

PROGRAMMING EXAMPLE: Election Results

Your local university is going to hold an election for president of the student council. For purposes of confidentiality, the election committee wants to computerize the analysis of the voting. The committee is looking for someone to write a program to analyze the data and to report the winner. Let's write a program to help the election committee.

The university has four major divisions. For the election, the four divisions are labeled as region 1, region 2, region 3, and region 4. The voting is reported directly to the election committee in the following form:

```
firstName lastName regionNumber numberOfVotes
```

The election committee wants the output in the following tabular form:

```
-----Election Results-----
```

Candidate Name	Rgn#1	Votes By Region			Total
		Rgn#2	Rgn#3	Rgn#4	
Sheila Bower	23	70	133	267	493
Danny Dillion	25	71	156	97	349
.					
.					
.					

```
Winner: ???, Votes Received: ???
Total votes polled: ???
```

The names of the candidates must be in alphabetical order in the output.

For this program, we assume that six candidates are seeking the student council's president post. This program can be enhanced to handle any number of candidates.

The data is provided in two files. One file, `candData.txt`, consists of the names of the candidates seeking the president's post. The names of the candidates in this file are in no particular order. In the second file, `voteData.txt`, each line consists of the voting results in the following form:

```
firstName lastName regionNumber numberOfVotes
```

That is, each line in the file `voteData.txt` consists of the candidate's name, the region number, and the number of votes received by the candidate in that region. There is one entry per line. For example, the input file containing the voting data looks like the following:

```

Amar Gupta 2 34
Mike Rizvi 1 56
Lisa Fisher 2 56
Peter Lamba 1 78
Danny Dillion 4 29
Sheila Bower 4 78
.
.
.

```

The first line indicates that **Amar Gupta** received **34** votes from region **2**.

Input: Two files: One containing the candidates' names, and the other containing the voting data as described previously

Output: The election results in a tabular form, as described previously, and the winner

PROBLEM ANALYSIS AND CLASS DESIGN

From the output, it is clear that the program must organize the voting data by region and calculate the total votes received for each candidate and polled for the election. Furthermore, the names of the candidates must appear in alphabetical order.

The main component of this program is a candidate. Therefore, first we design the **class** **Candidate** to implement a candidate object.

Candidate

The main component of this program is the candidate, which is described and implemented in this section. Every candidate has a first and a last name and receives votes. Because there are four regions, we declare an array of four components to keep track of the votes for each region. We also need a data member to store the total number of votes received by each candidate. Essentially, the members of the **class** **Candidate** are:

Instance Variables and Constants

```

private final int NO_OF_REGIONS = 4;

private String firstName;
private String lastName;
private int[] votesByRegion;
private int totalVotes;

```

Constructors and Instance Methods

```

Candidate()
    //Default constructor
    //Postcondition: firstName = " "; lastName = " "
    //              Creates the array votesByRegion and
    //              sets totalVotes to zero.

```

```

public void setName(String first, String last)
    //Method to set firstName and lastName according to
    //the parameters.
    //Postcondition: firstName = first; lastName = last;

public String getFirstName()
    //Method to return the firstName.
    //Postcondition: The value of firstName is returned.

public String getLastName()
    //Method to return the lastName.
    //Postcondition: the value of lastName is returned

public void setVotes(int region, int votes)
    //Method to set the votes of a candidate for a
    //particular region.
    //Postcondition: The votes specified by the parameter
    //                  votes are assigned to the region
    //                  specified by the parameter region.

public void updateVotesByRegion(int region, int votes)
    //Method to update the votes of a candidate for a
    //particular region.
    //Postcondition: The votes specified by the parameter votes
    //                  are added to the region specified by the
    //                  parameter region.

public void calculateTotalVotes()
    //Method to calculate the total votes received by a
    //candidate.
    //Postcondition: The votes received in each region are added.

public int getTotalVotes()
    //Method to return the total votes received by a
    //candidate.
    //Postcondition: The total votes received by the candidate
    //                  are returned.

public void printData()
    //Method to output the candidate's name, the votes
    //received in each region, and the total votes received.

public boolean equals(Candidate otherCan)
    //Method to determine if two candidates have the same
    //name.
    //Postcondition: Returns true if the name of this
    //                  candidate is the same as the name of
    //                  the candidate specified by the parameter
    //                  otherCan.

```

```

public int compareTo(Candidate otherCan)
    //Method to compare the names of two candidates.
    //Postcondition: Returns 0 if the name of this
    //                candidate is the same as the name of
    //                otherCan; returns < 0 if the name of
    //                this candidate is less then the name
    //                of otherCan; returns > 0 if the name
    //                of this candidate is greater than the
    //                name of otherCan.

```

Figure E-1 shows the UML diagram of the `class` `Candidate`.

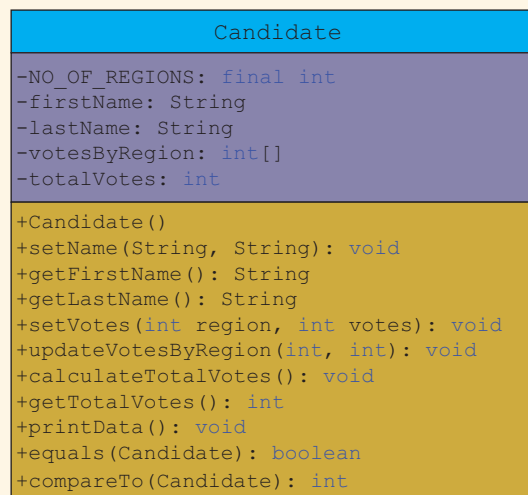


FIGURE E-1 UML class diagram of `class` `Candidate`

The definitions of the methods of the `class` `Candidate` are given next.

The default constructor initializes first name and last name to empty strings, creates the array to hold vote data, and sets `totalVotes` to 0. Its definition is:

```

Candidate()
{
    firstName = "";
    lastName = "";
    totalVotes = 0;
    votesByRegion = new int[NO_OF_REGIONS];
}

```

From the description, the methods `setName`, `getFirstName`, `getLastName` are straightforward. Their definitions are:

```

public void setName(String first, String last)
{
    firstName = first;
    lastName = last;
} //end setName

public String getFirstName()
{
    return firstName;
} //end getFirstName

public String getLastName()
{
    return lastName;
} //end getLastName

```

To set the votes of a particular region, the region number and the number of votes are passed as parameters to the method `setVotes`. Because an array index starts at 0, region 1 corresponds to the array component at position 0, and so on. Therefore, to set the value of the correct array component, 1 is subtracted from the `region`. The definition of the method `setVotes` is:

```

public void setVotes(int region, int votes)
{
    votesByRegion[region - 1] = votes;
} //end setVotes

```

To update the votes for a particular region, the region number and the number of votes for that region are passed as parameters. Then the votes are added to the region's previous value. The definition of the method `updateVotesByRegion` is:

```

public void updateVotesByRegion(int region, int votes)
{
    votesByRegion[region - 1] = votesByRegion[region - 1]
        + votes;
} //end updateVotesByRegion

```

The definitions of the methods `calculateTotalVotes`, `getTotalVotes`, and `printData` are given next.

```

public void calculateTotalVotes()
{
    totalVotes = 0;

    for (int i = 0; i < NO_OF_REGIONS; i++)
        totalVotes = totalVotes + votesByRegion[i];
} //end calculateTotalVotes

public int getTotalVotes()
{
    return totalVotes;
} //end getTotalVotes

public void printData()
{
    System.out.printf("%-14s ", firstName + " " + lastName);
}

```

```

        for (int i = 0; i < NO_OF_REGIONS; i++)
            System.out.printf("%6d ", votesByRegion[i]);

        System.out.printf("%5d\n", totalVotes);
    } //end printData

```

Next we give the definition of the methods `equals` and `compareTo`.

```

public boolean equals(Candidate otherCan)
{
    return (firstName.equals(otherCan.firstName)
        && lastName.equals(otherCan.lastName));
} //end equals

public int compareTo(Candidate otherCan)
{
    if (lastName.equals(otherCan.lastName))
        return (firstName.compareTo(otherCan.firstName));
    else
        return (lastName.compareTo(otherCan.lastName));
} //end compareTo

```

We can write the definition of the `class` `Candidate` as follows:

```

//*****
// Author: D.S. Malik
//
// This class implements a candidate as an object.
//*****

public class Candidate
{
    private final int NO_OF_REGIONS = 4;

    private String firstName;
    private String lastName;

    private int[] votesByRegion;
    private int totalVotes;

    //Place the definition of the default constructor and the
    //definitions of the methods, setName, getFirstName,
    //getLastName, setVotes, updateVotesByRegion,
    //calculateTotalVotes, getTotalVotes, printData, equals,
    //and compareTo here.
}

```

MAIN PROGRAM

Now that the `class` `Candidate` has been designed, we focus on designing the main program.

Because there are six candidates, we create a list, `candidateList`, containing six components of the type `Candidate`. The first thing that the program should do is read each candidate's name from the file `candData.txt` into the list `candidateList`. Once the candidates' names are stored in the array `candidateList`, we must sort this array.

The next step is to process the voting data from the file `voteData.txt`, which holds the voting data. After processing the voting data, the program should calculate the total votes received by each candidate and then print the data as shown previously. Thus, the general algorithm is:

1. Read each candidate's name into `candidateList`.
2. Sort `candidateList`.
3. Process the voting data.
4. Calculate the total votes received by each candidate.
5. Print the results.

The following statement creates the array `candidateList`.

```
Candidate[] candidateList = new Candidate[NO_OF_CANDIDATES];
```

Figure E-2 shows the object `candidateList`. Every component of the array `candidateList` is an object of `Candidate` type.

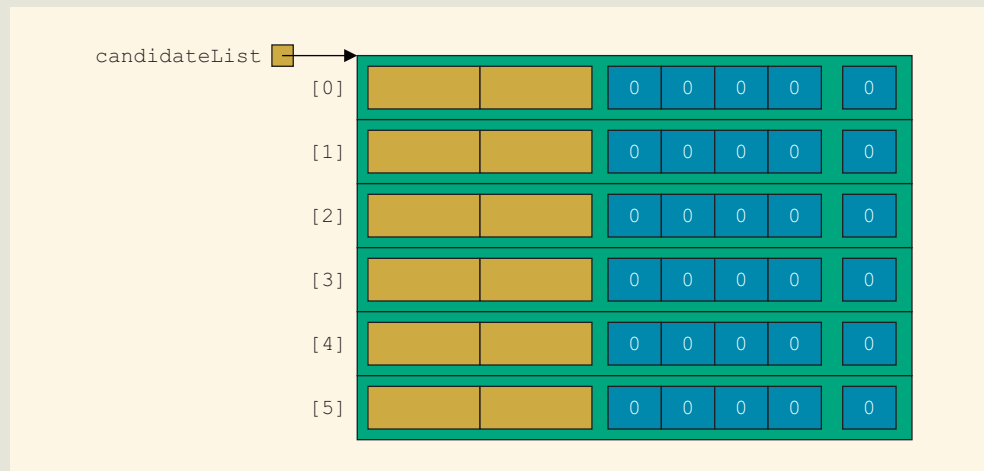


FIGURE E-2 `candidateList`

In Figure E-2, the array `votesByRegion` and the variable `totalVotes` are initialized to 0 by the default constructor of the `class` `Candidate`.

fillNames The first thing that the program must do is to read the candidates' names into `candidateList`. Therefore, we write a method to accomplish this task. The file `candData.txt` is opened in the method `main`. The name of the input file and `candidateList` are therefore passed as parameters to the method `fillNames`. The definition of the method `fillNames` is as follows:

```
public static void fillNames(Scanner inFile, Candidate[] cList,
                             int numOfCand)
{
    String firstN;
    String lastN;

    Candidate temp;

    for (int i = 0; i < numOfCand; i++)
    {
        firstN = inFile.next();
        lastN = inFile.next();
        temp = new Candidate();
        temp.setName(firstN, lastN);
        cList[i] = temp;
    }
} //end fillNames
```

Figure E-3 shows the `candidateList` after a call to the method `fillNames`.

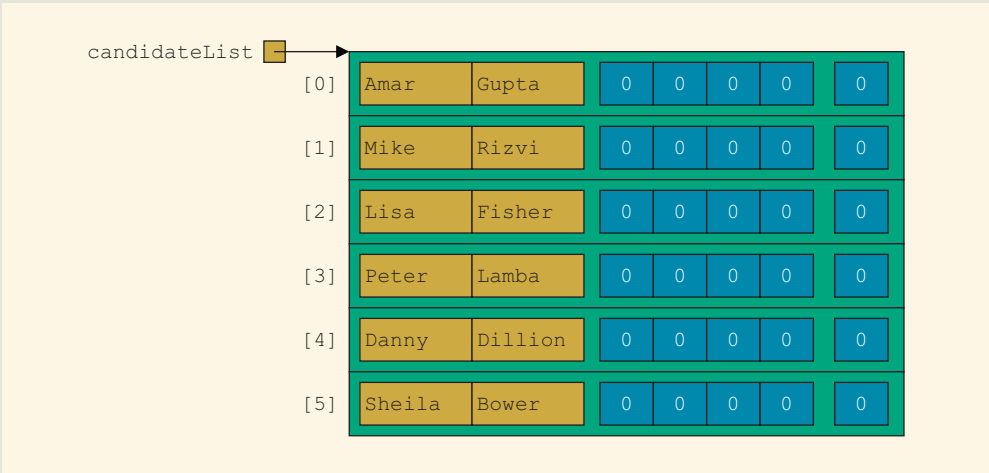


FIGURE E-3 Object `candidateList` after a call to the method `fillNames`

To save space in Figure E-3, we show the name of the candidate in the orange box, rather than draw arrows pointing to `String` objects containing the first name and last name.

Method This method sorts the array `candidateList`. This method has two parameters: a
insertion parameter corresponding to the array `candidateList` and another parameter to
Sort specify the length of the array. Essentially, this method is:


```

public static void insertionSort(Candidate[] list, int length)
{
    for (int firstOutOfOrder = 1; firstOutOfOrder < length;
        firstOutOfOrder++)
    {
        Candidate compElem = list[firstOutOfOrder];

        if (compElem.compareTo(list[firstOutOfOrder - 1]) < 0)
        {
            Candidate temp = list[firstOutOfOrder];

            int location = firstOutOfOrder;

            do
            {
                list[location] = list[location - 1];
                location--;
            }
            while (location > 0 &&
                temp.compareTo(list[location - 1]) < 0);

            list[location] = temp;
        }
    }
}
//end insertionSort

```

After a call to this method, the arrays are as shown in Figure E-4.

candidateList →

[0]	Sheila	Bower	0	0	0	0	0
[1]	Danny	Dillion	0	0	0	0	0
[2]	Lisa	Fisher	0	0	0	0	0
[3]	Amar	Gupta	0	0	0	0	0
[4]	Peter	Lamba	0	0	0	0	0
[5]	Mike	Rizvi	0	0	0	0	0

FIGURE E-4 Array candidateList after sorting names

Process Now we discuss how to process the voting data. Each entry in the file
Voting `voteData.txt` is of the form:
Data

```
firstName lastName regionNumber numberOfVotes
```

After reading an entry from the file `voteData.txt`, we locate the row in the array `candidateList` corresponding to the specific candidate, and update the entry specified by `regionNumber`. Now the array `candidateList` is sorted according to the `lastName` and `firstName`. So we do a binary search to find the row of this array for the specific candidate. The definition of the methods `processVotes` and `binarySearch` are:

```
public static void processVotes(Scanner inFile,
                               Candidate[] cList)
{
    String firstN;
    String lastN;
    int region;
    int votes;
    int candLocation;

    Candidate temp;

    while (inFile.hasNext())
    {
        firstN = inFile.next();
        lastN = inFile.next();
        region = inFile.nextInt();
        votes = inFile.nextInt();

        temp = new Candidate();
        temp.setName(firstN, lastN);
        temp.setVotes(region, votes);

        candLocation = binarySearch(cList, 6, temp);

        if (candLocation != -1)
        {
            temp = cList[candLocation];
            temp.updateVotesByRegion(region, votes);
        }
    }
} //end processVotes

public static int binarySearch(Candidate[] list, int length,
                              Candidate searchItem)
{
    int first = 0;
    int last = length - 1;
    int mid = 0;
```

```

boolean found = false;
while (first <= last && !found)
{
    mid = (first + last) / 2;

    Candidate compElem = list[mid];


    if (compElem.compareTo(searchItem) == 0)
        found = true;
    else if (compElem.compareTo(searchItem) > 0)
        last = mid - 1;
    else
        first = mid + 1;
}

if (!found)
    mid = -1;

return mid;
} //end binarySearch

```

Figure E-5 shows `candidateList` after processing votes.

candidateList 

[0]	Sheila	Bower	23	70	133	267	0
[1]	Danny	Dillion	25	71	156	97	0
[2]	Lisa	Fisher	110	158	0	0	0
[3]	Amar	Gupta	75	34	134	0	0
[4]	Peter	Lamba	285	56	0	46	0
[5]	Mike	Rizvi	120	141	156	67	0

FIGURE E-5 `candidateList` after processing votes

Add Votes After processing the voting data, the next step is to find the total votes received by each candidate. This is done by adding the votes received in each region. The definition of this method is:


```

public static void addVotes(Candidate[] cList, int numofCand)
{
    Candidate temp;

    for (int i = 0; i < numofCand; i++)
        cList[i].calculateTotalVotes();
} //end addVotes

```

Figure E-6 shows `candidateList` after adding the votes for each candidate—that is, after a call to the method `addVotes`.

candidateList 

[0]	Sheila	Bower	23	70	133	267	493
[1]	Danny	Dillion	25	71	156	97	349
[2]	Lisa	Fisher	110	158	0	0	268
[3]	Amar	Gupta	75	34	134	0	243
[4]	Peter	Lamba	285	56	0	46	387
[5]	Mike	Rizvi	120	141	156	67	476

FIGURE E-6 `candidateList` after a call to the method `addVotes`

Print Heading and Print Results To complete the program, we include a method to **print** the heading, which is the first four lines of the output. The following method accomplishes this task:

```

public static void printHeading()
{
    System.out.println("-----Election Results"
        + "-----\n");
    System.out.println("          "
        + "Votes By Region");
    System.out.println("Candidate Name  Rgn#1 \tRgn#2 \t"
        + "Rgn#3 \tRgn#4 \tTotal");
    System.out.println("----- \t----- "
        + "\t----- \t----- \t-----");
} //end printHeading

```

Now we describe the method `printResults`, which prints the results. Suppose that the variable `sumVotes` holds the total votes polled for the election, the variable `largestVotes` holds the largest number of votes received by a candidate, and the variable `winLoc` holds the index of the winning candidate in the array list. The algorithm for this method is:

1. Initialize `sumVotes`, `largestVotes`, and `winLoc` to 0.
2. For each candidate:
 - a. Access the candidate's data.
 - b. Print the candidate's name and relevant data.
 - c. Retrieve the total votes received by the candidate and update `sumVotes`.
3. Output the final lines of the output.

The definition of the method `printResults` is:

```
public static void printResults(Candidate[] cList, int numOfCand)
{
    int largestVotes = 0;
    int sumVotes = 0;
    int winLoc = 0;

    for (int i = 0; i < numOfCand; i++)
    {
        cList[i].printData();

        sumVotes = sumVotes + cList[i].getTotalVotes();

        if (largestVotes < cList[i].getTotalVotes())
        {
            largestVotes = cList[i].getTotalVotes();
            winLoc = i;
        }
    }

    System.out.print("\nWinner: ");
    System.out.print(cList[winLoc].getFirstName() + " "
        + cList[winLoc].getLastName());
    System.out.println(", Votes Received: "
        + cList[winLoc].getTotalVotes());
    System.out.println("\nTotal votes polled: " + sumVotes);
} //end printResults
```

To separate the definition of the method `main` from the definitions of other methods, we place the definitions of the methods `fillNames`, `insertionSort`, `binarySearch`, `processVotes`, `addVotes`, `printHeading`, and `printResults` in the `class ElectionResult`. Because these methods are public and static, we can call these methods using the name of the class and the dot operator. You can write the definition of the `class ElectionResult` as follows:

```
//*****
// Author: D.S. Malik
//
// This class contains methods for processing voting data for
// student council president's post.
//*****
```

```
import java.util.*;

public class ElectionResult
{
    //Place the definitions of the methods fillNames,
    //insertionSort, binarySearch, processVotes, addVotes,
    //printHeading, and printResults here.
}
```

PROGRAM LISTING (MAIN PROGRAM)

```
//*****
// Author: D.S. Malik
//
// This program processes voting data for student council
// president's post. It lists each candidate's name and the votes
// they received. The name of the winner is also printed.
//*****

import java.io.*;
import java.util.*;

public class TestProgElectionResult
{
    static final int NO_OF_CANDIDATES = 6;

    public static void main(String[] args) throws
        FileNotFoundException
    {
        Candidate[] candidateList =
            new Candidate[NO_OF_CANDIDATES];

        Candidate temp;

        Scanner inFile = new Scanner
            (new FileReader("candData.txt"));

        ElectionResult.fillNames(inFile, candidateList,
            NO_OF_CANDIDATES);

        ElectionResult.insertionSort(candidateList,
            NO_OF_CANDIDATES);

        inFile = null;

        inFile = new Scanner(new FileReader("voteData.txt"));

        ElectionResult.processVotes(inFile, candidateList);

        ElectionResult.addVotes(candidateList,
            NO_OF_CANDIDATES);

        ElectionResult.printHeading();
        ElectionResult.printResults(candidateList,
            NO_OF_CANDIDATES);
    } // end main
}
```

Sample Run:

-----Election Results-----

Candidate Name	Rgn#1	Votes By Region			Total
		Rgn#2	Rgn#3	Rgn#4	
Sheila Bower	23	70	133	267	493
Danny Dillion	25	71	156	97	349
Lisa Fisher	110	158	0	0	268
Amar Gupta	75	34	134	0	243
Peter Lamba	285	56	0	46	387
Mike Rizvi	112	141	156	67	476

Winner: Sheila Bower, Votes Received: 493

Total votes polled: 2216

Input Files: The files `candData.txt` and `voteData.txt` are provided in the Additional Student Files folder at www.cengagebrain.com, and the CD accompanying this book.