Please paste a screenshot of a successful program run, and copy-and-paste the source code from your .java file(s), here.

Your team member names must be a comment on the first line of each Java file.

**Debug.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.util.ArrayList;

/\*\*

\* This class holds methods used in debugging Robert Stepp and Ross Hemphill's

\* CS142 class project (primarily by printing to console to check stored data)

\*

\* @author Ross

\*

\*/

public class debug {

public static void printFilename(String file) {

System.out.println("FILENAME/PATH PASSED: \t" + file);

}

/\*\*

\* this printDates will print an array of LocalDate's with requested format,

\* one per line

\*

\* @param mydates

\* array of LocalDates to print

\* @param format

\* Formatting String for LocalDate/DateTimeFormatter

\*/

public static void printDates(LocalDate[] mydates, String format) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern(format);

System.out.print("DATES PASSED (" + format + "): \t");

for (LocalDate date : mydates) {

System.out.print(date.format(formatter) + ", ");

}

System.out.println();

}

/\*\*

\* this printDates will print a single LocalDate with required formatting

\* and newline

\*

\* @param mydate

\* LocalDate to print

\* @param format

\* Format String for DateTimeFormatter

\*/

public static void printDates(LocalDate mydate, String format) {

DateTimeFormatter formatter = DateTimeFormatter.ofPattern(format);

System.out.println(

"DATE PASSED (" + format + "): \t" + mydate.format(formatter));

}

/\*\*

\* this PrintArray is for an ArrayList of Strings, to be printed with a

\* delimiter after each String and a terminating newline

\*

\* @param al

\* ArrayList of Strings

\* @param delim

\* Delimiter between Strings

\*/

public static void printArray(ArrayList<String> al, String delim) {

System.out.print("ARRAYLIST<STRING> PASSED: ");

for (String s : al)

System.out.print(s + delim);

System.out.println();

}

/\*\*

\* this printArray is for an array of Strings, printed with delimiters and

\* terminating newline

\*

\* @param sa

\* Array of Strings

\* @param delim

\* Delimiter after Springs

\*/

public static void printArray(String[] sa, String delim) {

System.out.print("STRING[] PASSED: ");

for (String s : sa)

System.out.print(s + delim);

System.out.println();

}

/\*\*

\* The boolean is just to differentiate from printArray for

\* ArrayList-String's (vs. ArrayList-String[]'s)

\*

\* @param outtie

\* Our ArrayList of String Arrays (sic)

\* @param delim

\* Delimiter between items printed

\* @param special

\* Boolean value doesn't matter

\*/

public static void printArray(ArrayList<String[]> outtie, String delim,

boolean special) {

// TODO: Either add ranges as a param, or remove "j" conditionals when

// done testing

// int j = 0;

System.out.println("SOME OF ARRAYLIST<STRING[]> PASSED:");

for (String[] innie : outtie) {

// if ((j < 3) || (j > outtie.size() - 4))

printArray(innie, delim);

// ++j;

}

}

}

**Decisions.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.time.LocalDate;

public class decisions {

LocalDate[] daCh;

String[] bpCh;

String tyCh;

decisions(LocalDate[] daCh, String[] bpCh, String tyCh) {

this.daCh = daCh;

this.bpCh = bpCh;

this.tyCh = tyCh;

}

/\*\*

\* @return the daCh

\*/

public LocalDate[] getDaCh() {

return daCh;

}

/\*\*

\* @return the bpCh

\*/

public String[] getBpCh() {

return bpCh;

}

/\*\*

\* @return the tyCh

\*/

public String getTyCh() {

return tyCh;

}

/\*\*

\* @param tyCh

\* the tyCh to set

\*/

public void setTyCh(String tyCh) {

this.tyCh = tyCh;

}

/\*\*

\* @param bpCh

\* the bpCh to set

\*/

public void setBpCh(String[] bpCh) {

this.bpCh = bpCh;

}

/\*\*

\* @param daCh

\* the daCh to set

\*/

public void setDaCh(LocalDate[] daCh) {

this.daCh = daCh;

}

}

**magicTuble.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.time.LocalDate;

import java.util.ArrayList;

// https://stackoverflow.com/questions/19602601/create-an-arraylist-with-multiple-object-types

// I'd like to take this opportunity to thank Eclipse's "generate getters and setters!" :]

class magicTuple {

int[] colPos;

// first String[] is beats opts, second is prec opts

ArrayList<String[]> textVals = new ArrayList<String[]>();

// the extremes of dates represented in the records

LocalDate[] dateVals = new LocalDate[2];

ArrayList<String[]> theRecs = new ArrayList<String[]>();

magicTuple(int[] colPos, ArrayList<String[]> textVals, LocalDate[] dateVals,

ArrayList<String[]> theRecs) {

this.colPos = colPos;

this.textVals = textVals;

this.dateVals = dateVals;

this.theRecs = theRecs;

}

/\*\*

\* @return the colPos

\*/

public int[] getColPos() {

return colPos;

}

/\*\*

\* @return the textVals

\*/

public ArrayList<String[]> getTextVals() {

return textVals;

}

/\*\*

\* @return the dateVals

\*/

public LocalDate[] getDateVals() {

return dateVals;

}

/\*\*

\* @return the theRecs

\*/

public ArrayList<String[]> getTheRecs() {

return theRecs;

}

/\*\*

\* @param colPos

\* the colPos to set

\*/

public void setColPos(int[] colPos) {

this.colPos = colPos;

}

/\*\*

\* @param textVals

\* the textVals to set

\*/

public void setTextVals(ArrayList<String[]> textVals) {

this.textVals = textVals;

}

/\*\*

\* @param dateVals

\* the dateVals to set

\*/

public void setDateVals(LocalDate[] dateVals) {

this.dateVals = dateVals;

}

/\*\*

\* @param theRecs

\* the theRecs to set

\*/

public void setTheRecs(ArrayList<String[]> theRecs) {

this.theRecs = theRecs;

}

}

**options.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.time.LocalDate;

import java.util.LinkedHashMap;

import java.util.Map;

/\*\*

\* I'm currently confusing myself with "How did I create this and then not use

\* it?!" TODO: Is this trash, self? Testing for use for the crime classes? I bet

\* I was feeling hopeful about removing hardcodes by have figured out named

\* sublists by LinkedHashMap's... or sumpin

\*

\* @author Ross

\*

\*/

public class options {

LocalDate[] daRn;

String[] beOp;

String[] prOp;

Map<String, String[]> bpAs = new LinkedHashMap<String, String[]>();

String[] tyOp;

options(LocalDate[] daRn, String[] beOp, String[] prOp,

LinkedHashMap<String, String[]> bpAs, String[] tyOp) {

this.daRn = daRn;

this.beOp = beOp;

this.prOp = prOp;

this.bpAs = bpAs;

this.tyOp = tyOp;

}

/\*\*

\* @return the daRn

\*/

public LocalDate[] getDaRn() {

return daRn;

}

/\*\*

\* @return the beOp

\*/

public String[] getBeOp() {

return beOp;

}

/\*\*

\* @return the prOp

\*/

public String[] getPrOp() {

return prOp;

}

/\*\*

\* @return the bpAs

\*/

public Map<String, String[]> getBpAs() {

return bpAs;

}

/\*\*

\* @return the tyOp

\*/

public String[] getTyOp() {

return tyOp;

}

/\*\*

\* @param daRn

\* the daRn to set

\*/

public void setDaRn(LocalDate[] daRn) {

this.daRn = daRn;

}

/\*\*

\* @param beOp

\* the beOp to set

\*/

public void setBeOp(String[] beOp) {

this.beOp = beOp;

}

/\*\*

\* @param prOp

\* the prOp to set

\*/

public void setPrOp(String[] prOp) {

this.prOp = prOp;

}

/\*\*

\* @param bpAs

\* the bpAs to set

\*/

public void setBpAs(Map<String, String[]> bpAs) {

this.bpAs = bpAs;

}

/\*\*

\* @param tyOp

\* the tyOp to set

\*/

public void setTyOp(String[] tyOp) {

this.tyOp = tyOp;

}

}

**selecARec.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.util.ArrayList;

public class selecARec {

ArrayList<String[]> beatRecs = new ArrayList<String[]>();

ArrayList<String[]> precRecs = new ArrayList<String[]>();

selecARec(ArrayList<String[]> beatRecs, ArrayList<String[]> precRecs) {

this.beatRecs = beatRecs;

this.precRecs = precRecs;

}

/\*\*

\* @return the beatRecs

\*/

public ArrayList<String[]> getBeatRecs() {

return beatRecs;

}

/\*\*

\* @return the precRecs

\*/

public ArrayList<String[]> getPrecRecs() {

return precRecs;

}

/\*\*

\* @param beatRecs

\* the beatRecs to set

\*/

public void setBeatRecs(ArrayList<String[]> beatRecs) {

this.beatRecs = beatRecs;

}

/\*\*

\* @param precRecs

\* the precRecs to set

\*/

public void setPrecRecs(ArrayList<String[]> precRecs) {

this.precRecs = precRecs;

}

}

**WeGotDaBeats.java**

// Java Program by Ross Hemphill & Robert Stepp 11 Feb 2018

package src;

import java.awt.Dimension;

import java.awt.Image;

import java.awt.Toolkit;

import java.io.BufferedReader;

import java.io.FileReader;

import java.io.IOException;

import java.io.LineNumberReader;

import java.time.LocalDate;

import java.time.format.DateTimeFormatter;

import java.time.format.DateTimeParseException;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.Collections;

import java.util.HashMap;

import java.util.LinkedHashMap;

import java.util.List;

import java.util.Map;

import javax.swing.ImageIcon;

import javax.swing.JComboBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javafx.util.Pair;

public class WeGotDaBeat {

/////// 1st stage

/\*\*

\* First interaction with the user. Asks for filepath and uses default

\* passed to it if their entry is blank

\*

\* @param filename

\* The file which'll will be returned if the user enters nothing

\* @return String containing the user's entered filename or, barring that,

\* the default

\*/

public static String input(String filename) {

JTextField preFilename = new JTextField();

preFilename.setText(filename);

JPanel inputFilename = new JPanel();

inputFilename.add(new JLabel("Filename: (case sensitive)"));

inputFilename.add(preFilename);

JOptionPane.showConfirmDialog(null, inputFilename, "Comma Separated Values file (.csv)",

JOptionPane.DEFAULT\_OPTION);

String tempFilename = preFilename.getText();

if (tempFilename.length() > 0)

filename = tempFilename;

return filename;

}

/////// 2nd

/\*\*

\* TODO: Update method documentation

\*

\* structFromStream takes filepath, delimiter, date format, and column

\* keywords of interest. In return, it provides an ArrayList whose

\* first-dimension elements are: 0) the indices of columns of interest 1)

\* all relevant values which occur in those columns, and 2) the line-by-line

\* records themselves

\*

\* @param file

\* String path to the file to parse

\* @param del

\* String delimiter which separates fields

\* @param daFo

\* String date format which LocalDate parses with

\* @param cols

\* String[] containing textual column definitions (from the

\* header)

\* @return An ArrayList of Objects, each of which are super-fun themselves

\* @throws IOException

\*/

public static magicTuple structFromStream(String file, String del, String daFo, String[] cols) throws IOException {

// The columnar positions of the fields we're recording possible values

// for

// Index [0=beat, 1=sector, 2=precinct, 3=date]

// TODO: This is hardcoded and would probably be more appropriately

// represented by a linked list or something

int[] whichCol = { -1, -1, -1, -1 };

// Assumption: first column containing any occurrence of the keyword is

// the one we want

for (int q = 0; q < cols.length; ++q) {

String col = cols[q].toLowerCase();

if ((whichCol[0] == -1) && (col.contains("beat")))

whichCol[0] = q;

else if ((whichCol[1] == -1) && (col.contains("sector")))

whichCol[1] = q;

else if ((whichCol[2] == -1) && (col.contains("precinct")))

whichCol[2] = q;

else if ((whichCol[3] == -1) && (col.contains("date")))

whichCol[3] = q;

}

// For unique or extreme values for the fields we'll present for user

// selection

ArrayList<String> beats = new ArrayList<String>();

ArrayList<String> sectors = new ArrayList<String>();

ArrayList<String> precincts = new ArrayList<String>();

// https://www.mkyong.com/java/how-to-compare-dates-in-java/

// ArrayList<LocalDate> dates = new ArrayList<LocalDate>();

LocalDate[] daDates = new LocalDate[2];

// https://docs.oracle.com/javase/tutorial/datetime/iso/format.html

// Index 0 "is" (will be) earliest date found

daDates[0] = LocalDate.now();

// Index 1 is latest date found

daDates[1] = LocalDate.of(1900, 1, 1);

// currDate is date of the current record

LocalDate currDate = LocalDate.of(1901, 1, 1);

ArrayList<String[]> theRecords = new ArrayList<String[]>();

theRecords.ensureCapacity(numberOfRows(file));

BufferedReader br = new BufferedReader(new FileReader(file));

// Ditch first line (presumed header)

br.readLine();

// Should this be a do-while loop? Not very important

String curLine = br.readLine();

String[] splitLine;

while (curLine != null) {

splitLine = listFromString(curLine, del);

// Discard record if different number of columns than header

if (splitLine.length == cols.length) {

theRecords.add(splitLine);

// Add each unique occurrence of beats, sectors, and precincts

// to our lists of possibilities

if (!(beats.contains(splitLine[whichCol[0]])))

beats.add(splitLine[whichCol[0]]);

if (!(sectors.contains(splitLine[whichCol[1]])))

sectors.add(splitLine[whichCol[1]]);

if (!(precincts.contains(splitLine[whichCol[2]])))

precincts.add(splitLine[whichCol[2]]);

// Turn our date string into a LocalDate

currDate = LocalDate.parse(splitLine[whichCol[3]], DateTimeFormatter.ofPattern(daFo));

// If the current record's date is earlier and/or later than our

// thus-far-seen extremes, it becomes the new extreme(s)

if (currDate.isBefore(daDates[0]))

daDates[0] = currDate;

if (currDate.isAfter(daDates[1]))

daDates[1] = currDate;

}

curLine = br.readLine();

}

br.close();

ArrayList<String[]> textVals = new ArrayList<String[]>();

// Make our lists of textual possibilities "naturally ordered"

// (alphabetical) (We're preferring the order from the file for

// precincts (better reflects cardinal biases, orientation of map image,

// and reading direction))

Collections.sort(beats);

String[] daBeats = beats.toArray(new String[beats.size()]);

Collections.sort(sectors);

String[] daSectors = sectors.toArray(new String[sectors.size()]);

String[] daPrecincts = precincts.toArray(new String[precincts.size()]);

textVals.add(daBeats);

textVals.add(daSectors);

textVals.add(daPrecincts);

magicTuple allTheThings = new magicTuple(whichCol, textVals, daDates, theRecords);

return allTheThings;

}

/\*\*

\* getColDefs finds column definitions from a text file's header

\* (Assumption: single line header.)

\*

\* @param file

\* A String telling us the file's location

\* @param delim

\* The splitting delimiter as a String

\* @return A String ArrayList containing the column headings

\* @throws IOException

\*/

public static String[] getColDefs(String file, String delim) throws IOException {

BufferedReader br = new BufferedReader(new FileReader(file));

// ArrayList<String> cols = new

// ArrayList<>(Arrays.asList(br.readLine().split(delim)));

String[] columns = listFromString(br.readLine(), delim);

br.close();

return columns;

}

/\*\*

\* numberOfRows determines number of rows in a text file (Assumptions:

\* single line header, empty lines are valid (eg as from final newline).)

\*

\* @param file

\* A String of the file's location/name

\* @return An Int holding the number of lines

\* @throws IOException

\*/

public static int numberOfRows(String file) throws IOException {

LineNumberReader lineReader = new LineNumberReader(new FileReader(file));

lineReader.skip(Long.MAX\_VALUE);

int numRows = lineReader.getLineNumber();

lineReader.close();

return numRows;

}

/\*\*

\* listFromString takes a String and returns a list, split by delimiter and

\* escaped by double quotes. Also strips double quotes. (Assumption: double

\* quote is the escape character and the delimiter's a different character.

\* Untested: del being more than one character.)

\*

\* @param line

\* The String to be listified

\* @param del

\* The (non-double-quote) delimiter we're splitting on

\* @return The split String as a list

\*/

public static String[] listFromString(String line, String del) {

// https://stackoverflow.com/questions/15738918/splitting-a-csv-file-with-quotes-as-text-delimiter-using-string-split

// Split on del except within double quotes

String[] list = line.split(del + "(?=([^\"]|\"[^\"]\*\")\*$)");

for (int q = 0; q < list.length; ++q)

// Replace double quotes with nothing

list[q] = list[q].replace("\"", "");

return list;

}

/////// 3rd

/\*\*

\* getChoices (via submethods) obtains from the user their choices after

\* relevant options have been obtained from the data

\*

\* @param dateForm

\* The formatting String for parsing and display of dates

\* @param dateLimits

\* An LocalDate ArrayList with (0) the earliest permissible date

\* and (1) the latest permissible date

\* @param imgLoc

\* Path to the image (map) to be displayed

\* @param beatOpts

\* A String ArrayList of beat options

\* @param precOpts

\* String ArrayList of precinct options

\*/

public static decisions getChoices(String dateForm, LocalDate[] dateLimits, String imgLoc,

LinkedHashMap<String, String[]> bpAssoc, String[] beatOpts, String[] precOpts, String[] typeOpts) {

// Get user's chosen min/max dates

LocalDate[] daChoice = getDateRange(dateForm, dateLimits);

// Show user the beat(/sector)/precinct map to aid their areal choosings

displayMap(imgLoc);

// Get user's chosen beat and precinct

String[] bpChoice = getBeatPrecinct(bpAssoc, beatOpts, precOpts);

// Get user's crime type of focus

// GOODGOODGOOD

String tyChoice = getTypeOfCrime(typeOpts);

// Return the decisions/choices of all our submethods

decisions choices = new decisions(daChoice, bpChoice, tyChoice);

return choices;

}

/\*\*

\* getDateRange uses a dialog box to obtain from the user the range of dates

\* to consider. Informs of the range available to them and, incidentally, of

\* the formatting required. (Strict matching of format, including

\* superfluous zeroes.)

\*

\* @param dateFormat

\* A String determining the date format printed and expected

\* @param dateLims

\* ArrayList of LocalDate's. First element is earliest

\* permissible date, second latest

\*/

public static LocalDate[] getDateRange(String dateForm, LocalDate[] dateLims) {

LocalDate[] datesChosen = new LocalDate[2];

datesChosen[0] = LocalDate.MAX;

datesChosen[1] = LocalDate.MIN;

DateTimeFormatter formatter = DateTimeFormatter.ofPattern(dateForm);

String[] dateLimsS = new String[2];

boolean allIsWell = false;

while (!allIsWell) {

boolean begParsFailed = false, endParsFailed = false;

boolean anyParsFailed = false;

boolean begInRng = false, endInRng = false;

boolean begAfterEnd = true;

JPanel dPanel = new JPanel();

dPanel.add(new JLabel("Beginning date:"));

JTextField beginChoice = new JTextField();

dateLimsS[0] = dateLims[0].format(formatter);

beginChoice.setText(dateLimsS[0]);

dPanel.add(beginChoice);

dPanel.add(new JLabel("Ending date:"));

JTextField endChoice = new JTextField();

dateLimsS[1] = dateLims[1].format(formatter);

endChoice.setText(dateLimsS[1]);

dPanel.add(endChoice);

JOptionPane.showConfirmDialog(null, dPanel, "Timeframe of interest", JOptionPane.DEFAULT\_OPTION);

try {

// System.out.println("Begin choice: " + beginChoice.getText());

datesChosen[0] = LocalDate.parse(beginChoice.getText(), formatter);

} catch (Exception e) {

begParsFailed = true;

anyParsFailed = true;

}

try {

// System.out.println("End choice: " + endChoice.getText());

datesChosen[1] = LocalDate.parse(endChoice.getText(), formatter);

} catch (Exception e) {

endParsFailed = true;

anyParsFailed = true;

}

if (datesChosen[0].isAfter(datesChosen[1]))

System.out.println("That's not proper!");

if (anyParsFailed) {

if (begParsFailed && endParsFailed) {

JOptionPane.showConfirmDialog(null, "Unparseable dates.\nPlease enter in the format " + dateForm,

"Error", JOptionPane.DEFAULT\_OPTION);

} else {

if (begParsFailed)

JOptionPane.showConfirmDialog(null,

"Unparseable beginning date.\nPlease enter in the format " + dateForm, "Error",

JOptionPane.DEFAULT\_OPTION);

else if (endParsFailed)

JOptionPane.showConfirmDialog(null,

"Unparseable ending date.\nPlease enter in the format " + dateForm, "Error",

JOptionPane.DEFAULT\_OPTION);

}

} else {

if (!datesChosen[0].isBefore(dateLims[0]))

begInRng = true;

else

JOptionPane.showConfirmDialog(null,

"Beginning date too early.\nCan't be earlier than " + dateLimsS[0] + ".", "Error",

JOptionPane.DEFAULT\_OPTION);

if (!datesChosen[1].isAfter(dateLims[1]))

endInRng = true;

else

JOptionPane.showConfirmDialog(null,

"Beginning date too late.\nCan't be later than " + dateLimsS[1] + ".", "Error",

JOptionPane.DEFAULT\_OPTION);

if (!datesChosen[0].equals(LocalDate.MAX))

if (!datesChosen[1].equals(LocalDate.MIN))

if (!datesChosen[0].isAfter(datesChosen[1]))

begAfterEnd = false;

else

JOptionPane.showConfirmDialog(null, "Dates are in reversed order.", "Error",

JOptionPane.DEFAULT\_OPTION);

}

if (begInRng && endInRng)

if (!(begAfterEnd || anyParsFailed))

allIsWell = true;

}

return datesChosen;

}

/\*\*

\* Displays the Beat Map resized down.

\*\*/

public static void displayMap(String mapLoc) {

// https://alvinalexander.com/blog/post/jfc-swing/how-set-jframe-size-fill-entire-screen-maximize

// This reservedHeight is just enough for my high-DPI 13" Macbook and

// will be fine for the school computers

int reservedHeight = 45, scaleNumerator = 1, scaleDenominator = 1;

JFrame frame = new JFrame();

Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();

ImageIcon icon = new ImageIcon(new ImageIcon(mapLoc).getImage().getScaledInstance(-1,

(screenSize.height - reservedHeight) \* scaleNumerator / scaleDenominator, Image.SCALE\_SMOOTH));

frame.setSize(-1, (screenSize.height - reservedHeight) \* scaleNumerator / scaleDenominator);

frame.setTitle("Precinct and Beat options:");

JLabel label = new JLabel(icon);

frame.add(label);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.pack();

frame.setVisible(true);

}

/\*\*

\* getBeatPrecinct obtains from the user their beat and precinct preferences

\* via a dialog with dropdowns. It will only accept a beat which is within

\* the precinct chosen.

\*

\* @param bpAssoc

\*

\* @param theBeats

\* A String[] of all the beat possibilities

\* @param thePrecincts

\* A String[] of all the precinct possibilities

\* @return

\*/

public static String[] getBeatPrecinct(LinkedHashMap<String, String[]> bpAssoc, String[] theBeats,

String[] thePrecincts) {

boolean theWorldIsGood = false;

String bc = "", pc = "";

while (!theWorldIsGood) {

JPanel bpPanel = new JPanel();

JComboBox<?> prec = new JComboBox<Object>(thePrecincts);

bpPanel.add(new JLabel("Precinct:"));

bpPanel.add(prec);

JComboBox<?> be = new JComboBox<Object>(theBeats);

bpPanel.add(new JLabel("Beat:"));

bpPanel.add(be);

JOptionPane.showConfirmDialog(null, bpPanel, "Area of interest (Beat must be within precinct)",

JOptionPane.DEFAULT\_OPTION);

bc = theBeats[be.getSelectedIndex()];

pc = thePrecincts[prec.getSelectedIndex()];

if (Arrays.asList(bpAssoc.get(pc)).contains(bc))

theWorldIsGood = true;

}

String[] bpChoice = { pc, bc };

return bpChoice;

}

/\*\*

\* Ask the user the type of crime they would like to focus on

\*

\* @param types

\* Array of Strings containing the general types of crimes

\* available

\*/

public static String getTypeOfCrime(String[] types) {

JPanel tPanel = new JPanel();

JComboBox<?> type = new JComboBox<Object>(types);

tPanel.add(new JLabel("Type requested:"));

tPanel.add(type);

JOptionPane.showConfirmDialog(null, tPanel, "Crime class of interest", JOptionPane.DEFAULT\_OPTION);

return types[type.getSelectedIndex()];

}

/////// 4th

/\*\*

\* Perhaps "select records" would be a better name. Anyway, take the data

\* obtained, combine with the decisions made and parse format specified...

\* and return two ArrayLists of String[] records, one which matches

\* everything with just precinct for area, the second exactly everything,

\* down to beat

\*

\* @param allDat

\* A magicTuple containing our parsed data

\* @param thCh

\* A decisions containing choices made by the user

\* @param dateForm

\* A String indicating the date parse/format layout

\* @return A selecARec consisting of two sets of records as

\* ArrayList-String[]'s

\*/

private static selecARec cullRecords(magicTuple allDat, decisions thCh, String dateForm,

LinkedHashMap<String, String[]> crimeCateg) {

int dateCol = allDat.colPos[3], precCol = allDat.colPos[2], beatCol = allDat.colPos[0];

LocalDate minDate = thCh.daCh[0], maxDate = thCh.daCh[1];

String precReq = thCh.bpCh[0], beatReq = thCh.bpCh[1];

ArrayList<String[]> beatRecs = new ArrayList<String[]>();

ArrayList<String[]> precRecs = new ArrayList<String[]>();

for (String[] aRecord : allDat.theRecs) {

LocalDate recDate = LocalDate.parse(aRecord[dateCol], DateTimeFormatter.ofPattern(dateForm));

if (!((recDate.isAfter(maxDate)) || (recDate.isBefore(minDate))))

// Danger Will Robinson, hardcoded column position for

// CRIME\_TYPE! (aRecord[1])

if (Arrays.asList(crimeCateg.get(thCh.tyCh)).contains(aRecord[1]))

if (aRecord[precCol].equals(precReq)) {

precRecs.add(aRecord);

if (aRecord[beatCol].equals(beatReq))

beatRecs.add(aRecord);

}

}

selecARec recBag = new selecARec(beatRecs, precRecs);

return recBag;

}

// https://stackoverflow.com/questions/29920027/how-can-i-sort-a-list-of-pairstring-integer

private static void countOcc(LinkedHashMap<String, Integer> crSev, selecARec arealOcc, String[] whichCol,

int crCl) {

ArrayList<String[]> beatRecs = arealOcc.beatRecs;

ArrayList<String[]> precRecs = arealOcc.precRecs;

LinkedHashMap<String, Integer> weightOccBeat = new LinkedHashMap<String, Integer>();

LinkedHashMap<String, Integer> weightOccPrec = new LinkedHashMap<String, Integer>();

// Person is crCl=0

weightOccBeat.put("Homicide", 0);

weightOccBeat.put("Rape", 0);

weightOccBeat.put("Robbery", 0);

weightOccBeat.put("Assault", 0);

weightOccPrec.put("Homicide", 0);

weightOccPrec.put("Rape", 0);

weightOccPrec.put("Robbery", 0);

weightOccPrec.put("Assault", 0);

// Property is 1

weightOccBeat.put("Arson", 0);

weightOccBeat.put("Burglary", 0);

weightOccBeat.put("Larceny-Theft", 0);

weightOccBeat.put("Motor Vehicle Theft", 0);

weightOccPrec.put("Arson", 0);

weightOccPrec.put("Burglary", 0);

weightOccPrec.put("Larceny-Theft", 0);

weightOccPrec.put("Motor Vehicle Theft", 0);

int definedWeight = 0;

int recOccs = 0;

int existTyScore = 0;

int newTyScore = 0;

// String recOccT = "";

// String recNum = "";

String recCrimeType = "";

for (String[] rec : beatRecs) {

// recNum = rec[7];

// recOccT = rec[3];

//

//

// System.out.println("" + definedWeight + "\t" + recNum + "\t" +

// recOccT);

recCrimeType = rec[1];

existTyScore = weightOccBeat.get(rec[1]);

newTyScore = existTyScore + (recOccs \* definedWeight);

System.out.println(recCrimeType + "\t\t" + newTyScore);

definedWeight = crSev.get(recCrimeType);

recOccs = Integer.parseInt(rec[3]);

weightOccBeat.put(recCrimeType, weightOccBeat.get(recCrimeType) + (recOccs \* definedWeight));

}

}

/\*\*

\*

\* @param crSev

\* @param records

\* @param whichCol

\* @param pePr

\* If 0 return person, if else return property

\* @param bePr

\* If 0, return beat, if else return precinct

\*/

private static LinkedHashMap<String, Integer> cntOcc(LinkedHashMap<String, Integer> crSev, selecARec records,

String[] whichCol, int pePr, int bePr) {

boolean deb = false;

LinkedHashMap<String, Integer> ita = new LinkedHashMap<String, Integer>();

if (bePr == 0) {

ArrayList<String[]> beatRecs = records.beatRecs;

LinkedHashMap<String, Integer> beatOcc = new LinkedHashMap<String, Integer>();

if (pePr == 0) {

beatOcc.put("Homicide", 0);

beatOcc.put("Rape", 0);

beatOcc.put("Robbery", 0);

beatOcc.put("Assault", 0);

} else {

beatOcc.put("Burglary", 0);

beatOcc.put("Larceny-Theft", 0);

beatOcc.put("Motor Vehicle Theft", 0);

}

int prevOcc = 0;

int nowOcc = 0;

// int totOcc = 0;

String crimeType = "";

for (String[] rec : beatRecs) {

crimeType = rec[1];

prevOcc = beatOcc.get(crimeType);

if (deb) {

System.out.print(crimeType + "\t");

System.out.println("Prior to now, this many accumed:" + prevOcc);

}

nowOcc = Integer.parseInt(rec[3]);

if (deb)

System.out.println("This many occ now: " + nowOcc);

beatOcc.put(crimeType, (prevOcc + nowOcc));

nowOcc = beatOcc.get(crimeType);

if (deb)

System.out.println("Adding up: " + nowOcc);

}

int tval = 0;

int value = 0;

String key = "";

int wval = 0;

// https://stackoverflow.com/questions/12310914/how-to-iterate-through-linkedhashmap-with-lists-as-values

for (Map.Entry<String, Integer> entry : beatOcc.entrySet()) {

key = entry.getKey();

value = entry.getValue();

tval = crSev.get(key);

wval = tval \* value;

beatOcc.put(key, wval);

if (deb)

System.out.println(tval + "\t" + value + "\t" + beatOcc.get(key));

}

ita = beatOcc;

} else {

ArrayList<String[]> precRecs = records.precRecs;

LinkedHashMap<String, Integer> precOcc = new LinkedHashMap<String, Integer>();

if (pePr == 0) {

precOcc.put("Homicide", 0);

precOcc.put("Rape", 0);

precOcc.put("Robbery", 0);

precOcc.put("Assault", 0);

} else {

precOcc.put("Burglary", 0);

precOcc.put("Larceny-Theft", 0);

precOcc.put("Motor Vehicle Theft", 0);

}

int prevOcc = 0;

int nowOcc = 0;

// int totOcc = 0;

String crimeType = "";

for (String[] rec : precRecs) {

crimeType = rec[1];

prevOcc = precOcc.get(crimeType);

if (deb) {

System.out.print(crimeType + "\t");

System.out.println("Prior to now, this many accumed:" + prevOcc);

}

nowOcc = Integer.parseInt(rec[3]);

if (deb)

System.out.println("This many occ now: " + nowOcc);

precOcc.put(crimeType, (prevOcc + nowOcc));

nowOcc = precOcc.get(crimeType);

if (deb)

System.out.println("Adding up: " + nowOcc);

}

int tval = 0;

int value = 0;

String key = "";

int wval = 0;

// https://stackoverflow.com/questions/12310914/how-to-iterate-through-linkedhashmap-with-lists-as-values

for (Map.Entry<String, Integer> entr : precOcc.entrySet()) {

key = entr.getKey();

value = entr.getValue();

tval = crSev.get(key);

wval = tval \* value;

precOcc.put(key, wval);

if (deb)

System.out.println(tval + "\t" + value + "\t" + precOcc.get(key));

}

ita = precOcc;

}

if (deb)

System.out.println("END");

return ita;

}

public static void displayRecs(LinkedHashMap<String, Integer> listOfCrimeScores, String key) {

System.out.printf("%-10s: %d crime prevalence score\n", key, listOfCrimeScores.get(key));

}

///////

///////

public static void main(String[] args) throws IOException {

// Defaults/constants/hardcodes

String filename = "not-src/Seattle\_Crime\_Stats\_by\_Police\_Precinct\_2008-Present.csv";

String mapLoc = "not-src/beat-map-2.png";

String delimiter = ",";

String dateFormat = "M/d/yyyy";

// Beats are within precincts (we're ignoring sectors)

LinkedHashMap<String, String[]> pb = new LinkedHashMap<String, String[]>();

String[] n = { "B1", "B2", "B3", "J1", "J2", "J3", "L1", "L2", "L3", "N1", "N2", "N3", "U1", "U2", "U3" };

pb.put("N", n);

String[] w = { "D1", "D2", "D3", "K1", "K2", "K3", "M1", "M2", "M3", "Q1", "Q2", "Q3" };

pb.put("W", w);

String[] e = { "C1", "C2", "C3", "E1", "E2", "E3", "G1", "G2", "G3" };

pb.put("E", e);

String[] se = { "O1", "O2", "O3", "R1", "R2", "R3", "S1", "S2", "S3" };

pb.put("SE", se);

String[] sw = { "F1", "F2", "F3", "W1", "W2", "W3" };

pb.put("SW", sw);

// Two general classes of crime

LinkedHashMap<String, String[]> cc = new LinkedHashMap<String, String[]>();

String[] crimeClasses = { "Person", "Property" };

String[] persCrim = { "Homicide", "Rape", "Robbery", "Assault" };

cc.put(crimeClasses[0], persCrim);

String[] propCrim = { "Arson", "Burglary", "Larceny-Theft", "Motor Vehicle Theft" };

cc.put(crimeClasses[1], propCrim);

// Proportionate severity per crime

// See

// http://www.pewtrusts.org/en/research-and-analysis/issue-briefs/2016/03/the-punishment-rate

// and https://www.bjs.gov/index.cfm?ty=pbdetail&iid=2045

// ("Table 11. First releases from state prison, 2009: Sentence length

// and time served in prison, by offense and race")

// https://www.geeksforgeeks.org/pair-class-in-java/

LinkedHashMap<String, Integer> myMap = new LinkedHashMap<String, Integer>();

myMap.put("Homicide", 119);

myMap.put("Rape", 96);

myMap.put("Robbery", 52);

myMap.put("Assault", 31);

myMap.put("Arson", 38);

myMap.put("Burglary", 26);

myMap.put("Larceny-Theft", 17);

myMap.put("Motor Vehicle Theft", 19);

////

final boolean DEBUG = false;

////

// BEGIN user interaction

// Obtain from user which file has the records (which will provide our

// data and the parameters of our options)

filename = input(filename);

String[] colDefs = getColDefs(filename, delimiter);

// Now that we know our header, we can parse our file into structured

// data

magicTuple allDat = structFromStream(filename, delimiter, dateFormat, colDefs);

// With our structured data, we can ask the user their choice among the

// possibilities

decisions thCh = getChoices(dateFormat, allDat.dateVals, mapLoc, pb, allDat.getTextVals().get(0),

allDat.getTextVals().get(2), crimeClasses);

// Choices made, we can now cull our records

selecARec chosenOness = cullRecords(allDat, thCh, dateFormat, cc);

// END user interaction

// HACK 0=person!

int crCl = 0;

if (thCh.tyCh.equals("Property"))

crCl = 1;

System.out.println("Below follows data to assist you in determining the suitability of different areas for locating. \n" +

"The more severe the crime, the greater an impact a single occurrence has. \n" +

"We follow the methodology established by the Pew Charitable Trusts.");

System.out.println();

System.out.println("The weighted significances of criminal occurences within your chosen beat and timeframe:");

LinkedHashMap<String, Integer> beatWeigOcc = new LinkedHashMap<String, Integer>();

beatWeigOcc = cntOcc(myMap, chosenOness, colDefs, crCl, 0);

String key;

for (Map.Entry<String, Integer> entr : beatWeigOcc.entrySet()) {

key = entr.getKey();

displayRecs(beatWeigOcc, key);

}

System.out.println();

System.out.println("Compared to the same measure across the whole precinct:");

LinkedHashMap<String, Integer> precWeigOcc = new LinkedHashMap<String, Integer>();

precWeigOcc = cntOcc(myMap, chosenOness, colDefs, crCl, 1);

key = "";

for (Map.Entry<String, Integer> entr : precWeigOcc.entrySet()) {

key = entr.getKey();

displayRecs(precWeigOcc, key);

}

System.out.println();

System.out.println("Good luck with your choice, and thanks for utilizing our service!");

if (DEBUG) {

System.out.println("###INITIAL###");

src.debug.printDates(allDat.getDateVals(), dateFormat);

src.debug.printArray(allDat.getTextVals().get(0), ",");

src.debug.printArray(allDat.getTextVals().get(1), ",");

src.debug.printArray(allDat.getTextVals().get(2), ",");

System.out.println();

src.debug.printArray(colDefs, " \t ");

System.out.println("=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=");

// src.debug.printArray(allDat.getTheRecs(), " \t ", true);

System.out.println();

System.out.println("###EFFECTIVE###");

src.debug.printFilename(filename);

System.out.println();

System.out.println("###CHOSEN###");

src.debug.printDates(thCh.getDaCh(), dateFormat);

// src.debug.printArray(thCh.bpCh, ", ");

System.out.println("TYPE CHOSEN: " + thCh.tyCh);

src.debug.printArray(chosenOness.beatRecs, "\t", true);

// src.debug.printArray(chosenOness.precRecs, "\t", true);

}

}

}