

Project 3 - KenKen

CS 0445 — Data Structure

Due Sunday March 22, 2015 at 11:59pm

The purpose of this project is for you to learn recursion and backtracking together with a little bit of Abstract Data Type (depending on your design). The goal of this project is for you to write a program to solve a puzzle called KenKen.

Introduction to KenKen

KenKen is trademarked names for a style of arithmetic and logic puzzle invented in 2004 by Japanese math teacher Tetsuya Miyamoto, who intended the puzzles to be an instruction-free method of training the brain. An example of a 5 by 5 KenKen puzzle and its solution are shown below.

4-		2/	1-	1-
1-	30*			
		3-		3
4		15*	3-	1-
5+				

4-		2/	1-	1-
5	1	2	3	4
1-	30*			
3	2	1	4	5
		3-		3
2	5	4	1	3
4		15*	3-	1-
4	3	5	2	1
5+				
1	4	3	5	2

KenKen puzzles¹ can be as challenging as you want them to be. But learning how to complete them is totally easy. Just fill some numbers into a grid! OK, OK, there is slightly more to it than that. but you will conquer KenKen in no time by just following these instructions.

1. The numbers you can use in a puzzle depend on the size of the grid. If it is a 3 by 3 grid, you will use the numbers 1 to 3. In a 4 by 4 grid, use numbers 1 to 4. In a 5 by 5 grid... well, you can probably figure it out from there.
2. The heavily-outlined groups of squares in each grid are called "cages". In the upper-left corner of each cage, there is a "target number" and a math operator (+, -, ×, and ÷).
3. Fill in each square of a cage with a number. The numbers in a cage must combine – in any order, using only that cage's math operation – to form that cage's target number. For

¹This instruction came from www.kenken.com

example: Your target number is 5, your operation is addition, you are using the number 1 to 4, and the cage is made up of two squares. You could fill in 2 and 3 (because $2 + 3 = 5$) or 1 and 4 ($1 + 4 = 5$). But which number goes in which square? Read the next instruction!

4. Important: You may not repeat a number in any row or column. You can repeat a number within a cage, as long as those repeated numbers are not in the same row or column.
5. There is only one solution to each KenKen puzzle. As long as you follow the rules above, you will know you got it right!

What to Do?

For this project, I will give you a series of text files called puzzle files. Each file represents a KenKen puzzle. The format of the file will be explained later. Your program (`KenKenSolver.java`) simply asks user to enter a name of a puzzle file, solves the puzzle using recursion and backtracking, and then shows the result on the console screen. **Note** that you are allowed to create any additional classes as you wish. But do not forget to submit them as well.

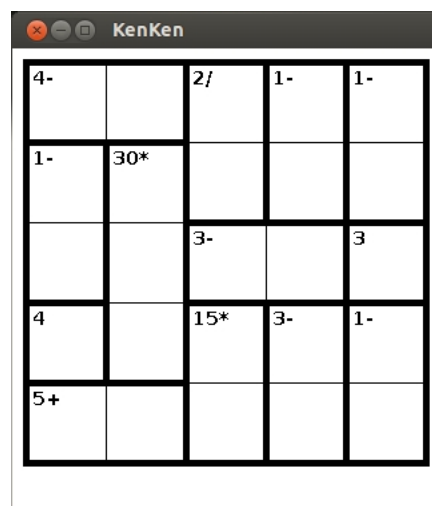
For this project, instead of showing the result on the console screen, you are going to show the result in graphic mode. Luckily, this course is not about Graphical User Interface (GUI), the code to display the result will be provided in a class called `KenKenComponent`. To use this class, simply construct a `JFrame` and construct a `KenKenComponent` as follows:

```
JFrame frame = new JFrame();
KenKenComponent kc = new KenKenComponent(fileName, frame);
```

where `filename` is a string representing the name of the puzzle file. After you construct the `KenKenComponent`, the program will create a frame and draw the puzzle. For example, if you construct the `KenKenComponent`, using the series of statement:

```
JFrame frame = new JFrame();
KenKenComponent kc = new KenKenComponent("5x5_1.txt", frame);
```

the following window will be popped up:



After your program successfully solves the puzzle, put the result in two-dimensional array of type integer (`int`). For example, for the 5 by 5 puzzle shown above, your result must be a 5 by 5 array. Suppose you have a 5 by 5 array named `results`, simply call the method `setNumber` of the `KenKenComponent` as follows:

```
kc.setNumber(results);
```

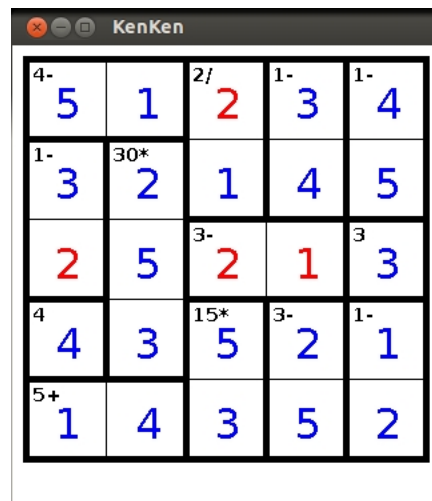
Note that your two-dimensional array must be organized as `row` and `column`. For example, the number of the top-left corner should be stored in `results[0][0]`, the number of the top-right corner should be stored in `results[0][4]`, the number of the bottom-left corner should be stored in `results[4][0]`, and the number of the bottom-right corner should be stored in `results[4][4]`. Note that the `row` and `column` start at 0.

After you call the `setNumber` method, the component will redraw and show result in the puzzle as shown below:

KenKen

4-	5	1	2/	1-	3	1-	4
1-	3	30*	2	1	4	5	
	2	5	3-	4	1	3	
4	4	3	15*	5	3-	1-	2
5+	1	4	3	5	2		

Note that the `KenKenComponent` also checks your solution. If a row or a column contains duplicate numbers, those numbers will be shown in red color. If a set of numbers in a cage cannot be computed to the cage's target number using cage's operator, all numbers in the cage will be shown in red color. For example, the solution below contain some error:



4-	5	1	2/	1-	3	1-	4
1-	3	30*	2	1	4	5	
	2	5	3-	2	1	3	
4	4	3	15*	5	2	1-	1
5+	1	4	3	5	2		

Puzzle File Format

The following is an example of the puzzle file associated with the puzzle shown above:

```
5
13
4,-,2
0,0
0,1
2,/,2
0,2
1,2
1,-,2
0,3
1,3
:
```

For each puzzle file, it is organized as follows:

1. The first line indicates the size of the puzzle. For example, if the first line is 5, the puzzle size is 5 by 5.
2. The second line indicates the number of cages in the puzzle. For example, if the second number is 13, the puzzle consists of 13 cages.
3. The rest are information about each cage which are organized as follows:
 - (a) The first line indicates target number, operator, and number of squares associated with this cage. For example, 4,-,2 indicates the target number is 4, the operator is minus (-), and there are two squares associated with this cage.
 - (b) The next x lines (depending on the number of squares associated with this cage) indicate position of each square in the form **row,column**. For example, the next two lines are 0,0 and 0,1. This means squares at row 0, column 0, and row 0, column 1 are associated with this cage. Note that for each puzzle file, the first row is row number 0 and the first column is column number 0.

Note that for any single square cage, those cages will not have any operator. For example, in the example above, the cage a row 2, column 4 and single square cage. In the puzzle file, lines associated with this cage will be as follows:

```
:
```

```
3, ,1
2,4
:
```

Note that in the above example, the target number is 3, operator is a space and the number of squares associated with this cage is 1, and it is located at row 2 column 4.

Note that the order of cages appears in a puzzle file does not matter. Similarly, the order of positions of squares for each cage does not matter as well.

Hints

As mention earlier, you have to solve these puzzles using recursion and backtracking. Note that the recursion/backtracking part is relatively small and easy. The hard part is setting up all environments for solving puzzles. One way to set up an environment to solve a KenKen puzzle is to think that a KenKen puzzle is an ADT. In other words, a KenKen puzzle is a collection cages. A cage contains a target number, an operator (may be none) and a collection of squares. Each square has a unique row and column associated with it and it contains an integer. For this project, you can use any pre-defined classes or create your own classes. You can design your software which ever way you want as long as the solving part is recursion/backtracking.

A number of puzzle files will be provided. Your program must be able to solve puzzle of any size. I also provide images of puzzles and their solutions. For example, for the puzzle file `5x5_1.txt`, the image of the puzzle will be in the file `5x5_1.jpg` and the image of its solution will be in the file `5x5_1.solution.jpg`.

Note that for a large puzzle, your program may take sometime to solve due to the nature of backtracking. Test your program with smaller puzzles to make sure that it works properly before solving large puzzle. We will test your program with a relatively large puzzle (not provided). The example below is the puzzle and its solution of the puzzle file `9x9_1.txt`.

Due Date and Submission

This project is due on Sunday March 22, 2015 at 11:59pm. No late submission will be accepted. All source files (`KenKenSolver.java`, and other class file that you created must be zipped into a single `.zip` file and submitted to the CourseWeb under Project 3.