Goal: Write programs to estimate the percolation threshold of a system, which is a measure of how porous the system needs be so that it percolates.

Part I: Warmup Problems

The problems in this part of the assignment are intended to give you solid practice on concepts (using and defining data types) needed to solve the problems in Part II.

Problem 1. (*Die Data Type*) Implement a data type called `Die` that represents a six-sided die and supports the following API:

`Die`

`Die()`	constructs a die with the face value -1
`void roll()`	rolls this die
`int value()`	returns the face value of this die
`boolean equals(Die other)`	returns `true` if this die is the same as `other`, and `false` otherwise
`String toString()`	returns a string representation of this die

`>_ ~/workspace/percolation`

```
$ javac -d out src/Die.java
$ java Die
*   *
  *
*   *
$ java Die
*

    *
```

Problem 2. (*Location Data Type*) Implement a data type called `Location` that represents a location on Earth and supports the following API:

`Location`

`Location(String name, double lat, double lon)`	constructs a new location given its name, latitude, and longitude
`double distanceTo(Location other)`	returns the great-circle distance† between this location and `other`
`boolean equals(Object other)`	returns `true` if the latitude and longitude of this location are the same as those of `other`, and `false` otherwise
`String toString()`	returns a string representation of this location

† See Problem 3 of Assignment 1 for formula.

`>_ ~/workspace/percolation`

```
$ javac -d out src/Location.java
$ java Location
x               = Paris (48.51, -2.17)
y               = Boston (42.36, -71.05)
x.distanceTo(y) = 5224.780334245809
x.equals(y)     = false
```

Problem 3. (*Rational Data Type*) Implement an immutable data type called `Rational` that represents a rational number, ie, a number of the form a/b where a and $b \neq 0$ are integers. The data type must support the following API:

Rational	
`Rational(long x)`	constructs a rational number whose numerator is `x` and denominator is 1
`Rational(long x, long y)`	constructs a rational number given its numerator `x` and denominator `y` (†)
`Rational add(Rational other)`	returns the sum of this rational number and `other`
`Rational multiply(Rational other)`	returns the product of this rational number and `other`
`boolean equals(Object other)`	returns `true` if this rational number is equal to `other`, and false otherwise
`String toString()`	returns a string representation of this rational number

† Use the private function `gcd()` to ensure that the numerator and denominator never have any common factors. For example, the rational number 2/4 must be represented as 1/2.

```
>_ ~/workspace/percolation

$ javac -d out src/Rational.java
$ java Rational 10
1 + 1/2 + 1/4 + ... + 1/2^10 = 1023/512
```

Problem 4. (*Harmonic Number*) Write a program called `Harmonic.java` that accepts n (int) as command-line argument, computes the nth harmonic number H_n as a rational number (using the `Rational` data type from the previous problem), and writes the value to standard output. Recall that H_n is defined as

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n-1} + \frac{1}{n}.$$

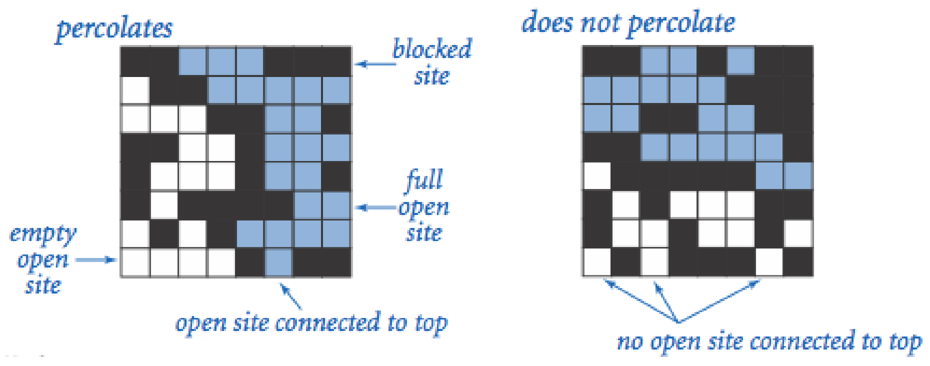
```
>_ ~/workspace/percolation

$ javac -d out src/Harmonic.java
$ java Harmonic 10
7381/2520
```

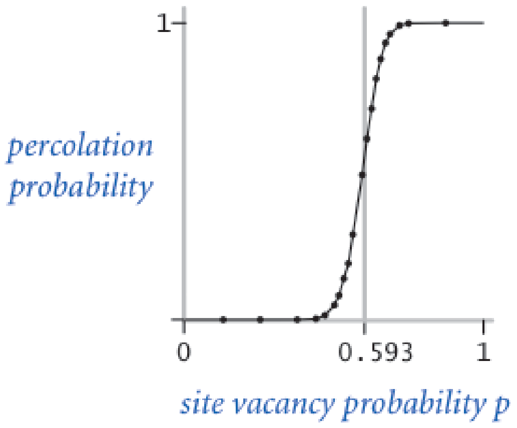
Part II: Percolation

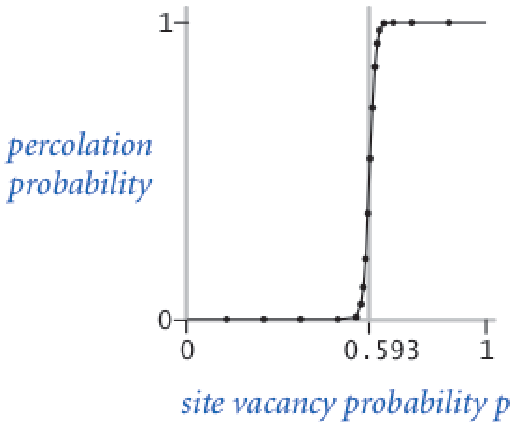
Percolation: Given a composite system comprising of randomly distributed insulating and metallic materials: what fraction of the system needs to be metallic so that the composite system is an electrical conductor? Given a porous landscape with water on the surface (or oil below), under what conditions will the water be able to drain through to the bottom (or the oil to gush through to the surface)? Scientists have defined an abstract process known as *percolation* to model such situations.

The Model: We model a percolation system using an $n \times n$ grid of sites. Each site is either open or blocked. A full site is an open site that can be connected to an open site in the top row via a chain of neighboring (left, right, up, down) open sites. We say the system percolates if there is a full site in the bottom row. In other words, a system percolates if we fill all open sites connected to the top row and that process fills some open site on the bottom row. For the insulating/metallic materials example, the open sites correspond to metallic materials, so that a system that percolates has a metallic path from top to bottom, with full sites conducting. For the porous substance example, the open sites correspond to empty space through which water might flow, so that a system that percolates lets water fill open sites, flowing from top to bottom.



The Problem: If sites are independently set to be open with probability p (and therefore blocked with probability $1 - p$), what is the probability that the system percolates? When p equals 0, the system does not percolate; when p equals 1, the system percolates. The plots below show the site vacancy probability p versus the percolation probability for 20×20 random grid (left) and 100×100 random grid (right).





When n is sufficiently large, there is a threshold value $p^\star$ such that when $p < p^\star$ a random $n \times n$ grid almost never percolates, and when $p > p^\star$, a random $n \times n$ grid almost always percolates. No mathematical solution for determining the percolation threshold $p^\star$ has yet been derived. Your task is to write a computer program to estimate $p^\star$.

Problem 5. (*Percolation Data Type*) Develop a data type called `Percolation` to model an $n \times n$ percolation system. The data type must support the following API.

Percolation	
`Percolation(int n)`	constructs an `n x n` percolation system, with all sites blocked
`void open(int i, int j)`	opens site `(i, j)` if it is not already open
`boolean isOpen(int i, int j)`	returns `true` if site `(i, j)` is open, and `false` otherwise
`boolean isFull(int i, int j)`	returns `true` if site `(i, j)` is full, and `false` otherwise
`int numberOfOpenSites()`	returns the number of open sites
`boolean percolates()`	returns `true` if this system percolates, and `false` otherwise

Corner cases:

- `Percolation()` should throw an `IllegalArgumentException("Illegal n")` if $n \leq 0$.
- `open()`, `isOpen()`, and `isFull()` should throw an `IndexOutOfBoundsException("Illegal i or j")` if i or j is outside the interval $[0, n-1]$.

Performance requirements:

- `Percolation()` should run in time $T(n) \sim n^2$.
- `isOpen()` and `numberOfOpenSites()` should run in time $T(n) \sim 1$.
- `open()`, `isFull()`, and `percolates()` should run in time $T(n) \sim \log n$.

~/workspace/percolation

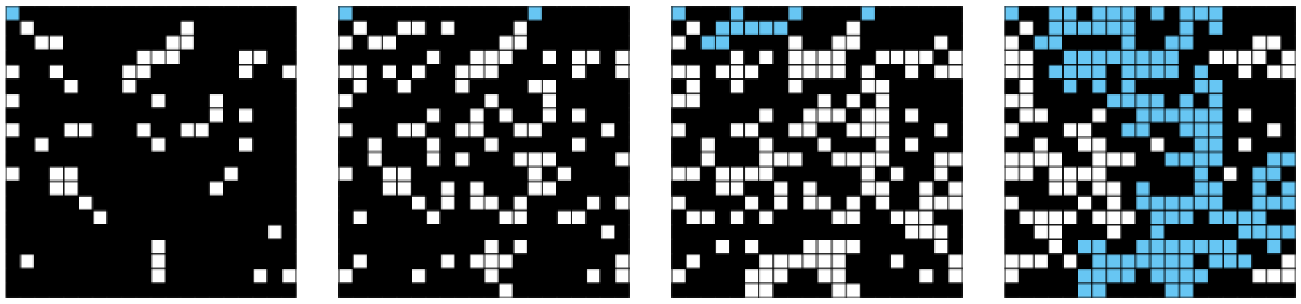
```
$ javac -d out src/Percolation.java
$ java Percolation data/input10.txt
10 x 10 system:
  Open sites = 56
  Percolates = true
$ java Percolation data/input10-no.txt
10 x 10 system:
```

```
Open sites = 55
Percolates = false
```

Problem 6. (*Estimation of Percolation Threshold*) To estimate the percolation threshold, consider the following computational (Monte Carlo simulation) experiment:

- Create an $n \times n$ percolation system (use the `Percolation` implementation) with all sites blocked.
- Repeat the following until the system percolates:
  - Choose a site (row i, column j) uniformly at random among all blocked sites.
  - Open the site (row i, column j).
- The fraction of sites that are open when the system percolates provides an estimate of the percolation threshold.

For example, if sites are opened in a 20×20 grid according to the snapshots below, then our estimate of the percolation threshold is $204/400 = 0.51$ because the system percolates when the 204th site is opened.



By repeating this computational experiment m times and averaging the results, we obtain a more accurate estimate of the percolation threshold. Let $x_1, x_2, \ldots, x_m$ be the fractions of open sites in computational experiments $1, 2, \ldots, m$. The sample mean μ provides an estimate of the percolation threshold, and the sample standard deviation σ measures the sharpness of the threshold:

$$\mu = \frac{x_1 + x_2 + \cdots + x_m}{m}, \qquad \sigma^2 = \frac{(x_1 - \mu)^2 + (x_2 - \mu)^2 + \cdots + (x_m - \mu)^2}{m - 1}.$$

Assuming m is sufficiently large (say, at least 30), the following interval provides a 95% confidence interval for the percolation threshold:

$$\left[\mu - \frac{1.96\sigma}{\sqrt{m}}, \mu + \frac{1.96\sigma}{\sqrt{m}}\right].$$

To perform a series of computational experiments, create an immutable data type called `PercolationStats` that supports the following API:

☰ `PercolationStats`	
`PercolationStats(int n, int m)`	performs `m` independent experiments on an `n x n` percolation system
`double mean()`	returns sample mean of percolation threshold
`double stddev()`	returns sample standard deviation of percolation threshold
`double confidenceLow()`	returns low endpoint of 95% confidence interval
`double confidenceHigh()`	returns high endpoint of 95% confidence interval

The constructor should perform m independent computational experiments (discussed above) on an $n \times n$ grid. Using this experimental data, it should calculate the mean, standard deviation, and the 95% confidence interval for the percolation threshold.

Corner cases:

- The constructor should throw an `IllegalArgumentException("Illegal n or m")` if either $n \leq 0$ or $m \leq 0$.

Performance requirements:

- `PercolationStats()` should run in time $T(n, m) \sim mn^2$.
- `mean()`, `stddev()`, `confidenceLow()`, and `confidenceHigh()` should run in time $T(n, m) \sim m$.

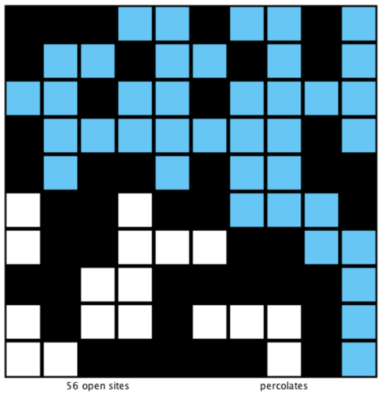
```
>_ ~/workspace/percolation
$ javac -d out src/PercolationStats.java
$ java PercolationStats 100 1000
Percolation threshold for a 100 x 100 system:
  Mean               = 0.592
  Standard deviation  = 0.016
  Confidence interval = [0.591, 0.593]
```

Data: The `data` directory contains some input (`.txt`) files for the percolation programs. The first number specifies the size of the percolation system and the pairs of numbers that follow specify the sites to open. Associated with each file is an output (`.png`) file that shows the desired output. For example, here is an input file:

```
>_ ~/workspace/percolation
$ cat data/input10.txt
10
9 1
1 9
...
7 9
```

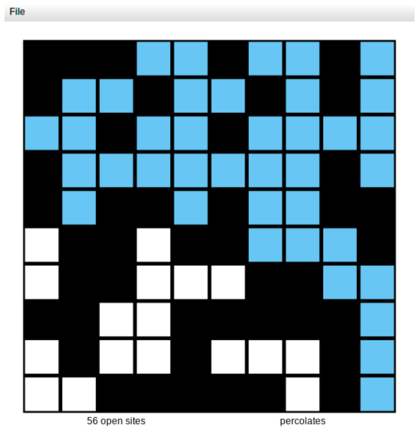
and here is the corresponding output file:

```
>_ ~/workspace/percolation
$ display data/input10.png
```



Visualization Programs: The program `PercolationVisualizer` accepts *filename* (String) as command-line argument and visually reports if the system represented by the input file percolates or not.

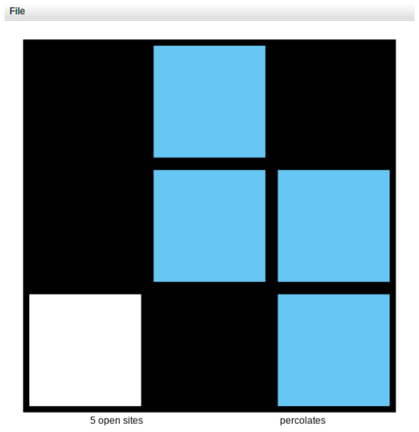
```
>_ ~/workspace/percolation
$ javac -d out src/PercolationVisualizer.java
$ java PercolationVisualizer data/input10.txt
```



The program `InteractivePercolationVisualizer` accepts n (int) as command-line argument, constructs an $n \times n$ percolation system, and allows you to interactively open sites in the system by clicking on them and visually inspect if the system percolates or not.

```
>_ ~/workspace/percolation

$ javac -d out src/InteractivePercolationVisualizer.java
$ java InteractivePercolationVisualizer 3
3
0 1
1 2
1 1
2 0
2 2
```



Files to Submit:

1. `Die.java`
2. `Location.java`
3. `Rational.java`
4. `Harmonic.java`
5. `Percolation.java`
6. `PercolationStats.java`
7. `notes.txt`

Before you submit your files, make sure:

- You do not use concepts from sections beyond *Defining Data Types*.
- Your code follows good programming principles (ie, it is clean and well-organized; uses meaningful variable names; and includes useful comments).
- You edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, you must include in nore more than 100 words: a short, high-level description of the problem; your approach to solve it; and any issues you encountered and if/how you managed to solve them.

Acknowledgement: Part II of this assignment is an adaptation of the Percolation assignment developed at Princeton University by Robert Sedgewick and Kevin Wayne.