Alexander Laudino

CSC 161-04

Dr. Farrett

Lab Assignment 9 – Lo Shu Magic Square

**Pseudo-code**

This program will:

Contain a two-dimensional array [3][3]

Contain a method that populates the two-dimensional array

The array contains the numbers 1 through 9 exactly

Randomly generate array

Contain a method that accepts a two-dimensional array as an argument

The method determines whether the array is a Lo Shu Magic Square

Local array to hold sum values

The sum of each row, column, and each diagonal all add up to the same number

For loop to sum each row

Add output to array

For loop to sum each column

Add output to array

Set diagonal values

Add values to array

For loop to check if sum values add up to the same number

Set variable to first element

Check if remaining 7 elements are equal to variable

A Lo Shu Magic square has the following properties:

* The grid contains the numbers 1 through 9 exactly.
* The sum of each row, each column, and each diagonal all add up to the same number.

This program simulates a magic square using a two-dimensional array. A random two-dimensional array is generated and the program determines whether the array is a Lo Shu Magic Square.

LoShuMagicSquareDriver

Begin

1. Create new LoShuMagicSquare object as test
2. Call isLoShuMagicSquare method to determine if generated array is a Lo Shu Magic Square
3. Generate a new square array
4. Call isLoShuMagicSquare method to determine if new square array is a Lo Shu Magic Square
5. Call getAllMagicSquares method to display all solutions to Lo Shu Magic Square

End

This class provides the variables, constructor, and methods to generate a square with the values 1 through 9 and determine if it is a Lo Shu Magic Square.

LoShuMagicSquare

// ----- VARIABLES -----

/\*\*

\* The MASTER\_SQUARES variable holds an array of solutions to the Lo Shu Magic Square

\*/

private final int[][][] MAGIC\_SQUARES = {

{{4,9,2}, {3,5,7}, {8,1,6}},

{{8,3,4}, {1,5,9}, {6,7,2}},

{{6,1,8}, {7,5,3}, {2,9,4}},

{{2,7,6}, {9,5,1}, {4,3,8}},

{{2,9,4}, {7,5,3}, {6,1,8}},

{{4,3,8}, {9,5,1}, {2,7,6}},

{{8,1,6}, {3,5,7}, {4,9,2}},

{{6,7,2}, {1,5,9}, {8,3,4}}

};

/\*\*

\* The solutions variable holds the number of elements/arrays in the MASTER\_SQUARES array

\*/

private int solutions = MAGIC\_SQUARES.length;

/\*\*

\* The square variable creates a new array to hold the generated square

\*/

private int[][] square = new int[3][3];

// ----- CONSTRUCTORS -----

/\*\*

\* The LoShuMagicSquare constructor calls the generateSquare method to populate

\* the square instance variable

\*/

public LoShuMagicSquare() {

this.square = generateSquare();

}

// ----- METHODS -----

/\*\*

\* The generateSquare method randomly populates a two-dimensional array

\* with numbers 1 through 9

\* @return Two-dimensional int[3][3] array

\*/

public int[][] generateSquare() {

// Create new Random object

Random random = new Random();

// Initialize two-dimensional array to be shuffled

int[][] square = {{1,2,3}, {4,5,6}, {7,8,9}};

// Loop through array and randomly reorder the elements

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

for (int column = 0; column < square[row].length; column++) {

// Generate an index m randomly

int m = random.nextInt(row + 1);

// Generate an index n randomly

int n = random.nextInt(column + 1);

// Swap square[row][column] with square[m][n]

int temp = square[row][column];

square[row][column] = square[m][n];

square[m][n] = temp;

}

}

return square;

}

/\*\*

\* The printSquare method takes an array and outputs its elements to screen

\* @param square Two-dimensional int[3][3] array

\*/

private void printSquare(int[][] square) {

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

for (int column = 0; column < square[row].length; column++) {

System.out.print(square[row][column]);

}

System.out.println();

}

}

/\*\*

\* The isLoShuMagicSquare method accepts a two-dimensional array to

\* determine whether the array is a Lo Shu Magic Square and returns a true/false value

\* @param square Two-dimensional int[3][3] array

\* @return true if array is a Lo Shu Magic Square, false if array is not a Lo Shu Magic Square

\*/

private boolean testSquare(int[][] square) {

// Display square

printSquare(square);

// Array to hold sum values

int[] sums = new int[8];

int sumIndex = 0;

// For loop to sum each row

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

int rowSum = 0;

for (int column = 0; column < square[row].length; column++) {

rowSum = rowSum + square[row][column];

}

sums[sumIndex] = rowSum; // Add sum to sums array

sumIndex++;

}

// For loop to sum each column

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

int columnSum = 0;

for (int row = 0; row < square[column].length; row++) {

columnSum = columnSum + square[row][column];

}

sums[sumIndex] = columnSum; // Add sum to sums array

sumIndex++;

}

// Set diagonal values

// [x][][]

// [][x][]

// [][][x]

sums[6] = square[0][0] + square[1][1] + square[2][2];

// [][][x]

// [][x][]

// [x][][]

sums[7] = square[0][2] + square[1][1] + square[2][0];

System.out.println();

boolean isMagicSquare = false;

// Set variable to first element

int sum = sums[0];

// For loop to check if sum values add up to the same number

for (int i : sums) {

// If sum values are equal, set isMagicSquare to true

if (sum == i) {

isMagicSquare = true;

}

// If sum values are not equal, set isMagicSquare to false and break out of loop

else {

isMagicSquare = false;

break;

}

}

return isMagicSquare;

}

/\*\*

\* The isLoShuMagicSquare method takes an array and passes it to

\* the testLoShuMagicSquare method to determine whether the array

\* is a Lo Shu Magic Square, and displays the results

\* @param square Two-dimensional int[3][3] array

\*/

public void isLoShuMagicSquare(int[][] square) {

if (testSquare(square) == true) {

System.out.println("This is a Lo Shu Magic Square!\n");

}

else {

System.out.println("This is not a Lo Shu Magic Square.\n");

}

}

/\*\*

\* The getSquare method returns the generated square

\* @return The square

\*/

public int[][] getSquare() {

return square;

}

/\*\*

\* The getAMasterSquare method returns one element in the MASTER\_SQUARES array

\* @param index An integer that represents the index of an element in the MASTER\_SQUARES array

\* @return A two-dimensional int[3][3] array representing one solution to the Lo Shu Magic Square

\*/

public int[][] getAMagicSquare(int index) {

try {

// If index is in range of array

return MAGIC\_SQUARES[index];

}

// If index is not in range of array

catch (ArrayIndexOutOfBoundsException error) {

System.out.println("The solution doesn't exist.\n");

// Create new Scanner object

Scanner input = new Scanner(System.in);

// Sentinel value for while loop

boolean continueInput = true;

int i = 0;

// While loop to prompt user to enter an index in range of array

while (continueInput) {

System.out.print("Please enter a number between 1 and " + (solutions) + ": ");

// Input validation

try {

i = input.nextInt() - 1;

}

catch (InputMismatchException e) {

System.out.println("Please enter a number.");

}

// Check if input is in range

if (i < solutions) {

continueInput = false; // Breaks out of loop

}

else {

System.out.println("Number entered is not between 1 and " + (solutions) + "!\n");

continue;

}

}

System.out.println();

input.close();

return MAGIC\_SQUARES[i];

}

}

/\*\*

\* The getAllMagicSquares method will loop through the MAGIC\_SQUARES array

\* and display all solutions to the Lo Shu Magic Square

\*/

public void getAllMagicSquares() {

// Loops through all solutions

for (int i = 0; i < solutions; i++) {

int [][] magicSquare = getAMagicSquare(i);

isLoShuMagicSquare(magicSquare);

}

}

**UML**

|  |
| --- |
| LoShuMagicSquareDriver |
|  |
|  |

|  |
| --- |
| LoShuMagicSquare |
| -MAGIC\_SQUARES : int[][][]  -solutions : int  -square : int[][] |
| +LoShuMagicSquare()  +generateSquare() : int[][]  -printSquare(int[][])  -testSquare(int[][]) : Boolean  -isLoShuMagicSquare(int[][])  +getSquare() : int[][]  +getAMagicSquare(int) : int[][]  +getAllMagicSquares() |