

**READ ALL INSTRUCTIONS BEFORE STARTING YOUR SOLUTION.**

**Topics:** chapter 3, analysis, top-down **design**, expressions, variables, loops, class constants, formatting

**Specifications:**

1. You must follow all the style requirements in *Java\_Program\_Style\_Requirements\_GradingComponents.pdf* to earn full credit. That document describes the scoring process used and some design hints.

(You may use printf. See pp. 253, 257,259.)

Your modular design choices will be critical for this exercise.

Problem: Perhaps someday you may be stranded on a deserted island, without the `java.lang.Math` class. Do not despair--you can estimate the value of  $\pi$  using only addition, subtraction and division with the following formula (the sum of an infinite series). The more terms you have on the right, the more accurate the estimate.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

2. Before landing on that deserted island, you should be sure that your algorithm is correct--plan and code a properly named class with the phrase `PiEstimator` in it. Your program should ask the user for the number of successive fractions or terms to include in the estimate. Your program will then employ the formula above (or a variation of it) to generate an estimate of  $\pi$ . In your analysis, you must use basic algebra to determine how to generate a proper value, not  $\frac{1}{4}$  of  $\pi$ . Note the alternating negative sign. Multiplying by  $-1$  is an idea.

3. Do NOT use `Math.PI` or any other features of `java.lang.Math` to generate your estimates.

You should, however, use `Math.PI` and `Math.abs(...)` to evaluate your own estimates.

Here's the display expected when 10 is the number of fractions the user requests be included in the estimate.

```
G:\141\Asgn3\Pi> java PiEstimator
```

```
Pi Estimator by Hypatia of Alexandria (for work on her planisphaerium)
```

```
Enter the number of estimates: 10
```

Num	Estimate	Difference
1	4.0000000000	0.8584073464
2	2.6666666667	0.4749259869
3	3.4666666667	0.3250740131
4	2.8952380952	0.2463545584
5	3.3396825397	0.1980898861
6	2.9760461760	0.1655464775
7	3.2837384837	0.1421458301
8	3.0170718171	0.1245208365
9	3.2523659347	0.1107732811
10	3.0418396189	0.0997530347

4. Your solution must include methods that do at least the following five methods:

- I. print the title of the program,
- II. get the user's request then, in a separate method,
- III. print the headings for the table.
- IV. a method that manages the details of one estimation cycle. It should return the next estimate to the program. It will need some parameters to generate that estimate --. It should NOT use any

methods from the Math class. It should NOT generate the output. (You may have it output during debugging if needed but not in the final version).

- V. a method to manage the output for each estimation cycle. This 5th method should use Math.PI and Math.abs to determine your estimate accuracy.

5. Add a text-graphic bar chart to illustrate the convergence of your estimates to the actual value of  $\pi$  – see the sample below. You should write a simple method named barBuilder with the following signature:

```
/**
 * Constructs a string of the requested character repeated count times
 * @param count number of copies of the character to put into the string
 * @param marker the character to copy into the string
 */
public static String barBuilder(int count, char marker)
```

Here's a sample use of barBuilder to print ##### on its own line :

```
System.out.println(barBuilder(5, '#'));
```

Use your barBuilder to enhance the output so that it looks like this (if the user enters 8):

Pi Estimator by Phineas Taylor

Enter the number of estimates: 8

Num	Estimate	Difference	Estimate Illustrated
1	4.0000000000	0.8584073464	MM
2	2.6666666667	0.4749259869	MM
3	3.4666666667	0.3250740131	MM
4	2.8952380952	0.2463545584	MM
5	3.3396825397	0.1980898861	MM
6	2.9760461760	0.1655464775	MM
7	3.2837384837	0.1421458301	MM
8	3.0170718171	0.1245208365	MM

To keep your output line on one screen line, you will have to scale the chart. Do not alter the design of barBuilder – your code must scale the argument that you send to barBuilder. *barBuilder should look a bit like something you have seen in class or in the book.*

After pi works, save it, print the listing. Make a copy of pi and make the minimal number of changes to produce a similar estimate for e the natural log base. Your code must allow a user to select up to 100 million iterations and print a **partial** table of your approximation vs. math.E. The table must have no more than 20 lines. You must approximate first using the factorial approximation:  $e = 1/0! + 1/1! + 1/2! + 1/3! + 1/4! + ..$  ). This version will break when tested with a fairly small number of repetitions. You must revise the class comment explaining the maximum number that works with factorial. Save and print.

Create a copy of this version. You must make changes to no more than 2 methods to replace the factorial with the exponential approximation. Being able to change only one method will earn more points than having to change 2 methods. Exponent approximation:  $e = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n}\right)^n$  . Your ultimate grade will also depend on minimizing the number of lines of code that must change between your pi solution and your final exponential approximation of e.