REVISION HISTORY

Date	Version	Description	Author
29/11/2021	0.1	Design of the Sections	Muhammed Esad Simitcioglu
30/11/2021	0.5	Diagrams added	Muhammed Esad Simitcioglu
16/12/2021	1.0	New informations added	Muhammed Esad Simitcioglu

TABLE OF CONTENTS

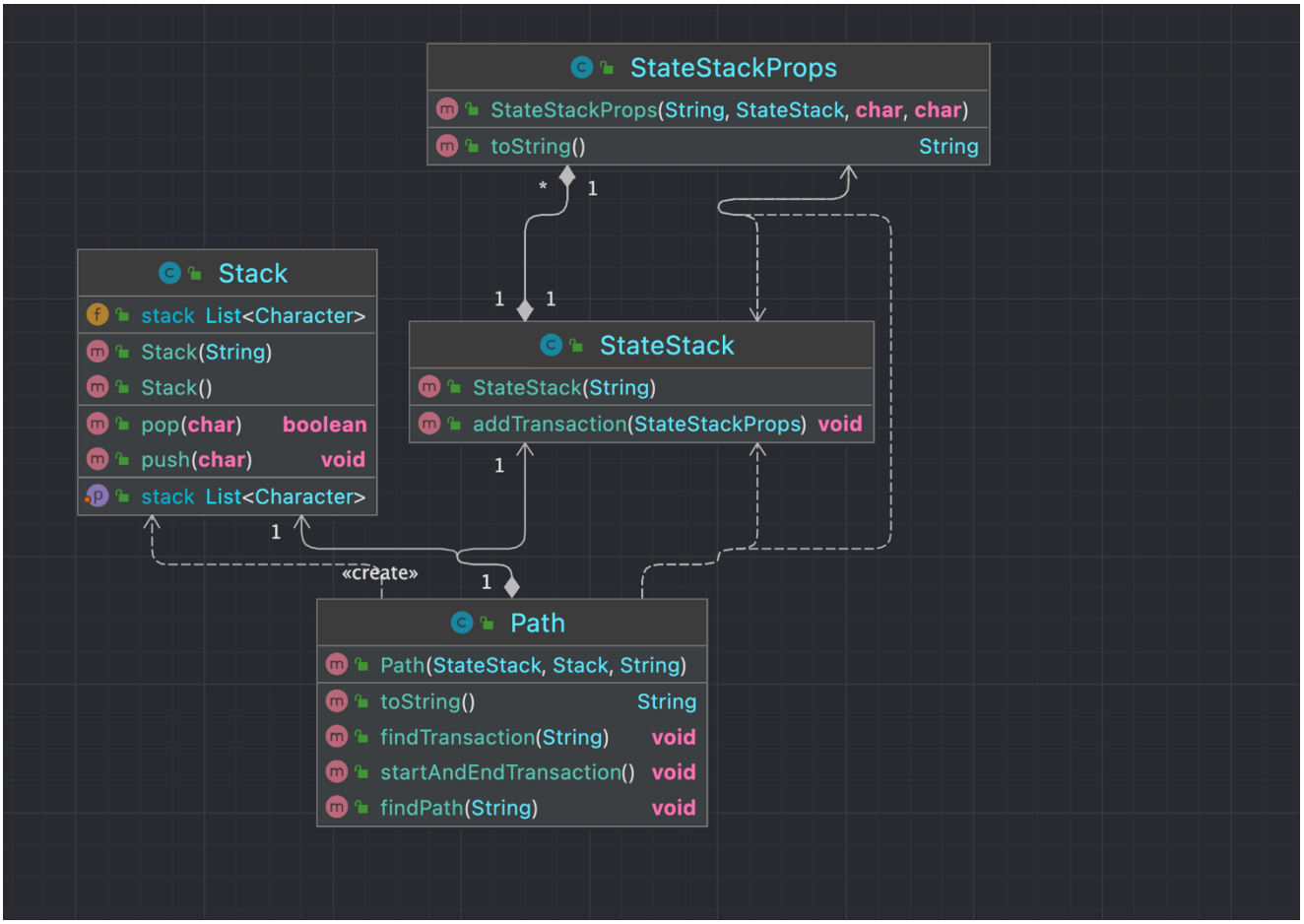
Table of Contents

1 Introduction

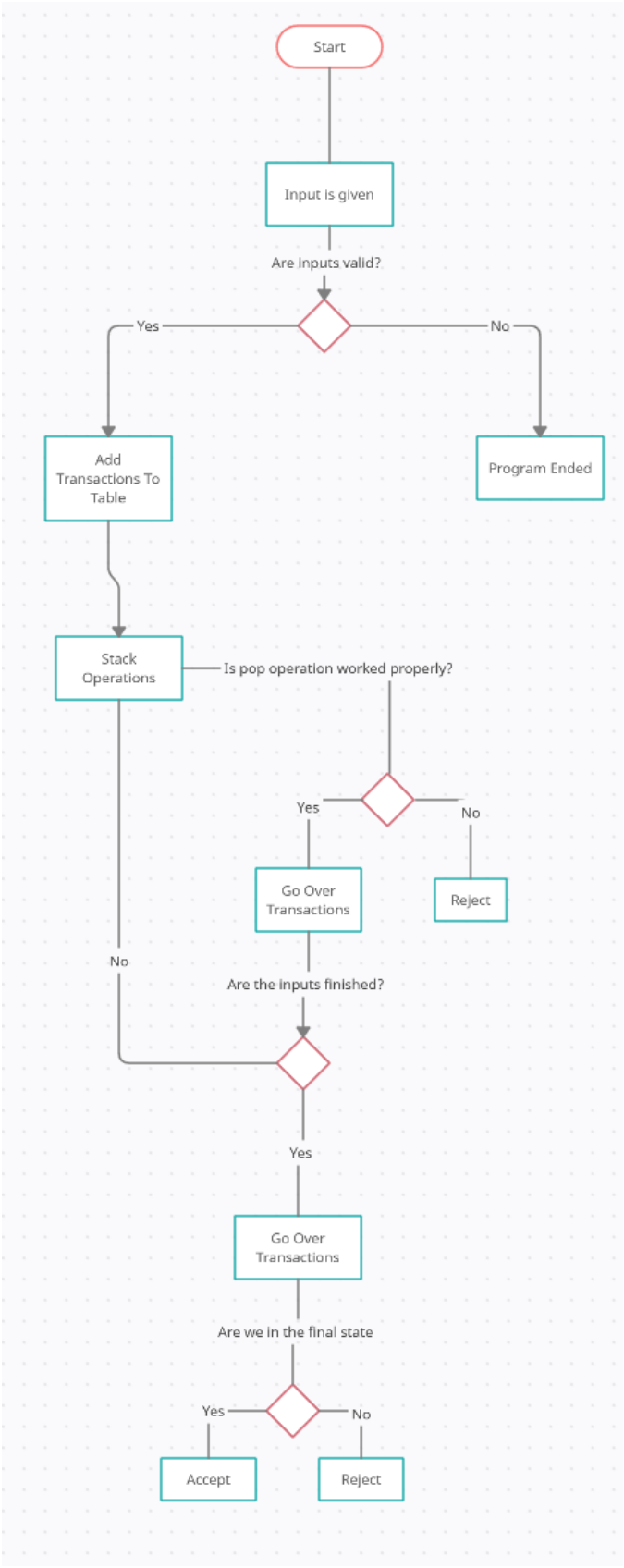
This document describes the design of a Push Down Automata (PDA). This program should be working exactly as a PDA would do. Given a string the simulated PDA should be able to tell if the string is accepted or rejected. Your program should output the information about the given string being accepted or rejected as well as which states it visited

2 Software Architecture overview

Class diagram is given below.



The Activity Diagram is given below:



3 Software design description

This program will not be desktop application and no need for database. So, in that case there won't be a UI or database implementation. The input and output format of the program should be like in the below.

Input:

2 (number of variables in input alphabet)
2 (number of variables in stack alphabet)
2 (number of goal states)
4 (number of states)
q1 q2 q3 q4 (states)
q1 (start state)
q1 q4 (goal state(s))
X Y (the stack alphabet)
X (initial stack symbol)
0 1 (the input alphabet)
q1 ε ε X q2 (q1 state'inden ε ile q2 state'ine gidiyor, ε popluyor, X pushluyor.)
q2 0 ε Y q2 (q2 state'inden 0 ile q2 state'ine gidiyor, ε popluyor, Y pushluyor.)
q2 1 Y ε q3 (q2 state'inden 1 ile q3 state'ine gidiyor, Y popluyor, ε pushluyor.)
q3 1 Y ε q3 (q3 state'inden 1 ile q3 state'ine gidiyor, Y popluyor, ε pushluyor.)
q3 ε X ε q4 (q3 state'inden ε ile q4 state'ine gidiyor, X popluyor, ε pushluyor.)
0011 (string to be detected)
0111 (string to be detected)

Output:

q1 q2 q2 q2 q3 q3 q4 (route taken)
Accepted
q1 q2 q2 q3 (route taken)
Rejected

The program will construct a List of "State" object to store all the given states. After constructing and connecting "States", the program will create transaction table for each "State". Each "State" will know where to go when the appropriate variable arrives. Stack operations will done after each iteration if there is any operation needed (e.g. pop or push) The name of each "State" will be printed to reveal the route the program took. Finally, the program will print "Accepted" if the last "State" is one of the target states. Otherwise the program will print "Rejected".

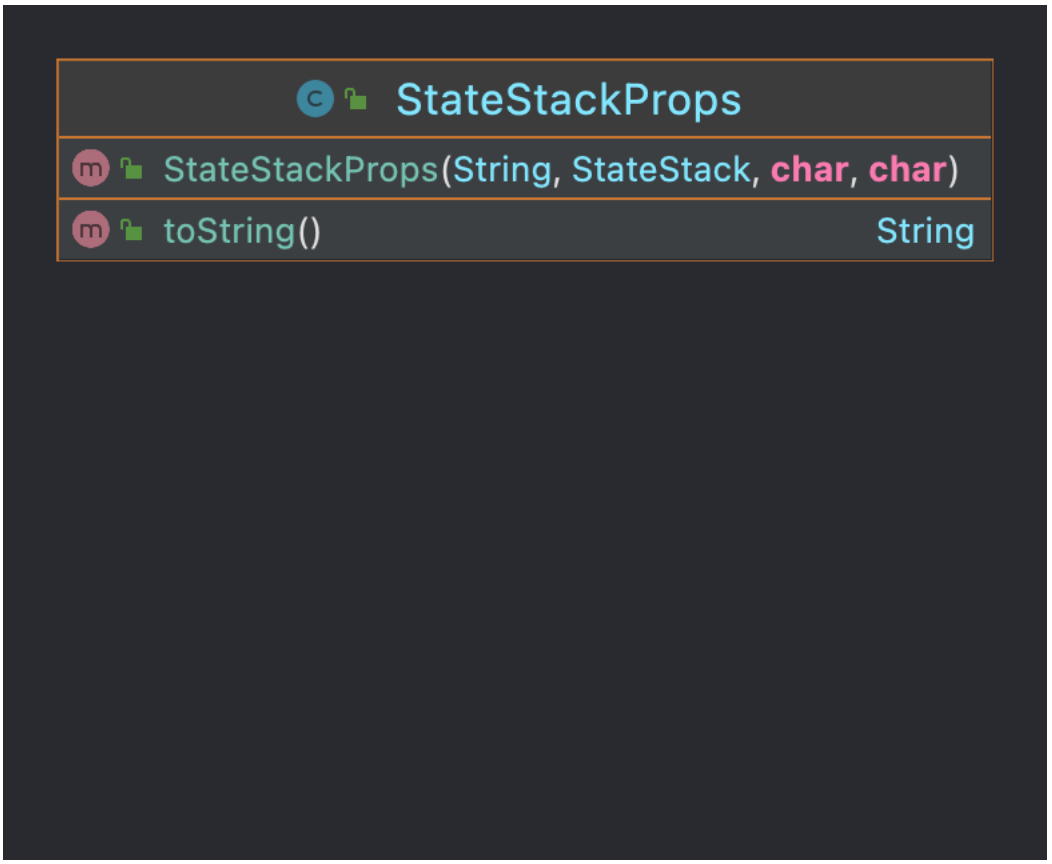
3.1 StateStackProps

3.1.1 Component Interface

Methods of State which are available from other components are:

public StateStackProps(String, State) {} – Creates the State properties as specific variable, pop , push variable and its next state.

3.1.2 Component Design Description



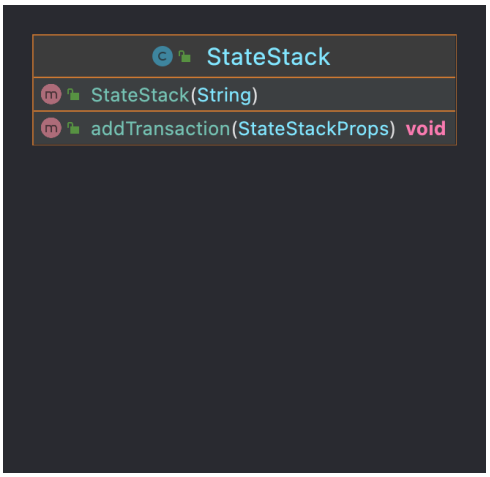
3.2 StateStack

3.2.1 Component Interface

Methods of State which are available from other components are:

public void addTransaction (StateStackProps) {} – Add StateStackProps to the specific state.

3.2.2 Component Design Description



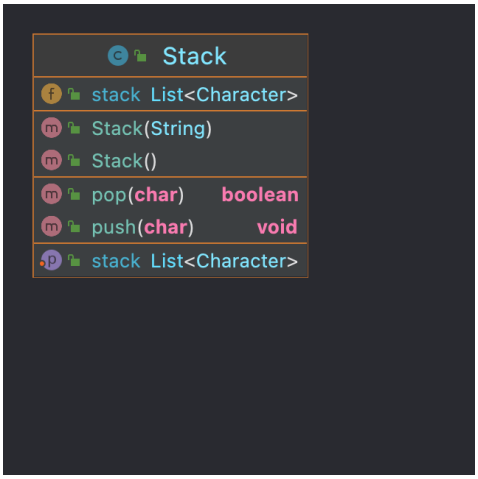
3.3 Stack

3.3.1 Component Interface

Methods of State which are available from other components are:

public void push (String variable) {} – push variable to the Stack
public State pop () {} –pop variable from the Stack.

3.3.2 Component Design Description

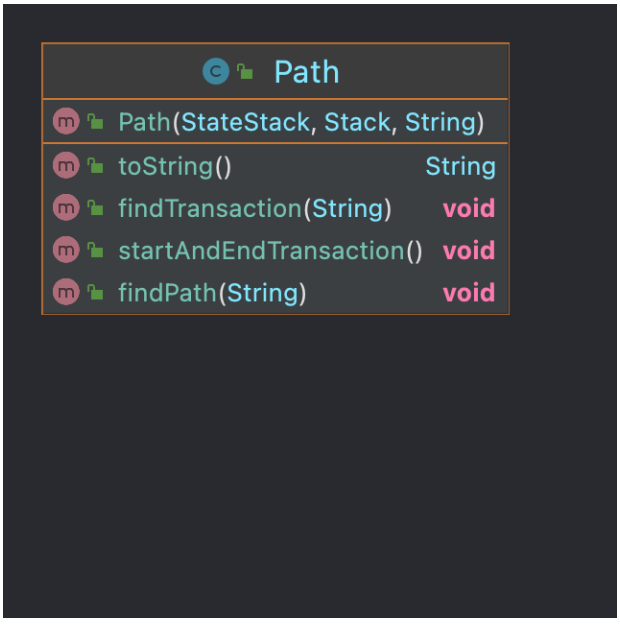


3.4 Path

3.4.1 Component Interface

public void findTransaction (String) {} – iterates the states except start and last state
public void startAndEndTransaction () {} –iterates the states only start and last state
public void findPath (String) {} – define a path from beginning to its end state.

3.4.2 Component Design Description



4 COTS Identification

COTS (Commercial off the shell) libraries used :

Not Applicable