Assignment 1

Due: Wednesday, January 25 by 11:59 PM

Objective

This assignment should help you gain practice with basic Java syntax using procedural programming, functions, arrays, and console I/O.

Task

Do the following three exercises each in a different file. Your filenames should be

- `Pi.java`
- `Reverse.java`
- `DiceStats.java`

Each file should have a comment including your name at the top of the file. Each file should also have appropriate comments throughout the program.

To do the console input for these exercises, use the `java.util.Scanner` class. For random numbers in the last exercise, you may use the `java.util.Random` class.

Declare any methods you write to be `public` and `static`. You may also use the `java.lang.Math` class if you need it. You may assume correct user input in these problems.

Exercise 1

Filename: `Pi.java`

Calculate the value of π from the infinite series:

$$\pi = 4 - 4/3 + 4/5 - 4/7 + 4/9 - 4/11 + \ldots$$

Print a table that shows the value of π approximated by computing one term of the series, approximated by two terms, three terms, and so on. Use default precision for output (do **not** set any decimal precision).

Start by asking the user how many terms to compute to and then let the user enter the information. Use this to print a table of the first N terms of the series (where N is the data entered by the user). Assume the user's input will be a non-negative integer. Try to match my sample output as closely as you can. Be aware that the default precision of `System.out.print` is different from that of `System.out.printf`, so if your precision does not match my output exactly, it is okay as long as you are using the default for whichever printing function you are using (I used `System.out.printf`).

Sample Run

(Sample user input is underlined)

```
Compute to how many terms of the series? 20
terms   PI approximation
1       4.000000
2       2.666667
3       3.466667
4       2.895238
5       3.339683
6       2.976046
7       3.283738
8       3.017072
9       3.252366
10      3.041840
11      3.232316
12      3.058403
13      3.218403
14      3.070255
15      3.208186
16      3.079153
17      3.200366
18      3.086080
19      3.194188
20      3.091624
```

Exercise 2

Filename: `Reverse.java`

Write a static method called `reverseDigits` that takes a long integer value and returns that number with its digits reversed. For example, given the value 1459, the method should return the value 9541 as a long integer.

Write a `main()` method that enters a loop in which the user is prompted and allowed to enter any long integer (0 to exit the loop) and the `reverseDigits` method is used to compute and return the reversed number. Print this from the main routine.

You may assume the user inputs a positive integer. Try to match the sample run exactly.

Sample Run

(Sample user input is underlined)

```
Please enter a long integer (0 to quit): 123456

The number reversed is: 654321

Please enter a long integer (0 to quit): 4837946852

The number reversed is: 2586497384

Please enter a long integer (0 to quit): 2345678

The number reversed is: 8765432

Please enter a long integer (0 to quit): 123456789012345678

The number reversed is: 876543210987654321

Please enter a long integer (0 to quit): 234005700

The number reversed is: 7500432

Please enter a long integer (0 to quit): 0

Goodbye!
```

Exercise 3

Filename: `DiceStats.java`

Write a program that does the following:

1. Ask the user to enter how many dice will constitute a roll (Some games require different numbers of dice per turn. Yahtzee takes 5, Monopoly takes 2, etc.).
2. Ask the user to enter how many rolls they would like to simulate.
3. Create and use an array to keep track of how many times each possible dice sum appears. Basically, it is a bunch of counters and how many you need depends on how many dice are rolled per "turn."
    - Hint: The idea is that this array is a frequency array like the example we went over in class (number 7.7 from the array lecture)

- Hint: You determine how many counters you will need based on the number of dice rolled per turn. The lowest possible total is all 1s, so the number of dice rolled. The highest possible total is all 6s, so it is (6 * number_of_dice). Use this to determine the size of your array.

4. Use a loop to roll the specified number of dice the desired number of times (and calculate the sum of each roll). Use the array to keep track the number of times each possible sum appears.

5. Display the results in a table with 3 columns:

   (a) the die total

   (b) the number of times that total appeared

   (c) the percentage of the total rolls that this sum appeared (print the percentage to 2 decimal places)

   Try to match the sample runs as closely as possible. Since randomness is involved, the number of times each sum appears (and the matching percentages) will be different, but if you use the same number of dice they should be similar to the sample run values (if you roll 2 dice at a time, you should not get a sum of 12 8% of the time, for example). If you get unexpected values, you may want to try to run it a few more times to make sure you did not just get unlucky (as any distribution is theoretically possible).

Sample Run 1

(user input is underlined)

```
How many dice will constitute one roll? 2
How many rolls? 100000
Sum     # of times      Percentage
2       2741            2.74 %
3       5540            5.54 %
4       8404            8.40 %
5       11228           11.23 %
6       13835           13.84 %
7       16662           16.66 %
8       13827           13.83 %
9       10989           10.99 %
10      8538            8.54 %
11      5480            5.48 %
12      2756            2.76 %
```

Sample Run 2

(user input is underlined)

```
How many dice will constitute one roll? 4
How many rolls? 100000
Sum     # of times      Percentage
4       66              0.07 %
5       312             0.31 %
6       773             0.77 %
7       1512            1.51 %
8       2633            2.63 %
9       4313            4.31 %
10      6312            6.31 %
11      8096            8.10 %
12      9660            9.66 %
13      10831           10.83 %
14      11179           11.18 %
15      10720           10.72 %
16      9654            9.65 %
17      7938            7.94 %
18      6158            6.16 %
19      4406            4.41 %
20      2658            2.66 %
21      1613            1.61 %
22      781             0.78 %
23      304             0.30 %
24      81              0.08 %
```

Compiling

Remember that the compile command is `javac` at the unix command prompt. Compile your code on `linprog.cs.fsu.edu` and run your program with the `java` command.

Preparing for Submission

Pack your files into a single jar-file called `hw1.jar` with the `jar` utility. To do this on `linprog.cs.fsu.edu` (or another terminal environment) use the following command:

```
jar cvf hw1.jar Pi.java Reverse.java DiceStats.java
```

Submitting

I have created a Blackboard submission link for the assignment. It is in the "Assignments" section of the Blackboard course site. Submit your `hw1.jar`

file there.