Assignment – Design and Code Review
Submission: via Blackboard
Points: 50

Objectives:

1. Follow software specifications provided to design and implement a program
2. Design your program using a flowchart and identify list of modules/methods/functions (Flowchart tutorial: http://creately.com/blog/diagrams/flowchart-guide-flowchart-tutorial/)
3. Conduct design review (self and peer)
4. Complete Software Size Estimation
5. Implement the program with appropriate JavaDoc Documentation (Javadoc tutorial: http://www.tutorialspoint.com/java/java_documentation.htm)
6. Conduct Code review (self)
7. Test case generation and testing

Program Specification:

An n x n matrix that is filled with the whole numbers 1, 2, 3, .. n^2 is a *magic square* if the sum of the elements in each row, in each column, and in the two diagonals is the same value.

Here is a magic square where n = 3:

8 1 6

3 5 7

4 9 2

Write a program that reads n^2 numbers from standard input and tests whether they form a magic square when put into matrix form. The value of n is **NOT** an input to the program; n must be determined from the number of inputs.

For example, the input that creates the example matrix above is 8 1 6 3 5 7 4 9 2.

The output is a single word, "true" if the input produces a magic square, "false" otherwise. Your program may assume that each input token is a whole number.

The program must verify:

1. The proper number of input values was provided.

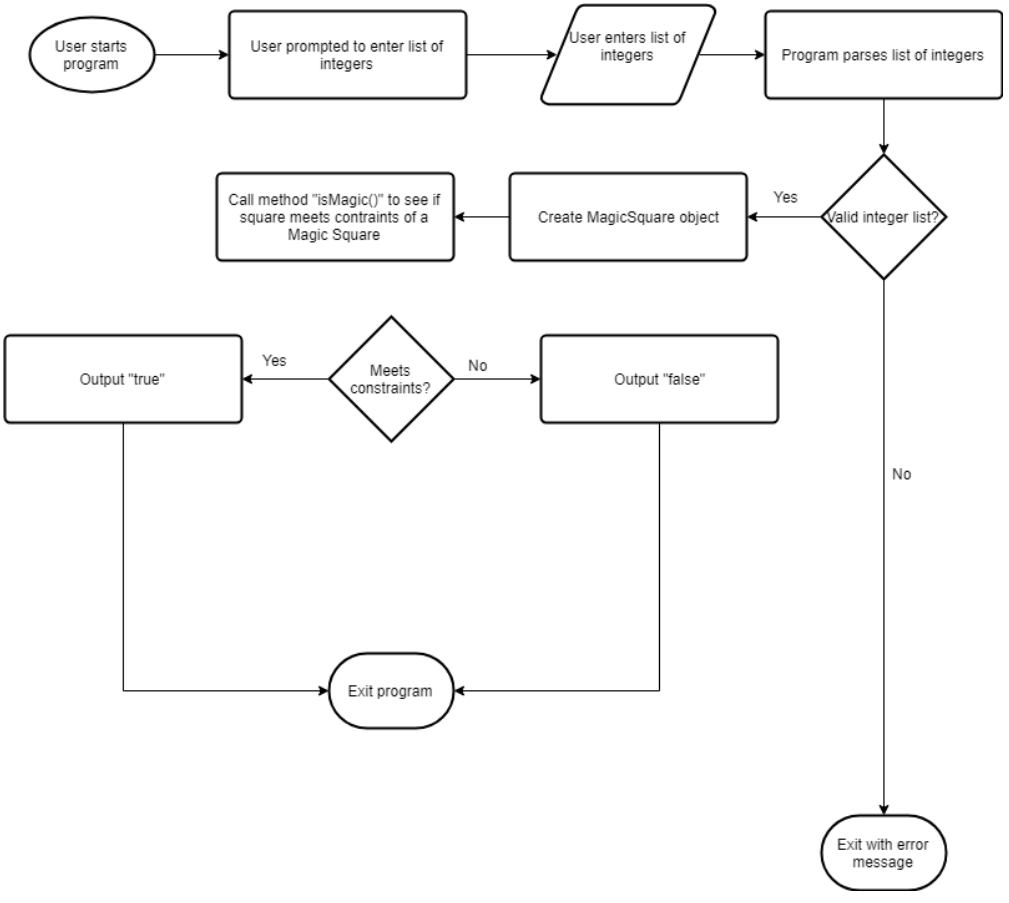
2. Each of the numbers between 1 and n^2 occurs exactly once in the input.

3. When the numbers are arranged in a matrix,
   - the sum of the rows,
   - columns,
   - and diagonals

   must be the same value.

Step 1: Design (5 points)

Design a Magic Squares program as per specifications provided above. Create a flow chart and identify a list of modules/functions/methods and data structures required for implementation.



Functions Required

main() - to run and test the program

MagicSquare(ArrayList) - to create the square from input

boolean isMagic() - to test if the square meets constraints of a Magic Square

boolean isUnique() - validate input integers don't repeat

boolean isSquare() - validate input is n^2 values

SumRows(), SumCols(), SumDiags() - check the sums

Data Structures

ArrayList<Integer> sqNumbers - hold user input, can be as large as necessary

int[][] square - 2d array to hold the square

Step 2: Design Review (Self) (5 points)

Review your design using the Design review checklist provided. Mark the checklist with your findings and make necessary changes.

Preliminary Design Review

i. Completeness

- Are software requirements reflected in the software architecture? ✓
- Are all referenced data defined? ✓
- Are all defined data used? ✓
- Are all referenced modules defined? ✓
- Are all defined modules used? ✓
- Are interfaces defined for modules and external elements? N/A
- Has maintainability been considered? X-changed locations of methods

ii. Consistency

- Is the data structure consistent with the information domain? ✓
- Is the data structure consistent with software requirements? ✓
- Is a standard design representation used? ✓
- Is a standard data usage representation used? ✓

iii. Traceability

- Did you ensure traceability of the design back to the systems specification and statement of requirements? ✓
- Is a scheme used for naming of modules, data, and interfaces? ✓ boolean returning methods "is ___ "()
- Are all modules, data, and interfaces uniquely identified? ✓

iiii. Efficiency

- Are data grouped for efficient processing? ✓
- Are storage requirements allocated to design? ✓
- Is effective modularity achieved? Are modules functionally independent? ✓

Step 3: Design Review (Peer) (5 points)

Ask one of your classmates to review your design (using the Design review checklist provided). Mark the checklist with their comments and make necessary changes.

Peer Review done by: Rachel Knoche

Preliminary Design Review

[1] Completeness

- Are software requirements reflected in the software architecture? ✓
- Are all referenced data defined? ✓
- Are all defined data used? ✓
- Are all referenced modules defined? ✓
- Are all defined modules used? ✓
- Are interfaces defined for modules and external elements? ✓
- Has maintainability been considered? ✓ - all methods should work together

[2] Consistency

- Is the data structure consistent with the information domain? ✓ - arrays are good fit
- Is the data structure consistent with software requirements? ✓ - 2d array fits square well
- Is a standard design representation used? ✓
- Is a standard data usage representation used? ✓

[3] Traceability

- Did you ensure traceability of the design back to the systems specification and statement of requirements? ✓
- Is a scheme used for naming of modules, data, and interfaces? ✓
- Are all modules, data, and interfaces uniquely identified? ✗ - give DS names that are unique

[4] Efficiency

- Are data grouped for efficient processing? ✓
- Are storage requirements allocated to design? ✓
- Is effective modularity achieved? Are modules functionally independent? ✓

Step 4: Software size estimation (5 points)

Estimate the program size. Use the following informal procedure for the estimation and mark the estimation table below.

Informal Size Estimating Procedure:

1. Study the requirements.
2. Sketch out a crude design.
3. Decompose the design into "estimable" chunks.
4. Make a size estimate for each chunk, using a combination of:
   * visualization.
   * recollection of similar chunks that you've previously written
   * intuition.
5. Add the sizes of the individual chunks to get a total.

SIZE ESTIMATION TABLE

Module Description	Estimated Size
main - read and process input	25 LOC
create object and call isMagic	15 LOC
method for is Square input	5 LOC
Constructor	10 LOC
isMagic method - logic	40 LOC
summing methods	40 LOC

TOTAL ESTIMATED SIZE: 135 Lines of code

Step 5: Implementation with JavaDoc Comments (15 points)

Implement the program in java. Name your program MagicSquare.java. Make sure you provide JavaDoc comments and generate documentation files (html files).

Step 6: Code Review (5 points)

Code Review Checklist – Java

1. **Specification / Design**
[X] Is the functionality described in the specification fully implemented by the code?
[X] Is there any excess functionality in the code but not described in the specification?

2. **Initialization and Declarations**
[X] Are all local and global variables initialized before use?
[X] Are variables and class members of the correct type and appropriate mode
[X] Are variables declared in the proper scope?
[X] Is a constructor called when a new object is desired?
[X] Are all needed import statements included?

3. **Method Calls**
[X] Are parameters presented in the correct order?
[X] Are parameters of the proper type for the method being called?
[X] Is the correct method being called, or should it be a different method with a similar name?
[X] Are method return values used properly? Cast to the needed type?

4. **Arrays**
[X] Are there any off-by-one errors in array indexing?
[X] Can array indexes ever go out-of-bounds?
[X] Is a constructor called when a new array item is desired?

5. **Object Comparison**
[/] Are all objects (including Strings) compared with "equals" and not "=="? N/A

6. **Output Format**
[X] Are there any spelling or grammatical errors in displayed output?
[X] Is the output formatted correctly in terms of line stepping and spacing?

7**. Computation, Comparisons and Assignments**
[X] Check order of computation/evaluation, operator precedence and parenthesizing
[/] Can the denominator of a division ever be zero? N/A - no division
[X] Is integer arithmetic, especially division, ever used inappropriately, causing unexpected truncation/rounding? Truncation used purposely to check square property
[X] Check each condition to be sure the proper relational and logical operators are used.
[X] If the test is an error-check, can the error condition actually be legitimate in some cases?
[X] Does the code rely on any implicit type conversions?

8**. Exceptions**
[X] Are all relevant exceptions caught?
[X] Is the appropriate action taken for each catch block?
[X] Are all appropriate exceptions thrown?

9. **Flow of Control**

[X] In a switch statement is every case terminated by break or return?] N/A
[X] Do all switch statements have a default branch?
[X] Check that nested if statements don't have "dangling else" problems.
[X] Are all loops correctly formed, with the appropriate initialization, increment and termination expressions?
[X] Are open-close parentheses and brace pairs properly situated and matched?

10. **Files**
[X] Are all files properly declared and opened?
[X] Are all files closed properly, even in the case of an error?
[X] Are EOF conditions detected and handled correctly?
[X] Are all file exceptions caught?

N/A - no file I/O

Step 7: Testing (5 points)

Identify 4 test cases; one where each of criteria 1, 2, 3 provided in the program specification fail, and one where they all pass. Test your program for all 4 test cases. Provide the outcome of your program for the 4 test cases.

Test Case 1 (provide a brief description): All conditions pass
Input parameter(s): 8 1 6 3 5 7 4 9 2
Expected Output: true
Actual Output: true

Test Case 2 (provide a brief description): Wrong # of inputs (not square
Input parameter(s): 1 2 3
Expected Output: false
Actual Output: false

Test Case 3 (provide a brief description): Values appear more than once (not unique) and all values are not represented.
Input parameter(s): 1 1 1 1 1 1 1 1 1
Expected Output: false
Actual Output: false

Test Case 4 (provide a brief description): Sums not the same
Input parameter(s): 1 2 3 4 5 6 7 8 9
Expected Output: false
Actual Output: false

Step 8: Actual Software Size (2 points)

Document the following:

Estimated Software Size: 135 LOC
Actual Software Size: 114 LOC

Step 9: Reflection (3 points)

[1] **What additional items would you add to the Design review checklist?**

Is the design easy to follow even for a non-coder?
Are there extraneous parts of the design that aren't part of the specification?

[2] **What is the muddiest point in this lab activity?**

The muddiest point in this activity was figuring out the proper way to make the design flowchart. Also it was very unclear how in-depth the design was supposed to be

SUBMISSION:

Name your Java files as MagicSquare.java and TestMagicSquare.java.
Submit this document along with your source code (java program) and JavaDoc documentation generated (html files) archived as ASURiteID-Assignment2.zip via Blackboard.