Lab 4: Using and Writing Methods

Introduction

In order to understand classes and objects better and to get used to using them in programs, it is a good idea to practice with them. Soon, we will build our own classes. But for now, we will just work with some predefined classes in Java and write some static methods.

Background

Java has many predefined classes that we can use in our programs in two ways:

1. Call helpful **static utility methods** that we otherwise would have to write ourselves. Recall from the lecture notes that static methods are associated with an entire class rather than with individual objects (which is why they are also called **class methods**). Recall also that static methods are invoked through the class directly. So, we would call them by `Classname.methodName(param list)`. For example, the Math class contains a number of static methods to calculate common functions such as `Math.abs()`, `Math.pow()`, and `Math.sin()`.

2. Create **objects** that we can also use in our programs. We have already seen an example of this with the `Scanner` class — we used a `Scanner` object to read tokens from the input stream in a logical way.

Before doing this lab, refer to the course lecture notes and Chapter 5 in the Gaddis text for more background on Java static method.

Math Class

The `Math` class is a predefined class that contains many useful static methods. We have used it already in several examples discussed in lecture. For details on the `Math` class, see the Java API. Note that many of the methods have multiple versions to accommodate the different primitive Java types. You will not need the `Math` class to complete this lab exercise. But it is useful to be familiarize yourself with the class nonetheless.

Random Class

The `Random` class in Java is a predefine class that enables the programmer to generate pseudo-random numbers. These are useful for simulations and scientific experiments. However, unlike the `Math` class, the methods in the `Random` class are instance methods, not static methods. Thus, to use it we must first create a `Random` object, then use that object to generate our random numbers. Look up the Random class in the Java API and note that many of the methods have the same name as those we saw in the `Scanner` class. This is because in a way they are similar — objects of both `Scanner` and `Random` produce sequences of values, but the `Scanner` class obtains them from the input stream while the `Random` class generates them using an algorithm.

What to do?

You are to write a complete Java program named `Lab04.java`. Your program will simulate rolling 2 six-sided dice, and keep track of how many times each possible roll $(2, 3, \ldots, 12)$ occurs.

First "roll" the dice 100 times and calculate the fraction of each of the value $(2, 3, \ldots, 12)$. In other words, calculate the number of occurrences of each value divided by 100. Compare these fractions with the probabilistic values for each number;

Value	Fraction	Approximation
2	1/36	0.0278
3	2/36	0.0556
4	3/36	0.0833
5	4/36	0.1111
6	5/36	0.1389
7	6/36	0.1667
8	5/36	0.1389
9	4/36	0.1111
10	3/36	0.0833
11	2/36	0.0556
12	1/36	0.0278

Next "roll" the dice 100000 times and calculate the fractions again (divide each occurrence by 100000). Again, compare them to the probabilistic values. Do they match up better with the values this time? Make sure you understand why.

Details

Complete your program in the following way:

- Write a `static void` method called `rollDice` that has two parameters, and `int` and a `Random`. The `int` parameter determines how many times to roll the dice and the `Random` is used to generate the actual values. **Note** that we could make the `Random` variable local to the method, but that would create a new object with each call, which is not necessary. Think carefully about which method in `Random` to call and how to appropriately generate the actual roll values. For example, consider why simply generating a single random number between 2 and 12 would **NOT** be correct. In the method, do the "rolls" and count how many times each number comes up. Then print out the number of times each number comes up and its fraction out of all the rolls.

- In the `main` program, create the `Random` object, then enter a conditional loop. At each iteration of the loop, ask the user to enter the number of rolls desired and call the `rollDice()` method with the appropriate parameters. Then as the user if he/she wants to continue. If so, repeat the process; if not terminate the program.

  The idea is that each iteration of the `main` program loop calls the `rollDice()` method and performs one "experiment", showing the empirical probabilities of the rolls which you can compare to the theoretical probabilities.

If you have trouble figuring out how to set this up or how to utilize a `Random` object, seek help from your Lab TA.

Grading

For this lab, your TA will use the following grading criteria

1. (2 points) Conditional loop in the `main` program
2. (2 points) Calling the `rollDice()` method from the `main` program
3. (2 points) Method Parameters
4. (2 points) Using the `Random` class to simulate rolling two dices correctly
5. (2 points) Probabilities of each outcome

The following is an example of a run:

```
Please enter a number of rolls: 100
Value: 2 Probability: 0.01
Value: 3 Probability: 0.06
Value: 4 Probability: 0.08
Value: 5 Probability: 0.05
Value: 6 Probability: 0.17
Value: 7 Probability: 0.26
Value: 8 Probability: 0.12
Value: 9 Probability: 0.07
Value: 10 Probability: 0.08
Value: 11 Probability: 0.06
Value: 12 Probability: 0.04
Would you like to continue? (y/n): y
Please enter a number of rolls: 100000
Value: 2 Probability: 0.02599
Value: 3 Probability: 0.05709
Value: 4 Probability: 0.08389
Value: 5 Probability: 0.11142
Value: 6 Probability: 0.14154
Value: 7 Probability: 0.16397
Value: 8 Probability: 0.13866
Value: 9 Probability: 0.11081
Value: 10 Probability: 0.0843
Value: 11 Probability: 0.05605
Value: 12 Probability: 0.02628
Would you like to continue? (y/n): n
Goodbye...
```

Note that your probabilities will be slightly different than the output shown above.

Due Date and Submission

Once you completed the program, you must demonstrate your program for your Lab TA. Once your TA already checked you, **DO NOT FORGET** to submit your `Lab04.java` file to the CourseWeb under this lab by the due date.

If you do not complete the lab this week, you may finish it and submit your code to the CourseWeb before the due date. However, you need to demonstrate it to your TA at the beginning of next

week's lab.

No late submission will be accepted.