Joanna Klukowska
joannakl@cs.nyu.edu

Project 5: `WayFinder`: A Recursive Puzzle Solver

Due date: Nov. 18, 11:55PM EST.

You may discuss any of the assignments with your classmates and tutors (or anyone else) but all work for all assignments must be entirely your own . Any sharing or copying of assignments will be considered cheating (this includes posting of partial or complete solutions on Piazza, GitHub or any other public forum). If you get significant help from anyone, you should acknowledge it in your submission (and your grade will be proportional to the part that you completed on your own). You are responsible for every line in your program: you need to know what it does and why. You should not use any data structures and features of Java that have not been covered in class (or the prerequisite class). If you have doubts whether or not you are allowed to use certain structures, just ask your instructor.

The goal of this project is to implement a program that finds all solutions to a number puzzle. The solution that you will implement has to use a recursive algorithm to find all the ways through the puzzle.

The puzzle uses an array of positive integers. The objective is to find a *path* from index zero to the last index in the array. At each step we need to know the distance to be traveled and the direction. Each entry in the array is a number indicating the distance to be traveled on this particular leg of the path. The player (your program) needs to decide the direction (if the move should be made to the right or to the left).

Here is an example. In each step the circled number indicates the current location. The starting position is always at index 0.

starting array	3	6	4	1	3	4	2	5	3	0
step 1: move right	(3)	6	4	1	3	4	2	5	3	0
step 2: move left	3	6	4	(1)	3	4	2	5	3	0
step 3: move right	3	6	(4)	1	3	4	2	5	3	0
step 4: move right	3	6	4	1	3	4	(2)	5	3	0
step 5: move left	3	6	4	1	3	4	2	5	(3)	0
step 6: move right	3	6	4	1	3	(4)	2	5	3	0
finished	3	6	4	1	3	4	2	5	3	(0)

This puzzle has more than this one solution. The directions for each step can be altered to produce other solutions.

Objectives

The goal of this programming project is for you to master (or at least get practice on) the following tasks:

- designing a program from scratch (deciding what classes and methods should be part of the program)
- designing and implementing recursive algorithms

Start early! This project may not seem like much coding, but debugging and testing always takes time.

For the purpose of grading, your project should be in the package called `project5`. This means that each of your submitted source code files should start with a line:

```
package project5;
```

Joanna Klukowska
joannakl@cs.nyu.edu

Program design

Unlike in the previous two projects, the class design in this project is (almost) entirely up to you. You have to decide what classes should be written, what methods they should contain and how the classes should interact.

But there are some restrictions as to the naming of the final program and program input and output.

The **name of your program** (i.e., the class with the `main` method) should be `WayFinder`.

The algorithm solving the puzzle **must be recursive**.

Input

The program should be given the array to work with via **command line arguments** (this is the `args` array parameter to the `main` method). This means that the program should be run using, for example:

```
java WayFinder 3 6 4 1 3 4 2 5 3 0
```

The command line arguments represent the values for the array. Restrictions:

- all values have to be non-negative integers in the range of 0 to 99 inclusive
- the last value has to be zero

If the program is executed with non-existent or invalid command line arguments, it should print an error message and terminate.

The program **should not be interactive**. All input should be provided as the command line arguments. The user should not be prompted for any additional information.

Results/Output

The program has to calculate and display **all possible solutions to the puzzle** or determine that no such solutions exist. The output of the program should match the format described below **exactly** (the correctness of the solutions will be determined based on textual matching of the program's output to the correct solution).

The outputs for multiple paths through the puzzle should printed on their own lines, with no blank linkes in between. The last line (after all the paths are printed) should conclude the total number of paths that can be successfully taken to solve the puzzle.

Each value in the array should be printed within a 2 digit field (right aligned) with additional space, or character `L` or `R` (for the current value) following the number. The values should be separated by the commas and spaces immediately following the commas. The entire array should be surrounded by square brackets.

The format of a 6-element array should be:

`[VVD,SVVD,SVVD,SVVD,SVVD,SVVD]`

where `VV` stands for the value (the first digit may be a space in case of one digit values), `S` stands for a space character, `D` stands for a direction (either `L`,`R` or a space if that value was not visited in the current path). Note: the value of `D` is always `R` for the value at index zero in the array and it is always a space for the last value in the array.

Here are a few examples:

`[ 3R,  4R, 11 ,  2L, 25 ,  0 ]` means that to follow this path, we first take three steps to the right (arriving at value 2), then take two steps to the left (arriging at value 4) and then take four steps to the right (arriving at the end of the array and hence solving the puzzle)

`[ 3R,  4 , 11 ,  2R, 25 ,  0 ]` means that to follow this path, we first take three steps to the right (arriving at value 2), then take two steps to the right (arriving at the end of the array and hence solving the puzzle)

Warning: Some arrays have values that may lead to infinite number of paths or to infinite loops. The program has to be able to handle those. For example:

- `3  4  11  3  25  0` has no solution, but may lead to an infinite loop if the program keeps taking the steps of `[ 3R,  4 , 11 ,  3L, 25 ,  0 ]`.

Joanna Klukowska
joannakl@cs.nyu.edu

- `2 0 2 3 0` has a solution of `[ 2R, 0 , 2R, 3 , 0 ]` . The program should not be following the path from index 0 to index 2, back to index 0, back to index 2, back to index 0, ..., finally to index 4.

In both cases, the solution to avoiding an infinite loop (or pointless repetiion) is to make sure that no value is visited more than once.

Program execution examples

Here are a few examples of the program execution.

Program run using:

```
java WayFinder
```

```
ERROR: incorrect usage. At least one argument is required.
```

Program run using:

```
java WayFinder  3 -2 9 0
```

```
ERROR: the puzzle values have to be positive integers in range [0, 99].
```

Program run using:

```
java WayFinder 0
```

```
[ 0 ]
There is 1 way through the puzzle.
```

Program run using:

```
java WayFinder  3 6 0 1 3 4 2 5 3 0
```

```
No way through this puzzle.
```

Program run using:

```
java WayFinder  2 1 2 0
```

```
No way through this puzzle.
```

Program run using:

```
java WayFinder  3 6 4 1 3 4 2 5 3 0
```

```
[ 3R,  6 ,  4R,  1L,  3 ,  4R,  2R,  5 ,  3L,  0 ]
[ 3R,  6R,  4R,  1R,  3L,  4R,  2R,  5L,  3L,  0 ]
[ 3R,  6 ,  4R,  1R,  3R,  4R,  2R,  5L,  3L,  0 ]
There are 3 ways through the puzzle.
```

Program run using:

```
java WayFinder 6 9 3 10 2 1 5 8 9 1 2 5 4 8 10 7 6 0
```

```
[ 6R,  9R,  3 , 10 ,  2 ,  1 ,  5L,  8 ,  9R,  1 ,  2L,  5 ,  4 ,  8 , 10 ,  7 ,  6 ,  0 ]
[ 6R,  9R,  3 , 10 ,  2 ,  1 ,  5L,  8 ,  9R,  1 ,  2R,  5 ,  4L,  8 , 10 ,  7 ,  6 ,  0 ]
[ 6R,  9 ,  3 , 10 ,  2 ,  1 ,  5R,  8 ,  9R,  1 ,  2L,  5R,  4 ,  8 , 10 ,  7 ,  6L,  0 ]
[ 6R,  9 ,  3 , 10 ,  2 ,  1 ,  5R,  8 ,  9R,  1 ,  2R,  5R,  4L,  8 , 10 ,  7 ,  6L,  0 ]
There are 4 ways through the puzzle.
```

Joanna Klukowska
joannakl@cs.nyu.edu

Programming Rules

You should follow the rules outlined in the document *Code conventions* posted on the course website at https://cs.nyu.edu/~joannakl/cs102_f19/notes/CodeConventions.pdf.

You may use any exception-related classes. You can use an array, `ArrayList` or any other form of a list implementation provided by Java API. You can use any of the family of classes representing strings.

Working on This Assignment

You should start right away!

Make sure that at all times you have a working program. You can implement methods that perform one task at a time. This way, if you run out of time, at least parts of your program will be functioning properly.

You should make sure that your program's results are consistent with what is described in this specification by running the program on carefully designed test inputs and examining the outputs produced to make sure they are correct. The goal in doing this is to try to find the mistakes you have most likely made in your code.

You should backup your code after each time you spend some time working on it. Save it to a flash drive, email it to yourself, upload it to your Google drive, do anything that gives you a second (or maybe third copy). Computers tend to break just a few days or even a few hours before the due dates - make sure that you have working code if that happens.

Grading

If your program does not compile or if it crashes (almost) every time it is run, you will get a zero on the assignment.

If the program does not adhere to the specification, the grade will be low and will depend on how easy it is to figure out what the program is doing.

50 points correctness

30 points design and the implementation of the classes

20 points proper documentation, program style and format of submission

How and What to Submit

For the purpose of grading, your project should be in the package called `project5`. This means that each of your submitted source code files should start with a line:

```
package project5;
```

Your should submit all your source code files (the ones with .java extensions only) in a single **zip** file to Gradescope. The Gradescope autograder will be available a few days before the assignment due date.

You can produce a zip file directly from Eclipse:

- right click on the name of the package (inside the `src` folder) and select Export...
- under General pick Archive File and click Next
- in the window that opens select appropriate files and settings:
  - in the right pane pick ONLY the files that are actually part of the project, but make sure that you select all files that are needed
  - in the left pane, make sure that no other directories are selected
  - click Browse and navigate to a location that you can easily find on your system (Desktop or folder with the course materials or ...)
  - in Options select "Save in zip format", "Compress the contents of the file" and "Create only selected directories"
- click Finish