SOFTWARE TESTING 2 FINAL EXAM

This is the final draft of the final exam. The exam is due Sunday night, December 7, at 11:59 pm.
I will not accept late submissions.

In this exam, you will create components of the VoteTest system specified in the National Institute of Science & Technology's beta draft specification, *VoteTest: Voting system logic testing* (August 12, 2008). A copy of that specification is in Angel, at the top of the *Course Materials* page.

Take a look at Section 6, starting on Page 102, paying particular attention to the definitions of classes for

- Political parties
- Choices
- Contests
- BallotStyles (ballot structure)
- Ballots (voted ballots)

You will have to skim the standard before Page 102 to understand Chapter 6 (beginning in page 102). Your system must include these classes.

OVERVIEW

You will create elections, and for each election, a large set of ballots. Think of these as if they were the actual pieces of paper marked up by voters. These make up the set of authoritative materials, the "truth" (the oracle) for each election. Some ballots will contain overvotes (too many choices marked in a contest) and some will contain undervotes (no choices marked, or too few marked).

You will (virtually) feed the ballots, in a random order, into a (virtual) scanner that interprets each ballot and adjusts its counts, contest by contest, of who has how many votes. Your scanners will sometimes make reading errors, or will reject choices that would be obvious to a human but confusing to a scanner (such as a choice that is crossed out, with an alternative choice. Your scanners might make random errors or they might make biased (election-rigging) errors. Your tests must recognize discrepancies between the totals obtained from scanning and the real totals (the ones you would get if you tabulated ballots perfectly). Your scanner must flag any ballot in which a contest was undervoted or overvoted, in a way that allows you to retrieve the ballot and examine it. (That is, you should be able to search your list for all the ballots marked as undervoted or overvoted.)

Because this is an experiment and not an election, you will be able to determine what order your ballots were actually fed into the scanner, and so you can always know what the real ballot says. When a scanner says that a ballot has a problem (undervote or overvote in at least one contest), you can ask the scanner what it sees as the votes in each contest on the ballot and compare them to the "real" ballot.

You will generate reports that give the election results, including overvotes and undervotes for each contest, and discrepancy reports showing all of the mismatches between the scanned ballot results and the actual ballots fed to the scanners.

In addition, you will write a component that reads ballotstructure objects and prints a "ballot" that shows each contest (and the choices for each contest).

Finally, for bonus points, you may choose to use Fitnesse (http://fitnesse.org/) to replace several of your tests. I have not used Fitnesse and will not specify what you should do with it, nor promise that you will be able to get it to work. Between now and Monday (when the exam starts) you are welcome to compare notes with classmates on getting Fitnesse to work. When I say "replace" I mean that you write the tests the non-Fitnesse way first and then show how you can do an alternative implementation with Fitnesse.

Of course, all of your code is TDD and done in iterations which are filed in Subversion. It would be helpful for me if you indicated in your check-in comments which 5-7 iterations you would most like me to read. The final iteration must be one of these. Please also deposit the following in a ZIP file in your dropbox in Angel:

- Your final code (source code)
- Your data files, named in a way that makes it easy for me to understand what file contains what information.

GROUND RULES

1. Our body of knowledge about Java for this exam is Langr, Chapters 1-12 and Chapter 15. If it's not in Chapters 1-12 or Chapter 15, it doesn't exist and you can't use it. (Well, you can use generics, but you are not required to.)
2. You may consult published sources. This includes websites and books. It is OK to look up algorithms online (or in books). It is OK to look up code online that implements exactly what I have asked you to implement.
3. IF YOU USE SOMETHING FROM A SOURCE, YOU MUST REFERENCE YOUR SOURCE. A COMMENT AT THE TOP OF THE CLASS OR THE TOP OF A METHOD IS AN APPROPRIATE PLACE FOR A REFERENCE. I am not speaking about basic Java (e.g. you don't need to reference the API). But if you start from someone else's code and modify it, reference the original code. If it is online, provide the URL.
4. You may not consult with live humans, and that includes not sending emails for help and not posting questions on websites. You may not look at any work done by any other student in this exam until after the exam is finished.
5. You must develop your work test-first. You must check your code into subversion on each iteration. An iteration includes:
    - Create a test
    - modify your code to pass the test and all the other tests you have created so far
    - run your suite of tests
    - fix your code
    - run your tests (repeat until eventually everything passes)
    - refactor your code

- run your tests (fix, test, refactor, test, until everything passes)
- check in your iteration

In some cases, you might create a few tests together rather than just one. Please keep this to a few tests that are obviously so closely related that it makes obvious sense to add them together. (What is "obvious"? Obvious to me. Think of me as a not-very-subtle person. If I have to think about why something is obvious, it's not obvious.)

6. Keep local copies of your code, so that if something terrible happens, you can recreate your work.

7. Your code must conform to our checkstyle standard. If your code has one or two consistent deviations from the standard, provide a note as a comment at the top of your main test class that explains why you are doing things this way. It is an acceptable comment to say that you don't know how to get rid of a certain type of checkstyle or Java warning without using Java techniques outside of Langr Chapters 1-12 and 15 (but if I see an obvious solution, I will drop points) and it is also acceptable to explicitly name a stylistic choice that you have made, as long as you follow it consistently in your code.

8. You should use EMMA to check your code coverage. I will. I will take off points for code that EMMA says is easy to reach (I'll check whether it actually is), that your tests don't cover. Remember, this is a testing course, and a testing exam. Weakly-tested good code will score less than well-tested not-so-good code.

9. Remember the policy on unit tests, published in the Syllabus:

- Every program that you submit must come with unit tests.
    - If the program does not compile successfully, I will assign a grade of zero.
    - If there are no unit tests, I will assign a grade of zero.
    - If the program does not pass its unit tests, I will assign a grade of zero UNLESS, when you submit your code, you also submit a memo that explicitly identifies the tests that the program fails and explain what you did to try to get the code to pass them.
    - If you write special code that creates an appearance that some part of the program is being tested and is passing the test, I will assign a grade of zero to the program.
    - These rules apply to every program, including your final exam.

10. The exam is due Sunday, December 7 at 11:59 pm. You must not submit any revisions of any kind after that time. Submit your iterations as you create them, not at the end.

DELIVERABLES

The first basic rule is that you will deliver a set of code that achieves everything I described in the overview. If there's anything from that section that I don't cover here, you are still responsible for achieving it.

Another basic rule is that anytime I say the program reads data from a text file, you should either be able to specify the values directly in JUnit or fetch them from a data file. For any given type of file should be only one point in the production code that cares where the input comes from. Create an interface that isolates your data collection from your computations.

Your text files should be named in ways that make it obvious what type of information is in them.

You should make it extremely easy and convenient for me to change your code in a way that lets me read your files on my disk. Don't assume that I have the same directory structure as you. Instead, let me change one constant, in one file, that specifies the path to your files. For each time, after the first, that I have to type in a path specifier, I will deduct one (1) point from your exam.

At a minimum, you will deliver the following:

1. A generator that reads a tab-delimited text file and:
    a. Creates a contest class that meets the specification in the file.
    b. If the file contains multiple records, it creates several contests.
    c. Creates a BallotStructure object that includes all the contests in the order they appeared in the original text file
    d. Saves the BallotStructure object to disk so that it can be read later.
2. A generator that reads a BallotStructure object (from disk) and generates a set of pairs of ballots according to user specifications. In each pair, the first is the "original" ballot and the second is the output from the scanner.
    a. For creating the original ballots, the user may specify:
        i. overall probability of an overvote (we will pretend this is constant across all contests), the overall probability of an undervote
        ii. for each contest, the probability that the user will vote for a given candidate.
    b. Here are some constraints:
        i. If there is a President contest, it is first on the ballot
        ii. At least half of the ballots, but not all of them, include a President
        iii. Suppose that P1 is a party and that a voter selects the P1 candidate for President. In that case, for any other contest in which a P1 candidate is represented, the probability that a voter will chose the P1 candidate in that contest is 0.50+R where R is a random number between 0.0 and 0.5.
    c. When creating an original ballot (that is, when simulating voting), the generator shall use a random number function to determine, for each contest, which candidate the voter has selected and whether there will be an undervote (select one fewer candidate than intended) or an overvote (select an extra candidate)
    d. For creating scanned ballots, the generator will create a random ordering for the ballots. If we were feeding the ballots into a scanner, this would be equivalent to shuffling the ballots before feeding them into the scanner.
        i. Example: suppose you stored the original ballots in an array (I am NOT specifying what data structure you should store ballots in) called ballot, and the scanned ballots in an array called scannedBallot.
        ii. Suppose ballot[1] contains the first ballot created and scannedBallot[1] contains the result of scanning the first ballot.

iii. Suppose that after creating a random order for feeding the ballots into the scanner, we determine that ballot[1] will be fed to the scanner as the 217th ballot.

iv. scannedBallot[1] should contain a field, scanningOrder, whose value is 217.

e. For creating scanned ballots, the user may specify (read this from a text file on disk):

i. overall probability of misreading the ballot in a way that misses a vote (in effect, creating an undervote or turning an overvote into an apparently normal vote) in a contest

ii. overall probability of misreading a ballot in a way that adds a vote (in effect, creating an overvote, or turning an undervote into an apparently normal vote) in a contest

iii. for each contest, a set of bias probabilities. Suppose that in a given contest, there are 4 candidates, C1, C2, C3 and C4. Suppose we want to rig the election in favor of C3 by stealing a few votes from the other candidates.

At this point, you have created two programs (or two components of one program) that read input from text files and create two files: BallotStructures and voted Ballots. You have also created various input files that specify what your BallotStructure and voted Ballots will contain.

3. Write a program that reads a data file and determine whether the file contains a BallotStructure object. If so, have it print a ballot that corresponds to the ballot structure.

   a. Your printout doesn't have to be pretty, but when I read the printout, I should be able to tell EASILY what contests are on the ballot and what choices there are within each contest. You should print to a text file. I will either read it in a word processor or do the printout from my printer.

   b. Your program should be able to correctly recognize and print any valid BallotStructure object.

4. Write a program that reads paired Ballots and ScannedBallots and reports discrepancies. Your program's report should print a summary that addresses these questions:

   a. Is there a systematic bias that favors any candidate?

   b. Is there a significant error rate?

   c. Does the accuracy of the scanner vary over time?

Up to this point, if I give your program a specification (by giving it an appropriate set of text files) of any ballot and any set of scanner characteristics (e.g. probability of a misread that leads to undervote or overvote), it can generate the ballot, the voted ballots and the scanned voted ballots. Your unit tests should give confidence that you can cope with any type of ballot.

Your next task is to create a generator that will create specification files for a good suite of tests.

5. Identify 10 ways that ballots can differ from each other. To start you off, here are 5 examples, so you have to identify 5 more:

   a. One ballot can have more contests than another. For exam purposes, I will set the minimum number of valid contests at 1 and the maximum at 50.

   b. One contest can have more choices than another. For exam purposes, I will set the minimum number of valid choices in a contest at 2 (there's no need to vote if the number of choices is 1) and the maximum at 10.

   c. A contest can be partisan (candidates reveal their party endorsements) or non-partisan.

   d. In a partisan contest, there are up to N possible parties. For exam purposes, I will set this to a maximum of 20.

   e. In a partisan contest, a candidate is either Independent (the endorsement is "Independent") or endorsed by 1 or more parties. An Independent candidate cannot be endorsed by any parties. Any party may endorse between 0 and all candidates in the contest.

6. Imagine a contest as a combination test.

   a. To specify a contest is to specify several attributes (variables that determine the details of the contest). To specify a ballot is to specify a bunch of contests, so we can create a pool of ballots that, between them, include a bunch of contests. Look back at your answer to Question 5 and consider only the variables that can vary from contest to contest.

   b. Your task is to create a set of BallotStructures that include a group of contests that collectively satisfy the all-pairs criterion. That is, if V1, V2, ..., VN are N variables that determine the details of the contest (e.g. V1 is the number of choices in the contest), then

      i. the number of possible tests is V1 x V2 x ... x VN, which is too many

      ii. you can reduce the testing space by subsetting the values of Vi to only include boundary cases and other special cases. For example, the number of choices in your contest might be 2 or 10, but nothing between. So let's pretend that the reduced set of values are the only values of the variables V1...VN.

      iii. You achieve all-pairs coverage if your set of tests includes all pairs of values of all variables. For example, if V3 can be 2, 5 or 22 and V7 can be 0 or 1, the set of tests would have to include (2, 0), (2, 1), (5, 0), (5, 1), (22, 0) and (22,1) along with all the pairs of values of V3 with V2 and with V1 and with V4, etc.

   c. List your variables, list the values of each variable, and create a set of contests that achieves all-pairs coverage in each of the following ways:

      i. use a test tool (see references below)

      ii. Write a program to:

         1. use a random number generator to create a set of contests.

         2. Check the set to see if it has met all-pairs

         3. continue adding another contest until a coverage check shows you have achieved all pairs.

        iii. You should now have two sets of contests. For each one, create one or more specification files (you may need one file per BallotStructure that you create) that you can feed to the generator (step 1), create 10 voted ballots based on each BallotStructure that you create, feed them to the scanner and run your report on the results.

    d. References for combination testing:

        i. http://www.testingeducation.org/BBST/BBSTCombinationTesting.html

        ii. http://www.developsense.com/testing/PairwiseTesting.html

        iii. http://www.stickyminds.com/getfile.asp?ot=XML&id=6488&fn=XDD6488filelistfilename1%2Epdf

    e. Tools for combination testing

        i. http://www.pairwise.org/tools.asp

        ii. http://www.satisfice.com/tools.shtml

        iii. http://www.stsc.hill.af.mil/consulting/sw_testing/improvement/cst.html

        iv. http://csrc.nist.gov/groups/SNS/acts/index.html

7. Bonus points (The exam to this point is worth 100 points. You can get up to 10 bonus points here.)

    a. Use Fitnesse (http://fitnesse.org/). Could you use this instead of JUnit to run some of your tests?

    b. Show how you would use Fitnesse in place of JUnit for some of your tests and explain how (and why) this could change the testing that you would do.

        i. "Show how" means, give some worked examples. Use screen shots or some other capture technique to show me what you did and what you got. Do not assume that I actually have Fitnesse up and running on my computer.

SIMPLIFICATIONS

Make your candidate names, party names, etc. one word long. Make the firm assumptions that names have no embedded commas or spaces (and then don't put any commas or spaces in them).

Use Excel or Open Office calc to create your configuration data files (contests, etc.) and just export the spreadsheets to tab-delimited files. If you do use these and you label the columns in a way that makes it easy to understand what you're doing, I would appreciate getting a copy of the files so that I can more easily understand your data files.