COS 126 **General Computer Science** **Spring 2014**

Programming Exam 2

This exam is semi-open: you may read the course website, the booksite, any direct links from those two, the textbook, any printed or written notes, and your past coursework on your computer. You may not use other websites. You must not post, write, or send information to anywhere. No other communication is permitted, except with course staff members. **Do not remove this exam from the exam room. Return it to course staff at the end of the exam.**

Upload your code to the dropbox. As with the COS 126 assignments, you can submit your code multiple times for testing, but only the last version will be graded.

Your code will be graded primarily on correctness. You will lose a substantial number of points if your program does not compile or has the wrong API. However, efficiency, clarity, and code style are also factors in your grade. Remember to include headers and comments in your code. Headers MUST include your name, netID and precept number.

Print your name, netID and precept number on this page (now), and write out and sign the Honor Code pledge before turning in this paper. It is a violation of the Honor Code to discuss this exam until everyone in the class has taken the exam. You have 90 minutes to complete the test.

Write out and sign the Honor Code pledge before turning in the test:
"I pledge my honor that I have not violated the Honor Code during this examination."

Pledge: ______________________________

____

Signature: ____________________

Name: ____________________

NetID: ____________________

Precept: ____________________

P01	12:30 TTh	David Pritchard
P01A	12:30 TTh	Donna Gabai
P01B	12:30 TTh	Aleksey Boyko
P02	1:30 TTh	Borislav Hristov
P02A	1:30 TTh	Terry Yannan Wang
P02B	1:30 TTh	Aleksey Boyko
P02C	1:30 TTh	Nanxi Kang
P02D	1:30 TTh	Xinyi Fan
P03	2:30 TTh	Borislav Hristov
P03A	2:30 TTh	Bebe Shi
P04	3:30 TTh	Kevin Lee
P04A	3:30 TTh	Victor Shaoqing Yang
P05	7:30 TTh	Kevin Lee
P06	10:00 WF	Mojgan Ghasemi
P07	11:00 WF	Mojgan Ghasemi
P08	12:30 WF	Donna Gabai
P08A	12:30 WF	Maia Ginsburg
P09	1:30 WF	David Pritchard
P09A	1:30 WF	Maia Ginsburg
P09B	1:30 WF	Judi Israel
P10	2:30 WF	Judi Israel

Part	**Value**	**Score**
1	18	
2A	7	
2B	5	
Total	**30**	

In this exam, you will create two Java classes to help run an election using *reverse instant runoff voting* (RIRV). This mechanism has *preferential ballots*: each voter ranks all candidates from best to worst, and the mechanism defines a rule for eliminating candidates until a majority of voters agree on who should win. This will be explained in more detail in part 2.

After completing each part and testing it on your own computer, you can test it online. Go to the Assignments page (refresh the page if necessary) and click on the Submit link for this exam. Examples, tests, and a sample client are described in the exam text. You can copy them from:

`http://www.cs.princeton.edu/~cos126/docs/data/Vote/`

Part 1: `Ballot` (18 points)

In this part of this exam you will define a data type `Ballot` that represents a voter's ballot in a RIRV election. When a voter fills out a ballot, they rank all candidates from best (favorite) to worst (least favorite). The constructor takes an array of strings representing this ranking, with the favorite at the start of the array and the least favorite at the end. The API is:

```
public Ballot(String[] prefs) // make preferential ballot (order: favorite to least)
public String toString()      // give string representation e.g. "Alice > Bob > Eve"
public String top()           // which remaining candidate is our favorite?
public String bottom()        // which remaining candidate is our least favorite?
public void cut(String name)  // remove candidate; throw RuntimeException if not on ballot
```

Constructor: Read all of part 1, including the top half of the next page, before deciding on what type(s) of instance variable(s) to use. Additionally,

- Any correct implementation is ok, you do not have to try to minimize the running time.
- If you use an array-based implementation, you should avoid exposing your private variables, by making a defensive copy in the constructor.
- You may assume that ballots always contain at least 1 candidate (are not empty) at all times.
- You may assume that all candidates have distinct names.

toString(): The `toString()` method must list all remaining candidates, with adjacent candidates separated by spaces and a `>` sign. For example,

```
String[] testPrefs = {"Doug", "Nanxi", "Bebe", "Aleksey"};
Ballot b = new Ballot(testPrefs);
System.out.println(b); // calls toString(). gives "Doug > Nanxi > Bebe > Aleksey"
```

top()/bottom(): The methods `top()` and `bottom()` should return the favorite and least favorite remaining candidates. E.g.,

```
System.out.println(b.top()); // gives "Doug"
System.out.println(b.bottom()); // gives "Aleksey"
```

cut(): Between rounds, candidates will be eliminated by calling `cut()`. E.g.,

```
b.cut("Doug");
System.out.println(b); // gives "Nanxi > Bebe > Aleksey"
System.out.println(b.top()); // gives "Nanxi" - note change from original top()
```

Note: `toString()`, `top()` and `bottom()` must take into account which people have been cut!

Anyone may be cut, including people who are in the middle of the ballot.

```
b.cut("Bebe");
System.out.println(b); // gives "Nanxi > Aleksey"
```

- You **must** throw a `RuntimeException` in `cut()` if the candidate does not exist like `b.cut("Scruffy")`, or if they were already cut from the ballot like calling `b.cut("Bebe")` a second time.
- You may assume we never try to `cut()` all candidates — ballots will never become empty.
- If you compare `String`s in `cut()`, remember to use `.equals()` instead of the `==` comparator.

Including a test `main()` method in your submission is optional; a sample one is provided for you at the URL listed earlier. **It includes all the tests listed earlier**, you do not need to type them in. *You can test your part 1 code online before starting part 2.* Don't upload `Stack.java`, `Queue.java`, or `ST.java` — the grader already has these files. If you are stuck on Part 1, you can still complete other parts of the exam — dropbox will also run your `Election` with a working version of `Ballot`.

Part 2: `Election` (12 points)

You will write a library `Election` of static methods for processing ballots in an election. It is not a data type — just a library of static methods. Here is the API; details are on the next page.

```
public class Election
// these methods should ALL be public AND static
int maxValue(ST<String, Integer> st)    // find max value in symbol table
ST<String, Integer> counts(Ballot[] a, boolean countTop) // get top or bottom counts
String winner(Ballot[] a)               // winner's name, or null if no majority yet
String loser(Ballot[] a)                // which person has the most bottom ballots?
```

As the API suggests, the abstraction that we are using for an election is an array of `Ballot` objects (`Ballot[] a`), representing one `Ballot` for each voter. The rules for a RIRV election are:

- Once a majority (more than half) of voters agree on the top candidate, that candidate wins the election.
- However, if no candidate has a majority of top votes, the candidate with the most bottom votes is cut from all ballots. (For the purposes of this exam, you may assume that ties never occur; we will never test any case where there would be any tie in any iteration.)

Here is code that implements these rules using the `Ballot` and `Election` APIs:

```
Ballot[] a;
// ... read the initial ballots into a ...
while (Election.winner(a) == null) {
  String elim = Election.loser(a);
  StdOut.println("Eliminating " + elim);
  for (Ballot b : a)
    b.cut(elim);
}
StdOut.println("The winner is " + Election.winner(a));
```

This code is used to define a test client `RIRV.java` that is already completed for you, and posted at the URL mentioned earlier.

The 4 methods that you must implement in the `Election` class are broken into parts 2A and 2B.

Part 2A (7 points) – Helper Methods

1. `public static int maxValue(ST<String, Integer> st)`

   `maxValue()` returns the maximum value in the symbol table. You may assume that all values are nonnegative, and that the symbol table is not empty (contains at least one key-value pair). When implementing this method, you will need to use an "enhanced for loop."

2. `public static ST<String, Integer> counts(Ballot[] a, boolean countTop)`

   When `countTop` is `true`, the `counts()` method creates a symbol table whose keys are candidates, and whose values count the number of times that each candidate occurred as the `top()` of a ballot in `a`. When `countTop` is `false`, it should do the same with the `bottom()` of each ballot. It need not have entries for candidates that appear zero times.

Part 2B (5 points) – `winner()` and `loser()`

Finally, define the methods `winner()` and `loser()` that control the election.

3. `public static String winner(Ballot[] a)`

   This method should consider all ballots in the array `a`. **If strictly more than half** of them agree on the same `top()` preference, `winner()` should return their name. Otherwise, if nobody is top of more than half, `winner()` should return `null`.

4. `public static String loser(Ballot[] a)`

   This method should return the candidate that occurs in `a` most often as a `bottom()` preference. You may assume there is never a tie — we will not test any such scenario.

Use the Part 2A methods as you see fit to help you in Part 2B. You can optionally include private methods. Here is one called `find()`; it is online for you to optionally copy and paste.

```
// search ST for target value. return associated key. exception if not found.
private static String find(ST<String, Integer> st, int target) {
    for (String key : st) { // enhanced for loop -- iterates through all keys
        if (st.get(key)==target)
            return key;
    }
    throw new RuntimeException("Not found!");
}
```

Including a test `main()` method in your submission is optional.

Test cases for these 4 methods appear at the URL given earlier, and also on the next page. We recommend using those tests first. After, we recommend you run the `RIRV` test client on the sample input files provided online; see the `RIRV.java` header for instructions on how to use it (and how to use it in "debug" mode).

Remember to submit your completed code online and to use the "Check All Submitted Files" button. Do not upload `ST.java` or `RIRV.java` — the grader already has these files.

Here is a copy of the tests for part 2 that appear online.

```
public static void main(String[] args) {
    // test maxValue()
    ST<String, Integer> st = new ST<String, Integer>();
    st.put("someKey", 5);
    st.put("anotherKey", 8);
    st.put("thirdKey", 6);
    StdOut.println("max value should be 8: " + maxValue(st));
    StdOut.println();

    // create 3 ballots with these preferences
    String[] prf1 = {"Claude", "David", "Ada", "Bjarne"};
    String[] prf2 = {"Bjarne", "Ada", "Claude", "David"};
    String[] prf3 = {"Ada", "David", "Claude", "Bjarne"};
    Ballot[] a = {new Ballot(prf1), new Ballot(prf2), new Ballot(prf3)};

    // test counts()
    ST<String, Integer> topCounts = counts(a, true);
    StdOut.println("topCounts - should be Ada 1, Bjarne 1, Claude 1: ");
    for (String cand : topCounts)
        StdOut.println(cand + " " + topCounts.get(cand));
    StdOut.println();

    ST<String, Integer> bottomCounts = counts(a, false);
    StdOut.println("bottomCounts - should be Bjarne 2, David 1: ");
    for (String cand : bottomCounts)
        StdOut.println(cand + " " + bottomCounts.get(cand));
    StdOut.println();

    // test winner() when no majority
    StdOut.println("winner() should be null, actually: " + winner(a));

    // test loser()
    StdOut.println("loser() should be Bjarne, actually: " + loser(a));

    StdOut.println("eliminating Bjarne");
    for (Ballot b : a)
        b.cut("Bjarne");

    /* at this point, the ballots should be
     * Claude > David > Ada
     * Ada > Claude > David
     * Ada > David > Claude                */

    // test winner() with majority
    StdOut.println("winner() should be Ada, actually: " + winner(a));
}
```

This is the same set of ballots used by the `RIRV` input file `sample.txt`. A few other sample runs will be provided online including `strictMajority.txt`, which checks that only a *strict* majority is accepted.