Francisco Rois Siso

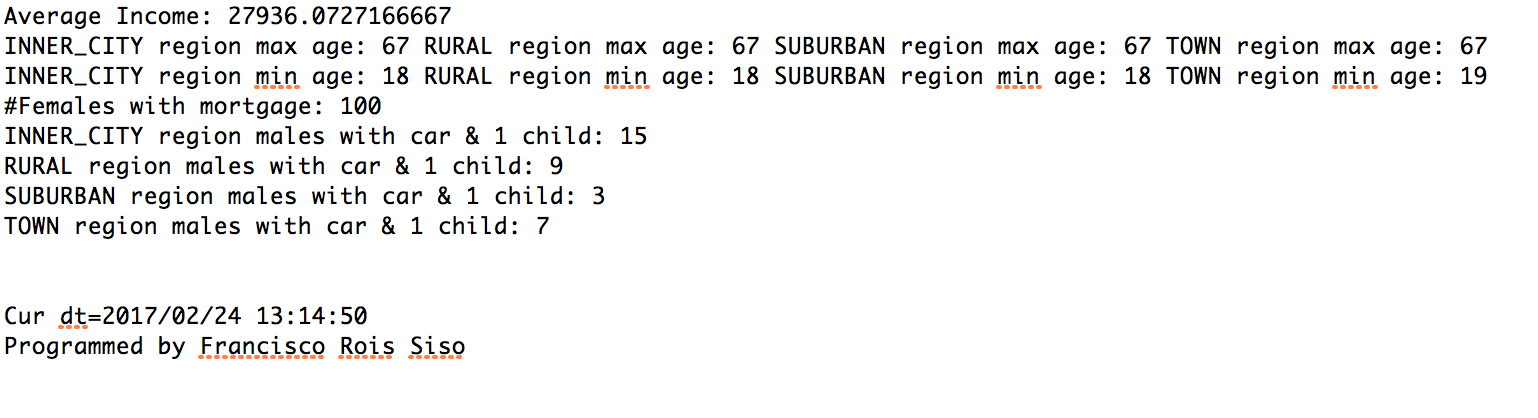
A20385948

02/24/2017

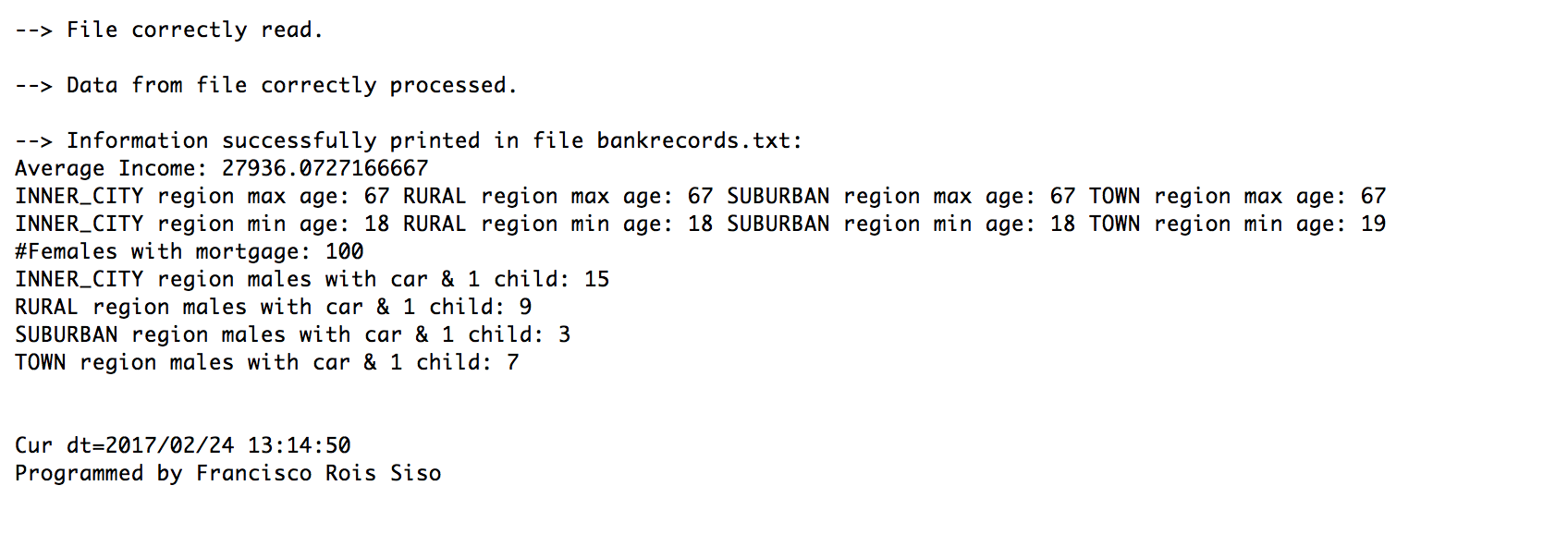
Lab3 - ITMD510 Object-Oriented Application Development

**Content**

1. Snapshots (text file and console)
2. Records\_test.java (source code)
3. Records.java
4. RecordsRegionAndAgeComparator.java
5. RecordsSexAndMortgageComparator.java
6. RecordsSexRegionCarAndNChildComparator.java
7. Client.java
8. BankRecords.java
9. **SNAPSHOTS**

****

*Content of the text file “bankrecords.txt” after running the program.*

****

*Information shown in the console while the different processes are performed.*

1. **Records\_test.java (source code)**

**import** java.io.BufferedWriter;

**import** java.io.FileWriter;

**import** java.io.IOException;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.Map;

/\*

Program to perform data analysis from a csv file with records from clients of a bank. After the data is retrieved and stored, the following

information is extracted from the data analysis:

- Overall average income.

- Max and min ages per location.

- Total number of females with mortgage.

- Number of males with both car and 1 child per location.

For efficiency purposes, the list of bank records is sorted in different ways during the program performance.

The results are shown in a text file.

- Programmer: Francisco Rois Siso

- Date: 02/24/2017

- Source File Name: Records\_Test.java

- Lab 3

- ITMD510 Object-Oriented Application Development

\*/

/\*\*

\* The purpose of this class is test the class Records by creating an object and calling the different methods of the class.

\* Then the information obtained is printed into a text file.

\*

\* **@author** Francisco Rois Siso

\*

\*/

**public** **class** Records\_Test {

**final** **static** String *FILENAME* = "bankrecords.txt";

**static** BufferedWriter *bw* = **null**;

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

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

// create a new object Records, which will be used all along the program

Records rec = **new** Records();

// read data from the default csv file and process it (storage)

rec.readData("default");

// variable for program performance measurement

**long** startTime = System.*nanoTime*();

// String content is created in order to add the information required to print on the text file. The info is also displayed in console.

String content = "";

// add average income

content += "Average Income: "+rec.getAverageIncome()+"\n";

// add max and min ages per region. map\_mins and map\_maxs store the min and max ages per location, respectively.

Map<String,Integer> map\_mins = rec.getMinAgesPerLocation();

Map<String,Integer> map\_maxs = rec.getMaxAgesPerLocation();

**for**(String s: rec.possible\_regions){

content += s + " region max age: "+ map\_maxs.get(s)+ " ";

}

content += "\n";

**for**(String r: rec.possible\_regions){

content += r + " region min age: "+ map\_mins.get(r)+ " ";

}

// add total number of females with mortgage

**int** numFemMort = rec.getNumberOfFemalesWithMortgages();

content += "\n";

content += "#Females with mortgage: " + numFemMort;

// add number of males with car and 1 child, per region. map\_malesCar1Child stores the information after data analysis.

Map<String,Integer> map\_malesCar1Child = rec.getNumberOfMalesWithCarAndOneChildPerLocation();

content += "\n";

**for**(String t: rec.possible\_regions){

content += t + " region males with car & 1 child: " + map\_malesCar1Child.get(t);

content += "\n";

}

// add name, date and time at the end of the file, for lab submission purposes

content += *showDateAndTime*();

// create FileWriter and BufferedWriter in order to write the content into the text file specified

**try** {

*fw* = **new** FileWriter(*FILENAME*);

*bw* = **new** BufferedWriter(*fw*);

*bw*.write(content);

}**catch** (IOException e) {

e.printStackTrace();

}**finally** {

**try** {

// close FileWriter and BufferedWriter

**if** (*bw* != **null**)

*bw*.close();

**if** (*fw* != **null**)

*fw*.close();

} **catch** (IOException ex) {

ex.printStackTrace();

}

}

// info also displayed in console

System.*out*.println("\n--> Information successfully printed in file " + *FILENAME* +":");

System.*out*.print(content);

// code used for time measuring purposes. Program performance time

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

System.*out*.println(String.*valueOf*(rec.getAverageIncome()));

**long** endTime = System.*nanoTime*();

System.*out*.println("Took "+(endTime - startTime) + " ns");

}

/\*\*

\* showDateAndTime allows to show the current date and time for lab submission purposes

\*/

**static** String showDateAndTime(){

String timeStamp = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(Calendar.*getInstance*().getTime());

**return**("\n\nCur dt=" + timeStamp + "\nProgrammed by Francisco Rois Siso\n");

}

}

1. **Records.java**

**import** java.util.Collections;

**import** java.util.HashMap;

**import** java.util.Map;

/\*\*

\* The class Records has methods for data processing and analysis of the data read and stored in BankRecords.java.

\* The class Records extends BankRecords, therefore it inherits its methods and variables.

\*

\* **@author** Francisco Rois Siso

\*

\*/

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

/\*\*

\* Constructor with no input parameters.

\*/

**public** Records(){}

/\*\*

\* method getAverageIncome goes through the list of bank records and calculate the total average income

\* **@return** averageIncome, as a double

\*/

**public** **double** getAverageIncome(){

**double** averageIncome = 0.0;

**for**(BankRecords b:**super**.bankRecords\_list){

averageIncome += b.getIncome();

}

averageIncome /= **super**.bankRecords\_list.size();

**return** averageIncome;

}

/\*\*

\* method getMinAgesPerLocation sorts the list of bank records by region and then creates a map with the minimum age for each of them.

\* **@return** map\_mins a map with the regions as key and the minimum age as values, as a Map<String,Integer>

\*/

**public** Map<String,Integer> getMinAgesPerLocation(){

Map<String,Integer> map\_mins = **new** HashMap<String,Integer>();

// sort the BankRecords list by region and age

Collections.*sort*(**super**.bankRecords\_list,**new** RecordsRegionAndAgeComparator());

// sort possible regions for info displaying purposes

Collections.*sort*(**super**.possible\_regions);

**for**(BankRecords b: **super**.bankRecords\_list){

// insert in map the first age value found for each region, which is the minimum

**if**(!map\_mins.containsKey(b.getRegion())){

// insert minimum in map

map\_mins.put(b.getRegion(), b.getAge());

}

}

**return** map\_mins;

}

/\*\*

\* method getMaxAgesPerLocation sorts the list of bank records by region and then creates a map with the maximum age for each of them.

\* **@return** map\_maxs a map with the regions as key and the maximum age as values, as a Map<String,Integer>

\*/

**public** Map<String,Integer> getMaxAgesPerLocation(){

Map<String,Integer> map\_maxs = **new** HashMap<String,Integer>();

// sort the BankRecords list by region and age

Collections.*sort*(**super**.bankRecords\_list,**new** RecordsRegionAndAgeComparator());

// sort possible regions for info displaying purposes

Collections.*sort*(**super**.possible\_regions);

// go through the sorted list, but this time backwards, so the first values found are the maximums

**for**(**int** i = **super**.bankRecords\_list.size()-1 ; i>=0 ; i--){

BankRecords b = **super**.bankRecords\_list.get(i);

**if**(!map\_maxs.containsKey(b.getRegion())){

// insert in map the only the first age value found for each region, which is the maximum

map\_maxs.put(b.getRegion(), b.getAge());

}

}

**return** map\_maxs;

}

/\*\*

\* method getNumberOfFemalesWithMortgages sorts the list of bank records by sex and mortgage. Then it goes through the sorted list

\* and counts the number of females with mortgage.

\* **@return** count the number of females with a mortgage, as an int

\*/

**public** **int** getNumberOfFemalesWithMortgages(){

// sort list by sex and mortgage

Collections.*sort*(**super**.bankRecords\_list,**new** RecordsSexAndMortgageComparator());

// go through the sorted list only until a record not female and without mortgage is found

**int** count = 0;

**int** e = 0;

**while**(**super**.bankRecords\_list.get(e).getSex().equals("FEMALE") && **super**.bankRecords\_list.get(e).getMortgage().equals("YES")){

e++;

count ++;

}

**return** count;

}

/\*\*

\* method getNumberOfMalesWithCarAndOneChildPerLocation first sorts the list of bank records by sex, region, car and number of children.

\* Then the method goes through the collection and counts the number of males with car and one child, per region.

\* **@return** map a map with the regions as key and the number of males with car and 1 child as value for each of the regions

\*/

**public** Map<String,Integer> getNumberOfMalesWithCarAndOneChildPerLocation(){

// sort list by sex, location, car and number of child

Collections.*sort*(**super**.bankRecords\_list, **new** RecordsSexRegionCarAndNChildComparator());

// map to store number of specified elements (value) per region (key)

Map<String,Integer> map = **new** HashMap<String,Integer>();

**int** i = 0;

**int** count = 0;

// while it is a man and he has car

**while**(i < **super**.bankRecords\_list.size() && **super**.bankRecords\_list.get(i).getSex().equals("MALE") && **super**.bankRecords\_list.get(i).getCar().equals("YES")){

// if he has 1 child, increment the count

**if**(**super**.bankRecords\_list.get(i).getChildren() == 1){

count++;

}

// if index in bound

**if**((i+1) < **super**.bankRecords\_list.size()) {

// if i is the last item of the certain region, put the count and the region into the map and restart the count for the next region

**if**(!**super**.bankRecords\_list.get(i).getRegion().equals(**super**.bankRecords\_list.get(i+1).getRegion())){

map.put(**super**.bankRecords\_list.get(i).getRegion(), count);

count = 0;

}

}

// move to the next element in the collection

i++;

}

**return** map;

}

}

1. **RecordsRegionAndAgeComparator.java**

**import** java.util.Comparator;

/\*\*

\* Class used to compare two objects BankRecords by their region and age.

\*

\* **@author** Francisco Rois Siso

\*

\*/

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

@Override

**public** **int** compare(BankRecords rec1, BankRecords rec2) {

// if different region

**if**(!rec1.getRegion().equals(rec2.getRegion())){

**return** rec1.getRegion().compareTo(rec2.getRegion());

}

// if equal region

**else**{

**return** rec1.getAge()-rec2.getAge();

}

}

}

1. **RecordsSexAndMortgageComparator.java**

**import** java.util.Comparator;

/\*\*

\* Class used to compare two objects BankRecords by their sex and mortgage.

\*

\* **@author** Francisco Rois Siso

\*

\*/

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

@Override

**public** **int** compare(BankRecords rec1, BankRecords rec2) {

// if different sex

**if**(!rec1.getSex().equals(rec2.getSex()))

**return** rec1.getSex().compareTo(rec2.getSex());

// if equal sex

**else**

**return** rec2.getMortgage().compareTo(rec1.getMortgage());

}

}

1. **RecordsSexRegionCarAndNChildComparator.java**

**import** java.util.Comparator;

/\*\*

\* Class used to compare two objects BankRecords by their sex, region, car and number of children.

\*

\* **@author** Francisco Rois Siso

\*

\*/

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

@Override

**public** **int** compare(BankRecords rec1,BankRecords rec2){

// if different sex

**if**(!rec1.getSex().equals(rec2.getSex())){

**return** rec2.getSex().compareTo(rec1.getSex());

}

**else**{

// if equal sex but different car

**if**(!rec1.getCar().equals(rec2.getCar())){

**return** rec2.getCar().compareTo(rec1.getCar());

}

**else**{

// if equal sex, equal car but different region

**if**(!rec1.getRegion().equals(rec2.getRegion())){

**return** rec1.getRegion().compareTo(rec2.getRegion());

}

// if equal sex, equal car, equal region, then compare by number of children

**else**{

**return** rec1.getChildren()-rec2.getChildren();

}

}

}

}

}

1. **Client.java**

/\*\*

\* The class Client provides three abstract methods the bank needs in order to process data from a file: readData, processData and printData.

\*

\* **@author** Francisco Rois Siso

\*

\*/

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

// abstract methods that will be defined in BankRecords.java.

// a String parameter called fileName is included in "readData" in order to be able

// to read information from different files in future implementations.

// The input String "default" triggers the reading from the default file ("bank-Detail.csv")

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

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

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

}

1. **BankRecords.java**

**import** java.io.BufferedReader;

**import** java.io.FileNotFoundException;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.text.SimpleDateFormat;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.Calendar;

**import** java.util.List;

/\*\*

\* The class BankRecords extends the abstract class Client and implements its methods readData, processData

\* and printData. The purposes of these methods are reading clients' information from a certain csv file, convert

\* each line into an object with the correct parameters and then print information from several of the records.

\*

\* **@author** Francisco Rois Siso

\*

\*/

**public** **class** BankRecords **extends** Client {

// default file from which the reading is done if not other file is specified.

**final** **static** String *DEFAULT\_FILE* = "bank-Detail.csv";

// list of objects BankRecords to store all the clients with their information.

List<BankRecords> bankRecords\_list = **new** ArrayList<>();

// list of lists of Strings. Each of the lists is a line (corresponding to a client) and each String is a client's parameter

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

// list with all the possible values for the field "region" in the Bank Records

List <String> possible\_regions = **new** ArrayList<>();

// parameters of the client

**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;

/\*\*

\* Constructor without input parameters.

\*/

**public** BankRecords(){}

/\*\*

\* Constructor with all the parameters of the client.

\* **@param** id Identifier as String

\* **@param** age Age as int

\* **@param** sex Sex as String. It can be FEMALE or MALE.

\* **@param** region Region as String. It can be INNER\_CITY, TOWN, RURAL or SUBURBAN.

\* **@param** income Income as double.

\* **@param** married Shows if the client is married or not, as a String. It can be NO or YES

\* **@param** children The number of children that the client has, as an int. It can be 0,1,2 or 3.

\* **@param** car Shows if the client has car or not, as a String. It can be NO or YES.

\* **@param** save\_act

\* **@param** current\_act

\* **@param** mortgage Shows if the client has mortgage or not. It can be NO or YES

\* **@param** pep Shows if the client is a "Politically Exposed Person"

\*/

**public** BankRecords(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){

**this**.id = id;

**this**.age = age;

**this**.sex = sex;

**this**.region = region;

**this**.age = age;

**this**.income = income;

**this**.married = married;

**this**.children = children;

**this**.car = car;

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

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

**this**.mortgage = mortgage;

**this**.pep = pep;

}

/\*\*

\* Getters and setters for the parameters of the client.

\*/

**public** String getId() {

**return** id;

}

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

**this**.id = id;

}

**public** **int** getAge() {

**return** age;

}

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

**this**.age = age;

}

**public** String getSex() {

**return** sex;

}

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

**this**.sex = sex;

}

**public** String getRegion() {

**return** region;

}

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

**this**.region = region;

}

**public** **double** getIncome() {

**return** income;

}

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

**this**.income = income;

}

**public** String getMarried() {

**return** married;

}

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

**this**.married = married;

}

**public** **int** getChildren() {

**return** children;

}

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

**this**.children = children;

}

**public** String getCar() {

**return** car;

}

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

**this**.car = car;

}

**public** String getSave\_act() {

**return** save\_act;

}

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

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

}

**public** String getCurrent\_act() {

**return** current\_act;

}

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

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

}

**public** String getMortgage() {

**return** mortgage;

}

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

**this**.mortgage = mortgage;

}

**public** String getPep() {

**return** pep;

}

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

**this**.pep = pep;

}

/\*\*

\* Override method toString in order to show the proper String with the parameters of the bank record

\* **@return** String The chain of parameters of the bank record.

\*/

@Override

**public** String toString() {

**return** "BankRecords [id=" + id + ", age=" + age + ", sex=" + sex

+ ", region=" + region + ", income=" + income + ", married="

+ married + ", children=" + children + ", car=" + car

+ ", save\_act=" + save\_act + ", current\_act=" + current\_act

+ ", mortgage=" + mortgage + ", pep=" + pep + "]";

}

/\*\*

\* readData method reads all the data from the file csv indicated and stores it into an ArrayList.

\* Then it calls to the method processData in order to continue the flow of actions.

\* **@param** fileName String indicating the name of the file to read the data from.

\*/

**public** **void** readData(String fileName){

// if the contrary is not specified, the reading is done from the default file.

String file2Read = *DEFAULT\_FILE*;

**if**(!fileName.equals(file2Read)&&!fileName.equals("default")){

file2Read = fileName;

}

// separator used between fields in the document.

**final** String SEPARATOR = ",";

// line that will correspond to a client with its data.

String line = "";

// try to read from file.

**try**(BufferedReader br = **new** BufferedReader(**new** FileReader(file2Read))){

// while there are more lines to read, continue reading.

**while**((line = br.readLine()) != **null**){

// add the new line to the list of lines. The output from the split method is an array,

//therefore it is converted into an ArrayList in order to include the line in the list of lines.

*listOfLines*.add(Arrays.*asList*(line.split(SEPARATOR)));

}

// inform the user when the process is correctly finished.

System.*out*.println("\n--> File correctly read.");

// continue the chain of actions by processing the data.

processData();

} **catch**(FileNotFoundException e){

System.*out*.println("The name of the file specified is not correct.");

}**catch**(IOException e){

System.*out*.println("Please, introduce a correct file name.");

}

}

/\*\*

\* processData method processes the ArrayList from readData and adds the data into each of the bank records objects.

\* It makes use of the setters defined in this class.

\* Then it calls the method printData in order to continue the flow of actions.

\*/

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

// create a new object BankRecords for each line in listOfLines and use setters to add the different features of the clients.

**for**(List<String> client\_fields: *listOfLines*){

BankRecords b = **new** BankRecords();

b.setId(client\_fields.get(0));

b.setAge(Integer.*parseInt*(client\_fields.get(1)));

b.setSex(client\_fields.get(2));

b.setRegion(client\_fields.get(3));

b.setIncome(Double.*parseDouble*(client\_fields.get(4)));

b.setMarried(client\_fields.get(5));

b.setChildren(Integer.*parseInt*(client\_fields.get(6)));

b.setCar(client\_fields.get(7));

b.setSave\_act(client\_fields.get(8));

b.setCurrent\_act(client\_fields.get(9));

b.setMortgage(client\_fields.get(10));

b.setPep(client\_fields.get(11));

// once the client's features are correctly set, the client is added to the list of bank records.

bankRecords\_list.add(b);

**if**(!possible\_regions.contains(b.getRegion()))

possible\_regions.add(b.getRegion());

}

// inform the user of successful processing.

System.*out*.println("\n--> Data from file correctly processed.");

// print on screen date and time for lab submission purposes.

//showDateAndTime();

// continue the chain of actions by printing data.

//printData();

}

/\*\*

\* printData method prints the first 25 records for various fields to the console.

\* It makes use of the getters defined in this class.

\* The records printed are ID, AGE, SEX, REGION, INCOME and MORTGAGE.

\*/

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

// Inform the user of the data to be printed on screen.

System.*out*.println("\n--> Here is some data from the first 25 clients in the document:");

// take the first 25 clients from the list of bank records and print information from them

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

BankRecords brec = bankRecords\_list.get(i);

// build String chain in order to print it.

String chain\_client= "\n\t>>>>> CLIENT " + (i+1) + " <<<<<\n"

+ "\t-------------------------" + "\n"

+ "\tID:\t\t" + brec.getId() + "\n"

+ "\tAGE:\t\t" + brec.getAge() + "\n"

+ "\tSEX:\t\t" + brec.getSex() + "\n"

+ "\tREGION:\t\t" + brec.getRegion() + "\n"

+ "\tINCOME:\t\t$" + brec.getIncome() + "\n"

+ "\tMORTGAGE:\t" + brec.getMortgage() + "\n"

+ "\n\n";

// once the chain is built, print it on screen.

System.*out*.print(chain\_client);

}

}

/\*\*

\* showDateAndTime allows to show the current date and time for lab submission purposes

\*/

**static** **void** showDateAndTime(){

String timeStamp = **new** SimpleDateFormat("yyyy/MM/dd HH:mm:ss").format(Calendar.*getInstance*().getTime());

System.*out*.println("\n\nCur dt=" + timeStamp + "\nProgrammed by Francisco Rois Siso\n");

}

}