# BankRecords.java

package bankRecords;

import java.io.BufferedReader;

import java.io.FileNotFoundException;

import java.io.FileReader;

import java.io.IOException;

import java.text.DateFormat;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Date;

import java.util.List;

/\*\*

\* This program reads in data from bank-details.csv file and processes the data

\* then stores it in an array of objects

\* @author Chris

\*

\*/

public class BankRecords extends Client {

//variable for file name

static final String fileName = "bank-Detail.csv";

//variable for the character the csv is split by

static final String csvSplit = ",";

//Variable to count the number of lines

long numOfLines = 0;

//instantiate the array of objects

static BankRecords objects[];

//array list of a list of string

static ArrayList<List<String>> array = new ArrayList<>();

//instantiate all the variables

String id;

int age;

String sex;

String region;

Double income;

String married;

int children;

String car;

String save\_act;

String current\_act;

String mortgage;

String pep;

/\*\*

\* @return the id

\*/

public String getId() {

return id;

}

/\*\*

\* @param id the id to set

\*/

public void setId(String id) {

this.id = id;

}

/\*\*

\* @return the age

\*/

public int getAge() {

return age;

}

/\*\*

\* @param age the age to set

\*/

public void setAge(int age) {

this.age = age;

}

/\*\*

\* @return the sex

\*/

public String getSex() {

return sex;

}

/\*\*

\* @param sex the sex to set

\*/

public void setSex(String sex) {

this.sex = sex;

}

/\*\*

\* @return the region

\*/

public String getRegion() {

return region;

}

/\*\*

\* @param region the region to set

\*/

public void setRegion(String region) {

this.region = region;

}

/\*\*

\* @return the income

\*/

public Double getIncome() {

return income;

}

/\*\*

\* @param income the income to set

\*/

public void setIncome(Double income) {

this.income = income;

}

/\*\*

\* @return the married

\*/

public String getMarried() {

return married;

}

/\*\*

\* @param married the married to set

\*/

public void setMarried(String married) {

this.married = married;

}

/\*\*

\* @return the children

\*/

public int getChildren() {

return children;

}

/\*\*

\* @param children the children to set

\*/

public void setChildren(int children) {

this.children = children;

}

/\*\*

\* @return the car

\*/

public String getCar() {

return car;

}

/\*\*

\* @param car the car to set

\*/

public void setCar(String car) {

this.car = car;

}

/\*\*

\* @return the save\_act

\*/

public String getSave\_act() {

return save\_act;

}

/\*\*

\* @param save\_act the save\_act to set

\*/

public void setSave\_act(String save\_act) {

this.save\_act = save\_act;

}

/\*\*

\* @return the current\_act

\*/

public String getCurrent\_act() {

return current\_act;

}

/\*\*

\* @param current\_act the current\_act to set

\*/

public void setCurrent\_act(String current\_act) {

this.current\_act = current\_act;

}

/\*\*

\* @return the mortgage

\*/

public String getMortgage() {

return mortgage;

}

/\*\*

\* @param mortgage the mortgage to set

\*/

public void setMortgage(String mortgage) {

this.mortgage = mortgage;

}

/\*\*

\* @return the pep

\*/

public String getPep() {

return pep;

}

/\*\*

\* @param pep the pep to set

\*/

public void setPep(String pep) {

this.pep = pep;

}

/\*\*

\* @return the numOfLines

\*/

public long getNumOfLines() {

return numOfLines;

}

/\*\*

\* @param numOfLines the numOfLines to set

\*/

public void setNumOfLines(long numOfLines) {

this.numOfLines = numOfLines;

}

//prints my name at the end of the console out

public void exitPrintout(){

DateFormat tf = new SimpleDateFormat("dd/MM/yy HH:mm:ss");

Date date = new Date();

System.out.println("Current Date " + tf.format(date));

System.out.println("Created By Chris Doherty \n");

}

/\*\*

\* This method will read in all the record data from the csv file and put

\* into an ArrayList

\*/

public void readData(){

String line = null;

int numLines = 0;

try {

//Wraps FileReader in BufferedReader so we can deal with one line at a time.

BufferedReader fileInput = new BufferedReader(new FileReader(fileName));

//reads each line to get the number of lines present in the file

while((line = fileInput.readLine()) != null) {

//adds line into array list

array.add(Arrays.asList(line.split(csvSplit)));

numLines++;

}

//sets the number of lines in the file

setNumOfLines(numLines);

// Close file

fileInput.close();

}

catch(FileNotFoundException ex) {

System.out.println(

"Unable to open file '" +

fileName + "'");

}

catch(IOException ex) {

System.out.println(

"Error reading file '"

+ fileName + "'");

}

//calls process data method

processData();

}

/\*\*

\* This method all the record data from your ArrayList and add the data into each

\* of your instance fields via your setters

\*

\*/

public void processData() {

//casts number of lines to int and makes the array that length

int lines = (int) getNumOfLines();

objects = new BankRecords[lines];

int x = 0;

//loop to iterate through the arraylist and add to objects array

for(List<String> row: array) {

objects[x] = new BankRecords();

objects[x].setId(row.get(0));

objects[x].setAge(Integer.parseInt(row.get(1)));

objects[x].setSex(row.get(2));

objects[x].setRegion(row.get(3));

objects[x].setIncome(Double.parseDouble(row.get(4)));

objects[x].setMarried(row.get(5));

objects[x].setChildren(Integer.parseInt(row.get(6)));

objects[x].setCar(row.get(7));

objects[x].setSave\_act(row.get(8));

objects[x].setCurrent\_act(row.get(9));

objects[x].setMortgage(row.get(10));

objects[x].setPep(row.get(11));

x++;

}

}

/\*\*

\* This method should print the first 25 records for various fields

\* to the console via your getters

\* ID, AGE, SEX, REGION, INCOME, and MORTGAGE

\*/

public void printData() {

//makes the header for columns

System.out.println("ID\t\tAge\tSex\tRegion\t\tIncome\t\tMortgage");

//iterate through the object array and prints out the first 25

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

System.out.printf("%s\t\t" //ID

+ "%d\t" //Age

+ "%s\t" //Sex

+ "%-15s\t" //Region

+ "%-9s\t" //Income

+ "%s\n", //Mortgage

objects[i].getId(),objects[i].getAge(),objects[i].getSex(),

objects[i].getRegion(),objects[i].getIncome(),objects[i].getMortgage());

}

}

}

# Client.java

package bankRecords;

/\*\*

\* This program is for Lab 2 of ITMD 511

\* This abstract class allows for three abstract methods the bank needs to process.

\*

\*/

/\*\*

\* @author Chris

\*

\*/

public abstract class Client {

public void readData() {

}

public void processData(){

}

public void printData(){

}

}

# Records.java

package bankRecords;

import java.io.BufferedWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.io.OutputStreamWriter;

import java.text.DateFormat;

import java.text.DecimalFormat;

import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Comparator;

import java.util.Date;

/\*\*

\* @author Chris

\* @param Type <BankRecords>

\*

\* This program is the third lab for ITMD 510.

\* It is designed to perform comparisons on the data read in from bankrecords.txt

\* It extend the BankRecords class

\* It uses 3 comparator classes implementing java.util.Comparator to perform data analysis

\*

\*

\*/

public class Records<R> extends BankRecords{

//This creates File Writer object to output results to a file

static File fout = null;

static FileOutputStream fos = null;

//Wrapper to write to file

static BufferedWriter filewriter = null;

//Object to format output

static DecimalFormat df = new DecimalFormat("##.##");

//constructor

public Records(){

//Try catch block to make/open file

try {

fout = new File("bankrecords.txt");

fos = new FileOutputStream(fout);

filewriter = new BufferedWriter(new OutputStreamWriter(fos));

} catch (IOException e) {

e.printStackTrace();

}

}

//Main method to perform actions

public static void main(String[] args) {

//Creates br object with type BankRecord

Records<BankRecords> br = new Records<BankRecords>();

//Calls readData method form bankrecord.java file

br.readData();

//call LocatComp to analyze average income for each location

LocatCompair();

//call LocatComp to compare max/min incomes for each location

MinMaxCompair();

//call LocatComp to analyze females with mortgage and savings accounts for each location

FwMaSCompair();

//call LocatComp to analyze number of males with a car and 1 child for each location

MwCaCCompair();

//prints out my name and current date

br.exitPrintout();

//Writes my name and current date to the file bankrecords.txt

try {

DateFormat tf = new SimpleDateFormat("dd/MM/yy HH:mm:ss");

Date date = new Date();

filewriter.newLine();

filewriter.write("Current Date " + tf.format(date));

filewriter.newLine();

filewriter.write("Created By Chris Doherty \n");

} catch (IOException e) {

e.printStackTrace();

}

//close out the buffer reader wrapper object after everything has been written

try {

filewriter.close();

} catch (IOException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

}

/\*\*

\* This method is used to compare each region's average income

\*/

private static void LocatCompair() {

//Sorts the array of objects names objects for Location

Arrays.sort(objects, new LocCompar());

//initialize variables for region and income for location

double numOfTownPeople = 0, numOfInnerPeople = 0, numOfSuburbanPeople = 0, numOfRuralPeople = 0,

townIncTotal = 0, innerIncTotal=0, suburbanIncTotal=0, ruralIncTotal=0;

//increments the number of people per region and retrieves their averages

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

if (objects[i].getRegion().equals("RURAL")) {

ruralIncTotal += objects[i].getIncome();

++numOfRuralPeople;

}

else if (objects[i].getRegion().equals("INNER\_CITY"))

{

innerIncTotal += objects[i].getIncome();

++numOfInnerPeople;

}

else if (objects[i].getRegion().equals("SUBURBAN"))

{

suburbanIncTotal += objects[i].getIncome();

++numOfSuburbanPeople;

}

else if (objects[i].getRegion().equals("TOWN"))

{

townIncTotal += objects[i].getIncome();

++numOfTownPeople;

}

else {}

//setup resulting averages

double ruralAvg = ruralIncTotal/(numOfRuralPeople);

double innerAvg = innerIncTotal/(numOfInnerPeople);

double suburbanAvg = suburbanIncTotal/(numOfSuburbanPeople);

double townAvg = townIncTotal/(numOfTownPeople);

//formats the double averages to 2 decimals

String rA = df.format(ruralAvg);

String iA = df.format(innerAvg);

String sA = df.format(suburbanAvg);

String tA = df.format(townAvg);

//Output to the console

System.out.println("Avg income for Inner City region $" + iA);

System.out.println("Avg income for Rural region $" + rA);

System.out.println("Avg income for Suburban region $" + sA);

System.out.println("Avg income for Town region $" + tA);

System.out.println("\n");

//Writes the information to the file bankrecords.txt

try {

filewriter.write("Avg income for Rural region $" + rA);

filewriter.newLine();

filewriter.write("Avg income for Inner City region $" + iA);

filewriter.newLine();

filewriter.write("Avg income for Suburban region $" + sA);

filewriter.newLine();

filewriter.write("Avg income for Town region $" + tA);

filewriter.newLine();

filewriter.newLine();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* This method is used to analyze the data and return the maximum and minimum

\* incomes per location

\*/

private static void MinMaxCompair() {

//Sorts the array of objects names objects for Location

Arrays.sort(objects, new minMaxIncome());

/\*

Created an ArrayLists to add the ordered value

I will take first and last element's value for the minimum and maximum

this is possible because the compare method ordered the information

\*/

ArrayList<Double> ruralArray = new ArrayList<Double>();

ArrayList<Double> cityArray = new ArrayList<Double>();

ArrayList<Double> suburbanArray = new ArrayList<Double>();

ArrayList<Double> townArray = new ArrayList<Double>();

//loop to add the values to the correct array list

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

if (objects[i].getRegion().equals("RURAL")) {

ruralArray.add(objects[i].getIncome());

}

else if (objects[i].getRegion().equals("INNER\_CITY"))

{

cityArray.add(objects[i].getIncome());

}

else if (objects[i].getRegion().equals("SUBURBAN"))

{

suburbanArray.add(objects[i].getIncome());

}

else if (objects[i].getRegion().equals("TOWN"))

{

townArray.add(objects[i].getIncome());

}

else {}

//Output to the console

System.out.println("Inner City region Minimum income $" + cityArray.get(0));

System.out.println("Rural region Minimum income $" + ruralArray.get(0));

System.out.println("Surburban region Minimum income $" + suburbanArray.get(0));

System.out.println("Town region Minimum income $" + townArray.get(0));

System.out.println("\n");

System.out.println("Inner City region Max income $" + cityArray.get(cityArray.size()-1));

System.out.println("Rural region Max income $" + ruralArray.get(ruralArray.size()-1));

System.out.println("Surburban region Max income $" + suburbanArray.get(suburbanArray.size()-1));

System.out.println("Town region Max income $" + townArray.get(townArray.size()-1));

System.out.println("\n");

//Writes the information to the file bankrecords.txt

try {

filewriter.write("Minimum income for Rural region " + ruralArray.get(0));

filewriter.newLine();

filewriter.write("Minimum income for Inner City region " + cityArray.get(0));

filewriter.newLine();

filewriter.write("Minimum income for Surburban region " + suburbanArray.get(0));

filewriter.newLine();

filewriter.write("Minimum income for Town region " + townArray.get(0));

filewriter.newLine();

filewriter.newLine();

filewriter.write("Max income for Rural region " + ruralArray.get(ruralArray.size()-1));

filewriter.newLine();

filewriter.write("Max income for Inner City region " + cityArray.get(cityArray.size()-1));

filewriter.newLine();

filewriter.write("Max income for Surburban region " + suburbanArray.get(suburbanArray.size()-1));

filewriter.newLine();

filewriter.write("Max income for Town region " + townArray.get(townArray.size()-1));

filewriter.newLine();

filewriter.newLine();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* This method is used to analyze the data and return the

\* number of females with both a mortgage and savings account per location

\* (F females w with M mortgage a and S savings

\*/

private static void FwMaSCompair() {

//Sorts the array of objects names objects for Location not really needed

Arrays.sort(objects, new FwMaSLocation());

//initialize variables for region and income for location

int numOfTownPeople = 0, numOfInnerPeople = 0, numOfSuburbanPeople = 0, numOfRuralPeople = 0;

//loop to increment the number of people per location

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

if (objects[i].getRegion().equals("RURAL") && objects[i].getSex().equals("FEMALE")

&& objects[i].getSave\_act().equals("YES") && objects[i].getMortgage().equals("YES")) {

++numOfRuralPeople;

}

else if (objects[i].getRegion().equals("INNER\_CITY") && objects[i].getSex().equals("FEMALE")

&& objects[i].getSave\_act().equals("YES") && objects[i].getMortgage().equals("YES"))

{

++numOfInnerPeople;

}

else if (objects[i].getRegion().equals("SUBURBAN") && objects[i].getSex().equals("FEMALE")

&& objects[i].getSave\_act().equals("YES") && objects[i].getMortgage().equals("YES"))

{

++numOfSuburbanPeople;

}

else if (objects[i].getRegion().equals("TOWN") && objects[i].getSex().equals("FEMALE")

&& objects[i].getSave\_act().equals("YES") && objects[i].getMortgage().equals("YES"))

{

++numOfTownPeople;

}

else {}

//Output to the console

System.out.println("Inner City region Females with Mortgage and Savings Account " + numOfInnerPeople);

System.out.println("Rural region Females with Mortgage and Savings Account " + numOfRuralPeople);

System.out.println("Suburban region Females with Mortgage and Savings Account " + numOfSuburbanPeople);

System.out.println("Town region Females with Mortgage and Savings Account " + numOfTownPeople);

System.out.println("\n");

//Writes the information to the file bankrecords.txt

try {

filewriter.write("Rural region Females with Mortgage and Savings Account " + numOfRuralPeople);

filewriter.newLine();

filewriter.write("Inner City region Females with Mortgage and Savings Account " + numOfInnerPeople);

filewriter.newLine();

filewriter.write("Suburban region Females with Mortgage and Savings Account " + numOfSuburbanPeople);

filewriter.newLine();

filewriter.write("Town region Females with Mortgage and Savings Account " + numOfTownPeople);

filewriter.newLine();

filewriter.newLine();

} catch (IOException e) {

e.printStackTrace();

}

}

/\*\*

\* This method is used to analyze the data and return the

\* number of males with both a car and 1 child per location

\* (M males w with C cars and C children)

\*/

private static void MwCaCCompair() {

//initialize variables for region and income for location

int numOfTownPeople = 0, numOfInnerPeople = 0, numOfSuburbanPeople = 0, numOfRuralPeople = 0;

//loop to increment the number of people per location

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

if (objects[i].getRegion().equals("RURAL") && objects[i].getSex().equals("MALE")

&& objects[i].getChildren()==1 && objects[i].getCar().equals("YES")) {

++numOfRuralPeople;

}

else if (objects[i].getRegion().equals("INNER\_CITY") && objects[i].getSex().equals("MALE")

&& objects[i].getChildren()==1 && objects[i].getCar().equals("YES"))

{

++numOfInnerPeople;

}

else if (objects[i].getRegion().equals("SUBURBAN") && objects[i].getSex().equals("MALE")

&& objects[i].getChildren()==1 && objects[i].getCar().equals("YES"))

{

++numOfSuburbanPeople;

}

else if (objects[i].getRegion().equals("TOWN") && objects[i].getSex().equals("MALE")

&& objects[i].getChildren()==1 && objects[i].getCar().equals("YES"))

{

++numOfTownPeople;

}

else {}

//Output to the console

System.out.println("Inner City region Males with 1 Child and a Car " + numOfInnerPeople);

System.out.println("Rural region Males with 1 Child and a Car " + numOfRuralPeople);

System.out.println("Suburban region Males with 1 Child and a Car " + numOfSuburbanPeople);

System.out.println("Town region Males with 1 Child and a Car " + numOfTownPeople);

System.out.println("\n");

//Writes the information to the file bankrecords.txt

try {

filewriter.write("Rural region Males with 1 Child and a Car " + numOfRuralPeople);

filewriter.newLine();

filewriter.write("Inner City region Males with 1 Child and a Car " + numOfInnerPeople);

filewriter.newLine();

filewriter.write("Suburban region Males with 1 Child and a Car " + numOfSuburbanPeople);

filewriter.newLine();

filewriter.write("Town region Males with 1 Child and a Car " + numOfTownPeople);

filewriter.newLine();

} catch (IOException e) {

e.printStackTrace();

}

}

}

/\*\*

\* Sorts the regions

\*/

class LocCompar implements Comparator<BankRecords>{

@Override

public int compare(BankRecords x, BankRecords y) {

// TODO Auto-generated method stub

int result = x.getRegion().compareTo(y.getRegion());

return result;

}

}

/\*\*

\* Sorts the array to order from minimum to maximum

\*/

class minMaxIncome implements Comparator<BankRecords>{

@Override

public int compare(BankRecords o1, BankRecords o2) {

// TODO Auto-generated method stub

int result = o1.getIncome().compareTo(o2.getIncome());

return result;

}

}

/\*\*

\*

\* Sorts the array based on sex, savings account, mortgage, and region

\* even though it is not really needed

\* I could have split the minimum and maximum sort but choose to do it all at once

\* added this compare to fulfill requirements of 3 comparator classes set forth in instructions

\*

\*/

class FwMaSLocation implements Comparator<BankRecords>{

@Override

public int compare(BankRecords x, BankRecords y) {

// TODO Auto-generated method stub

int result = x.getSex().compareTo(y.getSex());

if (result != 0) return result;

//Second sort

int result1 = x.getSave\_act().compareTo(y.getSave\_act());

if (result1 != 0) return result1;

//Third Sort

int result2 = x.getMortgage().compareTo(y.getMortgage());

if (result2 !=0) return result2;

int result3 = x.getRegion().compareTo(y.getRegion());

return result3;

}

}

# Console Output

Avg income for Inner City region $27752.87

Avg income for Rural region $30027.61

Avg income for Suburban region $28656.36

Avg income for Town region $26802.19

Inner City region Minimum income $6294.21

Rural region Minimum income $8639.24

Surburban region Minimum income $8143.75

Town region Minimum income $5014.21

Inner City region Max income $266222.8

Rural region Max income $57398.1

Surburban region Max income $63130.1

Town region Max income $59803.9

Inner City region Females with Mortgage and Savings Account 31

Rural region Females with Mortgage and Savings Account 9

Suburban region Females with Mortgage and Savings Account 4

Town region Females with Mortgage and Savings Account 27

Inner City region Males with 1 Child and a Car 15

Rural region Males with 1 Child and a Car 9

Suburban region Males with 1 Child and a Car 3

Town region Males with 1 Child and a Car 7

Current Date 04/03/18 14:32:47

Created By Chris Doherty

# Screenshot

