Assignment 8: Games of chance

CPSC 1150-006/007, Fall 2016
Instructor: Adina Goldberg
Langara College

Preparation

You are expected to be familiar with the following textbook sections and lectures *before* beginning the assignment. You should complete the following exercises before beginning the assignment. They will put you on the right track to doing the assignment, and will make it easier to create the required programs.

Textbook sections

- 4.2.5
- 7.2, 7.6, 7.7

Most relevant lectures

- L14

Exercises

1. Read through the lab.
2. Write down the **method header** for a method called `testArray` that takes an array of `int`s as an input and returns an array of `boolean`s.
3. Given a `String` as input, write an algorithm to find the length of the longest unbroken sequence of identical characters. For example, if the input is `"001101110"`, the longest identical sequence is `"111"`, so the output should be 3. If the input is `"Langara college"`, the longest identical seqence is `"ll"`, so the output should be 2.
4. Test your intuition before completing the lab. Then after the lab, you can use your programs to see how good your guesses were.
   (a) How many double rolls (matching dice) do you expect if you roll a pair of dice 10 times? 100 times? 1000 times?
   (b) What is the longest burst (sequence of identical flips) you expect if you toss a coin 10 times? 100 times? 1000 times?
   (c) Do your answers from part (a) increase at the same rate as your answers from part (b)? Why/why not?

Introduction

This assignment has several parts. **Make sure to complete the parts in order.** You may work in pairs. You will be using arrays and the `Math.random()` method to simulate some games of chance, and measure certain properties of these games.

Assignment 8: Games of chance

CPSC 1150-006/007, Fall 2016
Instructor: Adina Goldberg
Langara College

1 Utility class

1. Create a class called `RandomGames`. Create a main method which you will use to test the methods you develop.
2. Inside this class, create (and test) an overloaded method called `randomArray`.
   - One of the inputs should be the length of the desired array.
   - There should be two more inputs: `low` and `high`.
   - The method should return a random array of the specified length, where each element is a random number in the specified range (each element is $\geq$ `low` and $\leq$ `high`).
   - If `low` and `high` are `int`s, the returned array should be an `int` array.
   - If `low` and `high` are `double`s, the returned array should be a `double` array.
3. Create (and test) a method called `coinFlips`, which takes in a number of flips, and returns (*not prints!*) a `String` of random coin flips of the specified length (where `H` represents heads and `T` represents tails), e.g. if the input is `10`, the output might be `"HTTHTTHHHT"`.
   - `coinFlips` must call `randomArray`, and then use the output to generate the `String` of coin flips.
   - Heads and tails must each occur with equal probability.
4. Create a method called `dieRolls`, which takes in a number of rolls, and returns an `int` array of random die rolls of the specified length (a roll can be any number from `1-6`), e.g. if the input is `10`, the output might be {`1, 5, 3, 4, 3, 4, 1, 5, 2, 6`}.
   - `dieRolls` must call `randomArray` to generate the array of die rolls
5. Create (and test) a method called `toString`, which takes in an `int` array, and returns a `String` containing the elements of the array in a list, separated by commas, e.g. if the input is the array {`1, 5, 3, 4`}, the output would be the `String` `"1, 5, 3, 4"`.
6. Use `toString` to test the `dieRolls` method.
7. When all the methods in `RandomGames` are working properly, **delete the main method** (including the method header) from `RandomGames`. `RandomGames` can no longer be run by itself, but the methods inside can be invoked from other classes (in the same directory). You can invoke these methods the same way you invoke methods from the `Math` class. For example, if you want to call `coinFlips` from inside another class, you might type `RandomGames.coinFlips(12)`.

2 Heads/tails

You will write a program called `CoinFlipSequences` that simulates sequences of coin flips a certain number of times, and finds (on average) the length of the longest unbroken sequence of all heads or all tails.

Assignment 8: Games of chance

For this part and the next part: You will need to use methods from `RandomGames`. **Do not copy-paste the method definitions** into your new program! Instead, you must invoke the methods directly from `RandomGames`.

Your programs for this and the next part **should be modular** (made up of several methods). You are encouraged to use the top-down design process to create these programs. A program with everything in the main method may not receive full marks.

2.1 Specifications

Input: Your program should get two inputs from the user: `numExp` (a number of experiments) and `numFlips` (the number of coin flips to simulate).

Experiments: Conduct a total of `numExp` experiments. For **each** experiment:

- Generate a random sequence of `numFlips` coin flips (using the `coinFlips` method from `RandomGames`).
- Find the length of the longest *burst* – we define a burst as an unbroken sequence of heads (e.g. `HHHHH`) or tails (e.g. `TTTTTTT`).

Average: Take the average of the longest burst length from each experiment.

Output: Print a record of each experiment, and after all the experiments, print the average longest burst length.

2.2 Sample output for `CoinFlipSequences`

```
This program finds the longest burst length in sequences of coin flips.
Please enter a number of experiments to conduct:
10
Please enter a number of coin flips per experiment:
30
Exp. Longest burst  Flips
    1            9  HTHHHHHHHHHHTHTHHHTHTHHTTHTTTHT
    2            3  THTTHHTTTHHTTHTHTTHHTTTHHTTHTT
    3            4  HTHHTTTHHTTTTHHHHTTTHTHHTTHHHH
    4            6  THTTHTTTTHTTHTTHTHHHHHTTTTTTHT
    5            9  HTTTTTTTTTHHTHTHTHHTHTTTHHTHHT
    6            6  TTHHHTHHHHHTTHHHHTHTTTHTHHHHHH
    7            4  HHHHTHHHHTTTTHHHTTHTTTTHTHHTH
    8            7  HHTTTTTTTHTHTHTTTHHHTTTHHHTHTH
    9            6  TTTTHHHTTHHTHHTTHTTTTTTHTTTHHH
   10            5  HHHHTHTTTHTHHHHHTTTHTHTHTHTTTT
The average longest burst has length 5.
```

3 Count doubles

You will write a program called `CountDoubles` that simulates sequences of die rolls for two independent dice, and finds (on average) the number of times the dice match each other. When the two

dice are matching, we call this a "double roll" (not related to the floating-point data type).

3.1 Specifications

Input: Your program should get two inputs from the user: `numExp` (a number of experiments) and `numRolls` (the number of times to roll both dice).

Experiments: Conduct a total of `numExp` experiments. For **each** experiment:

- Generate a random sequence of `numRolls` die rolls for two separate dice (using the `dieRolls` method from `RandomGames`).
- Count the number of doubles that get rolled (the number of times the dice match each other).

Average: Take the average of the number of doubles from each experiment.

Output: Print a record of each experiment, and at the end, print the average number of doubles.

3.2 Sample output for `CountDoubles`

```
This program finds the number of double rolls in sequences of dice rolls.
Please enter a number of experiments to conduct:
4
Please enter a number of dice rolls per experiment:
20

Experiment 1: Num. doubles = 4
Die 1: 2, 1, 4, 2, 2, 2, 3, 1, 4, 4, 6, 1, 1, 2, 1, 4, 4, 2, 4, 4
Die 2: 2, 2, 2, 1, 2, 3, 6, 3, 6, 1, 2, 3, 4, 4, 1, 3, 4, 3, 6, 1

Experiment 2: Num. doubles = 3
Die 1: 6, 2, 5, 1, 3, 3, 3, 4, 3, 3, 6, 2, 6, 6, 5, 3, 1, 6, 3, 1
Die 2: 5, 2, 6, 5, 1, 6, 5, 2, 2, 6, 4, 1, 5, 1, 2, 2, 1, 1, 3, 2

Experiment 3: Num. doubles = 3
Die 1: 3, 6, 4, 4, 4, 2, 4, 3, 2, 2, 5, 6, 4, 2, 2, 2, 4, 1, 4, 5
Die 2: 3, 4, 2, 6, 1, 1, 5, 5, 3, 5, 6, 1, 5, 5, 3, 2, 1, 2, 5, 5

Experiment 4: Num. doubles = 5
Die 1: 1, 2, 6, 6, 6, 6, 3, 4, 5, 1, 1, 2, 6, 2, 4, 3, 6, 2, 2, 3
Die 2: 5, 6, 6, 6, 2, 2, 3, 5, 1, 1, 1, 5, 4, 3, 2, 4, 4, 3, 4, 2

The average number of double rolls out of 20 rolls is 3.
```

Submission

Recall that submission instructions are in the **Lab Guide**. Your group is required to submit **one** `.zip` folder (in one person's D2L dropbox) containing the modular, internally documented, and

properly styled source code files

- `RandomGames.java` (should not have a main method)
- `CoinFlipSequences.java`
- `CountDoubles.java`

External documentation is not necessary. Javadoc style for method/class documentation is not necessary but is highly recommended. The files should contain both partners' names in the headers. *Make sure both partners save copies of the finished code to their personal H: drive.*