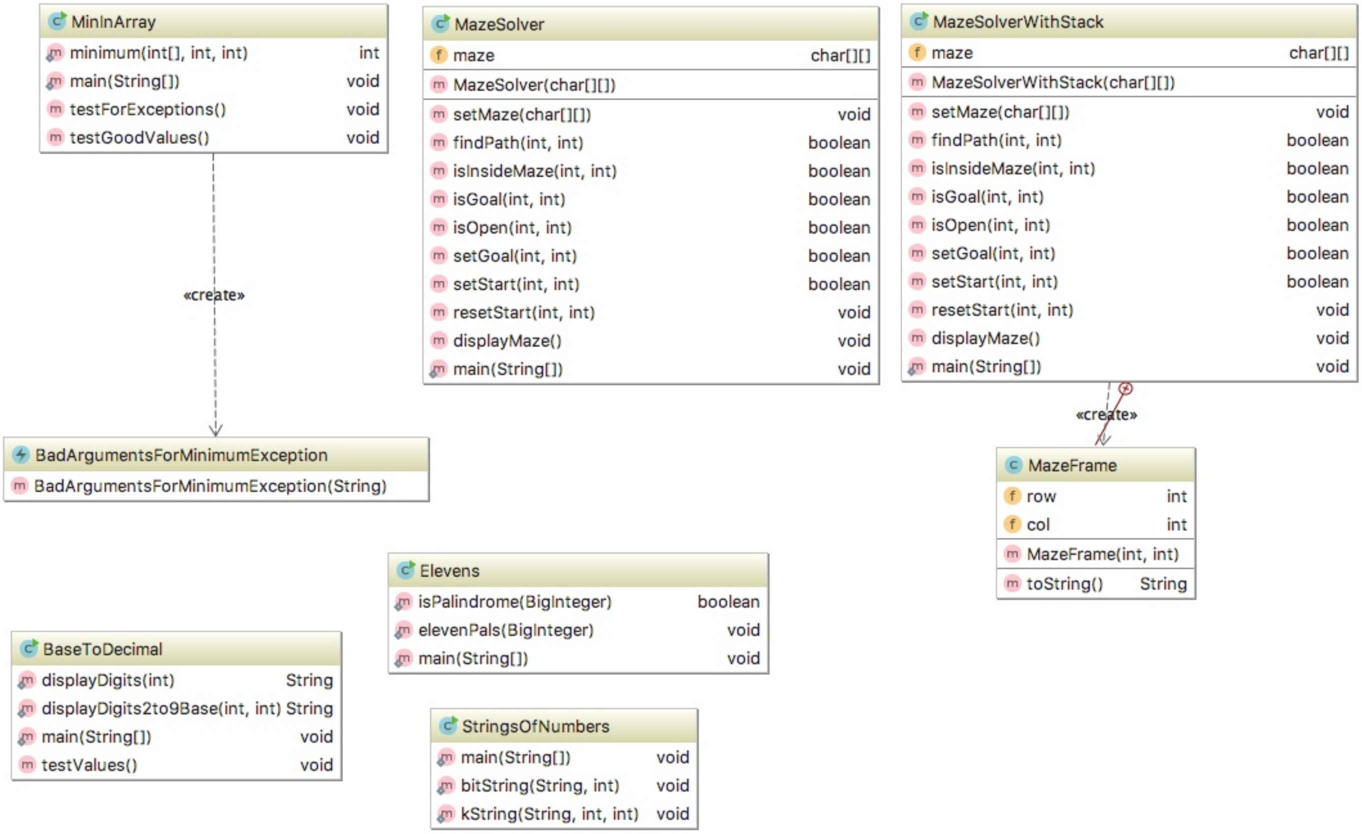
Comp151 Lab05

In Lab05 you will be working on **six** separate applications:

I. `BaseToDecimal`
II. `StringsOfNumbers`
III. `Elevens`
IV. `MazeSolver`
V. `MazeSolverWithStack`
VI. `MinInArray`

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:


MinInArray
minimum(int[], int, int) int
main(String[]) void
testForExceptions() void
testGoodValues() void
«create»
BadArgumentsForMinimumException
BadArgumentsForMinimumException(String)
MazeSolver
maze char[][]
MazeSolver(char[][])
setMaze(char[][]) void
findPath(int, int) boolean
isInsideMaze(int, int) boolean
isGoal(int, int) boolean
isOpen(int, int) boolean
setGoal(int, int) boolean
setStart(int, int) boolean
resetStart(int, int) void
displayMaze() void
main(String[]) void
MazeSolverWithStack
maze char[][]
MazeSolverWithStack(char[][])
setMaze(char[][]) void
findPath(int, int) boolean
isInsideMaze(int, int) boolean
isGoal(int, int) boolean
isOpen(int, int) boolean
setGoal(int, int) boolean
setStart(int, int) boolean
resetStart(int, int) void
displayMaze() void
main(String[]) void
«create»
MazeFrame
row int
col int
MazeFrame(int, int)
toString() String
Elevens
isPalindrome(BigInteger) boolean
elevenPals(BigInteger) void
main(String[]) void
BaseToDecimal
displayDigits(int) String
displayDigits2to9Base(int, int) String
main(String[]) void
testValues() void
StringsOfNumbers
main(String[]) void
bitString(String, int) void
kString(String, int, int) void

I. 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`.

II. Elevens

Write a recursive method that checks the result of sequence of 1s multiplied by the same sequence of 1s for palindrome. In this application, you need to utilize **methods** and **constants** from `BigInteger` class. Implement the algorithm that you designed as part of your pre-lab.

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
```

III. StringsOfNumbers

Write two recursive methods that generate string of numbers. The first method generates all the possible strings that contain the combinations of *n* bits, where *n* is given by the user. The second method is generalization of the first method. It also generates all the permutations of *n* numbers, but the numbers are drown from [0..k); where the *k* is given by the user. See sample runs below. Implement the algorithms that you designed as part of your pre-lab.

SAMPLE RUN:

```
Please enter an integer value of n representing the number of digits in a string
3

Generating binary-Strings:
000
001
010
011
100
101
110
111

Please enter an integer value k; strings of length n will be drown from 0..k-1
4
Generating k-Strings:
000
001
002
```

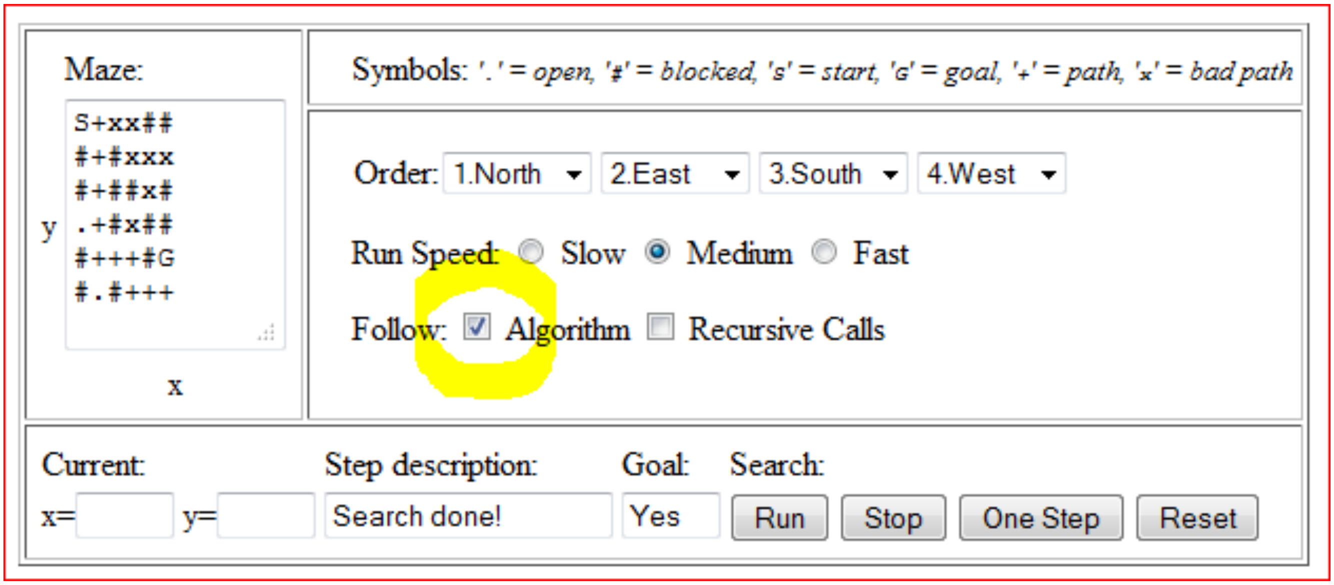
```
003
010
011
012
013
020
021
022
023
030
031
032
033
100
101
102
103
110
111
112
113
120
121
122
123
130
131
132
133
200
201
202
203
210
211
212
213
220
221
222
223
230
231
232
233
300
301
302
303
310
311
312
313
320
321
322
323
330
331
332
333

Process finished with exit code 0
```

IV. Maze Solver

a. Go to https://www.cs.bu.edu/teaching/alg/maze/ and analyze the given algorithm

b. Run the applet at the bottom of the page with the “Algorithm” box checked:



Maze:

```
S+xx##
#+#xxx
#+##x#
.+#x##
#+++#G
#.#+++
```

Symbols: '.' = *open*, '#' = *blocked*, 's' = *start*, 'G' = *goal*, '+' = *path*, 'x' = *bad path*

Order: 1.North 2.East 3.South 4.West

Run Speed: ○ Slow ● Medium ○ Fast

Follow: ☑ Algorithm □ Recursive Calls

Current: x= y=

Step description: Search done!

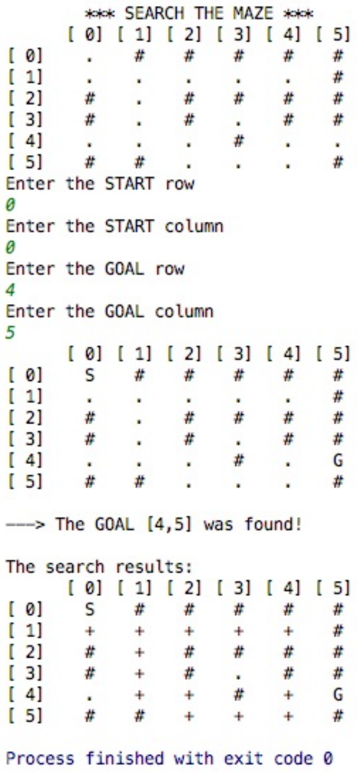
Goal: Yes

c. Implement the following maze search algorithm using recursion, use `MazeSolver.java` as the starting point (please notice that our algorithm does not unmark `[x,y]` if it is not in the solution path):

```
FIND-PATH(x, y)

•  if ([x,y] outside maze) return false
•  if ([x,y] is goal) return true
•  if ([x,y] not open) return false
•  mark [x,y] as part of solution path
•  if (FIND-PATH(North of x,y) == true) return true
•  if (FIND-PATH(East of x,y) == true) return true
•  if (FIND-PATH(South of x,y) == true) return true
•  if (FIND-PATH(West of x,y) == true) return true
•  return false
```

d. See two sample runs below:

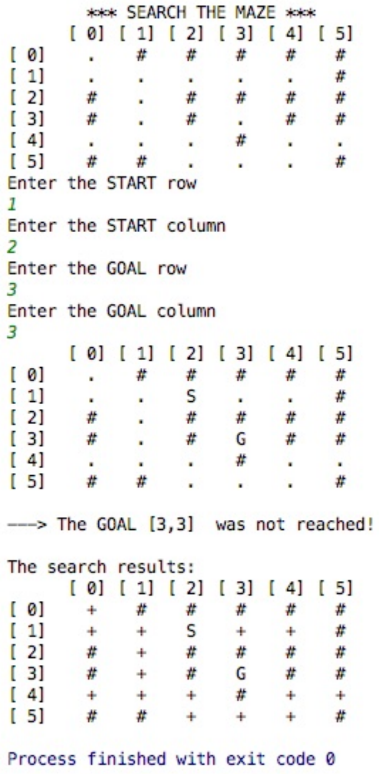


```
        *** SEARCH THE MAZE ***
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    .    #    #    #    #    #
[ 1]    .    .    .    .    .    #
[ 2]    #    .    #    #    #    #
[ 3]    #    .    #    .    #    #
[ 4]    .    .    .    #    .    .
[ 5]    #    #    .    .    .    #
Enter the START row
0
Enter the START column
0
Enter the GOAL row
4
Enter the GOAL column
5
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    S    #    #    #    #    #
[ 1]    .    .    .    .    .    #
[ 2]    #    .    #    #    #    #
[ 3]    #    .    #    .    #    #
[ 4]    .    .    .    #    .    G
[ 5]    #    #    .    .    .    #

---> The GOAL [4,5] was found!

The search results:
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    S    #    #    #    #    #
[ 1]    +    +    +    +    +    #
[ 2]    #    +    #    #    #    #
[ 3]    #    +    #    .    #    #
[ 4]    .    +    +    #    +    G
[ 5]    #    #    +    +    +    #

Process finished with exit code 0
```



```
        *** SEARCH THE MAZE ***
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    .    #    #    #    #    #
[ 1]    .    .    .    .    .    #
[ 2]    #    .    #    #    #    #
[ 3]    #    .    #    .    #    #
[ 4]    .    .    .    #    .    .
[ 5]    #    #    .    .    .    #
Enter the START row
1
Enter the START column
2
Enter the GOAL row
3
Enter the GOAL column
3
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    .    #    #    #    #    #
[ 1]    .    .    S    .    .    #
[ 2]    #    .    #    #    #    #
[ 3]    #    .    #    G    #    #
[ 4]    .    .    .    #    .    .
[ 5]    #    #    .    .    .    #

---> The GOAL [3,3]  was not reached!

The search results:
      [ 0] [ 1] [ 2] [ 3] [ 4] [ 5]
[ 0]    +    #    #    #    #    #
[ 1]    +    +    S    +    +    #
[ 2]    #    +    #    #    #    #
[ 3]    #    +    #    G    #    #
[ 4]    +    +    +    #    +    +
[ 5]    #    #    +    +    +    #

Process finished with exit code 0
```

V. MazeSolverWithStack

Use stack instead of recursion to implement the previous algorithm, use `MazeSolverWithStack.java` as the starting point. Your program should produce the same results as the `MazeSolver.java.`

See segment “Using a Stack Instead of Recursion” on page 224 of your textbook.

VI. 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`.