**Averaging Numbers**

1. Write a program Average.java that just prints the strings that it is given at the command line, one per line. If nothing is given at the command line, print “No arguments”.
2. Modify your program (Make a copy and rename it as Average1.java) so that it assumes the arguments given at the command line are integers. If there are no arguments, print a message. If there is at least one argument, compute and print the average of the arguments. Note that you will need to use the parseInt method of the Integer class to extract integer values from the strings that are passed in. If any non-integer values are passed in, your program will produce an error, which is unavoidable at this point.
3. Test your program thoroughly using different numbers of command line arguments.

*Note: command line arguments are those which are passed to the main method.*

**Exploring Variable Length Parameter Lists**

The file *Parameters.java* contains a program to test the variable length method *average* from Section 7.5 of the text. Note that *average* must be a static method since it is called from the static method *main.*

1. Compile and run the program.
2. Add a call to find the average of a single integer, say 13. Print the result of the call.
3. Add a call with an empty parameter list and print the result. Is the behavior what you expected?
4. Add a method *minimum* that takes a variable number of integer parameters and returns the *minimum* of the parameters. Invoke your method on each of the parameter lists used for the average function.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Parameters.java

//

// Illustrates the concept of a variable parameter list.

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class Parameters

{

//-----------------------------------------------

// Calls the average and minimum methods with

// different numbers of parameters.

//-----------------------------------------------

public static void main(String[] args)

{

double mean1, mean2;

mean1 = average(42, 69, 37);

mean2 = average(35, 43, 93, 23, 40, 21, 75);

System.out.println ("mean1 = " + mean1);

System.out.println ("mean2 = " + mean2);

}

//----------------------------------------------

// Returns the average of its parameters.

//----------------------------------------------

public static double average (int ... list)

{

double result = 0.0;

if (list.length != 0)

{

int sum = 0;

for (int num: list)

sum += num;

result = (double)sum / list.length;

}

return result;

}

}

**Magic Squares**

One interesting application of two-dimensional arrays is *magic squares.* A magic square is a square matrix in which the sum of every row, every column, and both diagonals is the same. Magic squares have been studied for many years, and there are some particularly famous magic squares. In this exercise you will write code to determine whether a square is magic.

File *Square.java* contains the shell for a class that represents a square matrix. It contains headers for a constructor that gives the size of the square and methods to read values into the square, print the square, find the sum of a given row, find the sum of a given column, find the sum of the main (or other) diagonal, and determine whether the square is magic. The read method is given for you; you will need to write the others. Note that the read method takes a Scanner object as a parameter.

File *SquareTest.java* contains the shell for a program that reads input for squares from a file named *magicData* and tells whether each is a magic square. Following the comments, fill in the remaining code. Note that the main method reads the size of a square, then after constructing the square of that size, it calls the *readSquare* method to read the square in. The readSquare method must be sent the Scanner object as a parameter.

You should find that the first, second, and third squares in the input are magic, and that the rest (fourth through seventh) are not. Note that the -1 at the bottom tells the test program to stop reading.

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Square.java

//

// Define a Square class with methods to create and read in

// info for a square matrix and to compute the sum of a row,

// a col, either diagonal, and whether it is magic.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class Square

{

int[][] square;

//--------------------------------------

//create new square of given size

//--------------------------------------

public Square(int size)

{

}

//--------------------------------------

//return the sum of the values in the given row

//--------------------------------------

public int sumRow(int row)

{

}

//--------------------------------------

//return the sum of the values in the given column

//--------------------------------------

public int sumCol(int col)

{

}

//--------------------------------------

//return the sum of the values in the main diagonal

//--------------------------------------

public int sumMainDiag()

{

}

//--------------------------------------

//return the sum of the values in the other ("reverse") diagonal

//--------------------------------------

public int sumOtherDiag()

{

}

//--------------------------------------

//return true if the square is magic (all rows, cols, and diags have

//same sum), false otherwise

//--------------------------------------

public boolean magic()

{

}

//--------------------------------------

//read info into the square from the input stream associated with the

//Scanner parameter

//--------------------------------------

public void readSquare(Scanner scan)

{

for (int row = 0; row < square.length; row++)

for (int col = 0; col < square.length; col ++)

square[row][col] = scan.nextInt();

}

//----------------------------------------

//print the contents of the square, neatly formatted

//----------------------------------------

public void printSquare()

{

}

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// SquareTest.java

//

// Uses the Square class to read in square data and tell if

// each square is magic.

//

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.util.Scanner;

public class SquareTest

{

public static void main(String[] args) throws IOException

{

Scanner scan = new Scanner(new File("magicData"));

int count = 1; //count which square we're on

int size = scan.nextInt(); //size of next square

//Expecting -1 at bottom of input file

while (size != -1)

{

//create a new Square of the given size

//call its read method to read the values of the square

System.out.println("\n\*\*\*\*\*\*\*\* Square " + count + " \*\*\*\*\*\*\*\*"*);*

//print the square

//print the sums of its rows

//print the sums of its columns

//print the sum of the main diagonal

//print the sum of the other diagonal

//determine and print whether it is a magic square

//get size of next square

size = scan.nextInt();

}

}

}

//magicData file is attached.

//In chapter05, there is an example about how to read from a file.