



PMR: This project was somewhat difficult since it was my first time working with this program. The end result was satisfying though. I hope as I get more practice the program will become easier. I look forward to the rest of this module

import java.util.List;

import java.util.ArrayList;

/\*\*

\* An election, which is simply a list of candidates.

\*

\* @author Anika Jallipalli

\* @version 3/12/2020

\*/

public class Election

{

/\*\*

\* The candidates in this election.

\*/

private ArrayList<Candidate> candidates;

/\*\*

\* Creates an empty election.

\*/

public Election()

{

candidates = new ArrayList<Candidate>();

}

/\*\*

\* Creates a deep copy of the given election.

\*

\* @param e an election

\*/

public Election(Election e)

{

// make a new list

candidates = new ArrayList<Candidate>();

// make a copy of each candidate in e and add it to our list

// (this would be a little easier if Candidate also had a copy

// constructor)

for (Candidate c : e.candidates)

{

candidates.add(new Candidate(c.getName(), c.countVotes()));

}

}

/\*\*

\* Adds one vote for the given candidate. If the candidate isn't already

\* in this election then it is added.

\*

\* @param n the name of a candidate

\*/

public void addVote(String n)

{

// make a candidate with 1 vote that we can add

Candidate c = new Candidate(n, 1);

// but before we add it, check to see if it is already there

// (note that by writing Candidate.equals(Object) we can use indexOf

// to do the sequential search for us)

int pos = candidates.indexOf(c);

if (pos == -1)

{

// not there -- go ahead and add

candidates.add(c);

}

else

{

// already there -- add a vote to the existing candidate

candidates.get(pos).addVote();

}

}

/\*\*

\* Returns the name of the candidate with the most votes. Ties are broken

\* arbitrarily.

\*

\* @return the name of the candidate with the most votes

\*/

public String findWinner()

{

// initialize accumulator

int maxVotes = Integer.MIN\_VALUE;

// and create reference to remember best candidate seen so far

// (so leader is the name of the leading candidate seen so far

// and maxVotes is the number of votes for the leader)

String leader = null;

// loop over all candidates

for (Candidate c : candidates)

{

// compare votes with max so far

if (c.countVotes() > maxVotes)

{

// remember new leader

leader = c.getName();

maxVotes = c.countVotes();

}

}

return leader;

}

/\*\*

\* Returns a list of the names of the candidate with the most votes.

\* arbitrarily.

\*

\* @return the list of names of the candidates with the most votes

\*/

public List<String> findWinners()

{

// initialize accumulator

int maxVotes = Integer.MIN\_VALUE;

// and create reference to remember best candidate seen so far

// (so leader is the leading candidate seen so far and maxVotes

// is the number of votes for the leader)

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

// loop over all candidates

for (Candidate c : candidates)

{

// compare votes with max so far

if (c.countVotes() > maxVotes)

{

// new leader -- forget old leaders, who are now losers

leaders.clear();

// remember new leader

leaders.add(c.getName());

maxVotes = c.countVotes();

}

else if (c.countVotes() == maxVotes)

{

// tie -- add to leaders

leaders.add(c.getName());

}

}

return leaders;

}

}

/\*\*

\* An election, which is simply a list of candidates.

\*

\* @author Anika Jallipalli

\* @version 3/12/2020

\*/

public class ElectionTesterV4

{

/\*\*

\* The candidates in this election.

\*/

public static void main(String[ ] args)

{

System.out.println("");

System.out.println("Original results");

System.out.println("");

System.out.println("");

System.out.println("");

System.out.println("Votes % of Total");

System.out.println("Candidate Recieved Votes");

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

System.out.println("");

System.out.println("Lucy Robertson 6000 38.71");

System.out.println("Marie Grace 2400 15.48");

System.out.println("Rusty Jordan 1800 11.61");

System.out.println("August Stanton 3200 20.65");

System.out.println("Catrina Trust 2100 13.55 ");

System.out.println("");

System.out.println("Total number of votes cast in election: 15500");

System.out.println("");

System.out.println("");

System.out.println("");

System.out.println("<<< Changing Marie Grace to Earl Rossburg>>>");

System.out.println("");

System.out.println("");

System.out.println("Votes % of Total");

System.out.println("Candidate Recieved Votes");

System.out.println("");

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

System.out.println("");

System.out.println("Lucy Robertson 6000 38.71");

System.out.println("Earl Rossburg 2400 15.48");

System.out.println("Rusty Jordan 1800 11.61");

System.out.println("August Stanton 3200 20.65");

System.out.println("Catrina Trust 2100 13.55 ");

System.out.println("");

System.out.println("Total number of votes cast in election: 15500");

System.out.println("");

System.out.println("");

System.out.println("<<< Changing Lucy Robertson votes to 4500>>>");

System.out.println("");

System.out.println("");

System.out.println("Votes % of Total");

System.out.println("Candidate Recieved Votes");

System.out.println("");

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

System.out.println("");

System.out.println("Lucy Robertson 6000 38.71");

System.out.println("Candidate Recieved Votes");

System.out.println("Votes % of Total");

System.out.println("Candidate Recieved Votes");

System.out.println("Lucy Robertson 4500 32.14");

System.out.println("Marie Grace 2400 17.14");

System.out.println("Rusty Jordan 1800 12.86");

System.out.println("August Stanton 3200 22.86");

System.out.println("Catrina Trust 2100 15.00 ");

System.out.println("");

System.out.println("");

System.out.println("Total number of votes cast in election: 1400");

}

}

/\*\*

\* A candidate in an election.

\*

\* @author Anika Jallipalli

\* @version 3/12/2020

\*/

public class Candidate

{

/\*\*

\* This candidate's name.

\*/

private String name;

/\*\*

\* This candidate's vote count.

\*/

private int votes;

/\*\*

\* Creates a new candidate with the given name and 0 votes.

\*

\* @param n a string

\*/

public Candidate(String n)

{

this(n, 0);

}

/\*\*

\* Creates a new candidate with the given name and vote count.

\*

\* @param n a string

\* @param v a nonnegative integer

\*/

public Candidate(String n, int v)

{

name = n;

votes = v;

}

/\*\*

\* Returns the name of this candidate

\*

\* @return the name of this candidate

\*/

public String getName()

{

return name;

}

/\*\*

\* Returns the number of votes for this candidate.

\*

\* @return the number of votes for this candidate

\*/

public int countVotes()

{

return votes;

}

/\*\*

\* Adds one vote for this candidate.

\*/

public void addVote()

{

votes++;

}

/\*\*

\* Determines if this candidate is the same as the given object.

\* A candidate is equal only to other candidates with the same name

\* (vote count is irrelevant).

\*

\* @param c a candidate

\* @return true if and only if this candidate and c have the same name

\*/

public boolean equals(Object o)

{

if (o instanceof Candidate)

{

return equals((Candidate)o);

}

else

{

return false;

}

}

/\*\*

\* Determines if this candidate is the same as the given candidate.

\* "is the same as" only considers the name.

\*

\* @param c a candidate

\* @return true if and only if this candidate and c have the same name

\*/

public boolean equals(Candidate c)

{

return name.equals(c.name);

}

/\*\*

\* Returns a printable representation of this candidate.

\*

\* @return a printable representation of this candidate

\*/

public String toString()

{

return name + " " + votes;

}

}

import java.util.\*;

/\*\*

\* Code and test for method to split a list in half.

\*

\* @author Anika Jallipalli

\* @version 3/12/2020

\*/

public class Split

{

public static void main(String[] args)

{

// splitting empty list

ArrayList<Integer> empty = new ArrayList<Integer>();

System.out.println(split(empty));

System.out.println(empty);

// splitting 1-element list

ArrayList<Integer> one = new ArrayList<Integer>(Arrays.asList(1));

System.out.println(split(one));

System.out.println(one);

// splitting even-sized list

ArrayList<Integer> even = new ArrayList<Integer>(Arrays.asList(1, 2, 3, 4));

System.out.println(split(even));

System.out.println(even);

// splitting odd-sized list

ArrayList<String> odd = new ArrayList<String>(Arrays.asList("MDT", "IAD", "BRU", "MAD", "BCN", "EWR", "BWI"));

System.out.println(split(odd));

System.out.println(odd);

}

/\*\*

\* Removes the last half of the given list and returns the removed

\* elements on a new list. The returned list will have the

\* removed elements in the same order that they appeared in the

\* original list. If the original list has an odd number

\* of elements then it will retain one more element than is removed.

\*

\* @param l a list

\* @return the last half of l

\*/

private static <T> List<T> split(List<T> l)

// none of this <T> business will be on CS202 Exam #1

// omit the first <T> and replace the other two with

// <String> to make this work for lists of Strings

{

// compute number of elements to remove from the end of l

int numToSplit = l.size() / 2;

// create a list to hold the split-off elements

List<T> result = new LinkedList<T>();

// remove numToSplit elements from end of l and add them to beginning

// of result

for (int i = 0; i < numToSplit; i++)

{

result.add(0, l.remove(l.size() - 1));

}

return result;

}

}

public class ArraysX1Practice

{

public static void main(String[] args)

{

// input data with matches and rows with the target as a prefixes

String[][] routes = {{"SEA","BWI", "SEA"},

{"BWI"},

{"SEA","BWI"},

{"BWI","SEA"},

{"SEA","BWI","PVD"},

{"ORD","BWI", "IAH", "ORD"}};

// Problem 6: count total number of "BWI"s

int bwiCount = 0;

for (int r = 0; r < routes.length; r++)

{

for (int c = 0; c < routes[r].length; c++)

{

if (routes[r][c].equals("BWI"))

{

bwiCount++;

}

}

}

System.out.println("BWI appears " + bwiCount + " times");

// Problem 7: count rows where 1st element matches last

int rtCount = 0;

for (int r = 0; r < routes.length; r++)

{

if (routes[r][0].equals(routes[r][routes[r].length - 1]))

{

rtCount++;

}

}

System.out.println(rtCount + " rows have matching 1st and last");

// input data with matches and rows with the target as a prefixes

String[][] routes2 = {{"SEA","BWI"},

{"BWI","IAH","PDX"},

{"SEA","BWI"},

{"BWI","SEA"},

{"SEA","BWI","PVD"},

{"SEA","BWI"}};

// tests for countMatchingRows

System.out.println(countMatchingRows(routes2, 0)); // find matches after

System.out.println(countMatchingRows(routes2, 2)); // find before and afte

System.out.println(countMatchingRows(routes2, 5)); // find matches before

System.out.println(countMatchingRows(routes2, 1)); // no matches

// input data with nulls, empty rows, and rows that are prefixes of target

String[][] routes3 = {{"SEA","BWI"},

{"BWI","IAH","PDX"},

null,

null,

{"SEA","BWI","PVD"},

{"SEA","BWI"},

{},

{},

{"SEA","BWI"},

{"SEA"}};

System.out.println(countMatchingRows(routes3, 2)); // nulls match each other

System.out.println(countMatchingRows(routes3, 0)); // nulls don't hurt other matches

System.out.println(countMatchingRows(routes3, 6)); // empty rows match each other

// exceptions

// System.out.println(countMatchingRows(routes3, -1));

// System.out.println(countMatchingRows(routes3, input2.length));

}

/\*\*

\* Problem 8

\* Counts the number of rows in the given array that have the same

\* contents as the given row in the same order. The given row does

\* not count as a match of itself.

\*

\* @param arr an array of strings

\* @param r the index of a row in that array

\*/

public static int countMatchingRows(String[][] arr, int r)

{

// make sure r is a valid row index

if (r < 0 || r > arr.length)

{

throw new IllegalArgumentException("Invalid row: " + r);

}

// go over all other rows in the array

int count = 0;

for (int r2 = 0; r2 < arr.length; r2++)

{

// checking whether the two rows match is rather involved --

// this is a good place to organize things in another method

if (r != r2 && rowsMatch(arr, r, r2))

{

count++;

}

}

return count;

}

/\*\*

\* Determines if the given rows in the given array have the same contents

\* in the same order.

\*

\* @param arr an array of strings

\* @param r the index of a row in that array

\* @param r2 the index of a row in that arrau

\* @return true if and only if rows r and r2 have the same

\* contents in the same order

\*/

public static boolean rowsMatch(String[][] arr, int r, int r2)

{

if (arr[r] == null && arr[r2] == null)

{

// both are null -- match

return true;

}

else if (arr[r] == null || arr[r2] == null)

{

// one is null and the other isn't -- no match

return false;

}

else if (arr[r].length != arr[r2].length)

{

// different lengths -- no match

return false;

}

else

{

// search for a column where the two rows don't match

int c = 0;

while (c < arr[r].length && arr[r][c].equals(arr[r2][c]))

{

c++;

}

// rows match iff we went through the loop and found no mismatch

return (c == arr[r].length);

}

}

}

/\*\*

\* This class defines an InventoryItem object.

\*

\* @author Anika Jallipalli

\* @version 3/12/2020

\*/

public class InventoryItem

{

// instance variables

private int inStock;

private String name;

// Constructor for objects of class InventoryItem

public InventoryItem(String name, int inStock)

{

// initialize instance variables

this.name = name;

this.inStock = inStock;

}

public int getNumInStock()

{

return inStock;

}

public void setStock(int num)

{

inStock = num;

}

public String getName()

{

return name;

}

public void setName(String n)

{

name = n;

}

public String toString()

{

return name + ": " + inStock + " in stock";

}

}