# ***Goals***

* To deepen your understanding of loops, conditionals, and arrays.
* To learn how to debug code.

# ***Background***

Read Section 1.3 and 1.4 of the textbook. You may also find it instructive to work through some of the other exercises and look at the solutions on the booksite afterwards.

# ***Implementation Tasks***

## **Overall Requirements**

* You MUST implement five programs:
  + Bits.java
  + NoonSnooze.java
  + RandomWalker.java
  + RandomWalkers.java
  + RollDice.java
* You MUST submit a readme.txt

**Bits.java**

Write a program Bits.java that takes an integer command-line argument n and uses a while loop to compute the number of times you need to divide n by 2 until it is strictly less than 1. Print the error message **Illegal input** if n is negative.

| This program computes the number of bits in the binary representation of n, which also equals when is positive. This quantity arises in information theory and the analysis of algorithms. |
| --- |
|  |

## 

Examples:

| | > java-introcs Bits 0  0  > java-introcs Bits 1  1  > java-introcs Bits 2  2  > java-introcs Bits 4  3 | > java-introcs Bits 8  4    > java-introcs Bits 16  5    > java-introcs Bits 1000  10    > java-introcs Bits -23  Illegal input | | --- | --- | |
| --- | --- | --- |

**NoonSnooze.java**

Write a program NoonSnooze.java that takes an integer command-line argument representing the number of minutes, snooze, that have elapsed since 12:00pm (noon) and prints the resulting time. Assume a 12-hour clock. **You must not use loops.** You may assume that the value of snooze is a non-negative integer.

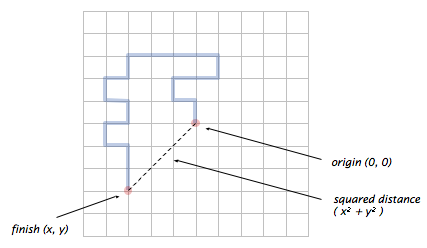
| **Hints!**   * *Use integer division (*/*) and integer remainder (*%*) operators to help compute the new values for the hours and minutes.* * *Use the integer remainder (*%*) operator to determine if the clock should read am or pm.* * *Use conditionals* (if*) to help format the output.* |
| --- |

| **Exercises!**   * *Try solving this problem by hand, e.g., using the following values: 0, 9, 15, 60, 719, 720, 721, 1440, 1441, ..., 5039, 5040, 5041, … Use only integer division (*/*) and integer remainder (*%*).* |
| --- |

Examples:

| > java-introcs NoonSnooze 50  12:50pm  > java-introcs NoonSnooze 100  1:40pm  > java-introcs NoonSnooze 721  12:01am  > java-introcs NoonSnooze 11111  5:11am |
| --- |

| ***Advanced Programming (Optional)***   * Implement NoonSnooze.java without using any if statements. * Handle negative snooze values. * Before attempting an approach that meets these requirements, make sure you can solve the problem using if statements. |
| --- |

A drone begins flying aimlessly, starting at Nassau Hall (0, 0). At each time step, the drone flies one meter in a random direction, either north, east, south, or west, with probability 25%. How far will the drone be from Nassau Hall after n steps? This process is known as a *two-dimensional random walk*. 

Write a program RandomWalker.java that takes an integer command-line argument n and simulates the motion of a random walk for n steps. Print the location at each step (including the starting point), treating the starting point as the origin (0, 0). Also, print the square of the final Euclidean distance from the origin.

Examples:

| | > java-introcs RandomWalker 10  (0, 0)  (0, -1)  (0, 0)  (0, 1)  (0, 2)  (-1, 2)  (-2, 2)  (-2, 1)  (-1, 1)  (-2, 1)  (-3, 1)  squared distance = 10 | > java-introcs RandomWalker 20  (0, 0)  (0, 1)  (-1, 1)  (-1, 2)  (0, 2)  (1, 2)  (1, 3)  (0, 3)  (-1, 3)  (-2, 3)  (-3, 3)  (-3, 2)  (-4, 2)  (-4, 1)  (-3, 1)  (-3, 0)  (-4, 0)  (-4, -1)  (-3, -1)  (-3, -2)  (-3, -3)  squared distance = 18 | | --- | --- | |
| --- | --- | --- |

| ***Possible Progress Steps***  *First, think about which variables you need to maintain. You certainly need to parse and store the command-line argument n. You also need to store the current location (x, y) of the random walker. What should be the type of the variables x and y? What should be their initial values? To choose a random direction, consider using the idiom from Section 1.2 to generate a random integer in a given range.* |
| --- |

**RandomWalkers.java**

Write a program RandomWalkers.java that takes two integer command-line arguments n and trials. In each of trials independent experiments, simulate a random walk of n steps and compute the squared distance. Output the *mean squared distance* (the average of the trials squared distances).

| ***Background***  *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.* |
| --- |

| > java-introcs RandomWalkers 100 10000  mean squared distance = 101.446  > java-introcs RandomWalkers 100 10000  mean squared distance = 99.1674  > java-introcs RandomWalkers 200 1000  mean squared distance = 195.75  > java-introcs RandomWalkers 400 2000  mean squared distance = 383.12    > java-introcs RandomWalkers 800 5000  mean squared distance = 811.8264    > java-introcs RandomWalkers 1600 100000  mean squared distance = 1600.13064 |
| --- |

| ***Possible Progress Steps***   1. Copy the code (in main) from RandomWalker.java to the file RandomWalkers.java. 2. Which additional variables do you need?    1. You certainly need to read and store the command-line arguments n and trials.    2. In addition to the current location (x, y) of the random walker, you need an accumulator variable, say totalSquaredDistance, that stores the total sum of squared distances so far.    3. Nest the loop inside an outer loop that repeats trials times and add code to update totalSquaredDistance after each time through the outer loop. |
| --- |

**RollDice.java**

Write a program RollDice.java that takes an integer command-line argument n, and rolls ten (10) fair six-sided dice, n times. Use an integer array to tabulate the number of times each possible total (between 10 and 60) occurs. Then print a text histogram of the results, as illustrated below:

| > java-introcs RollDice 1000  10:  11:  12:  13:  14:  15:  16:  17:  18: \*  19: \*\*\*\*  20:  21: \*\*\*  22: \*\*\*\*\*\*  23: \*\*\*\*\*\*\*\*  24: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  25: \*\*\*\*\*\*\*\*\*\*\*\*\*  26: \*\*\*\*\*\*\*\*\*\*  27: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  28: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  29: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  30: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  31: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  32: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  33: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  34: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  35: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  36: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  37: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  38: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  39: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  40: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  41: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  42: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  43: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  44: \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  45: \*\*\*\*\*\*\*\*\*  46: \*\*\*\*\*\*\*\*\*\*\*  47: \*\*\*\*\*\*\*  48: \*\*\*  49: \*\*  50:  51:  52: \*  53:  54:  55:  56:  57:  58:  59:  60: |
| --- |

| ***Possible Progress Steps***   1. Write a program that prints the result of rolling one fair die. To generate a random die roll, use the idiom from Section 1.2 to generate a random integer between 0 and n-1, and modify it slightly to get one in the desired range. 2. Add a loop to print the sum of rolling ten (10) fair dice. 3. Add a second loop to repeat this n times, printing the sum after each trial. (Printing the sum is for testing purposes only; be sure to remove it before you submit.) 4. Maintain an array tally[] so that tally[k] stores the number of times the sum is exactly k. 5. Review your code and eliminate unnecessary magic numbers, such as 6, 10, 51, 60, and 61:    1. Define constant variables (such as SIDES and NUMBER\_OF\_DICE) and use these symbolic names to refer to the corresponding magic numbers.    2. Avoid introducing magic numbers that can be derived from existing constants. For example, don't introduce the magic number 60; instead, use SIDES \* NUMBER\_OF\_DICE. |
| --- |

# ***Analysis***

As n increases, we expect the random walk to end up farther and farther away from the origin. But how much farther? Use RandomWalkers to formulate a hypothesis as to how the mean squared distance grows as a function of n.

What is the proportional relationship between the number of steps n of the random walk and the mean squared distance?

Briefly justify your answer based on computational experiments. Describe the experiments and list several data points. Provide your answer in your readme.txt file.

| ***Hints****!*  Using RandomWalkers   * run a series of experiments keeping the number of trials fixed, but changing n - try large values for n (e.g., 100,000) * run a series of experiments keeping n filed but changing the the number of trials - try large values for trials (e.g., 100,000) |
| --- |

**readme.txt**

Edit the text file named readme.txt that is a narrative description of your work.

# ***Submitting the assignment***

Submit the files Bits.java, NoonSnooze.java, RandomWalker.java, RandomWalkers.java, RollDice.java, and a completed readme.txt file.

# ***Enrichment***

[Here](https://www.cs.cmu.edu/~pattis/quotations.html) are some famous and not-so-famous quotations about learning to program.