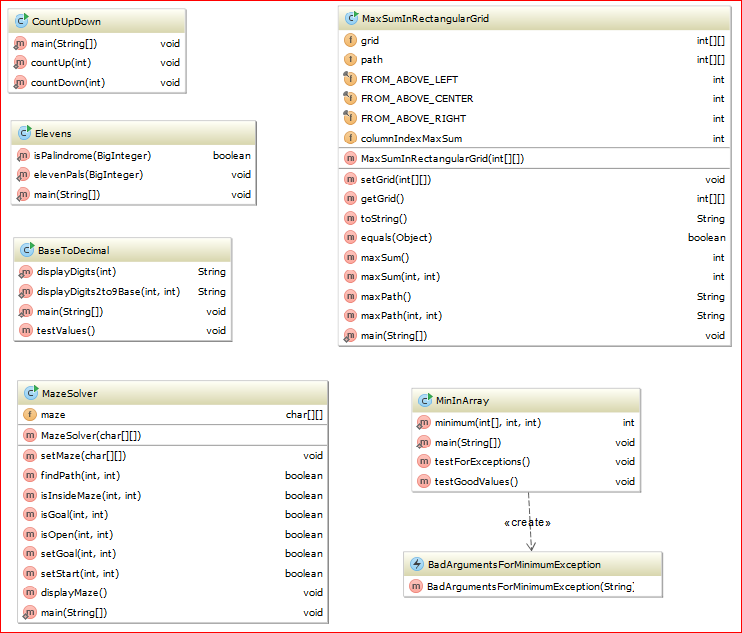
# Comp151 Lab05

In Lab05 you will be working on six separate applications:

1. CountUpDown
2. Elevens
3. BaseToDecimal
4. MazeSolver (started in the class on Tuesday)
5. MinInArray
6. MaxSumInRectangularGrid

Each application has a main inside the corresponding .java file and there are no dependencies between them. See the descriptions below. The skeletons for each program are provided.

### UML Diagram:



1. **CountUpDown**

Implement two simple recursive methods that do not compute or return anything; they simply print integer values as described below. Explanation in section 7 of your textbook may be helpful here.

1. Run CountUpDown file: the program will ask you for an integer value. Enter any value. A couple of messages will be displayed, but no integer will be displayed.
2. Implement recursive method countUp(), so when you run the CountUpDown program and enter 5 the method produces the following output: **1 2 3 4 5**
3. Implement recursive method countDown(), so when you run the CountUpDown program and enter 5 the method produces the following output: **5 4 3 2 1**

## Elevens

Write a recursive method that checks the result of sequence of 1s multiplied by the same sequence of 1s for palindrome. Implement the algorithm that you designed as part of your homework.

Your program should produce the following output:

**1 \* 1 is 1 - and it is a PALINDROME  
11 \* 11 is 121 - and it is a PALINDROME  
111 \* 111 is 12321 - and it is a PALINDROME  
1111 \* 1111 is 1234321 - and it is a PALINDROME  
11111 \* 11111 is 123454321 - and it is a PALINDROME  
111111 \* 111111 is 12345654321 - and it is a PALINDROME  
1111111 \* 1111111 is 1234567654321 - and it is a PALINDROME  
11111111 \* 11111111 is 123456787654321 - and it is a PALINDROME  
111111111 \* 111111111 is 12345678987654321 - and it is a PALINDROME  
1111111111 \* 1111111111 is 1234567900987654321 - and it is NOT a PALINDROME**

1. **BaseToDecimal**

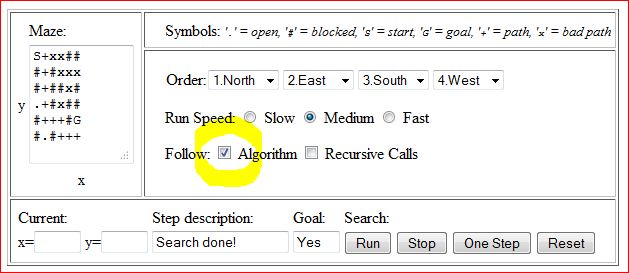
If n is a positive integer in Java, n % 10 is its rightmost digit and n / 10 is the integer obtained by dropping the rightmost digit from n. Using these facts, write a recursive method called displayDigits that displays the digits of an integer n in decimal. For example the integer number 345 should be displayed as a String "3 4 5"; where the negative integer number -345 should be displayed as a String "-3 4 5". Note the spaces between the digits.

Now observe that you can display n in any base between 2 and 9 by replacing 10 with the new base. Write displayDigits2to9Base method that is a revised version of yours displayDigits method, to accommodate a given base. Please note that the modified method will essentially convert a decimal number into its equivalent in the given base. For example 10 in base 8 will be displayed as a String "1 2"; 5 in base 2 will be displayed as a String "1 0 1", and so on.

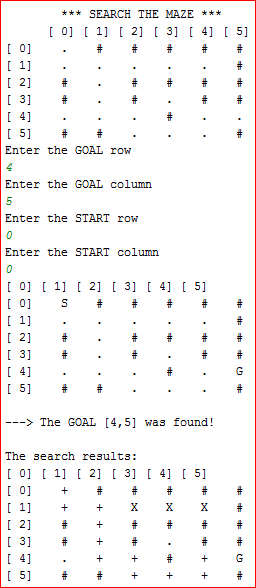
Test your methods with the driver provided in main.

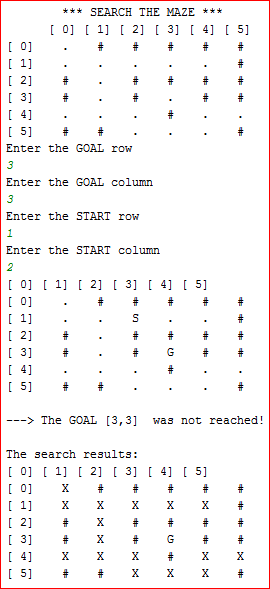
## Maze Solver

* 1. Go to <https://www.cs.bu.edu/teaching/alg/maze/> and analyze the given algorithm
  2. Run the applet at the bottom of the page with the “Algorithm” box checked:



* 1. Implement the algorithm using recursion, use MazeSolver.java as the starting point
  2. See sample runs:





## MinInArray

Write a recursive method that returns the smallest integer in an array of integers.

If you divide the array into two pieces - halves, for example - and find the smallest integer in each of the pieces, the smallest integer in the entire array will be the smaller of these two integers. Since you will be searching a portion of the array - for example, the elements array[first] through array[last] - it will be convenient for your method to have three parameters: the array and two indices: first and last.

NOTE: You can refer to the method displayArray in Segment 7.18 in the textbook for the inspiration.

Test your methods with the driver provided in main.

## MaxSumInRectangularGrid

Consider a rectangular grid of integers. We are interested in computing recursively the largest sum of any path from a top position to a bottom position. A valid path is defined as follows:

* It should start at a number in the top row and end at a number in the bottom row
* It should include a number in every row
* From row *r* to row (*r+1*), a valid path can be created:
  + Down vertically (in the same column)
  + Down diagonally one column to the left (if possible)
  + Down diagonally one column to the right (if possible)

For example, let’s assume we have the following grid of numbers:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 5 | 17 | 12 | 3 |
| 15 | 8 | 4 | 11 | 10 |
| 9 | 18 | 6 | 20 | 16 |
| 14 | 13 | 12 | 1 | 7 |

Examples of valid paths are:

2 -> 8 -> 18 -> 14

17 -> 4 -> 18 -> 14

5 -> 4 -> 20 -> 12

In this grid, the path generating the largest sum is:

17 -> 11 -> 20 -> 12 for the total of (17 + 11 + 20 + 12 = 60)

The program should recursively compute and output the path that generates the largest sum.

Here are sample runs:

Run#1

**Enter the number of rows of the grid (between 1 and 10) > 5**

**Enter the number of columns of the grid (between 1 and 20) > 7**

**21 14 8 11 9 5 1**

**17 5 3 6 22 14 12**

**14 15 5 6 3 4 11**

**10 22 6 1 12 15 21**

**21 15 18 3 22 16 15**

**The max sum is 96**

**The max path is [0,0]:21 ==> [1,0]:17 ==> [2,1]:15 ==> [3,1]:22 ==> [4,0]:21**

Run#2

**Enter the number of rows of the grid (between 1 and 10) > 10**

**Enter the number of columns of the grid (between 1 and 20) > 20**

**9 14 8 1 9 23 23 13 10 22 2 10 22 6 13 25 5 24 12 10**

**20 11 5 25 23 18 9 3 10 7 18 21 11 6 13 9 13 13 19 20**

**17 14 20 11 12 11 1 2 9 1 17 15 12 25 2 20 8 13 14 25**

**22 18 8 7 19 7 2 2 11 3 9 23 11 3 23 10 1 15 17 13**

**19 23 8 19 9 1 18 1 7 10 13 16 24 8 1 23 11 21 22 19**

**10 14 5 25 21 14 20 22 19 13 8 9 24 4 8 2 1 1 18 2**

**23 2 22 9 12 20 12 10 12 4 3 8 13 24 11 24 18 10 3 4**

**18 19 12 22 14 13 25 23 12 3 9 21 18 13 9 13 24 21 21 7**

**12 3 2 17 19 10 11 17 17 21 20 21 2 3 4 12 3 24 18 20**

**11 7 8 21 8 10 2 21 20 3 8 14 17 3 9 11 16 11 1 16**

**The max sum is 211**

**The max path is [0,12]:22 ==> [1,11]:21 ==> [2,10]:17 ==> [3,11]:23 ==> [4,12]:24 ==> [5,12]:24 ==> [6,13]:24 ==> [7,12]:18 ==> [8,11]:21 ==> [9,12]:17**

Run#3

**Enter the number of rows of the grid (between 1 and 10) > 1**

**Enter the number of columns of the grid (between 1 and 20) > 10**

**9 6 13 25 6 23 19 2 13 19**

**The max sum is 25**

**The max path is [0,3]:25**

Run#4

**Enter the number of rows of the grid (between 1 and 10) > 10**

**Enter the number of columns of the grid (between 1 and 20) > 1**

**9**

**10**

**19**

**24**

**10**

**12**

**8**

**17**

**9**

**13**

**The max sum is 131**

**The max path is [0,0]:9 ==> [1,0]:10 ==> [2,0]:19 ==> [3,0]:24 ==> [4,0]:10 ==> [5,0]:12 ==> [6,0]:8 ==> [7,0]:17 ==> [8,0]:9 ==> [9,0]:13**