Introduction to Computer Programming (CS 1323) Project 6

Instructor: Andrew H. Fagg

November 3, 2014

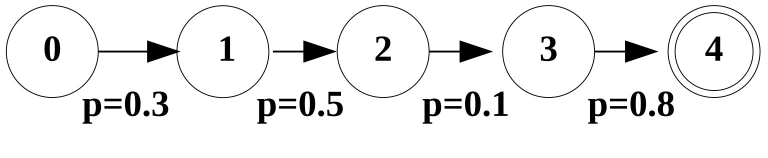
Introduction

A *Markov Chain* (MC) is an abstract model of probabilistic behavior. Like a Finite State Machine, a MC is composed of a set of states. At any one time, a MC can be in exactly one state. A *transition function* also defines how the MC proceeds from one state to the next. However, unlike a FSM, the transition from one state to another is stochastic – it may or may not happen at a given time. MCs are tools that can be used to model human behavior, biological phenomena and robots performing tasks.

In this project, we will implement a simple form of a Markove Chain that is defined by:

- A finite set of states, numbered 0, 1, ... $N-1$, and
- A *transition function* that defines for each state (s), what the probability is of transitioning to the next state in the sequence ($s+1$).

The following figure illustrates a MC with a total of $N = 5$ states. States are represented with labeled circles. Within the circle is the number that corresponds to that state. The arrows represent the transition function. If the MC starts in state 0, at a given *time step*, it will transition to state 1 with probability 0.3. If it does not transition, then the MC remains in state 0 for the next time step (where the transition is "attempted" again). Once the MC is in state 1, it will transition on a given time step to state 2 with probability 0.5. Once the MC reaches state 4, it is considered as complete (or terminated).



One way to represent the transition function represented in this figure is through a table. For this MC, the transition function is:

State	**Probability**
0	0.3
1	0.5
2	0.1
3	0.8

There is one row in the table for each arrow in the figure. Note that although there are 5 states, there are only 4 probabilities listed. This comes from the fact that no transitions can happen from state 4.

In this project, we will implement this simple model of a MC. The input to your program will be:

1. A non-negative integer that will be used as a seed for our random number generator.
2. The numbers in the **Probability** column
3. The String "DONE"

The program will then respond with the sequence of states that the MC visits. This sequence will end once the MC reaches the last state ($N - 1$).

Objectives

By the end of this project, you will be able to:

1. create and manipulate **primitive arrays**,
2. use a random number generator to produce random samples, and
3. write a loop that traverses through an array.

Project Requirements

1. Copy your **getNonNegativeInt()** method from project 4.

2. Write a method with this prototype:

   ```
       public static double getDoubleConstrained(String prompt,
                                    Scanner keyboard, double min, double max)
   ```

   This method displays the prompt and returns a double entered by the user that is between min and max (inclusive). **If the user enters "done" with any capitalization, then this method must return Double.NaN**. Any other entered values result in a reprompting of the user. Note that this method is not unlike methods that you have written in other projects.

3. Write a method with this prototype:

   ```
       public static int sample(Random rand, double p)
   ```

   This method takes as input a Random object and a probability (p) and returns 1 with probability p and 0 otherwise.

4. Write a method with this prototype:

   ```
       public static double[] addDouble(double[] list, double val)
   ```

   This method takes as input an existing primitive array of doubles and a new value to add, and returns a new array with the value appended onto the end of the array.

   If the original array is empty (the reference is *null*), then this method creates and returns an array of length one.

5. Write a **main()** method that prompts the user for an integer random seed, creates a Random object instance with that seed, prompts the user for a set of transition probabilities, and prints out the sequence of visited states.

   The advantage of creating Random with a fixed seed is that the random numbers will always come in the same sequence.

Examples

Here are some example inputs and corresponding outputs from a properly executing program. You should carefully consider the cases with which you test your program. In particular, think about the key "boundary cases" (the cases that cause your code to do one thing or another, based on very small differences in input). User inputs are shown in bold.

Example 1

```
Please enter a seed: 10
Please enter a probability for state 0: .3
Please enter a probability for state 1: .5
Please enter a probability for state 2: .1
Please enter a probability for state 3: .8
Please enter a probability for state 4: done
Probabilities: 0.3 0.5 0.1 0.8
State: 0
State: 0
State: 1
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 3
State: 4
```

(program ends)

Notes: 1) given these inputs, your program will **always** produce these outputs, and 2) although four probabilities are given, we have five states.

Example 2

```
Please enter a seed: 11
Please enter a probability for state 0: .3
Please enter a probability for state 1: .5
Please enter a probability for state 2: .1
Please enter a probability for state 3: .8
Please enter a probability for state 4: done
Probabilities: 0.3 0.5 0.1 0.8
State: 0
State: 0
State: 0
State: 0
State: 1
State: 2
State: 3
State: 4
```

(program ends)

Example 3

```
Please enter a seed: **-5**
Only non-negative values are allowed: **12**
Please enter a probability for state 0: **-1**
Value must be between 0.0 and 1.0: **0.1**
Please enter a probability for state 1: **1.4**
Value must be between 0.0 and 1.0: **0.8**
Please enter a probability for state 2: **0.15**
Please enter a probability for state 3: **0.5**
Please enter a probability for state 4: **DONe**
Probabilities: 0.1 0.8 0.15 0.5
State: 0
State: 0
State: 0
State: 0
State: 0
State: 1
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 2
State: 3
State: 4
```

(program ends)

Project Documentation

Follow the same documentation procedures as we used in project 1. In particular, you must include:

- Java source file documentation (top of file)
- Method-level documentation
- Inline documentation

For full credit, you must execute Javadoc on your source file and include the results (in the *doc* directory) in your zip file.

Hints

Implement and test your program one method at a time (we call this *incremental development*). Convince yourself that each method is working properly before you move on to the next one.

The project requirements define specific method prototypes In order to receive full credit for the project, you must adhere to these requirements.

Random.nextDouble() will return a double in the range of 0 (inclusive) to 1 (exclusive). All values are equally likely.

Project Submission

This project must be submitted no later than 2:00pm on Tuesday, November 4th to the project 6 D2L dropbox. The file must be called *Project6.zip* and be generated in the same way as for Projects 1..5.

Code Review

For the office hour sessions surrounding the project deadline, we will put together a "Doodle Poll" through which you can sign up for a 5-minute code review appointment. During this review, we will execute test examples and review the code with you. If you fail to show up for your reserved time, then you will forfeit your appointment and you must attend at a later time that is not already reserved by someone else.

If either the instructor or TA are free during office hours (or another mutually-agreed upon time), then we are happy to do code reviews, as well as provide general help.

We will only perform reviews on code that has already been submitted to the dropbox on D2L. Please prepare ahead of time for your code review by performing this submission step.

Code reviews must be completed no later than Tuesday, November 11th. If you fail to do a code review, then you will receive a score of zero for the project.

Anyone receiving a 95 or greater on Project 5 may opt to do an “offline” code review (which does not require your presence). Send the instructor email once you have submitted your project in order to schedule an offline review.

Rubric

The project will be graded out of 100 points. The distribution is as follows:

Implementation: 35 points

Program formatting: 15 points

(15) The program is properly formatted (including indentation, curly brace and semicolon locations).
(8) There is one problem with program formatting.
(0) The program is not properly formatted.

Data types and method calls: 10 points

(10) The program is using proper data types and method calls.
(5) There is one error in data type or method call selection.
(0) There are multiple errors in data type and method call selection.

Required Methods: 10 points

(10) All of the required methods are implemented correctly.
(0) The required methods are not implemented correctly.

Proper Execution: 30 points

Output: 15 points

(15) The program passes all tests.
(12) The program fails one test.
(9) The program fails two tests.
(6) The program fails three tests.
(3) The program fails four tests.
(0) The program fails five or more tests.

Execution: 15 points

(15) The program executes with no errors (thrown exceptions).
(8) The program executes, but there is one minor error.
(0) The program does not execute.

Documentation and Submission: 35 points

Project Documentation: 5 points

(5) The java file contains all of the required documentation elements at the top of the file.

(3) The java file is missing one of the required documentation elements.

(2) The java file is missing two of the required documentation elements.

(0) The java file is missing more than two of the required documentation elements.

Method-Level Documentation: 10 points

(10) Every method contains all of the required documentation elements ahead of the method prototype, and the Javadocs have been generated and included in the submission.

(7) The method documentation is missing one of the required documentation elements.

(3) The method documentation is missing two of the required documentation elements, or the Javadocs have not been submitted.

(0) The method documentation is missing more than two of the required documentation elements.

Inline: 10 points

(10) Every method contains appropriate inline documentation.

(7) There is one missing or incorrect line of inline documentation.

(3) There are two missing or incorrect lines of inline documentation.

(0) There are more than two missing or incorrect lines of inline documentation.

Submission: 10 points

(10) The correct zip file name is used.

(0) An incorrect zip file name is used.

Bonus: 6 points For each unique and meaningful error in this project description that is reported to the instructor, a student will receive an extra two points.

References

- Markov Chains: http://en.wikipedia.org/wiki/Markov_chain