Rahul Nagaraju | **A20543969 |** [rnagaraju@hawk.iit.edu](mailto:rnagaraju@hawk.iit.edu)

**Lab3**

**Object-oriented App Development ( ITMD-510 )**

**PROJECT Bank record generations 100 points**

Objective: To write a program that performs data analysis from class objects created in lab #2.

***PROJECT DESCRIPTION***

Bank of IIT is in desperate need of analytics from its clients for its loan application process. Currently records show 600 clients exist and the bank is hoping to expand its clientele and its potential by offering premium loans to deserved grantees.

Perform the data analysis as follows for this lab.

***Project Details***

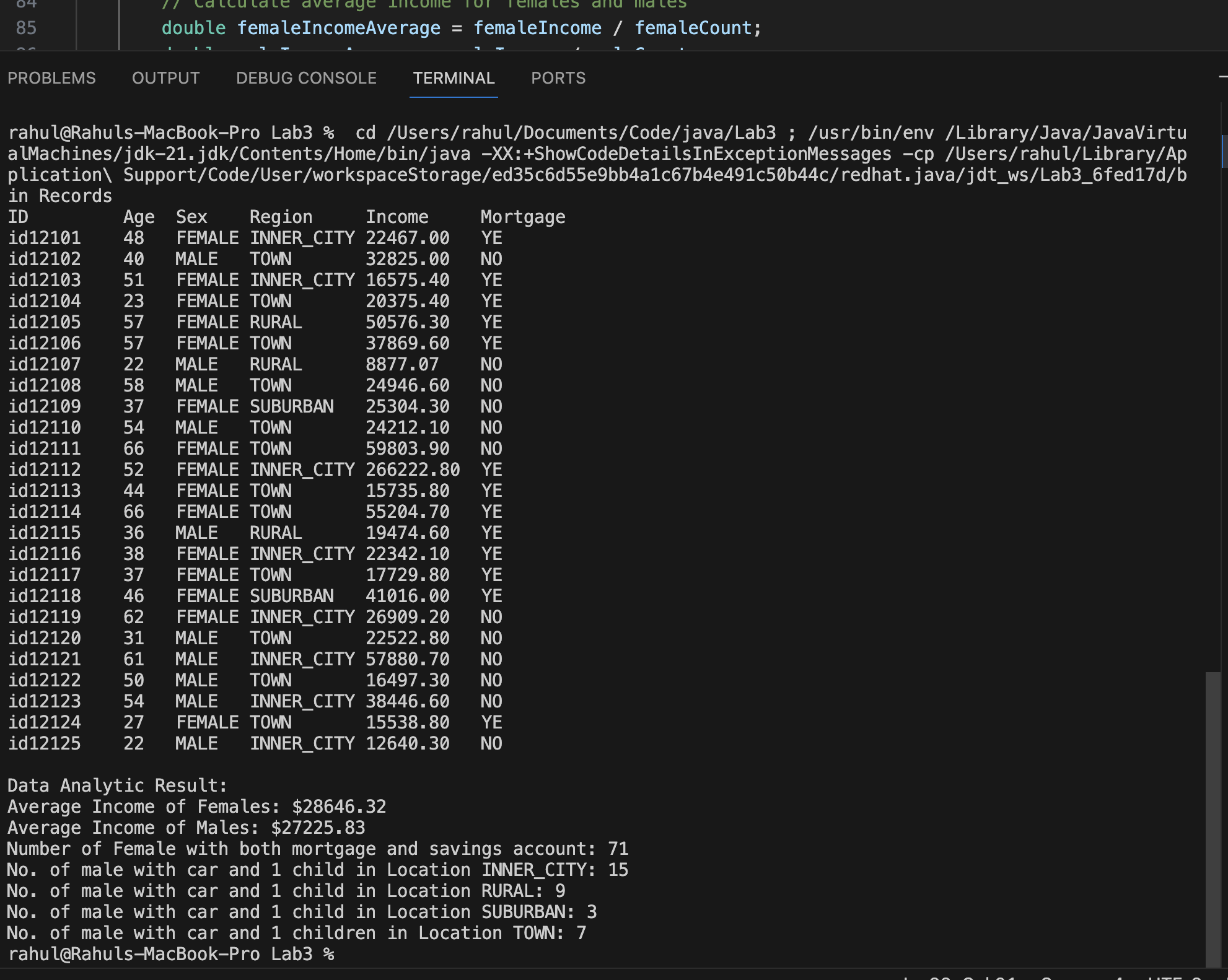
- Add to your existing project files from lab 2, a new class called **Records**. Have the class *extend* the BankRecords class to grab hold of its instance methods plus the BankRecords object array.

- Provide at least 2 comparator classes implementing the **java.util.Comparator** interfacefor comparing various fields for the following data analysis requirements.

- Perform the following analysis requirements and output detail for the Records class.

* Display the following data analytics in a coherent manner to the console:
* average income for males vs. females
* number of females with a mortgage and savings account
* number of males with both a car and 1 child per location
* Write all displayed data to a text file called **bankrecords.txt**relative to your project path as well. Append your name within the text file at the end of the file plus the date/time.
* Make sure to include a try catch block when reading from or writing to any file! Include also proper exception handling.
* Include your project’s entire source code/.csv/bankrecords.txt file into a zip file, and in a separate Word file, your console output snapshot and snapshot of your **bankrecords.txt** file, a program description and all your source code (all java files) as well.

**Snapshot :-**

****

**bankRecords.txt**

Data Analytic Result:

Average Income of Females: $28646.32

Average Income of Males: $27225.83

Number of Female with both mortgage and savings account: 71

No. of male with car and 1 child in Location INNER\_CITY: 15

No. of male with car and 1 child in Location RURAL: 9

No. of male with car and 1 child in Location SUBURBAN: 3

No. of male with car and 1 children in Location TOWN: 7

Name: Rahul Nagaraju

Current Date and Time: 2023-10-22 22:29:29 SUNDAY America/Chicago

Source Code:

**Client.java**

/\*

--------------------------------------------------------------------

- Author Rahul Nagaraju

- Assignment: Lab3

- FileName: Client.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

----------------------------------------------------------------------

\*/

public abstract class Client {

    public abstract void readData();

    public abstract void processData();

    public abstract void printData();

}

**BankRecords.java**

/\*

--------------------------------------------------------------------

- Author Rahul Nagaraju

- Assignment: Lab3

- FileName: BankRecords.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

----------------------------------------------------------------------

\*/

import java.io.BufferedReader; // Import the BufferedReader class for file reading

import java.io.FileReader; // Import the FileReader class for file reading

import java.io.File; // Import the File class for file manipulation

import java.io.FileNotFoundException; // Import the FileNotFoundException class for handling missing files

import java.io.IOException; // Import the IOException class for handling input/output errors

import java.util.ArrayList; // Import the ArrayList class for working with lists

import java.util.Arrays; // Import the Arrays class for working with arrays

import java.util.Comparator; // Import the Comparators class for comparing objects

import java.util.List; // Import the List class for working with lists

public class BankRecords extends Client{

// Necessary variables

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;

// Create a list to store lists of strings (2D list)

List<List<String>> arrayOfLists = new ArrayList<>();

// Create an array to hold BankRecords objects

BankRecords[] recordObjects = new BankRecords[600];

// Constructor

public BankRecords() {

// Default constructor

}

// Getter method for retrieving the ID of the bank record.

public String getId() {

return id;

}

// Setter method for setting the ID of the bank record.

public void setId(String id) {

this.id = id;

}

// Getter method for retrieving the age of the bank record.

public int getAge() {

return age;

}

// Setter method for setting the age of the bank record.

public void setAge(int age) {

this.age = age;

}

// Getter method for retrieving the sex of the bank record.

public String getSex() {

return sex;

}

// Setter method for setting the sex of the bank record.

public void setSex(String sex) {

this.sex = sex;

}

// Getter method for retrieving the region of the bank record.

public String getRegion() {

return region;

}

// Setter method for setting the region of the bank record.

public void setRegion(String region) {

this.region = region;

}

// Getter method for retrieving the income of the bank record.

public double getIncome() {

return income;

}

// Setter method for setting the income of the bank record.

public void setIncome(double income) {

this.income = income;

}

// Getter method for retrieving the marital status of the bank record.

public String getMarried() {

return married;

}

// Setter method for setting the marital status of the bank record.

public void setMarried(String married) {

this.married = married;

}

// Getter method for retrieving the number of children of the bank record.

public int getChildren() {

return children;

}

// Setter method for setting the number of children of the bank record.

public void setChildren(int children) {

this.children = children;

}

// Getter method for retrieving the car ownership status of the bank record.

public String getCar() {

return car;

}

// Setter method for setting the car ownership status of the bank record.

public void setCar(String car) {

this.car = car;

}

// Getter method for retrieving the savings account status of the bank record.

public String getSave\_act() {

return save\_act;

}

// Setter method for setting the savings account status of the bank record.

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

this.save\_act = save\_act;

}

// Getter method for retrieving the current account status of the bank record.

public String getCurrent\_act() {

return current\_act;

}

// Setter method for setting the current account status of the bank record.

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

this.current\_act = current\_act;

}

// Getter method for retrieving the mortgage status of the bank record.

public String getMortgage() {

return mortgage;

}

// Setter method for setting the mortgage status of the bank record.

public void setMortgage(String mortgage) {

this.mortgage = mortgage;

}

// Getter method for retrieving the PEP (Personal Equity Plan) status of the bank record.

public String getPep() {

return pep;

}

// Setter method for setting the PEP (Personal Equity Plan) status of the bank record.

public void setPep(String pep) {

this.pep = pep;

}

// Override the abstract methods from the abstract class by adding instance fields to read, process, and print the data in concrete subclasses.

/\*\*

\* Read data from a CSV file and store it in arrayOfLists.

\*/

@Override

public void readData() {

BufferedReader br = null; // Declare a BufferedReader variable

try {

// Initialize a reader object and set the file path to the project's root

br = new BufferedReader(new FileReader(new File("bank-Detail.csv"))); // Create a FileReader and BufferedReader for reading the CSV file

String line; // Declare a variable to store each line of the file

// Read each record in the CSV file

while ((line = br.readLine()) != null) { // Read each line of the file until the end

// Split each record in the CSV file by a comma (,) and add it to the list

List<String> parts = Arrays.asList(line.split(",")); // Split the line into a list of strings using a comma as the separator

arrayOfLists.add(parts); // Add the list of strings to arrayOfLists

}

} catch (FileNotFoundException e) { // Handle the FileNotFoundException

System.err.println("File not found: " + e.getMessage()); // Print an error message if the file is not found

} catch (IOException e) { // Handle the IOException

System.err.println("Error reading the file: " + e.getMessage()); // Print an error message if there is an IO error

} finally {

try {

if (br != null) {

br.close(); // Close the BufferedReader if it is not null

}

} catch (IOException e) {

System.err.println("Error closing the file: " + e.getMessage()); // Print an error message if there is an error while closing the file

}

}

// Call the processData method to further process the data

processData();

}

/\*\*

\* Process the data from arrayOfLists and populate the recordObjects array.

\*/

@Override

public void processData(){

// Iterate through the data and populate recordObjects array

int idx = 0; // Initialize an index variable

for (List<String> rowData : arrayOfLists) { // Loop through each list of strings in arrayOfLists

recordObjects[idx] = new BankRecords(); // Create a new BankRecords object

recordObjects[idx].setId(rowData.get(0)); // Get the 1st column and set it as the ID in the BankRecords object

recordObjects[idx].setAge(Integer.parseInt(rowData.get(1))); // Get the 2nd column, parse it as an integer, and set it as the age in the BankRecords object

recordObjects[idx].setSex(rowData.get(2)); // Get the 3rd column and set it as the sex in the BankRecords object

recordObjects[idx].setRegion(rowData.get(3)); // Get the 4th column and set it as the region in the BankRecords object

recordObjects[idx].setIncome(Double.parseDouble(rowData.get(4))); // Get the 5th column, parse it as a double, and set it as the income in the BankRecords object

recordObjects[idx].setMarried(rowData.get(5)); // Get the 6th column and set it as the marital status in the BankRecords object

recordObjects[idx].setChildren(Integer.parseInt(rowData.get(6))); // Get the 7th column, parse it as an integer, and set it as the number of children in the BankRecords object

recordObjects[idx].setCar(rowData.get(7)); // Get the 8th column and set it as the car ownership status in the BankRecords object

recordObjects[idx].setSave\_act(rowData.get(8)); // Get the 9th column and set it as the savings account status in the BankRecords object

recordObjects[idx].setCurrent\_act(rowData.get(9)); // Get the 10th column and set it as the current account status in the BankRecords object

recordObjects[idx].setMortgage(rowData.get(10)); // Get the 11th column and set it as the mortgage status in the BankRecords object

recordObjects[idx].setPep(rowData.get(11)); // Get the 12th column and set it as the PEP (Personal Equity Plan) status in the BankRecords object

// Increment the index for the next BankRecords object

idx++;

}

// Call the printData method to print the data

printData();

}

/\*\*

\* Print the data in a formatted tabular format.

\*/

public void printAllData() {

// Print column headers

System.out.printf("%-10s %-4s %-6s %-10s %-10s %-10s %-8s %-4s %-6s %-8s %-8s %-3s%n",

"ID", "Age", "Sex", "Region", "Income", "Married", "Children", "Car", "Savings", "Current", "Mortgage", "PEP");

// Print the first 25 records (or fewer if there are fewer than 25 records)

int maxRecords = Math.min(25, recordObjects.length);

// Print records

for (int i = 0; i < maxRecords; i++) { // Loop through each BankRecords object in recordObjects

BankRecords record = recordObjects[i];

// Print the data in a formatted tabular format

System.out.printf("%-10s %-4d %-6s %-10s %-10.2f %-10s %-8d %-4s %-6s %-8s %-8s %-3s%n",

record.getId(), record.getAge(), record.getSex(), record.getRegion(), record.getIncome(),

record.getMarried(), record.getChildren(), record.getCar(), record.getSave\_act(),

record.getCurrent\_act(), record.getMortgage(), record.getPep());

}

}

/\*\*

\* Print only 7 certain columns of data in a formatted tabular format.

\*/

@Override

public void printData() {

// Print column headers

System.out.printf("%-10s %-4s %-6s %-10s %-10s %-10s%n",

"ID", "Age", "Sex", "Region", "Income", "Mortgage");

// Print the first 25 records (or fewer if there are fewer than 25 records)

int maxRecords = Math.min(25, recordObjects.length);

// Print records

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

BankRecords record = recordObjects[i];

// Print the selected fields in a formatted tabular format

System.out.printf("%-10s %-4d %-6s %-10s %-10.2f %-10.2s%n",

record.getId(), record.getAge(), record.getSex(), record.getRegion(), record.getIncome(),

record.getMortgage());

}

}

/\*\*

\* Comparator for arranging BankRecords by the 'Sex' attribute.

\*/

public class SexComparator implements Comparator<BankRecords> {

/\*\*

\* Compares two BankRecords based on their 'Sex' attribute.

\*

\* @param record1 The first BankRecords object to compare.

\* @param record2 The second BankRecords object to compare.

\* @return A value indicating the order of the two records.

\*/

@Override

public int compare(BankRecords record1, BankRecords record2) {

return record1.getSex().compareTo(record2.getSex());

}

}

/\*\*

\* Comparator for organizing BankRecords based on the 'Location' attribute.

\*/

public class LocationComparator implements Comparator<BankRecords> {

/\*\*

\* Compares two BankRecords using their 'Location' attribute.

\*

\* @param record1 The first BankRecords object to compare.

\* @param record2 The second BankRecords object to compare.

\* @return A result indicating the relative order of the two records.

\*/

@Override

public int compare(BankRecords record1, BankRecords record2) {

return record1.getRegion().compareTo(record2.getRegion());

}

}

}

**BankRecordsTest.java**

/\*

--------------------------------------------------------------------

- Author Rahul Nagaraju

- Assignment: Lab3

- FileName: BankRecordsTest.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

----------------------------------------------------------------------

\*/

public class BankRecordsTest {

    public static void main(String[] args) {

        // Create an instance of the BankRecords class

        BankRecords bankRecords = new BankRecords();

        // Call the readData() method to read data from a CSV file and process it

        bankRecords.readData();

    }

}

**Records.java**

/\*

--------------------------------------------------------------------

- Author Rahul Nagaraju

- Assignment: Lab3

- FileName: Records.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

----------------------------------------------------------------------

\*/

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.time.DateTimeException;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.Arrays;

import java.util.TimeZone;

// The "Records" class extends the "BankRecords" class to inherit and possibly extend its functionality.

public class Records extends BankRecords {

private FileWriter fw;

// Constructor for Records class

public Records() {

try {

// Initialize the FileWriter for writing to "bankrecords.txt"

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

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

// Handle file not found errors

} catch (SecurityException e) {

// Handle security-related errors

System.err.println("Security Exception: " + e.getMessage());

} catch (UnsupportedEncodingException e) {

// Handle unsupported encoding errors

System.err.println("Unsupported Encoding: " + e.getMessage());

} catch (IOException e) {

// Handle IO-related errors

System.err.println("IO Exception: " + e.getMessage());

}

}

public static void main(String[] args) {

Records records = new Records();

// Read data from BankRecords

records.readData();

// Analyze income data

records.analyzeIncome();

// Find the count of females with specific accounts

records.findFemalesCount();

// Find the count of males with specific attributes

records.findMalesCount2();

try {

// Write name and date/time information to the output file

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

records.fw.write("Name: Rahul Nagaraju");

// Format date and time

LocalDateTime currentDateTime = LocalDateTime.now();

DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");

String str = "Current Date and Time: " + currentDateTime.format(formatter) + " " +

currentDateTime.getDayOfWeek() + " " + TimeZone.getDefault().getID();

// Write date and time information to the output file

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

records.fw.write(str);

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

// Close the FileWriter

records.fw.close();

}catch (FileNotFoundException e) {

// Handle file not found errors

e.printStackTrace();

} catch (SecurityException e) {

// Handle security-related errors

e.printStackTrace();

} catch (UnsupportedEncodingException e) {

// Handle unsupported encoding errors

e.printStackTrace();

} catch (NullPointerException e) {

// Handle null reference errors

e.printStackTrace();

} catch (DateTimeException e) {

// Handle date and time formatting errors

e.printStackTrace();

} catch (IOException e) {

// Handle IO-related errors

e.printStackTrace();

}

}

// Method to analyze income of females and males separately

public void analyzeIncome() {

// Sort records by 'Sex' using the 'SexComparator'

Arrays.sort(recordObjects, new SexComparator());

int maleCount = 0, femaleCount = 0;

double maleIncome = 0, femaleIncome = 0;

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

if (recordObjects[i].getSex().equals("FEMALE")) {

femaleCount++;

femaleIncome += recordObjects[i].getIncome();

} else {

maleCount++;

maleIncome += recordObjects[i].getIncome();

}

}

// Calculate average income for females and males

double femaleIncomeAverage = femaleIncome / femaleCount;

double maleIncomeAverage = maleIncome / maleCount;

System.out.println("\nData Analytic Result:");

System.out.println("Average Income of Females: $" + String.format("%.2f", femaleIncomeAverage));

System.out.println("Average Income of Males: $" + String.format("%.2f", maleIncomeAverage));

try {

// Write data analytic results to the output file

fw.write("Data Analytic Result:\n");

fw.write("\n");

fw.write("Average Income of Females: $" + String.format("%.2f", femaleIncomeAverage));

fw.write("\n");

fw.write("Average Income of Males: $" + String.format("%.2f", maleIncomeAverage));

fw.write("\n");

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

// Handle file not found errors

} catch (SecurityException e) {

// Handle security-related errors

System.err.println("Security Exception: " + e.getMessage());

} catch (UnsupportedEncodingException e) {

// Handle unsupported encoding errors

System.err.println("Unsupported Encoding: " + e.getMessage());

} catch (IOException e) {

// Handle IO-related errors

System.err.println("IO Exception: " + e.getMessage());

}

}

// Method to find Females Count with Mortgage and Savings Account

public void findFemalesCount() {

// Sort records by 'Sex' using the 'SexComparator'

Arrays.sort(recordObjects, new SexComparator());

int femaleCount = 0;

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

if (recordObjects[i].getSex().equals("FEMALE")) {

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

femaleCount++;

}

}

}

System.out.println("Number of Female with both mortgage and savings account: " + femaleCount);

try {

// Write the count of females with specific accounts to the output file

fw.write("\n");

fw.write("Number of Female with both mortgage and savings account: " + femaleCount);

fw.write("\n");

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

// Handle file not found errors

} catch (SecurityException e) {

// Handle security-related errors

System.err.println("Security Exception: " + e.getMessage());

} catch (UnsupportedEncodingException e) {

// Handle unsupported encoding errors

System.err.println("Unsupported Encoding: " + e.getMessage());

} catch (IOException e) {

// Handle IO-related errors

System.err.println("IO Exception: " + e.getMessage());

}

}

// Method to find Males Count with car and 1 child for each location

public void findMalesCount2() {

// Sort records by 'Location' using the 'LocationComparator'

Arrays.sort(recordObjects, new LocationComparator());

String currentLocation = recordObjects[0].getRegion();

int maleCountForLocation = 0;

try {

fw.write("\n");

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

if (recordObjects[i].getSex().equals("MALE")) {

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

if (recordObjects[i].getRegion().equals(currentLocation)) {

maleCountForLocation++;

} else {

StringBuilder str = new StringBuilder("No. of male with car and 1 child in Location ");

str.append(currentLocation);

str.append(": ").append(maleCountForLocation);

System.out.println(str.toString());

fw.write(str.toString());

fw.write("\n");

currentLocation = recordObjects[i].getRegion();

maleCountForLocation = 1;

}

}

}

}

// String builder object to create the desired string

StringBuilder str = new StringBuilder("No. of male with car and 1 children in Location ");

str.append(currentLocation);

str.append(": ").append(maleCountForLocation);

System.out.println(str.toString());

fw.write(str.toString());

fw.write("\n");

} catch (FileNotFoundException e) {

System.err.println("File not found: " + e.getMessage());

// Handle file not found errors

} catch (SecurityException e) {

// Handle security-related errors

System.err.println("Security Exception: " + e.getMessage());

} catch (UnsupportedEncodingException e) {

// Handle unsupported encoding errors

System.err.println("Unsupported Encoding: " + e.getMessage());

} catch (IOException e) {

// Handle IO-related errors

System.err.println("IO Exception: " + e.getMessage());

}

}

// End of class

}