NAME: **login ID:**
precept:

COS 126 Midterm 2 Programming Exam, Fall 2009

This exam is like a mini-programming assignment. You will create several classes, compile them with provided files, and run it with the provided test data on your laptop. Debug your programs as needed. This exam is open book, open browser. You may use code from your assignments or code found on the COS126 website. When you are done, submit your program via the course website using the submit link for Precept Exam 2 on the Assignments page.

Grading. Each programs will be graded on correctness, clarity (including comments), design, and efficiency. You will lose a substantial number of points if your programs do not compile or if they crashes on typical inputs.

Print your name, login ID, and precept number on this page (*now*), and write out and sign the Honor Code pledge before turning in this paper. *Note*: It is a violation of the Honor Code to discuss this midterm exam question with anyone until after everyone in the class has taken the exam. You have 50 minutes to complete the exam.

"I pledge my honor that I have not violated the Honor Code during this examination."

_____

Signature

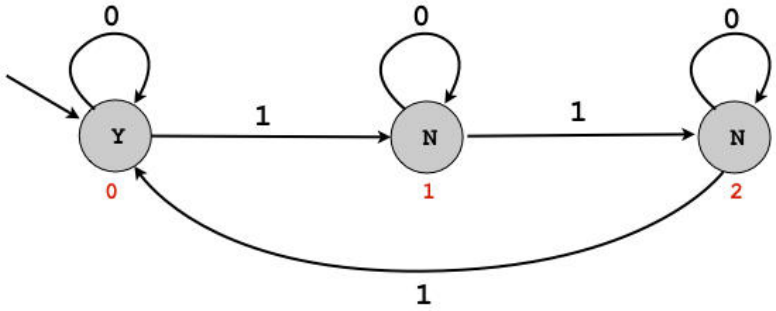
1	/20
2	/10

December 18, 2009

Part 1 (20 points)

Files. This part of the assignment uses the partially completed client program `DFA.java`, the data file `example.txt`, `StdIn.java`, `In.java`, and `StdOut.java`, so *you must create a directory for this exam and download these five files (four .java files and the data file) from the course assignments page before you begin.*

The client program `DFA.java` is equivalent to a universal DFA: its constructor builds a DFA from a description in a file and its `run()` method simulates the operation of the DFA on the string given as argument (it returns `accept` or `reject`). For simplicity this universal DFA assumes the two-character alphabet of '0' and '1' . For reference, this diagram shows a DFA (like the one from lecture, which recognizes bitstrings whose number of 1s is a multiple of 3) as described by `example.txt`.




```
% more example.txt
3
accept 0 1
reject 1 2
reject 2 0
```

The first line of `example.txt` gives the number of states `N`. The state names are numbers from `0` to `N-1`. Subsequent lines of `example.txt` give the state types and the transitions (the `i`th line describes state `i`) : each line has `accept` or `reject` followed by two state names, the first giving the transition if the input is `0` and the second giving the transition if the input is `1`. Assume that the start state is always state `0`.

Your first task. Write a Java class `State.java` that implements for `DFA.java` the functionality of simulating a DFA state. You need a constructor, a method to return the state's type, and a method to implement transitions (given the input character, return the next state). Specifically, you must implement the following public API:

`public class State`

`public State(String type, int next0, int next1)`
Constructor; build a `State` of the given type (`"accept"` or `"reject"`) with the given transitions.

`public String type()`
Return this state's type.

`public int next(char c)`
Do a transition: Assuming that the input character c is either '0' or '1' and the DFA is in this state, return the next DFA state.

Instance variables. Your implementation should maintain three instance variables.

- A string defining the state's type, either `"accept"` or `"reject"`.
- An integer giving the transition if the DFA is in this state and the input is `'0'`.
- An integer giving the transition if the DFA is in this state and the input is '1'.

Your second task. Complete the implementation of `DFA.java` by filling in the code for its `run()` method. Once you have completed this task and have implemented `State.java`, you are looking for the following behavior:

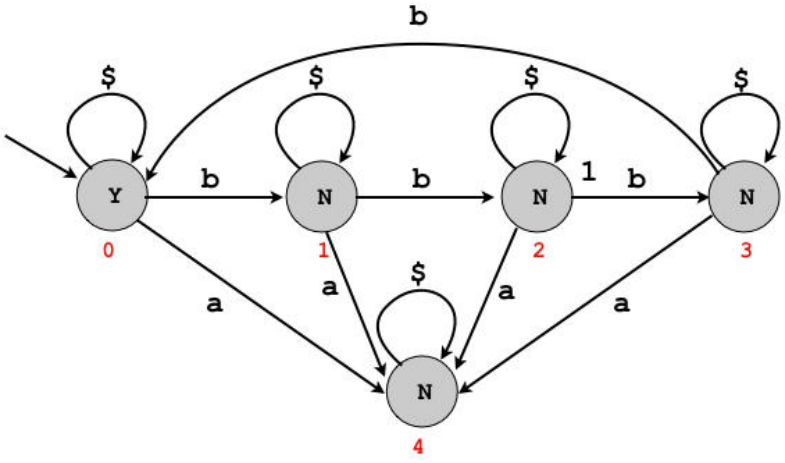
```
% javac DFA.java
% java DFA example.txt
111
accept
1011001000101
accept
110111101
reject
```

Note the first line and all odd lines are typed by you (`StdIn`) and the DFA program produces `accept` or `reject`. There is a `toString()` method which you can use for testing. You probably will want to add some trace prints to check that your program is behaving as it should; if you get the program working, you should remove them, but not `toString()`; if not, the quality of your debugging effort will be a factor in your grade.

Remember that to send an EOF to StdIn from the keyboard we use ctrl-d on terminal (mac), and ctrl-z on command-prompt (windows).

Submit your program `State.java` and your completed `DFA.java` via the link on the Assignments page. DO NOT GO TO THE NEXT PAGE UNTIL YOU HAVE SUBMITTED.

Part 2. (10 points) *Do not attempt this part unless you have completed Part 1 successfully.* Change `DFA.java` and `State.java` to make classes `DFA2.java` and `State2.java` that use our standard generic symbol table `ST.java` to simulate DFAs that can read an *arbitrary* alphabet. For example, the following input file `example2.txt` describes a DFA that recognizes strings whose number of `b`s is a multiple of four, and which contains no `a`:




```
% more example2.txt
5
accept b 1 a 4 $ 0
reject b 2 a 4 $ 1
reject b 3 a 4 $ 2
reject b 0 a 4 $ 3
reject $ 4
```

As before, the first line gives the number of states. Each line after the first gives the state type followed by a sequence of pairs, each giving a character and the transition state when the DFA is in that state and that character is the input character. The `$` character is a meta-character that matches any character not specified for that state. *Important note*: The line for *every* state must end with a `$` followed by a transition state.

Helpful hints:

- The `readChar()` method in `In` and `StdIn` does not skip over whitespace. One easy way to avoid having to deal with whitespace is to read a single character into a char variable `c` with the code `c = in.readString().charAt(0);`
- We suggest you create a symbol table for each state.
- When writing the code for `next()`, think about what you want the code to do for `example2.txt` when you find a char such as `'T'` that is not in the symbol table.
- Character is the wrapper type for char.

Your goal is the following behavior:

```
% java DFA2 example2.txt
bbbb
accept
The_quick_brown_fox_jumped_over_the_lazy_dog.
reject
Bibbidi-Bobbidi-Boo
accept
bbbab
reject
```

Submit your programs `DFA2.java` and `State2.java` via the link on the Assignments page.