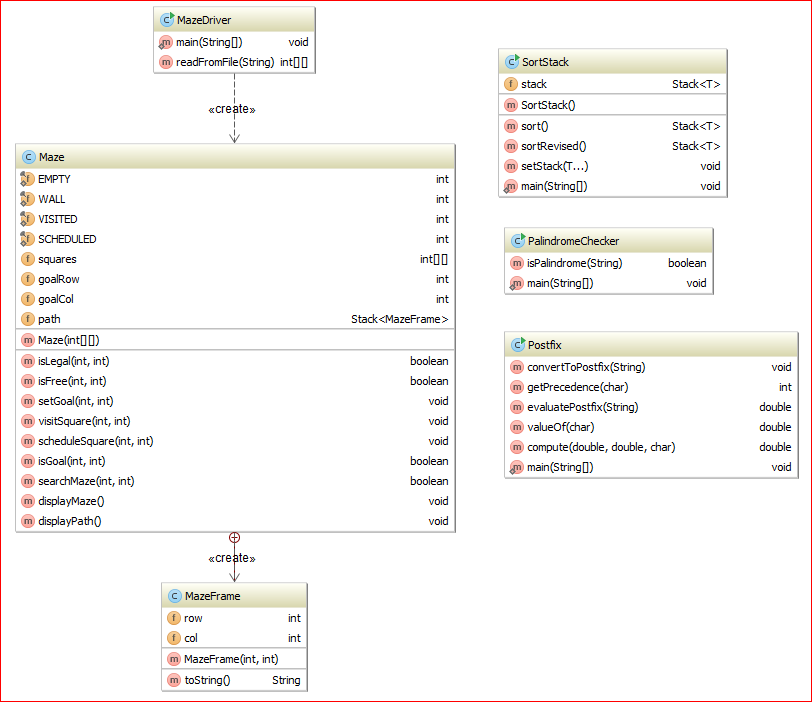
# Comp151 Lab04

Using the Stack class from java.util write four independent applications as per descriptions below:

### UML DIAGRAM



## Application #1

Write a Java program that uses a stack to test whether an input string is a palindrome. A palindrome is a string of characters (a word, phrase, or sentence) that is the same regardless of whether you read it forward or backward – assuming that you ignore spaces, punctuation, and case. Design an algorithm and implement it.

Skeleton of PalindromeChecker class is provided.

### Sample Run:

**\*\*\* This program determines whether a string is a palindrome.**

**A palindrome is spelled the same from left to right as it is from right to left,**

**if we ignore punctuation, spaces, and case. \*\*\***

**Enter a string that you want to check (or enter "stop" to stop):**

**Race car**

**---> Checking: "racecar"**

**"Race car" is a palindrome!**

**Enter a string that you want to check (or enter "stop" to stop):**

**A man, a plan, a canal: Panama!**

**---> Checking: "amanaplanacanalpanama"**

**"A man, a plan, a canal: Panama!" is a palindrome!**

**Enter a string that you want to check (or enter "stop" to stop):**

**abracadabra**

**---> Checking: "abracadabra"**

**"abracadabra" is not a palindrome!**

**Enter a string that you want to check (or enter "stop" to stop):**

**stop**

**Done!**

## Application #2

### Part1

Implement the following algorithm (method sort()) that sorts entries on a stack: ***original*** stack contains entries that are not sorted. First create two empty stacks, ***destination*** and ***temp***. At any given time, ***destination*** stack must hold the entries in sorted order, with the smallest at the top of the stack. Move the top entry of ***original*** stack to ***destination*** stack. Pop and consider the top entry ***topO*** of ***original*** stack. Using a while loop pop entries of ***destination*** stack and push them onto ***temp*** stack until you reach the correct place to put ***topO***. Then push ***topO*** onto ***destination*** stack. Next using a while loop move all the entries from ***temp*** stack to ***destination*** stack. Repeat the process for as long as the original stack is not empty.

### Part2

Now consider the following revision of the above algorithm (method sortRevised()) : after moving the top entry of ***original*** stack to ***destination*** stack, compare the new top entry ***topO*** of the ***original*** stack with the top entry of ***destination*** stack. Then either move entries from ***destination*** stack to ***temp*** stack or from ***temp*** stack to ***destination*** stack until you locate the correct position for ***topO***. Push ***topO*** onto ***destination*** stack. Continue until ***original*** stack is empty. Finally move any entries remaining in ***temp*** stack to ***destination*** stack.

Your finished program should produce the following output:

**Setting the original stack to:**

**03 09 01 04 06 05 07 08 00 02**

**\*\*\*Calling sort method\*\*\***

**push 02 from original to destination**

**Moving entries from destination to temp**

**push 00 to destination**

**Moving entries from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**push 08 to destination**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**push 07 to destination**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**push 05 to destination**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**--> push 05 from destination to temp**

**push 06 to destination**

**Moving entries from temp to destination**

**--> push 05 from temp to destination**

**--> push 02 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**push 04 to destination**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**push 01 to destination**

**Moving entries from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 01 from destination to temp**

**--> push 02 from destination to temp**

**--> push 04 from destination to temp**

**--> push 05 from destination to temp**

**--> push 06 from destination to temp**

**--> push 07 from destination to temp**

**--> push 08 from destination to temp**

**push 09 to destination**

**Moving entries from temp to destination**

**--> push 08 from temp to destination**

**--> push 07 from temp to destination**

**--> push 06 from temp to destination**

**--> push 05 from temp to destination**

**--> push 04 from temp to destination**

**--> push 02 from temp to destination**

**--> push 01 from temp to destination**

**--> push 00 from temp to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 01 from destination to temp**

**--> push 02 from destination to temp**

**push 03 to destination**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 01 from temp to destination**

**--> push 00 from temp to destination**

**Stack should be sorted (with sort()) ....**

**00 01 02 03 04 05 06 07 08 09**

**===================================**

**Testing the revised method**

**Setting the original stack to:**

**03 09 01 04 06 05 07 08 00 02**

**\*\*\*Calling sortRevised method\*\*\***

**--> push 02 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**push 00 from original to destination**

**Moving entries from destination to temp**

**--> push 00 from destination to temp**

**--> push 02 from destination to temp**

**Moving entries from temp to destination**

**push 08 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**push 07 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**push 05 from original to destination**

**Moving entries from destination to temp**

**--> push 05 from destination to temp**

**Moving entries from temp to destination**

**push 06 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**--> push 05 from temp to destination**

**push 04 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**--> push 02 from temp to destination**

**push 01 from original to destination**

**Moving entries from destination to temp**

**--> push 01 from destination to temp**

**--> push 02 from destination to temp**

**--> push 04 from destination to temp**

**--> push 05 from destination to temp**

**--> push 06 from destination to temp**

**--> push 07 from destination to temp**

**--> push 08 from destination to temp**

**Moving entries from temp to destination**

**push 09 from original to destination**

**Moving entries from destination to temp**

**Moving entries from temp to destination**

**--> push 08 from temp to destination**

**--> push 07 from temp to destination**

**--> push 06 from temp to destination**

**--> push 05 from temp to destination**

**--> push 04 from temp to destination**

**push 03 from original to destination**

**Moving any remaining entries from temp to destination**

**--> push 02 from temp to destination**

**--> push 01 from temp to destination**

**--> push 00 from temp to destination**

**Stack should be sorted (with sortRevised()) ....**

**00 01 02 03 04 05 06 07 08 09**

## Application #3

Write an application that converts infix expression to postfix and evaluates it.

Class Postfix includes convertToPostfix and evaluatePostfix methods that should implement algorithms given in Segments 5.16 and 5.18 respectively. Assume that the given algebraic expressions are syntactically correct. The String method replaceAll, and StringBuilder class should be utilized.

### Your Task:

1. Analyze provided skeleton, UML diagram and the sample run.
2. When working with the Stack use only the stack’s "vanilla" methods: push, pop, and peek.

### Sample Run:

**Converting infix expressions to postfix expressions:**

**Infix: a+b**

**Postfix: ab+**

**Infix: (a + b) \* c**

**Postfix: ab+c\***

**Infix: a \* b / (c - d)**

**Postfix: ab\*cd-/**

**Infix: a / b + (c - d)**

**Postfix: ab/cd-+**

**Infix: a / b + c - d**

**Postfix: ab/c+d-**

**Infix: a^b^c**

**Postfix: abc^^**

**Infix: (a^b)^c**

**Postfix: ab^c^**

**Infix: a\*(b/c+d)**

**Postfix: abc/d+\***

**Infix: (a+b)/(c-d)**

**Postfix: ab+cd-/**

**Infix: a/(b-c)\*d**

**Postfix: abc-/d\***

**Infix: a-(b/(c-d)\*e+f)^g**

**Postfix: abcd-/e\*f+g^-**

**Infix: (a-b\*c)/(d\*e^f\*g+h)**

**Postfix: abc\*-def^\*g\*h+/**

**Evaluating postfix expressions with**

**a = 2, b = 3, c = 4, d = 5, e = 6**

**Assuming correct input!!!**

**ae+bd-/ : -4.0**

**abc\*d\*- : -58.0**

**abc-/d\* : -10.0**

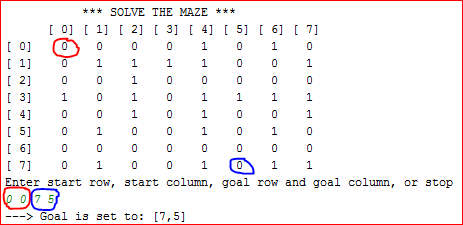
**ebca^\*+d- : 49.0**

**Done.**

## Application #4

You will be searching the given *maze* using a stack.

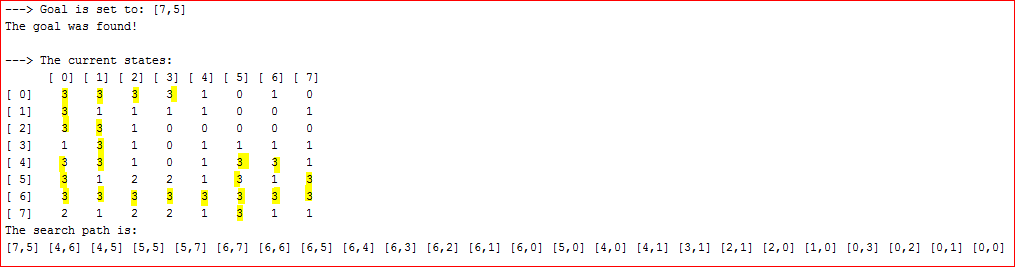
A *maze* is represented by two-dimensional array filled with 0s and 1s, where 1 represents a wall and 0 represents a free cell which will allow the search path to pass thru. You can move only up, down, left, and right.



Each square in the *maze* can be in one of the four states: **EMPTY (0)**, **WALL (1)**, **SCHEDULED (2)** or **VISITED (3)**. Initially each square is either **WALL** or **EMPTY**. Which way to move? Is there anything preventing moving to a neighboring square? Clearly there is a need to mark the visited position so they are not visited again. But there were four directions to choose from at the first step, so how to go back?

We will be using MazeFrame objects and a local stack of MazeFrame objects:

* Anytime a square is placed on the stack as a potential direction will be marked as **SCHEDULED**.
* We will not visit a **SCHEDULED** square until it is taken off from the stack.
* When the square is taken off from the stack we will:
  + mark it **VISITED**,
  + add it to the search path (see the instance variable path) and
  + check if it is the goal we are searching for, if yes we stop the search
* If it is not the goal we schedule visits for its four neighbors if possible (see Sample Runs).
* The result should show the search path:



*The basic algorithm for searching maze in general terms is as follow:*

1. *Create a stack local to the method*
2. *Put the initial frame on the local stack*
3. *While local stack is not empty and the goal is not found*
   1. *Take the top frame from the local stack and mark the square visited (each visited square should be pushed on the search path stack)*
   2. *If the top frame represents the goal, mark the square visited and set result to found*
   3. *Otherwise schedule visits for all four neighbors if they are free (push appropriate frames on the local stack)*
4. *Output the search path stack as the solution*

The *maze* is defined in a text file. The first line in the file contains the number of rows and the number of columns. The subsequent lines contain the data. The file is processed by the MazeDriver class and the maze object is created. The *maze array* is displayed and the user is prompted for the coordinates as to where to start the search from, and what are the coordinates of the target the search is going to look for. If the goal is behind walls it will not be found, otherwise the goal should be found. In both cases the search path is displayed.

The MazeDriver class is fully implemented; MazeFrame class is implemented as an inner class of the Maze class; you will be working with the Maze class as defined in the UML diagram above, some of the methods are already implemented.

Six different mazes are provided for your testing: maze1.txt, … , maze6.txt.

The following is a sample run of the finished application that utilizes maze2.txt. Please note two different scenarios: one where the goal was found and one where the goal was not found because it was unreachable. Since the search path was recorded on the stack, the first coordinates displayed represent the last square visited.

Please note that the implementation shown in the sample runs below schedules visits in the following order: ***south* (row+1), *north* (row-1), *east* (column+1), and *west* (column-1).**

See sample run below.

### Sample Run:

**\*\*\* SOLVE THE MAZE \*\*\***

**[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]**

**[ 0] 0 0 0 0 1 0 1 0**

**[ 1] 0 1 1 1 1 0 0 1**

**[ 2] 0 0 1 0 0 0 0 0**

**[ 3] 1 0 1 0 1 1 1 1**

**[ 4] 0 0 1 0 1 0 0 1**

**[ 5] 0 1 0 0 1 0 1 0**

**[ 6] 0 0 0 0 0 0 0 0**

**[ 7] 0 1 0 0 1 0 1 1**

**Enter start row, start column, goal row and goal column, or stop**

**0 0 7 5**

**---> Goal is set to: [7,5]**

**[0,0] set to VISITED**

**[1,0] set to SCHEDULED**

**[0,1] set to SCHEDULED**

**[0,1] set to VISITED**

**[0,2] set to SCHEDULED**

**[0,2] set to VISITED**

**[0,3] set to SCHEDULED**

**[0,3] set to VISITED**

**[1,0] set to VISITED**

**[2,0] set to SCHEDULED**

**[2,0] set to VISITED**

**[2,1] set to SCHEDULED**

**[2,1] set to VISITED**

**[3,1] set to SCHEDULED**

**[3,1] set to VISITED**

**[4,1] set to SCHEDULED**

**[4,1] set to VISITED**

**[4,0] set to SCHEDULED**

**[4,0] set to VISITED**

**[5,0] set to SCHEDULED**

**[5,0] set to VISITED**

**[6,0] set to SCHEDULED**

**[6,0] set to VISITED**

**[7,0] set to SCHEDULED**

**[6,1] set to SCHEDULED**

**[6,1] set to VISITED**

**[6,2] set to SCHEDULED**

**[6,2] set to VISITED**

**[7,2] set to SCHEDULED**

**[5,2] set to SCHEDULED**

**[6,3] set to SCHEDULED**

**[6,3] set to VISITED**

**[7,3] set to SCHEDULED**

**[5,3] set to SCHEDULED**

**[6,4] set to SCHEDULED**

**[6,4] set to VISITED**

**[6,5] set to SCHEDULED**

**[6,5] set to VISITED**

**[7,5] set to SCHEDULED**

**[5,5] set to SCHEDULED**

**[6,6] set to SCHEDULED**

**[6,6] set to VISITED**

**[6,7] set to SCHEDULED**

**[6,7] set to VISITED**

**[5,7] set to SCHEDULED**

**[5,7] set to VISITED**

**[5,5] set to VISITED**

**[4,5] set to SCHEDULED**

**[4,5] set to VISITED**

**[4,6] set to SCHEDULED**

**[4,6] set to VISITED**

**[7,5] set to VISITED**

**The goal was found!**

**---> The current states:**

**[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]**

**[ 0] 3 3 3 3 1 0 1 0**

**[ 1] 3 1 1 1 1 0 0 1**

**[ 2] 3 3 1 0 0 0 0 0**

**[ 3] 1 3 1 0 1 1 1 1**

**[ 4] 3 3 1 0 1 3 3 1**

**[ 5] 3 1 2 2 1 3 1 3**

**[ 6] 3 3 3 3 3 3 3 3**

**[ 7] 2 1 2 2 1 3 1 1**

**The search path is:**

**[7,5] [4,6] [4,5] [5,5] [5,7] [6,7] [6,6] [6,5] [6,4] [6,3] [6,2] [6,1] [6,0] [5,0] [4,0] [4,1] [3,1] [2,1] [2,0] [1,0] [0,3] [0,2] [0,1] [0,0]**

**\*\*\* SOLVE THE MAZE \*\*\***

**[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]**

**[ 0] 0 0 0 0 1 0 1 0**

**[ 1] 0 1 1 1 1 0 0 1**

**[ 2] 0 0 1 0 0 0 0 0**

**[ 3] 1 0 1 0 1 1 1 1**

**[ 4] 0 0 1 0 1 0 0 1**

**[ 5] 0 1 0 0 1 0 1 0**

**[ 6] 0 0 0 0 0 0 0 0**

**[ 7] 0 1 0 0 1 0 1 1**

**Enter start row, start column, goal row and goal column, or stop**

**0 0 0 7**

**---> Goal is set to: [0,7]**

**[0,0] set to VISITED**

**[1,0] set to SCHEDULED**

**[0,1] set to SCHEDULED**

**[0,1] set to VISITED**

**[0,2] set to SCHEDULED**

**[0,2] set to VISITED**

**[0,3] set to SCHEDULED**

**[0,3] set to VISITED**

**[1,0] set to VISITED**

**[2,0] set to SCHEDULED**

**[2,0] set to VISITED**

**[2,1] set to SCHEDULED**

**[2,1] set to VISITED**

**[3,1] set to SCHEDULED**

**[3,1] set to VISITED**

**[4,1] set to SCHEDULED**

**[4,1] set to VISITED**

**[4,0] set to SCHEDULED**

**[4,0] set to VISITED**

**[5,0] set to SCHEDULED**

**[5,0] set to VISITED**

**[6,0] set to SCHEDULED**

**[6,0] set to VISITED**

**[7,0] set to SCHEDULED**

**[6,1] set to SCHEDULED**

**[6,1] set to VISITED**

**[6,2] set to SCHEDULED**

**[6,2] set to VISITED**

**[7,2] set to SCHEDULED**

**[5,2] set to SCHEDULED**

**[6,3] set to SCHEDULED**

**[6,3] set to VISITED**

**[7,3] set to SCHEDULED**

**[5,3] set to SCHEDULED**

**[6,4] set to SCHEDULED**

**[6,4] set to VISITED**

**[6,5] set to SCHEDULED**

**[6,5] set to VISITED**

**[7,5] set to SCHEDULED**

**[5,5] set to SCHEDULED**

**[6,6] set to SCHEDULED**

**[6,6] set to VISITED**

**[6,7] set to SCHEDULED**

**[6,7] set to VISITED**

**[5,7] set to SCHEDULED**

**[5,7] set to VISITED**

**[5,5] set to VISITED**

**[4,5] set to SCHEDULED**

**[4,5] set to VISITED**

**[4,6] set to SCHEDULED**

**[4,6] set to VISITED**

**[7,5] set to VISITED**

**[5,3] set to VISITED**

**[4,3] set to SCHEDULED**

**[4,3] set to VISITED**

**[3,3] set to SCHEDULED**

**[3,3] set to VISITED**

**[2,3] set to SCHEDULED**

**[2,3] set to VISITED**

**[2,4] set to SCHEDULED**

**[2,4] set to VISITED**

**[2,5] set to SCHEDULED**

**[2,5] set to VISITED**

**[1,5] set to SCHEDULED**

**[2,6] set to SCHEDULED**

**[2,6] set to VISITED**

**[1,6] set to SCHEDULED**

**[2,7] set to SCHEDULED**

**[2,7] set to VISITED**

**[1,6] set to VISITED**

**[1,5] set to VISITED**

**[0,5] set to SCHEDULED**

**[0,5] set to VISITED**

**[7,3] set to VISITED**

**[5,2] set to VISITED**

**[7,2] set to VISITED**

**[7,0] set to VISITED**

**The goal was not found!**

**---> The current states:**

**[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]**

**[ 0] 3 3 3 3 1 3 1 0**

**[ 1] 3 1 1 1 1 3 3 1**

**[ 2] 3 3 1 3 3 3 3 3**

**[ 3] 1 3 1 3 1 1 1 1**

**[ 4] 3 3 1 3 1 3 3 1**

**[ 5] 3 1 3 3 1 3 1 3**

**[ 6] 3 3 3 3 3 3 3 3**

**[ 7] 3 1 3 3 1 3 1 1**

**The search path is:**

**[7,0] [7,2] [5,2] [7,3] [0,5] [1,5] [1,6] [2,7] [2,6] [2,5] [2,4] [2,3] [3,3] [4,3] [5,3] [7,5] [4,6] [4,5] [5,5] [5,7] [6,7] [6,6] [6,5] [6,4] [6,3] [6,2] [6,1] [6,0] [5,0] [4,0] [4,1] [3,1] [2,1] [2,0] [1,0] [0,3] [0,2] [0,1] [0,0]**

**\*\*\* SOLVE THE MAZE \*\*\***

**[ 0] [ 1] [ 2] [ 3] [ 4] [ 5] [ 6] [ 7]**

**[ 0] 0 0 0 0 1 0 1 0**

**[ 1] 0 1 1 1 1 0 0 1**

**[ 2] 0 0 1 0 0 0 0 0**

**[ 3] 1 0 1 0 1 1 1 1**

**[ 4] 0 0 1 0 1 0 0 1**

**[ 5] 0 1 0 0 1 0 1 0**

**[ 6] 0 0 0 0 0 0 0 0**

**[ 7] 0 1 0 0 1 0 1 1**

**Enter start row, start column, goal row and goal column, or stop**

**stop**

**Bye!**