

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 1 / 8

Revision History

Date	Version	Description	Author
02.01.2024	1.0	Design of the program.	Elif Özmen

TABLE OF CONTENTS

1 Introduction	2
1.1 References	2
1.1.1 Project References	2
2 Software Architecture overview	2
3 Software design description	3
3.1.1 Component interfaces	3
3.1.2 Designing Description	3
3.1.3 Workflows and algorithms	4
4 COTS Identification	6
5 Testing and Error Handling	7

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 2 / 8

1 Introduction

In this report I mainly tried to explain how I implement the assignment given in my CS410 class. Report mainly focuses on the introduction to the problem, design considerations, algorithms and data structures used in implementation, input/output format, error handling, testing, code structure and challenges faced during implementation.

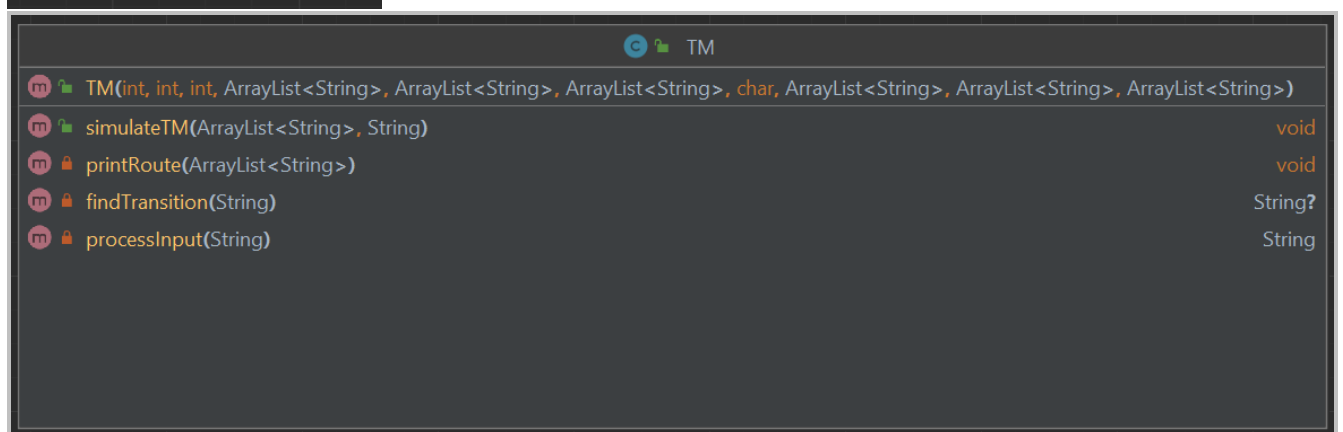
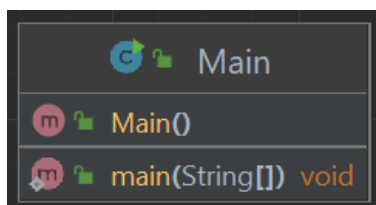
The problem in this project was simulating a Turing Machine (TM) which works with any given alphabet, number of states, number of variables and transitions. The simulation has to work correctly as a Turing Machine would do and it should read the information from the given input file then print the output information about the given string, states visited, string being accepted or rejected.

1.1 References

1.1.1 Project References

#	Document Identifier	Document Title
[SRS]	TM-SRS-2	TM Software Requirements Specifications

2 Software Architecture overview



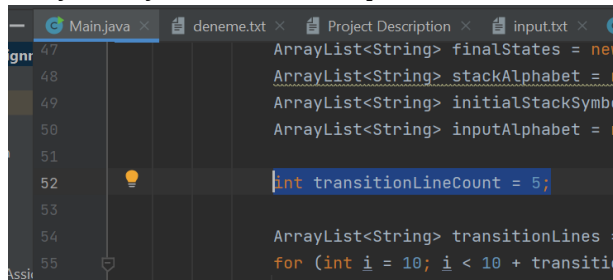
Design of TM		
Doc # TM-SDD	Version: 1.0	Page 3 / 8

3 Software design description

3.1.1 Component interfaces

Input Data: Input is taken from the text file constructed with the same way with the example input file given. Input file's name is taken from the user in the console.

It is important to not to forget to change the "transitionLineCount" in the main according to how many lines you have in the input file.



```

47 ArrayList<String> finalStates = new
48 ArrayList<String> stackAlphabet = n
49 ArrayList<String> initialStackSymbo
50 ArrayList<String> inputAlphabet = n
51
52 int transitionLineCount = 5;
53
54 ArrayList<String> transitionLines =
55 for (int i = 10; i < 10 + transitio

```

Output Data: In the program the results are written in a output file and also to the console. Output file's name is taken from the user in the console.

3.1.2 Designing Description

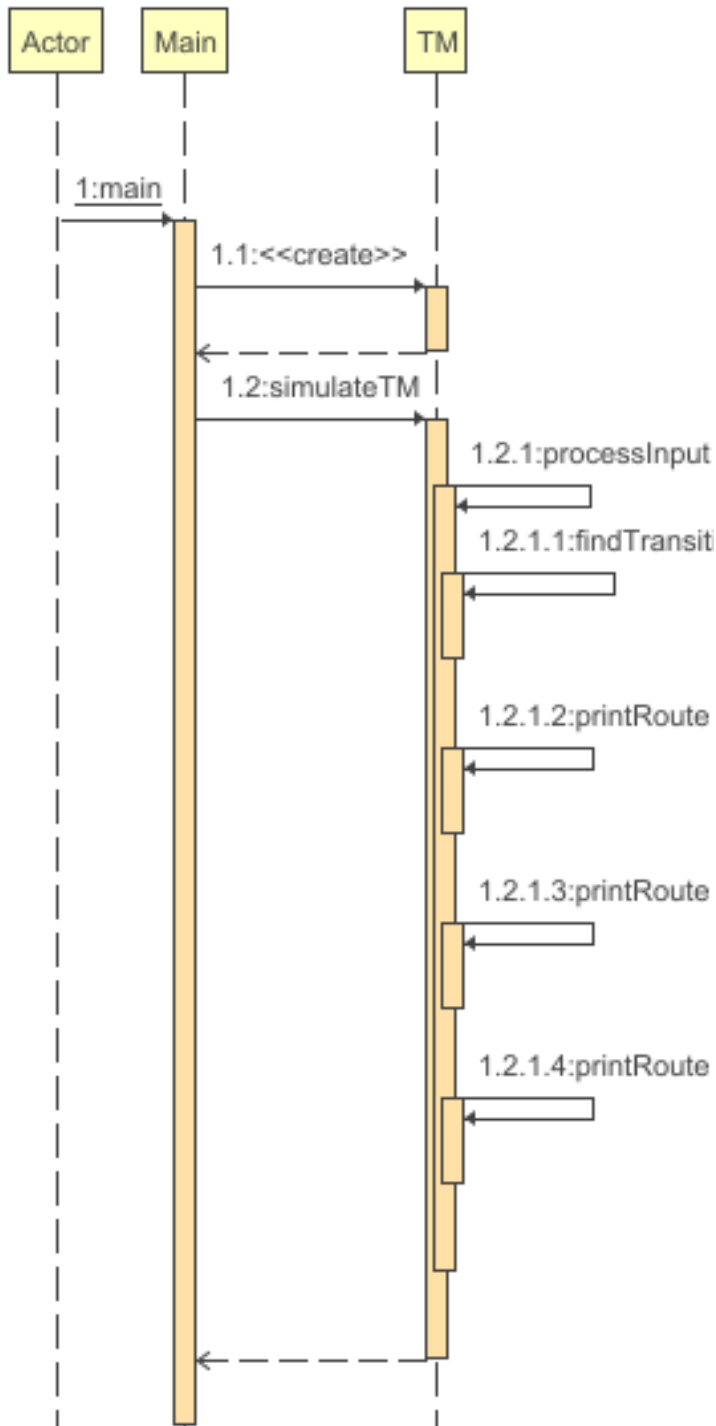
Before starting to implement, I considered how to read the information from the input file. I stored each line in an ArrayList. In the given example input file it shows that the first line is the number of variables in the input alphabet, second line is the number of variables in the tape alphabet,, third line is the number of states in TM. For the first three lines, since they are the count of the elements, I decided to parse them to integer and make each one integer parameters. For the fourthline of the input file, it is the information about the states names. I decided to keep them in a ArrayList of strings, each state name is one element in the list. As it is similar in the sixth, seventh, eighth, ninth and the tenth lines which are the names of start state, accept state, reject states, blank symbol, tape alphabet and input alphabet symbols, I applied the same process to them. It is always precise, the number of the first 10 lines in the input file and what information they contain. However, it starts to move to transition lines on line 11, and this can change to any number depending on the number of states and variables. The transition lines do not include every element of the transition table's every row and column. But it may change according to the input file and I could not find a way to determine it with a generic code. So I assigned it to 15 as it was in the example input file. If there is an input file with more or less lines it needs to be changed before running. After the transition lines end, in the input file, there starts the string lines that will be taken to check if it is accepted or not in the TM. So it starts with the (11 + transitionLineCount) line and continues until the input file ends. Again each string is an element of the ArrayList.

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 4 / 8

After reading the input file and retrieving the information, I had to construct the TM. So I created a class called TM and I created a constructor for this class with the parameters I took from the input file. In TM class there are four methods. “simulateTM” method is for analyzing each input string and deciding if the TM will accept it or reject it or loops. simulateTM method takes the inputLines, calls the printRoute to keep the route. Here, processInput method is called with inputLines to make the TM work. Also the information of whether the strings are accepted or not is printed. In the processInput method, it applies the transition algorithms by calling findTransiton method. It parts the transition lines as a String and for each input symbol it checks the transition lines which fits to make a transition with the findTransiton method. The processInput runs a while loop, which checks if the reached state is whether accept state or reject state, returns the strings. After I implemented these I called the TM constructor in the main method. I decided to print the results to both the console and to an output file. I decided to use Scanner because I wanted to take the input file’s name from the user to make it more flexible. Also I decided to take the output file’s name from the user, again with the same reason. So, this was my decision and implementation phase.

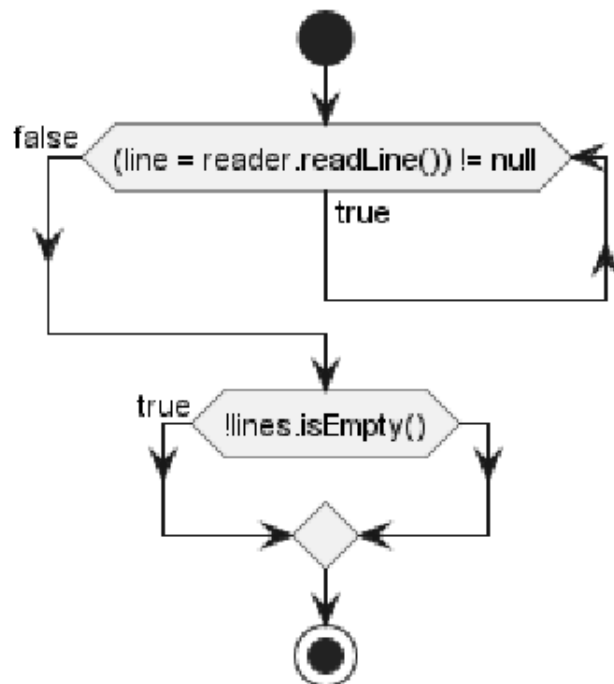
3.1.3 Workflows and algorithms

The Turing Machine (TM) implementation use key data structures and algorithms to simulate the behavior of a Turing Machine. Fundamental data structures include arrays and ArrayLists used to store states, transitions, tape alphabet, and input alphabet. Strings and char arrays are used for symbol representation, while StringBuilder handles the construction of the route taken during the simulation. Primitive data types such as int and char store essential information like the number of input alphabets and the blank symbol. The core simulation algorithm, encapsulated in the simulateTM method, iterates through input symbols, updating the tape and transitioning between states based on defined rules. The findTransition algorithm searches for the appropriate transition rule, while printRoute constructs the route taken. The processInput algorithm manages input string processing, tape updates, and state transitions until acceptance, rejection, or looping conditions arise.

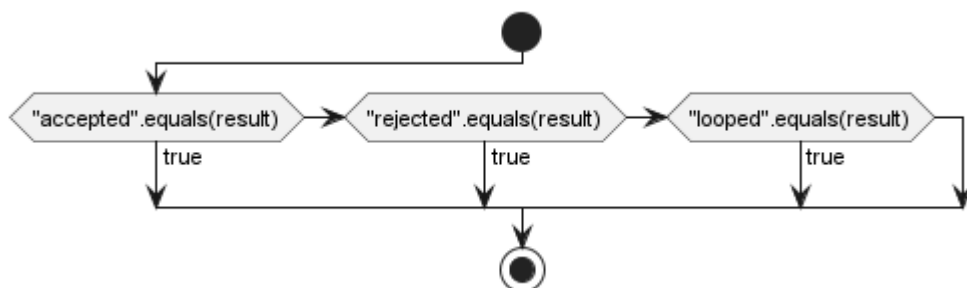


Main method's Flow Diagram:

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 6 / 8

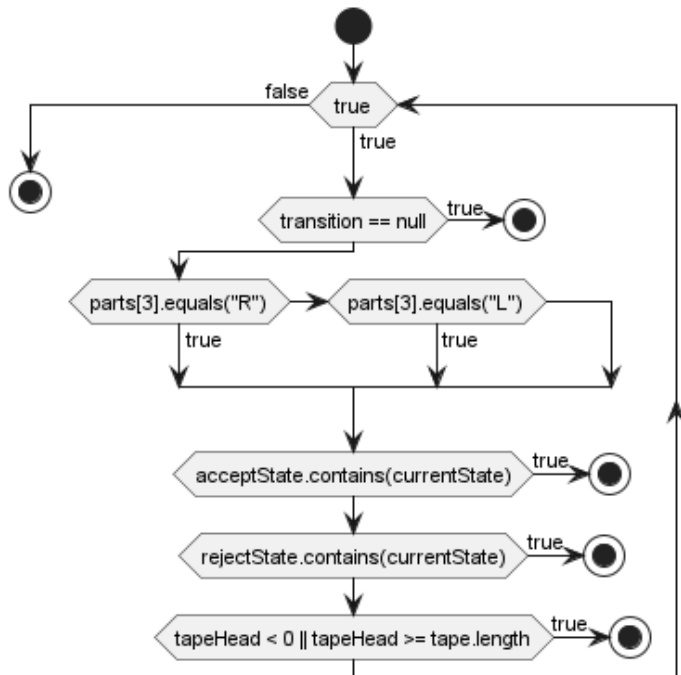


simulateTM Flow Diagram:



processInput Flow Diagram:

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 7 / 8



4 COTS Identification

COTS (commercial of the shelf) libraries used in DFA are the following:

- java.io.IOException;
- java.util.Scanner;
- java.io.BufferedReader;
- java.io.FileReader;
- java.io.FileWriter;
- java.io.BufferedWriter;
- java.io.PrintWriter;
- java.util.Arrays;
- java.util.ArrayList;

5 Testing and Error Handling

To test whether my program was working correctly, I used the Turing Machine that given in the example input file.

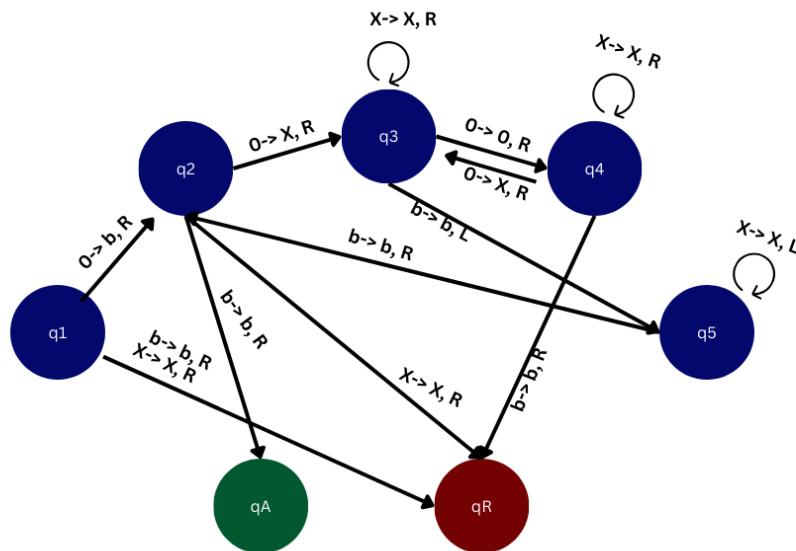
For the given example TM , it was working correct for every string.

The most important thing about my code is not to forget to change the “transitionLineCount” according to how many transition line you wrote on the input file.

For the error handling I used the IDE’s tools. I have three exceptions: IOException, FileNotFoundException and NumberFormatException.

Design of TM		
Doc # TM-SDD	Version: 1.0	Page 8 / 8

Here is the TM constructed:



6 Challenges Faced

In this assignment, it was hard to determine the logic and implement it since there are so many things to check while working with the tape structure. There were so many problems about the logic and while trying to solve them I used so many debug prints, it helped me to check which transitions are done, what it writes and read from the tape, what is the route and what are the current states. Another problem I faced was it did not print the qA and qR states to the route when it has reached. Again I used the debug prints to check if it correctly takes the states and I realized that the problem was because of adding the currentState to the route at the end of the if statements in the while loop, so I moved it to the beginning of the if statements and the problem was fixed.