4 A magic square puzzle corresponds to a matrix of n × n cells.

Consider, for example, the following 3 × 3 matrix:

The magic square must then be filled with integers ranging from 1 to n^2. From the above example, we thus have:

6	1	8
7	5	3
2	9	4

Also notice that when we sum each row (3), column (3) and diagonal (2 – thus, altogether 8 sets of 3 numbers), the sum is always the same. This corresponds to a valid solution to the magic square.

You are tasked to implement an object-oriented class for a magic square puzzle using the UML class diagram below.

MagicSquare
-n: INTEGER -grid: ARRAY OF ARRAY OF INTEGER
+constructor(INTEGER) +swap(INTEGER, INTEGER, INTEGER, INTEGER) +solved(): BOOLEAN +print()

The following are the descriptions of the various attributes and methods within this class.

Attribute/Method	Description

`MagicSquare.n`
`MagicSquare.grid`
`MagicSquare.constructor (INTEGER)`
	The `MagicSquare` class contains two attributes: `n`, the length of each side of the matrix; and `grid`, the actual contents of the matrix. The maximum value for `n` should be 10. If a value larger than 10 is specified, an exception should result.
	On construction, the contents should be the integers 1 to n^2, in order (left column to right column, then top row to bottom row). For example:
	1 2 3
	4 5 6
	7 8 9

`MagicSquare.swap()`	This method takes in 4 integer parameters:
	`AR` ROW index of CELL `A`
	`AC` COLUMN index of CELL `A`
	`BR` ROW index of CELL `B`
	`BC` COLUMN index of CELL `B`
	Each of the above must be in the range 0 and (n – 1). The method will then swap the values in `grid[AR, AC]` and `grid[AR, AC]`.
	Specification of a value outside the required range must result in an exception.

`MagicSquare.solved()`	Checks if the `grid` is ordered in such a way that it satisfies the conditions for a solved magic square. This method will return True if this is the case, else returns False.
`MagicSquare.print()`	Prints the contents of `grid`. This output should include pretty printing of cell boundaries.

Task 4.1

Implement the `MagicSquare` class. Only implement the constructor and print methods. Do **not** implement the `swap` and `solved` methods (yet).

Evidence 20

- The program code for the `MagicSquare` class (excluding the `swap` and `solved` methods). [3]

Task 4.2

Implement the `swap` method for the `MagicSquare` class.

Evidence 21

- The program code for the `swap` method. [2]

Task 4.3

Implement the `solved` method in the `MagicSquare` class.

Evidence 22

- The program code for the `solved` method in the `MagicSquare` class. [5]

Now that you have a working implementation of a magic square. You are to write a program to solve a magic square.

You should only utilise the `swap` method within the `MagicSquare` class to manipulate the `grid` attribute.

Your program should work for any value of n.

Task 4.4

The program code to initialise a random sized magic square between 5 and 10, and to solve that magic square.

Evidence 23

- The program code for **Task 4.4**. [10]