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

**Lab4**

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

**PROJECT Bank record loan generation report 100 points**

Objective: To write a program with an MVC ‘simulated’ approach that performs a Loan analysis from class objects created in lab #2.

***PROJECT DESCRIPTION***

Bank of IIT now needs your help in deciphering whom from its records should be exclusive to premium loans versus those offered micro or less premium loans.

Use a database to store then present Loan analysis information from your data BankRecords objects you worked on in prior labs.

***Project Details***

For this lab you will continue using your current project **src** folder files created for labs 2 & 3 and you'll create the following packages & their containments in an MVC style as follows:

**Package**: **models**

**File** **DbConnect:** will allow an object to connect / close a database connection.

**File** **DaoModel:** where Dao stands for Data Access Object, defines CRUD (Create Read Update Delete) like operations.

**Package**: **controllers**

**File** **LoanProcessing:** acts as a controller or “driver” file (i.e., includes main function) to run database CRUD methods and fires up some resulting output.

**Package**: **views**

**File LoanView:** shows JTable output.

**Package**: **records**

**File Client.java  
 File BankRecords.java**

**File BankRecordsTest.java**

**File Records.java**

**File RecordSerialize.java**

**Snapshot :-**

**A screenshot of a computer

Description automatically generated**

**A screenshot of a computer

Description automatically generated**

Extra credits:-

Serialization and deserialization with time difference displayed(5060 ms).

Also .ser file included in the snapshots.

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Extra credits –

Insert Records using prepared statement:-

A computer screen shot of a code

Description automatically generated

Detailed Loan Analysis Report:-

A screenshot of a computer

Description automatically generated

Source Code:

**Package Records: 1. Client.java**

/\*

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

- Author Rahul Nagaraju

- Assignment: Lab4

- 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();

}

**2. BankRecords.java**

/\*

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

- Author Rahul Nagaraju

- Assignment: Lab4

- 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 implements Serializable{

// 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

protected static 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());

}

}

}

**3. 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();

    }}

**4. 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

}

**Package Models: 1. DaoModel.java**

/\*

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

- Author Rahul Nagaraju

- Assignment: Lab4

- FileName: DaoModel.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package models;

import java.sql.\*;

import records.BankRecords;

public class DaoModel {

// Declare DB objects

DBConnect conn = null;

Statement stmt = null;

// constructor

public DaoModel() { //create db object instance

conn= new DBConnect();

//createTable();

}

// CREATE TABLE METHOD

public void createTable() {

try {

// Open a connection

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

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

// Execute create query

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

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

String sql = "CREATE TABLE r\_naga\_tab " +

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

" id VARCHAR(10), " +

" income numeric(8,2), " +

" pep VARCHAR(3), " +

" PRIMARY KEY ( pid ))";

stmt.executeUpdate(sql);

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

conn.connect().close(); //close db connection

}catch (SQLException e) { // Handle errors for JDBC

e.printStackTrace();

}

}

// INSERT INTO METHOD

public void insertRecords(BankRecords[] robjs) {

try {

// Execute a query

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

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

String sql = "INSERT INTO r\_naga\_tab(id, income, pep) VALUES( ?, ?, ?)";

PreparedStatement insertRecordsStatement = conn.connect().prepareStatement(sql);

// Include all object data to the database table

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

insertRecordsStatement.setString(1, robjs[i].getId());

insertRecordsStatement.setDouble(2, robjs[i].getIncome());

insertRecordsStatement.setString(3, robjs[i].getPep());

insertRecordsStatement.executeUpdate();

}

System.out.println("All Records inserted!");

conn.connect().close();

} catch (SQLException e) {

System.out.println("Error Occurred while inserting the data :"+ e.getMessage());

e.printStackTrace();

}

}

public ResultSet retrieveRecords() {

ResultSet rs = null;

try {

// Attempt to create a statement

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

// Execute the SQL query

// String sql = "SELECT \* from r\_naga\_tab";

String sql="select pid, id,income, pep from r\_naga\_tab order by pep desc";

rs = stmt.executeQuery(sql);

System.out.println("Records retrieved!");

} catch (SQLException e) {

// Handle SQL exceptions

e.printStackTrace();

} finally {

try {

// Close the connection in a finally block to ensure it's always closed

conn.connect().close();

} catch (SQLException e) {

// Handle closing connection exceptions

System.out.println("Error Occurred while retrieving the data :"+ e.getMessage());

e.printStackTrace();

}

}

return rs;

}

public void deleteRecords() {

try{

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

String sql = "DELETE from r\_naga\_tab";

System.out.println("Deleting Records from the table...");

stmt.executeUpdate(sql);

System.out.println("All Records deleted!");

conn.connect().close();

}catch(SQLException e){

System.out.println("Error Occurred while deleting the data :"+ e.getMessage());

e.printStackTrace();

}

}

public void dropTable() {

try {

// Execute the drop table query

System.out.println("Deleting the table...");

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

String sql = "DROP TABLE r\_naga\_tab";

stmt.executeUpdate(sql);

System.out.println("Table 'r\_naga\_tab' deleted successfully.");

// Close the database connection

conn.connect().close();

} catch (SQLException e) {

System.out.println("Error Occurred while deleting the table :"+ e.getMessage());

e.printStackTrace();

}

}

}

**2. DBConnect.java**

/\*

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

- Author Rahul Nagaraju

- Assignment: Lab4

- FileName: DBConnect.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package models;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnect {

// Code database URL

static final String DB\_URL = "jdbc:mysql://www.papademas.net:3307/510labs?autoReconnect=true&useSSL=false";

// Database credentials

static final String USER = "db510", PASS = "510";

public Connection connect() throws SQLException {

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

}

}

**Package Views: LoanView.java**

/\*

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

- Author: Rahul Nagaraju

- Assignment: Lab4

- File: LoanView.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package views;

import java.sql.\*;

import java.util.Vector;

import javax.swing.JFrame;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.table.DefaultTableModel;

public class LoanView {

public void runView(ResultSet rs) {

// Create vector objects to store column and row data for JTable

Vector<Vector<Object>> data = new Vector<Vector<Object>>();

Vector<String> column = new Vector<String>();

try {

// Get metadata of the ResultSet to determine column count

ResultSetMetaData metaData = rs.getMetaData();

int columns = metaData.getColumnCount();

// Get column names from the table and add them to the 'column' vector

for (int i = 1; i <= columns; i++) {

column.add(metaData.getColumnName(i));

}

// Get row data from the table and populate the 'data' vector

while (rs.next()) {

Vector<Object> row = new Vector<Object>(columns);

for (int i = 1; i <= columns; i++) {

row.addElement(rs.getObject(i));

}

data.addElement(row);

}

// Create a DefaultTableModel with the data and column vectors

DefaultTableModel model = new DefaultTableModel(data, column);

// Create a JTable and set its model

JTable table = new JTable(model);

// Create a JFrame to display the JTable

JFrame frame = new JFrame("Loan Details");

frame.setSize(700, 200);

frame.add(new JScrollPane(table));

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE); // Set the default close operation

frame.pack();

frame.setVisible(true);

rs.close(); // Close the ResultSet instance

} catch (SQLException e) {

e.printStackTrace();

}

}

}

**Package Controllers: LoanProcessing.java**

/\*

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

- Author: Rahul Nagaraju

- Assignment: Lab4

- File: LoanProcessing.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package controllers;

import java.sql.ResultSet;

import models.DaoModel;

import records.BankRecords;

import views.LoanView;

public class LoanProcessing extends BankRecords {

public static void main(String args[]) {

ResultSet rs;

// Instantiate a new BankRecords object

BankRecords bankRecords = new BankRecords();

bankRecords.readData(); // Load and process data from a CSV file

// Initialize a Data Access Object (DAO) model

DaoModel dataAccessModel = new DaoModel();

// Create a database table for storing bank records

dataAccessModel.createTable();

// Insert bank records into the database

dataAccessModel.insertRecords(BankRecords.recordObjects);

rs = dataAccessModel.retrieveRecords();

new LoanView().runView(rs);

// Delete all bank records from the database

//dataAccessModel.deleteRecords();

//dataAccessModel.dropTable();

//rs = dataAccessModel.retrieveRecords();

//new LoanView().runView(rs);

}

}

**Extra Credits:- Serialisation and deserialization(snapshots below)  
Package Records: RecordSerialize.java**

/\*

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

- Author Rahul Nagaraju

- Assignment: Lab4

- FileName: RecordSerialize.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package records;

import java.io.\*;

import java.util.HashMap;

import java.util.Map;

public class RecordSerialize extends BankRecords{

public static void main(String[] args) {

// Create a new BankRecords instance

BankRecords br = new BankRecords();

// Read and process data from the CSV file

br.readData();

// Create a map to store BankRecords objects with ID as the key

Map<String, BankRecords> recordsMap = new HashMap<>();

int i=0;

// Iterate through recordObjects and populate the map

for (BankRecords record : recordObjects) {

recordsMap.put(record.getId(), record);

i++;

}

// Display the number of records to be serialized

System.out.println("Serializing "+i+" Records");

// Serialization

long startTime = System.currentTimeMillis();

try (ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("bankrecords.ser"))) {

out.writeObject(recordsMap);// Serialize and write the recordsMap to a file

} catch (IOException e) {

//System.out.println("Error here 1");

e.printStackTrace();

}

System.out.println("Serialized!!");

// Sleep for 5 seconds

try {

System.out.println("Sleeping 5 secs");

Thread.sleep(5000);

} catch (InterruptedException e) {

e.printStackTrace();

}

// Deserialization

long endTime = System.currentTimeMillis();

System.out.println("DeSerializing Records...");

try (ObjectInputStream in = new ObjectInputStream(new FileInputStream("bankrecords.ser"))) {

@SuppressWarnings("unchecked")

Map<String, BankRecords> deserializedRecordsMap = (Map<String, BankRecords>) in.readObject();

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

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

// Perform operations with deserialized data if needed

for (Map.Entry<String, BankRecords> entry : deserializedRecordsMap.entrySet()) {

//System.out.println("ID: " + record.getKey());

//System.out.println("DeSerializing "+i+" Record");

// Performing operations with deserialized data

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

entry.getValue().getId(), entry.getValue().getAge(), entry.getValue().getSex(), entry.getValue().getRegion(), entry.getValue().getIncome(),

entry.getValue().getMortgage());

i--;

}

} catch (IOException | ClassNotFoundException e) {

e.printStackTrace();

}

System.out.println("DeSerialized!!");

// Calculate and display the time difference

long timeDifference = endTime - startTime;

System.out.println("Time Difference (Serialization to Deserialization with sleep 5 secs): " + timeDifference + " ms");

timeDifference-=5000;

System.out.println("Time Difference (Serialization to Deserialization without sleep): " + timeDifference+ " ms");

}

}

Extra credits:-

Time difference is also printed i.e, 5060 ms

A screenshot of a computer

Description automatically generated

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Extra credits :–

Insert Records using prepared statement

A computer screen shot of a code

Description automatically generated

Detailed Loan Analysis Report:

A screenshot of a computer

Description automatically generated

**Package Controllers: LoanProcessing.java**

/\*

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

- Author: Rahul Nagaraju

- Assignment: Lab4

- File: LoanProcessing.java

- Course: ITMD-510 Object-Oriented App Development

- Instructor: James Papademas

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

\*/

package controllers;

import java.io.FileNotFoundException;

import java.io.FileWriter;

import java.io.IOException;

import java.io.UnsupportedEncodingException;

import java.sql.ResultSet;

import java.time.DateTimeException;

import java.time.LocalDateTime;

import java.time.format.DateTimeFormatter;

import java.util.TimeZone;

import models.DaoModel;

import records.BankRecords;

import records.Records;

import views.LoanView;

public class LoanProcessing extends BankRecords {

public static void main(String args[]) {

ResultSet rs;

// Instantiate a new BankRecords object

BankRecords bankRecords = new BankRecords();

bankRecords.readData(); // Load and process data from a CSV file

// Initialize a Data Access Object (DAO) model

DaoModel dataAccessModel = new DaoModel();

// Create a database table for storing bank records

dataAccessModel.createTable();

// Insert bank records into the database

dataAccessModel.insertRecords(BankRecords.recordObjects);

rs = dataAccessModel.retrieveRecords();

new LoanView().runView(rs);

// Delete all bank records from the database

//dataAccessModel.deleteRecords();

//dataAccessModel.dropTable();

//rs = dataAccessModel.retrieveRecords();

//new LoanView().runView(rs);

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");

System.out.println("\nProgrammed by Rahul Nagaraju");

System.out.println(str);

// 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();

}

}

}