Assignment 1

Due: Tuesday, January 22 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. These are all based on problems in the textbook (with some modifications) Your filenames should be

- Pi.java
- Prime.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 or the java.security.SecureRandom class. These two classes are functionally identical for the purposes you will use them in this assignment (SecureRandom produces cryptographically strong random numbers, but slower than Random).

Declare any methods you write to be public and static. Each class should be declared public. 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 $\pi$ 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 $\pi$ 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: Prime.java

This is based on exercise 6.25 from your book.

A prime number is a number greater than 1 that is divisible by only 1 and itself. For example: 2, 3, 7, and 13 are prime, but 4, 6, 9, and 114 are not. Note that by this definition, 1 is not a prime number.

Write a static method called isPrime that takes a long integer value and returns whether or not that number is a prime number. If the number is prime,
it should return true. If the number is not prime, the method should return false.

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 isPrime method is used to determine whether or not the number is prime. Print this from the main routine. Continue looping until the user inputs 0.

**Hint:** The simplest way to determine if a number is prime is to brute-force it, rather than trying to use a more complicated algorithm. This is fine for this assignment. One way to reduce the time the brute-forcing algorithm to take is to remember that the largest number you must test to determine if a number \( n \) is prime is \( \sqrt{n} \) (this may be a good time to use the Math class mentioned in the Task section, Math.sqrt(double a) returns the square root of a). Doing it this way means that you should be able to do all of the numbers in the sample run in a reasonable amount of time.

You may assume the user inputs a positive integer (or 0 to quit). Try to match the sample run exactly.

**Sample Run**

(Sample user input is underlined)

Input a positive long integer (0 to quit): 5
5 is a prime number.

Input a positive long integer (0 to quit): 6
6 is not a prime number.

Input a positive long integer (0 to quit): 1
1 is not a prime number.

Input a positive long integer (0 to quit): 2
2 is a prime number.

Input a positive long integer (0 to quit): 10000
10000 is not a prime number.

Input a positive long integer (0 to quit): 10007
Input a positive long integer (0 to quit): 67280421310721
67280421310721 is a prime number.
Input a positive long integer (0 to quit): 9223372036854775807
9223372036854775807 is not a prime number.
Input a positive long integer (0 to quit): 0
Goodbye!

Exercise 3
Filename: DiceStats.java

This is adapted from Exercise 7.17 in your book. 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 7.07 from the Deitel examples linked in 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.
   - In order to generate a random number, it is recommended you use the nextInt(int bound) method from either the Random or SecureRandom classes, which will return an integer between 0 and (bound - 1).
• Again, see example 7.07 from the Deitel book (found here) for an example of doing this.

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).

**Note:** You should only create one instance of your random number generator, so make sure you are not creating it inside of a loop or inside of a function that you call multiple times.

**Sample Run 1**

(underlined)

<table>
<thead>
<tr>
<th>Sum</th>
<th># of times</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2741</td>
<td>2.74 %</td>
</tr>
<tr>
<td>3</td>
<td>5540</td>
<td>5.54 %</td>
</tr>
<tr>
<td>4</td>
<td>8404</td>
<td>8.40 %</td>
</tr>
<tr>
<td>5</td>
<td>11228</td>
<td>11.23 %</td>
</tr>
<tr>
<td>6</td>
<td>13835</td>
<td>13.84 %</td>
</tr>
<tr>
<td>7</td>
<td>16662</td>
<td>16.66 %</td>
</tr>
<tr>
<td>8</td>
<td>13827</td>
<td>13.83 %</td>
</tr>
<tr>
<td>9</td>
<td>10989</td>
<td>10.99 %</td>
</tr>
<tr>
<td>10</td>
<td>8538</td>
<td>8.54 %</td>
</tr>
<tr>
<td>11</td>
<td>5480</td>
<td>5.48 %</td>
</tr>
<tr>
<td>12</td>
<td>2756</td>
<td>2.76 %</td>
</tr>
</tbody>
</table>
Sample Run 2

(user input is underlined)

How many dice will constitute one roll? 4
How many rolls? 1000000

<table>
<thead>
<tr>
<th>Sum</th>
<th># of times</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>795</td>
<td>0.08 %</td>
</tr>
<tr>
<td>5</td>
<td>3170</td>
<td>0.32 %</td>
</tr>
<tr>
<td>6</td>
<td>7599</td>
<td>0.76 %</td>
</tr>
<tr>
<td>7</td>
<td>15519</td>
<td>1.55 %</td>
</tr>
<tr>
<td>8</td>
<td>27142</td>
<td>2.71 %</td>
</tr>
<tr>
<td>9</td>
<td>43289</td>
<td>4.33 %</td>
</tr>
<tr>
<td>10</td>
<td>61462</td>
<td>6.15 %</td>
</tr>
<tr>
<td>11</td>
<td>80011</td>
<td>8.00 %</td>
</tr>
<tr>
<td>12</td>
<td>96423</td>
<td>9.64 %</td>
</tr>
<tr>
<td>13</td>
<td>108106</td>
<td>10.81 %</td>
</tr>
<tr>
<td>14</td>
<td>112426</td>
<td>11.24 %</td>
</tr>
<tr>
<td>15</td>
<td>108277</td>
<td>10.83 %</td>
</tr>
<tr>
<td>16</td>
<td>96436</td>
<td>9.64 %</td>
</tr>
<tr>
<td>17</td>
<td>80226</td>
<td>8.02 %</td>
</tr>
<tr>
<td>18</td>
<td>62070</td>
<td>6.21 %</td>
</tr>
<tr>
<td>19</td>
<td>43285</td>
<td>4.33 %</td>
</tr>
<tr>
<td>20</td>
<td>26794</td>
<td>2.68 %</td>
</tr>
<tr>
<td>21</td>
<td>15457</td>
<td>1.55 %</td>
</tr>
<tr>
<td>22</td>
<td>7688</td>
<td>0.77 %</td>
</tr>
<tr>
<td>23</td>
<td>3048</td>
<td>0.30 %</td>
</tr>
<tr>
<td>24</td>
<td>777</td>
<td>0.08 %</td>
</tr>
</tbody>
</table>

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 Prime.java DiceStats.java
```

Make sure you use this exact command, changing the order of the parameters may overwrite your files.
Submitting

Submit your hw1.jar file to the Assignment 1 link in the "Assignments" section of Canvas.