4**);**

private final int value**;** // current menu option

// constructor

private MenuOption**(**int value**)**

**{**

**this.**value **=** value**;**

**}**

**}** // end enum MenuOption

The application uses the Enum (Enumeration) class to present the balance types. The enumeration class helps the code to be more readable. So instead of giving numbers to every type of balances, the types will be represented by their own names. Later on, when a programmer needs to change or add something new to the application, this programmer will be able to understand the process related to the balance types.

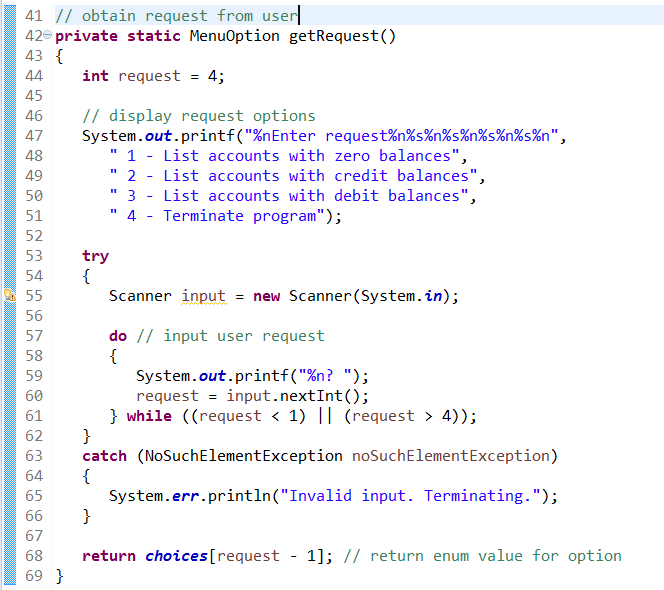


Firstly, import some packets like java.io.IOException, java.lang.IllegalStateException, java.nio.file.Paths, java.util.NoSuchElementException and java.util.Scanner for later use.

The class CreditInquiry start with creating a private final static MenuOption array choices and store the four type of MenuOption in the array.

In main method it first calls method getRequest to display the menu options, translates the number typed by the user into a MenuOption and stores the result in MenuOption variable accountType. While block loops until the user specifies that the program should terminate, and according the accountType ZERO\_BALANCE, CREDIT\_BALANCE and DEBIT\_BALANCE, switch statement displays a header for the current set of records to be output to the screen using statement “System.out.printf()”. Also, using break statement between each two cases. Finally, the block calls method readRecords using accountType as the variable, which loops through the file and reads every record. And then, assign the return value of the method getRequest to variable accountType.

The program declares and initializes an array of the type **MenuOption** with the name **“choices”**. The array is initialized by the calling the **values()** method of the **MenuOption** enum. The main method accepts a single argument: an array of elements of type String. This array is the mechanism through which the runtime system passes information to the application. The static keyword allows the main() to be called without having to instantiate a particular instance of the class. The keyword public is an access modifier which allows the member to be called from outside the class. **Void** indicates that **main()** does not return any value. The **values()** method is a built in method and all the constants of an enum type can be obtained by calling the implicit public static **values()** method of that type. The choices array is referenced many times later on within the program.



The programs execution begins from the main which is the entry point for the application and will subsequently invoke all the other methods required by the program. The method **getRequest()** is called in the first line in main method. The **getRequest()** method performs the function of obtaining a request from the user. Now, the user can request the list of all the accounts that fall in one of the following categories:

1. Accounts with 0 balance
2. Accounts with credit balance
3. Accounts with debit balance

The **getRequest()** method prompts the user to provide an input. The program continues to prompt for an input until a valid input option is provided i.e. an option in the range of 1 to 4. The program uses the java.utils.Scanner class for reading the inputs provided by the user. The code for reading the input is written in a try catch block which handle the exceptions raised by the user providing an input that is not integer. Once a valid input is received from the user the program generates the index value where the enum value for the requested option has to be fetched from the array “choices”. The fetched value is then retuned back to the caller. This returned value is then passed through a switch case that determines the type of account and then prints the same. The readRecord() is then called that reads all the user account entries from a text file. The file is read in a sequential manner. The above steps are recursively repeated until the user provides an input that corresponds to terminating the program.

**Method readRecords:**

// read records from file and display only records of appropriate type

private static void readRecords**(**MenuOption accountType**)**

**{**

// open file and process contents

**try** **(**Scanner input **=** **new** Scanner**(**Paths**.**get**(**"clients.txt"**)))**

**{**

**while** **(**input**.**hasNext**())** // more data to read

**{**

int accountNumber **=** input**.**nextInt**();**

String firstName **=** input**.**next**();**

String lastName **=** input**.**next**();**

double balance **=** input**.**nextDouble**();**

// if proper acount type, display record

**if** **(**shouldDisplay**(**accountType**,** balance**))**

System**.**out**.**printf**(**"%-10d%-12s%-12s%10.2f%n"**,** accountNumber**,**

firstName**,** lastName**,** balance**);**

**else** input**.**nextLine**();**

// discard the rest of the current record**}}**

**catch** **(**NoSuchElementException **|**

IllegalStateException **|** IOException e**)**

**{**System**.**err**.**println**(**"Error processing file. Terminating."**);**

System**.**exit**(**1**);}** **}** // end method readRecords

**Explanation of this method:**

Method readRecords uses a try-with-resources statement to create a Scanner that opens the file for reading. Also try-with-resources will close this file when the try block terminates successfully or due to an exception. The file will be opened for reading with a new Scanner object each time readRecords is called, so that we can again read from the beginning of the file.

In method readRequest, the statement in try block opens the file for reading by instantiating a Scanner object- input, here we uses Paths static method get to converts the sting”clients.txt” to a Path object, then we pass the Path object to the Scanner constructor, which specifies that the Scanner object will read from the file "clients.txt" (using a relative path) located in the directory from which the application executes. If the file cannot be found or cannot be opened correctly, an exception (NoSuchElementException or IllegalStateException or IOException) occurs.

In while block of method readRecords, if input.hasNext is true, that is, there is still data to read in the file, we read a record (current line), then it calls method shouldDisplay to determine whether the current record satisfies the account type requested. If shouldDisplay returns true, the program displays the account information. When the end-of-file marker is reached, the loop terminates and the try-with-resources statement closes the Scanner and the file. Once all the records have been read, control returns to main and getRequest is again called to retrieve the user’s next menu option.

**private static boolean** shouldDisplay(

MenuOption accountType, **double** balance)

{

**if** ((accountType == **MenuOption.CREDIT\_BALANCE**) && (balance < **0**))

**return true**;

**else if** ((accountType == **MenuOption.DEBIT\_BALANCE**) && (balance > **0**))

**return true**;

**else if** ((accountType == **MenuOption.ZERO\_BALANCE**) && (balance == **0**))

**return true**;

**return false**;

}

This part of the code is written to display the record by determining the record type. For this, we create a private method shouldDisplay which returns Boolean solution i.e., True to display the record or False to not display the record. This method takes two parameters namely accountType of type MenuOptions and a double floating point variable balance. In this method we have four conditions namely if the person is using a credit card and his balance is under zero the method returns True, if the person is using a debit card and his balance is greater than zero the method returns True and finally the methods returns true for querying zero balance. For any conditions other than this the method returns False. These conditions are implemented using the if and else if statements as shown in the program. For the first condition we use if statement and logical AND operator to check whether the query is for credit card and if so whether the corresponding balance is below zero , only if both the conditions are true , then the method returns True. Similarly, we check for the second and third conditions using else if statement and logical AND operator as shown in the program. Finally if the query is other than the above mentioned three conditions the method returns False. When this method is called it determines whether the current record satisfies the account type requested. If shouldDisplay returns true, the program displays the account information for the corresponding query record type as shown in the output.

The Output is shown and by looking at the output we can see that this program allows a credit manager to obtain lists of customers with *zero balances* i.e., customers who do not owe any money, customers with *credit balances* i.e., customers to whom the company owes money and customers with *debit balances* i.e., customers who owe the company money for goods and services received. A credit balance is a *negative* amount, a debit balance a *positive* amount. The output lists all the options and asks the user to enter the required query number an displays the result accordingly as shown.

**Database**

Database is used to store data as records in columns and rows. To retrieve, insert, update and delete specific language called SQL (Structured Query Language) is used. DBMS like Oracle, MySQL, SQL Server are used to perform SQL operations. This program uses MySQL as its DBMS. This program uses MySQL specific JDBC Driver(com.mysql.jdbc.Driver).

Once the program branches to DB part of this program JDBC Driver is loaded onto the memory. Database credentials such as URL, Username, Password and Table name are passed as arguments to DirverManager.getConnection(). The program then attempts to connect to the database with the provided credentials. Once the program connects to the Database the user can then select an option to query the database. To send the Query and receive ResultSet from DB statement Object is used. The execute statement is used to execute the query and fetch the ResultSet.. If the user wants to fetch information about Credit users SQL Statement ‘SELECT \* FROM BANKDATA WHERE ACCOUNT BALANCE >0” is sent. This query selects users with account balance greater than 0 and returns the tuple as ResultSet object which can then be broken down and stored in variables.

**Program**

private static void sqlCommand(String cmd)

try{

Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/db","root","");

ResultSet rs;

Statement smt = con.createStatement();

long lStartTime = System.nanoTime();

rs = smt.executeQuery(cmd);

while(rs.next()){

String id =rs.getString(1);

String name =rs.getString(2);

String type = rs.getString(3);

int balance = rs.getInt(4);

System.out.format(align,id,name,type,"$"+Integer.toString(balance));

}

}

catch(Exception e){}

}

private static void requestList(int option)

{

switch(option){

case 1:

sqlCommand("select \* from bankdata where account\_balance < 0 ");

break;

case 2:

sqlCommand("select \* from bankdata where account\_balance>0 ");

break;

case 3:

sqlCommand("select \* from bankdata where account\_balance = 0 ");

break;

case 4:

sqlCommand("select \* from bankdata");

break;

default:

System.out.println("Please enter the right option");

break;

}

}