Snapshot 1: Display of the statements and results

Table

Description automatically generated

Snapshot 2: Screenshot of the created table on the phpMyAdmin with the 600 records

Graphical user interface, text, application, email

Description automatically generated

Snapshot 3: (Extra Credit) Show the total time that it took between start and end

Text

Description automatically generated

**Source Code**

**BankRecords.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-ITMD 411

\* Purpose: To write a program that will read, process and store the

\* data in variables and the print it very nicely in columns back to the

\* user

\*/

//Import the packages that will be used throughout the program

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.Serializable;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

//Declare a new class called BankRecords that calls on the other class Client for the methods and implement Serializable

**public** **class** BankRecords **extends** Client **implements** Serializable{

//Declare all of the variables needed

**private** String id;

**private** **int** age;

**private** String sex;

**private** String region;

**private** **double** income;

**private** String married;

**private** **int** children;

**private** String car;

**private** String save\_act;

**private** String current\_act;

**private** String mortgage;

**private** String pep;

//Declare a new array to import all of the data and then create a new ArrayList

**static** BankRecords *robjs*[] = **new** BankRecords[600];

**static** ArrayList<List<String>> *array1* = **new** ArrayList<>();

//Define the readData method that will be used to read all of the data from bank-Detail

**public** **void** readData() {

//Create a new Buffered Reader to go through the file

BufferedReader br1;

//Declare a new try block to do if the file is defined and there

**try** {

//Look for a file called bank-Detail

br1 = **new** BufferedReader(**new** FileReader(**new** File("bank-Detail.csv")));

//Call on a new variable called line1

String line1;

//Keep reading the file and adding to a list until they get to the very end

**while** ((line1 = br1.readLine()) != **null**) {

*array1*.add(Arrays.*asList*(line1.split(",")));

}

//Close the buffered reader

br1.close();

processData();

}

//If the file isn't present print the exception to the user

**catch** (Exception e){

System.***out***.println("The following exception occured: " + e.getMessage());

}

}

//Define the next abstract method called processData to go through all of the data

**public** **void** processData() {

//Declare a new variable to keep track of the count

**int** cag=0;

//Add all of the data into an array and get the data from each row and add it to variables to display

**for** (List<String> rowData: *array1*) {

*robjs*[cag] = **new** BankRecords();

*robjs*[cag].setterId(rowData.get(0));

*robjs*[cag].setterAge(Integer.*parseInt*(rowData.get(1)));

*robjs*[cag].setterSex(rowData.get(2));

*robjs*[cag].setterRegion(rowData.get(3));

*robjs*[cag].setterIncome(Double.*parseDouble*(rowData.get(4)));

*robjs*[cag].setterMarried(rowData.get(5));

*robjs*[cag].setterChildren(Integer.*parseInt*(rowData.get(6)));

*robjs*[cag].setterCar(rowData.get(7));

*robjs*[cag].setterSave\_Act(rowData.get(8));

*robjs*[cag].setterCurrent\_Act(rowData.get(9));

*robjs*[cag].setterMortgage(rowData.get(10));

*robjs*[cag].setterPep(rowData.get(11));

cag++;

}

//Call the printData() method

//printData();

}

//Define the last abstract method to print all of the data for the user

**public** **void** printData() {

//Print the headings for each data field

System.***out***.println("ID\t\tAGE\t\tSEX\t\tREGION\t\tINCOME\t\tMORTGAGE");

//Keep going through and printing the data for the first 25 data fields

**for**(**int** i=0; i<25; i++) {

//Print all of the different fields

System.***out***.print(*robjs*[i].getterId()+"\t\t"+*robjs*[i].getterAge()+"\t\t"+*robjs*[i].getterSex()+"\t\t");

//Depending on the length of the region, print either one or two spaces after printing the region name

**if**(*robjs*[i].getterRegion().contentEquals("INNER\_CITY") || *robjs*[i].getterRegion().contentEquals("SUBURBAN")) {

System.***out***.print(*robjs*[i].getterRegion()+"\t");

}

**else** {

System.***out***.print(*robjs*[i].getterRegion()+"\t\t");

}

//Convert the int of Income to a String

String length = ""+ *robjs*[i].getterIncome();

//Depending on the length of the income, print either one or two spaces after the income

**if**(length.length() == 8) {

System.***out***.print(*robjs*[i].getterIncome()+"\t"+*robjs*[i].getterMortgage());

}

**else** {

System.***out***.print(*robjs*[i].getterIncome()+"\t\t"+*robjs*[i].getterMortgage());

}

//Print a new line after each so that all of the data is not on one line

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

}

}

//Declare all of the getter and setter methods that allow for the program to get all of the variables

**public** String getterId() {

**return** id;

}

**public** **void** setterId(String id) {

**this**.id = id;

}

**public** **int** getterAge() {

**return** age;

}

**public** **void** setterAge(**int** age) {

**this**.age = age;

}

**public** String getterSex() {

**return** sex;

}

**public** **void** setterSex(String sex) {

**this**.sex = sex;

}

**public** String getterRegion() {

**return** region;

}

**public** **void** setterRegion(String region) {

**this**.region = region;

}

**public** **double** getterIncome() {

**return** income;

}

**public** **void** setterIncome(**double** income) {

**this**.income = income;

}

**public** String getterMarried() {

**return** married;

}

**public** **void** setterMarried(String married) {

**this**.married = married;

}

**public** **int** getterChildren(){

**return** children;

}

**public** **void** setterChildren(**int** children) {

**this**.children = children;

}

**public** String getterCar() {

**return** car;

}

**public** **void** setterCar(String car) {

**this**.car = car;

}

**public** String getterSave\_Act() {

**return** save\_act;

}

**public** **void** setterSave\_Act(String save\_act) {

**this**.save\_act = save\_act;

}

**public** String getterCurrent\_Act() {

**return** current\_act;

}

**public** **void** setterCurrent\_Act(String current\_act) {

**this**.current\_act = current\_act;

}

**public** String getterMortgage() {

**return** mortgage;

}

**public** **void** setterMortgage(String mortgage) {

**this**.mortgage = mortgage;

}

**public** String getterPep() {

**return** pep;

}

**public** **void** setterPep(String pep) {

**this**.pep = pep;

}

//Call the main method that calls the BankRecords program

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

BankRecords brs = **new** BankRecords();

//Call the readData method to start the program

brs.readData();

}

}

**Client.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: To name the three methods that the bank needs to process

\* the names are: readData(), processData(), and printData()

\*/

//Declare a new abstract class called Client that can be extended to the other program

**public** **abstract** **class** Client{

//read all of the details of the file with all the information

**public** **abstract** **void** readData();

//process and store all of the information of each variable in arrays

**public** **abstract** **void** processData();

//print all of the data that was read and processed to the user

**public** **abstract** **void** printData();

}

**Dao.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: Write a program that creates a table, updates

\* and then writes a select statement that is used to select

\* all of the data to the table

\*/

//Import all of the necessary packages

**import** java.sql.SQLException;

**import** java.sql.Statement;

**import** java.sql.ResultSet;

//Make a new public class named Dao

**public** **class** Dao {

//Declare DB objects that will be used in the program

DBConnect conn = **null**;

Statement stmt = **null**;

//Create a new constructor called Dao

**public** Dao() {

//Create a new connectivity instance

conn = **new** DBConnect();

}

//Write a new method called createTable used to create a new table

**public** **void** createTable() {

**try** {

//Open a connection and tell the user that they connected successfully

System.***out***.println("Connecting to a selected database to create Table...");

System.***out***.println("Connected database successfully...");

//Execute the create query in SQL to create a table

System.***out***.println("Creating table in given database...");

stmt = conn.connect().createStatement();

//Write the SQL statement used to create the table

String sql = "CREATE TABLE C\_GARD\_tab1 "+

"(pid INTEGER not NULL AUTO\_INCREMENT, " +

" id VARCHAR(10), " +

" income numeric(8,2), " +

" pep VARCHAR(4), " +

"PRIMARY KEY (pid))";

//Execute the sql statement to update the changes

stmt.executeUpdate(sql);

//Tell the user that the table was created

System.***out***.println("Created table in given database...");

//Close the DB connection

conn.connect().close();

}

**catch** (SQLException se) {

//Handle errors for JDBC

se.printStackTrace();

}

}

//Write a new method used to insert all of the records into the table

**public** **void** insertRecords(BankRecords[] robjs) {

**try** {

//Let the user know that they connected successfully and are trying to insert

System.***out***.println("Connecting to a selected database to insert...");

System.***out***.println("Connected database successfully...");

//Start to create a new statement and declare the local variable

stmt = conn.connect().createStatement();

String sql = **null**;

//Tell the user that they are inserting the records into the table

System.***out***.println("Inserting records into the table...");

//Include all object data into the database table

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

//Insert id, income and pep into the database table with all of the values using an SQL statement

sql = "INSERT INTO C\_GARD\_tab1(id,income,pep) " + "VALUES (' "+robjs[i].getterId()+" ', ' "+robjs[i].getterIncome()+" ', ' "+robjs[i].getterPep()+" ' )";

//Execute the sql statements

stmt.executeUpdate(sql);

}

//Tell the user that the table was created and then close it out

System.***out***.println("Created table in given database...");

conn.connect().close();

}

//Catch if the try block does not work

**catch** (SQLException se) {

se.printStackTrace();

}

}

//Declare a new method called Result set that will retrieve the Records

**public** ResultSet retrieveRecords() {

//Declare a new local variable

ResultSet rs = **null**;

//Tell the user what they are doing

System.***out***.println("Connecting to a selected database for record retrievals...");

//Try to connect and create a statement and tell the user they successfully connected

**try** {

stmt = conn.connect().createStatement();

System.***out***.println("Connected to database successfully...");

}

//Throw an exception if the try block is not caught

**catch** (SQLException e) {

e.printStackTrace();

}

//Write a new sql statement for a select statement

String sql = "SELECT id,income,pep FROM C\_GARD\_tab1 ORDER BY pep DESC";

//Try to execute the SQL query and tell the user they are creating a select statement

**try** {

rs = stmt.executeQuery(sql);

System.***out***.println("Creating Select statement");

}

//Throw a catch if the try block does not work

**catch** (SQLException e) {

e.printStackTrace();

}

//Try to close the connection

**try** {

conn.connect().close();

}

//If it does not work throw an exception

**catch** (SQLException e) {

e.printStackTrace();

}

//Return the result set

**return** rs;

}

}

**DBConnect.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: To call the sql database and enter the credentials to connect

\* to the database connection

\*/

//import the necessary packages needed for the program

**import** java.sql.Connection;

**import** java.sql.DriverManager;

**import** java.sql.SQLException;

//Declare a new class called DB Connect

**public** **class** DBConnect {

//Add the URL for the database to be a string called DB\_URL

**static** **final** String ***DB\_URL*** = "jdbc:mysql://www.papademas.net:3307/411labs?autoReconnect=true&useSSL=false";

//Add all of the database credentials to login

**static** **final** String ***USER*** = "db411", ***PASS*** = "411";

//Declare a new connection called connect that throws SQLException

**public** Connection connect() **throws** SQLException {

//Return the new connection that is created

**return** DriverManager.*getConnection*(***DB\_URL***, ***USER***, ***PASS***);

}

}

**LoanProcessing.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: Write a program that will serialize and deserialize an object.

\* Additionally, it will print all of the data for the record set after reading data

\* It will also show the extra credit needed.

\*/

//Import all of the necessary packages that are needed for the program

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** java.util.HashMap;

**import** java.util.Map;

**import** java.util.Map.Entry;

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

//Create a new public class named LoanProcessing that extended to the BankRecords program

**public** **class** LoanProcessing **extends** BankRecords {

//Call the main method

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

//Call the bank records program and then read the data

BankRecords br = **new** BankRecords();

br.readData();

//Declare a new HashMap called bankMap

Map<Long, BankRecords> bankMap = **new** HashMap<Long, BankRecords>();

//Declare all of the local variables

**long** t = 0;

**long** start = 0, stop = 0;

//As long as the number is in robjs add it into the bankMap HashMap

**for** (BankRecords number : *robjs*) {

bankMap.put(++t, number);

}

//Extra Credit

//Serialize an object called bankrecords.ser

//Try to declare new variables to create the object

**try** {

FileOutputStream fileOut = **new** FileOutputStream("bankrecords.ser");

ObjectOutputStream objectOut = **new** ObjectOutputStream(fileOut);

//Start the timer for the initial starting time

start = System.*currentTimeMillis*();

//Write the object bankMap

objectOut.writeObject(bankMap);

//Both flush and close out of both objectOut and fileOut

objectOut.flush();

objectOut.close();

fileOut.flush();

fileOut.close();

}

//Throw an exception if it does not work

**catch** (IOException e) {

e.printStackTrace();

}

//Write a timer to sleep for 5 seconds

**try**{

Thread.*sleep*(5000);

}

//Throw a new exception if it does not work

**catch** (InterruptedException e1) {

e1.printStackTrace();

}

//Desearalize the object that was created earlier called bankrecords.ser

//Declare new variables needed for the deserialization

FileInputStream fileIn;

ObjectInputStream objectIn;

**int** count = 0;

//Try to create new objects and then read the object

**try** {

fileIn = **new** FileInputStream("bankrecords.ser");

objectIn = **new** ObjectInputStream(fileIn);

bankMap = (Map<Long, BankRecords>) objectIn.readObject();

//Find the current stop time when the process ended

stop = System.*currentTimeMillis*();

//Print the HashMap with the key and values

/\*for (Entry<Long, BankRecords> mapEntry : bankMap.entrySet()) {

Long key = (Long) mapEntry.getKey();

String value = (String) mapEntry.getValue().getterId();

System.out.println("Data=> " + "Key val: " + key + " id val: " + value);

//Add one to the count variable every time that the for loop goes through

count++;

}\*/

}

//Throw two different exceptions if the try block does not work

**catch** (IOException e) {

e.printStackTrace();

}

**catch** (ClassNotFoundException e) {

e.printStackTrace();

}

//Call the dao method to read it

Dao dao = **new** Dao();

//Create and insert records into the table created and then retrieve the records

dao.createTable();

dao.insertRecords(robjs);

ResultSet rs = dao.retrieveRecords();

//Create heading for display of all of the results

System.***out***.println("\nBANK-DETAIL REPORT:\n");

System.***out***.println(" ID\t Income\t Pep");

//Add the pep, id and income of all of the various datas after converting them

**try** {

**while** (rs.next()) {

String id = rs.getString("id").toUpperCase();

Double income = rs.getDouble("income");

String pep = rs.getString("pep");

System.***out***.printf("%7s %10.2f %7s\n", id, income, pep);

}

}

//Catch if the SQL does not work

**catch** (SQLException se) {

se.printStackTrace();

}

//Print the total time that it took between the start and stop

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

System.***out***.println("Total Time: " + (stop - start) + " Milliseconds");

//Try to close out of the result set

**try** {

rs.close();

}

//Catch if the closing does not work

**catch** (SQLException se) {

se.printStackTrace();

}

}

}

**LocationComparator.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: Write a program that will compare the strings of two different regions

\* and then return the result to the user

\*/

//Import the java.util.Comparator that will be used in the program

**import** java.util.Comparator;

//Declare a class called LocationComparator that will compare data of multiple locations when all of the other things are similar

**public** **class** LocationComparator **implements** Comparator<BankRecords>{

//Override the last result to start it fresh for the next

@Override

//Compare two records in the Bank Records Program

**public** **int** compare(BankRecords o1, BankRecords o2) {

//Compare the two strings and turn the result into an integer

**int** result = o1.getterRegion().compareTo(o2.getterRegion());

//Return the result of the comparison back to the BankRecords program that will give you the number

**return** result;

}

}

**Records.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: Write a program that will both display all the comparison results to the user

\* but also write the program in a FileWriter

\*/

//import the necessary packages that are needed for the program

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.text.DateFormat;

**import** java.text.SimpleDateFormat;

**import** java.util.Arrays;

**import** java.util.Date;

//Create a new public class called Records that will draw in all of the data from the BankRecords class

**public** **class** Records **extends** BankRecords {

//create formatted object to write output directly to console & file

**static** FileWriter *fw* = **null**;

//Declare a new class called Records that will try to write a new file called "bankrecords.txt"

**public** Records() {

**try** {

*fw* = **new** FileWriter("bankrecords.txt");

}

//If it cannot be done, throw an exception to the user

**catch** (IOException e5) {

e5.printStackTrace();

}

}

//Call the main method of the program

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

Records br = **new** Records();

//Call the readData function from BankRecords to allow for the data to be read which can then be used for comparisons

br.readData();

//Print the header for all of the results displayed

System.***out***.println("Data analytic results:");

//Try to write it also to the file writer and if it cannot be done then throw an exception

**try** {

*fw*.write("Data analytic results:");

}

**catch** (IOException e4) {

e4.printStackTrace();

}

// call functions to perform analytics

//AvgComp(); // analyze average income per loc

//femsComp(); // female count w. mort/savings accounts

//malesComp(); // male counts per loc. w. car & 1 child

// \*\*\* close out file object \*\*\*//

//Try the following block and if it does not work then throw an exception

**try** {

//Write name to file

*fw*.write("\n");

*fw*.write("\nCheryl Gardner ");

//Format the day for writing and then declare a new Date called day

DateFormat dateFormat = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

Date day = **new** Date();

//Write the new date to file which follows the format that is delcared above

*fw*.write(dateFormat.format(day) + "\n");

//Close out the file

*fw*.close();

} **catch** (IOException e3) {

e3.printStackTrace();

}

}

//Declare a new private method called AvgComp() to calculate the average income for both males and females

**private** **static** **void** AvgComp() {

//Sort all of the robjs data using the SexComparator class that will separate the data based on gender

Arrays.*sort*(*robjs*, **new** SexComparator());

//Declare all of the variables needed to keep track of the number of males and females as well as the Income for all of them

**int** maleCt = 0, femCt = 0;

**double** maleInc =0, femInc = 0;

//Keep going through the for loop as long as the value of i does not exceed the length of the array, adding one to i every time

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

//If the sex is female then add one to the female count and add that income to the total income

**if** (*robjs*[i].getterSex().equals("FEMALE")) {

++femCt;

femInc += *robjs*[i].getterIncome();

}

//If the sex is not female (Male) then add one to the male count and add that income to the total income

**else** {

++maleCt;

maleInc += *robjs*[i].getterIncome();

}

//Display the calculated averages to the console for the user to see

System.***out***.printf("\nAvg inc for Females: $%.2f", (femInc/femCt));

System.***out***.printf("\nAvg inc for Males: $%.2f", (maleInc/maleCt));

//Try to write the averages to the file and if it cannot be done then throw an exception

**try** {

*fw*.write("\n");

*fw*.write("\nAvg inc. for Females: $" + String.*format*("%.2f", (femInc/femCt)));

*fw*.write("\nAvg inc. for Males: $" + String.*format*("%.2f", (maleInc/maleCt)));

} **catch** (IOException e2) {

e2.printStackTrace();

}

}

//Declare a new method called femsComp() to find the number of females that have both a mortgage and savings acct

**private** **static** **void** femsComp() {

//Sort the array by sex so that all of the data is supported with the females being first

Arrays.*sort*(*robjs*, **new** SexComparator());

//Declare a new local variable to keep track of the number of females that have both accounts

**int** total = 0;

//Keep the for loop going as long as the value of i does not exceed the array length, adding one to i every time

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

//If that row is a female and has a mortgage and savings account do this

**if** (*robjs*[i].getterSex().contentEquals("FEMALE") && *robjs*[i].getterMortgage().contentEquals("YES") && *robjs*[i].getterSave\_Act().contentEquals("YES")) {

//Add one to the total count

total += 1;

}

//Make a new line in between the various data displayed and then display the number of females that meet the requirement

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

System.***out***.println("\nNum. of Females with Mortgage & savings acct: " + total);

//Try to write the data to the file and if it cannot be done then throw an exception

**try** {

*fw*.write("\n");

*fw*.write("\nNum. of Females with Mortgage & savings acct: " + total);

}

**catch** (IOException e1) {

e1.printStackTrace();

}

}

//Declare a new method called malesComp() that will compare the males that have a car and one child

**private** **static** **void** malesComp() {

//Sort all of the data based on the location to have all the data entries of one location next to each other

Arrays.*sort*(*robjs*, **new** LocationComparator());

//Declare the local variables needed to keep track of all the locations

**int** innercity = 0, rural = 0, suburban = 0, town = 0;

//Keep the for loop going as long as the value of i does not exceed the length of the array adding one every time

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

//If the person is a male has a car and one child do this program

**if**(*robjs*[i].getterSex().contentEquals("MALE") && *robjs*[i].getterCar().contentEquals("YES") && *robjs*[i].getterChildren() == 1) {

//If the location is innercity add one to the innercity count

**if**(*robjs*[i].getterRegion().contentEquals("INNER\_CITY")) {

innercity +=1;

}

//If the location is rural then add one to the rural count

**else** **if**(*robjs*[i].getterRegion().contentEquals("RURAL")) {

rural +=1;

}

//If the location is suburban add one to the suburban count

**else** **if**(*robjs*[i].getterRegion().contentEquals("SUBURBAN")) {

suburban +=1;

}

//If it is not any of these three then add one to the town count

**else**{

town +=1;

}

}

//Add a space between the input and display the number of males with a car and one child in each of the locations

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

System.***out***.println("Innercity region males with car & 1 child: " + innercity);

System.***out***.println("Rural region males with car & 1 child: " + rural);

System.***out***.println("Suburban region males with car & 1 child: " + suburban);

System.***out***.println("Town region males with car & 1 child: " + town);

//Try to write the data to the file and if it cannot be done then throw an exception

**try** {

*fw*.write("\n");

*fw*.write("\nInnercity region males with car & 1 child: " + innercity);

*fw*.write("\nRural region males with car & 1 child: " + rural);

*fw*.write("\nSuburban region males with car & 1 child: " + suburban);

*fw*.write("\nTown region males with car & 1 child: " + town);

}

**catch** (IOException e) {

e.printStackTrace();

}

}

}

**SexComparator.java**

/\*

\* Cheryl Gardner

\* 11/24/2020

\* Lab 04-411

\* Purpose: Write a program that will compare the strings of males and females

\* and then return the result to the user

\*/

//Import the java.util.Comparator package that will be used later in the program

**import** java.util.Comparator;

//Declare a new public class called SexComparator that will compare the data of two different sexes

**public** **class** SexComparator **implements** Comparator<BankRecords>{

//Override the last result to start the new comparator fresh

@Override

//Compare two different strings of o1 and o2 and return an integer

**public** **int** compare(BankRecords o1, BankRecords o2) {

// Use the compare to function to compare the two strings to get an integer result

**int** result = o1.getterSex().compareTo(o2.getterSex());

//Return the result to be pushed back into the main program

**return** result;

}

}