Loops and Conditionals

The purpose of this assignment is to give you practice writing programs with loops and conditionals.

1. **Generalized harmonic numbers.** Write a program `GeneralizedHarmonic.java` that takes two integer command-line arguments *n* and *r* and uses a `for` loop to compute the n^{th} *generalized harmonic number* of order *r*, which is defined by the following formula:

   $$H(n, r) = \frac{1}{1^r} + \frac{1}{2^r} + \cdots + \frac{1}{n^r}.$$

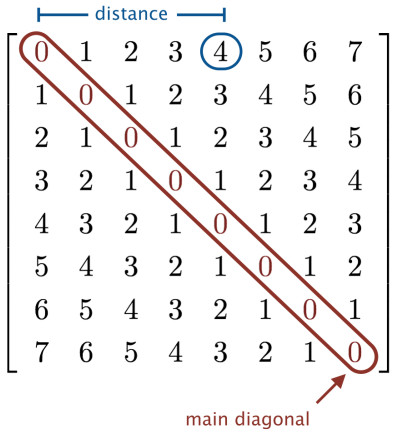
   For example, $H(3, 2) = \frac{1}{1^2} + \frac{1}{2^2} + \frac{1}{3^2} = \frac{49}{36} \approx 1.361111.$

   ```
   ~/Desktop/loops> java GeneralizedHarmonic 1 1
   1.0

   ~/Desktop/loops> java GeneralizedHarmonic 2 1
   1.5

   ~/Desktop/loops> java GeneralizedHarmonic 3 1
   1.8333333333333333

   ~/Desktop/loops> java GeneralizedHarmonic 1 2
   1.0

   ~/Desktop/loops> java GeneralizedHarmonic 2 2
   1.25

   ~/Desktop/loops> java GeneralizedHarmonic 3 2
   1.3611111111111112
   ```

   Note: you may assume that *n* is a positive integer.

   The generalized harmonic numbers are closely related to the Riemann zeta function, which plays a central role in number theory.

2. **Band matrices.** Write a program `BandMatrix.java` that takes two integer command-line arguments `n` and `width` and prints an `n`-by-`n` pattern like the ones below, with a zero (`0`) for each element whose distance from the main diagonal is strictly more than `width`, and an asterisk (`*`) for each entry that is not, and two spaces between each `0` or `*`.



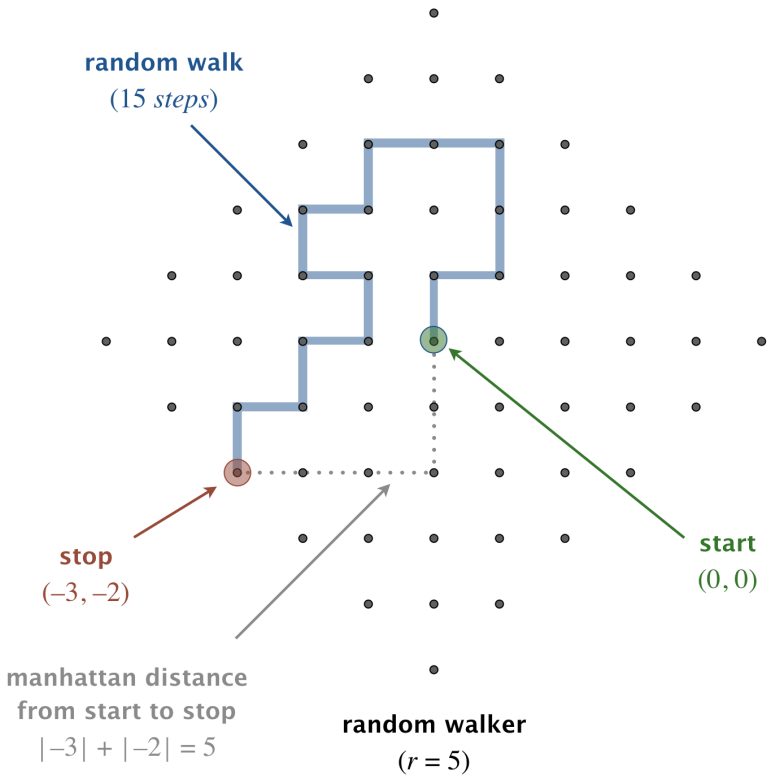
Here, *distance* means the minimum number of cells you have to move (either left, right, up, or down) to reach any element on the main diagonal.

```
~/Desktop/loops> java BandMatrix 8 0
*  0  0  0  0  0  0  0
0  *  0  0  0  0  0  0
0  0  *  0  0  0  0  0
0  0  0  *  0  0  0  0
0  0  0  0  *  0  0  0
0  0  0  0  0  *  0  0
0  0  0  0  0  0  *  0
0  0  0  0  0  0  0  *

~/Desktop/loops> java BandMatrix 8 1
*  *  0  0  0  0  0  0
*  *  *  0  0  0  0  0
0  *  *  *  0  0  0  0
0  0  *  *  *  0  0  0
0  0  0  *  *  *  0  0
0  0  0  0  *  *  *  0
0  0  0  0  0  *  *  *
0  0  0  0  0  0  *  *

~/Desktop/loops> java BandMatrix 8 2
*  *  *  0  0  0  0  0
*  *  *  *  0  0  0  0
*  *  *  *  *  0  0  0
0  *  *  *  *  *  0  0
0  0  *  *  *  *  *  0
0  0  0  *  *  *  *  *
0  0  0  0  *  *  *  *
0  0  0  0  0  *  *  *

~/Desktop/loops> java BandMatrix 8 3
*  *  *  *  0  0  0  0
*  *  *  *  *  0  0  0
*  *  *  *  *  *  0  0
*  *  *  *  *  *  *  0
0  *  *  *  *  *  *  *
0  0  *  *  *  *  *  *
0  0  0  *  *  *  *  *
0  0  0  0  *  *  *  *
```

Note: you may assume that `n` and `width` are non-negative integer.

Band matrices are matrices whose nonzero entries are restricted to a diagonal band. They arise frequently in numerical linear algebra.

3. **Random walk.** A Java programmer begins walking aimlessly. At each time step, she takes one step in a random direction (either north, east, south, or west), each with probability 25%. She stops once she is at Manhattan distance `r` from the starting point. How many steps will the random walker take? This process is known as a two-dimensional *random walk*.



Write a program `RandomWalker.java` that takes an integer command-line argument `r` and simulates the motion of a random walk until the random walker is at Manhattan distance `r` from the starting point. Print the coordinates at each step of the walk (including the starting and ending points), treating the starting point as (0, 0). Also, print the total number of steps taken.

```
~/Desktop/loops> java RandomWalker 5
(0, 0)
(0, 1)
(1, 1)
(1, 2)
(1, 3)
(0, 3)
(-1, 3)
(-1, 2)
(-2, 2)
(-2, 1)
(-1, 1)
(-1, 0)
(-2, 0)
(-2, -1)
(-3, -1)
(-3, -2)
steps = 15
```

This process is a discrete version of a natural phenomenon known as Brownian motion. It serves as a scientific model for an astonishing range of physical processes from the dispersion of ink flowing in water, to the formation of polymer chains in chemistry, to cascades of neurons firing in the brain.

4. **Random walkers.** Write a program `RandomWalkers.java` that takes two integer command-line arguments `r` and `trials`. In each of `trials` independent experiments, simulate a random walk until the random walker is at Manhattan distance `r` from the starting point. Print the *average* number of steps.

```
~/Desktop/loops> java RandomWalkers 5 1000000
average number of steps = 14.98188

~/Desktop/loops> java RandomWalkers 5 1000000
average number of steps = 14.93918

~/Desktop/loops> java RandomWalkers 10 100000
average number of steps = 59.37386

~/Desktop/loops> java RandomWalkers 20 100000
average number of steps = 235.6288

~/Desktop/loops> java RandomWalkers 40 100000
average number of steps = 949.14712

~/Desktop/loops> java RandomWalkers 80 100000
average number of steps = 3775.7152

~/Desktop/loops> java RandomWalkers 160 100000
average number of steps = 15113.61108
```

As `r` increases, we expect the random walker to take more and more steps. But how many more steps? Use `RandomWalkers.java` to formulate a hypothesis as to how the average number of steps grows as a function of `r`.

Estimating an unknown quantity by generating random samples and aggregating the results is an example of Monte Carlo simulation—a powerful computational technique that is used widely in statistical physics, computational finance, and computer graphics.

Submission. Submit a `.zip` file containing `GeneralizedHarmonic.java`, `BandMatrix.java`, `RandomWalker.java`, and `RandomWalkers.java`. You may not call library functions except those in the `java.lang` (such as `Integer.parseInt()` and `Math.sqrt()`). Use only Java features that have already been introduced in the course (e.g., loops and conditionals, but not arrays).

This assignment was developed by Kevin Wayne.