همچنین برنامه دیگری برای نمایش روند کار در console نوشتیم که یک نمونه ورودی و خروجی آن به شکل زیر است

$L = \{a^n b^n e\}$

input

```
Run - Test-Train
Run:  Main (1)
"C:\Program Files\Java\jdk-11.0.8\bin\java
      .223\lib\idea_rt.jar=58508:C:\Users\
       -classpath "D:\university\term 2\01
(q0, a) -> (q1, x, R)
(q0, b) -> (q3, b, R)
(q0, y) -> (q3, y, R)
(q1, a) -> (q1, a, R)
(q1, y) -> (q1, y, R)
(q1, b) -> (q2, y, L)
(q2, a) -> (q2, a, L)
(q2, y) -> (q2, y, L)
(q2, x) -> (q0, x, R)
(q3, y) -> (q3, y, R)
(q3, e) -> (q4, e, R)
end of rules
q4
end of accepts
end of rejects
a a a b b e
```

output

```
end of rejects
a a a b b e
(a) a  a  b  b  e
 x (a) a  b  b  e
 x  a (a) b  b  e
 x  a  a (b) b  e
 x  a (a) y  b  e
 x (a) a  y  b  e
(x) a  a  y  b  e
 x (a) a  y  b  e
 x  x (a) y  b  e
 x  x  a (y) b  e
 x  x  a  y (b) e
 x  x  a (y) y  e
 x  x (a) y  y  e
 x (x) a  y  y  e
 x  x (a) y  y  e
 x  x  x (y) y  e
 x  x  x  y (y) e
 x  x  x  y  y (e)
   ~~~ reject

Process finished with exit code 0
```

output of previous Turing Machine with new Strings

```
a a a b b b e
(a) a  a  b  b  b  e
 x (a) a  b  b  b  e
 x  a (a) b  b  b  e
 x  a  a (b) b  b  e
 x  a (a) y  b  b  e
 x (a) a  y  b  b  e
(x) a  a  y  b  b  e
 x (a) a  y  b  b  e
 x  x (a) y  b  b  e
 x  x  a (y) b  b  e
 x  x  a  y (b) b  e
 x  x  a (y) y  b  e
 x  x (a) y  y  b  e
 x (x) a  y  y  b  e
 x  x (a) y  y  b  e
 x  x  x (y) y  b  e
 x  x  x  y (y) b  e
 x  x  x  y  y (b) e
 x  x  x  y (y) y  e
 x  x  x (y) y  y  e
 x  x (x) y  y  y  e
 x  x  x (y) y  y  e
 x  x  x  y (y) y  e
 x  x  x  y  y (y) e
 x  x  x  y  y  y (e)
     ~~~ accept
```

```
a a a a b e
(a) a  a  a  b  e
 x (a) a  a  b  e
 x  a (a) a  b  e
 x  a  a (a) b  e
 x  a  a  a (b) e
 x  a  a (a) y  e
 x  a (a) a  y  e
 x (a) a  a  y  e
(x) a  a  a  y  e
 x (a) a  a  y  e
 x  x (a) a  y  e
 x  x  a (a) y  e
 x  x  a  a (y) e
 x  x  a  a  y (e)
     ~~~ reject

Process finished with exit code 0
```

```
a b b b b e
(a) b  b  b  b  e
 x (b) b  b  b  e
(x) y  b  b  b  e
 x (y) b  b  b  e
 x  y (b) b  b  e
     ~~~ reject

Process finished with exit code 0
```

یک مثال دیگر از ماشین تورینگ این است که ماشین تورینگ برای مثلا حاصلضرب ۲ عدد را در tape نمایش دهد که این کار هم با استفاده از این برنامه قابل انجام است

سپس ورودی را در این قالب به tape ماشین میدهیم که تعداد 1 نمایانگر مقدار a و تعداد 2 نمایانگر مقدار b است

```
# 1 1 # 2 2 2
```

و در آخر خروجی را به این شکل بر روی تیپ به کاربر نشان میدهیم که تعداد 0 ها نمایانگر مقدار a * b است و B در ابتدای تیپ یعنی Blank

```
 B  #  a  a  # (2) 2  2  #  0  0  0  0  0  0
     ~~~ accept

Process finished with exit code 0
```

ابتدا ماشین تورینگ را برای برنامه تعریف میکنیم

```
(q0, #) -> (q_i, #, R)
(q_i, 1) -> (q_i, 1, R)
(q_i, #) -> (q_i, #, R)
(q_i, 2) -> (q_i, 2, R)
(q_i, B) -> (q_j, #, L)

(q_j, #) -> (q_j, #, L)
(q_j, 1) -> (q_j, 1, L)
(q_j, 2) -> (q_j, 2, L)
(q_j, B) -> (q_0, B, R)

(q_0, #) -> (q_1, #, R)

(q_1, 1) -> (q_2, a, R)
(q_1, #) -> (q_accept, #, R)

(q_2, 1) -> (q_2, 1, R)
(q_2, #) -> (q_3, #, R)

(q_3, 2) -> (q_4, b, R)
(q_3, #) -> (q_8, #, L)

(q_8, b) -> (q_8, 2, L)
(q_8, #) -> (q_9, #, L)

(q_9, 1) -> (q_9, 1, L)
(q_9, a) -> (q_1, a, R)

(q_4, 2) -> (q_4, 2, R)
(q_4, #) -> (q_5, #, R)

(q_5, 0) -> (q_5, 0, R)
(q_5, B) -> (q_6, 0, L)

(q_6, 0) -> (q_6, 0, L)
(q_6, #) -> (q_7, #, L)

(q_7, 2) -> (q_7, 2, L)
(q_7, b) -> (q_3, b, R)

end of rules
q_accept
end of accepts
end of rejects
```

در ادامه نیز خروجی کامل آورده شده است

```
(#) 1  1  #  2  2  2
 # (1) 1  #  2  2  2
 #  1 (1) #  2  2  2
 #  1  1 (#) 2  2  2
 #  1  1  # (2) 2  2
 #  1  1  #  2 (2) 2
 #  1  1  #  2  2 (2)
 #  1  1  #  2  2  2 (B)
 #  1  1  #  2  2 (2) #
 #  1  1  #  2 (2) 2  #
 #  1  1  # (2) 2  2  #
 #  1  1 (#) 2  2  2  #
 #  1 (1) #  2  2  2  #
 # (1) 1  #  2  2  2  #
(#) 1  1  #  2  2  2  #
(B) #  1  1  #  2  2  2  #
 B (#) 1  1  #  2  2  2  #
 B  # (1) 1  #  2  2  2  #
 B  #  a (1) #  2  2  2  #
 B  #  a  1 (#) 2  2  2  #
 B  #  a  1  # (2) 2  2  #
 B  #  a  1  #  b (2) 2  #
 B  #  a  1  #  b  2 (2) #
 B  #  a  1  #  b  2  2 (#)
 B  #  a  1  #  b  2  2  # (B)
```

```
B  #  a  1  #  b  2  2  # (B)
B  #  a  1  #  b  2  2 (#) 0
B  #  a  1  #  b  2 (2) #  0
B  #  a  1  #  b (2) 2  #  0
B  #  a  1  # (b) 2  2  #  0
B  #  a  1  #  b (2) 2  #  0
B  #  a  1  #  b  b (2) #  0
B  #  a  1  #  b  b  2 (#) 0
B  #  a  1  #  b  b  2  # (0)
B  #  a  1  #  b  b  2  #  0 (B)
B  #  a  1  #  b  b  2  # (0) 0
B  #  a  1  #  b  b  2 (#) 0  0
B  #  a  1  #  b  b (2) #  0  0
B  #  a  1  #  b (b) 2  #  0  0
B  #  a  1  #  b  b (2) #  0  0
B  #  a  1  #  b  b  b (#) 0  0
B  #  a  1  #  b  b  b  # (0) 0
B  #  a  1  #  b  b  b  #  0 (0)
B  #  a  1  #  b  b  b  #  0  0 (B)
B  #  a  1  #  b  b  b  #  0 (0) 0
B  #  a  1  #  b  b  b  # (0) 0  0
B  #  a  1  #  b  b  b (#) 0  0  0
B  #  a  1  #  b  b (b) #  0  0  0
B  #  a  1  #  b  b  b (#) 0  0  0
B  #  a  1  #  b  b (b) #  0  0  0
B  #  a  1  #  b (b) 2  #  0  0  0
B  #  a  1  # (b) 2  2  #  0  0  0
B  #  a  1 (#) 2  2  2  #  0  0  0
```

```
B  #  a  1 (#) 2  2  2  #  0  0  0
B  #  a (1) #  2  2  2  #  0  0  0
B  # (a) 1  #  2  2  2  #  0  0  0
B  #  a (1) #  2  2  2  #  0  0  0
B  #  a  a (#) 2  2  2  #  0  0  0
B  #  a  a  # (2) 2  2  #  0  0  0
B  #  a  a  #  b (2) 2  #  0  0  0
B  #  a  a  #  b  2 (2) #  0  0  0
B  #  a  a  #  b  2  2 (#) 0  0  0
B  #  a  a  #  b  2  2  # (0) 0  0
B  #  a  a  #  b  2  2  #  0 (0) 0
B  #  a  a  #  b  2  2  #  0  0 (0)
B  #  a  a  #  b  2  2  #  0  0  0 (B)
B  #  a  a  #  b  2  2  #  0  0 (0) 0
B  #  a  a  #  b  2  2  #  0 (0) 0  0
B  #  a  a  #  b  2  2  # (0) 0  0  0
B  #  a  a  #  b  2  2 (#) 0  0  0  0
B  #  a  a  #  b  2 (2) #  0  0  0  0
B  #  a  a  #  b (2) 2  #  0  0  0  0
B  #  a  a  # (b) 2  2  #  0  0  0  0
B  #  a  a  #  b (2) 2  #  0  0  0  0
B  #  a  a  #  b  b (2) #  0  0  0  0
B  #  a  a  #  b  b  2 (#) 0  0  0  0
B  #  a  a  #  b  b  2  # (0) 0  0  0
B  #  a  a  #  b  b  2  #  0 (0) 0  0
B  #  a  a  #  b  b  2  #  0  0 (0) 0
B  #  a  a  #  b  b  2  #  0  0  0 (0)
B  #  a  a  #  b  b  2  #  0  0  0  0 (B)
```

```
B  #  a  a  #  b  b  2  #  0  0  0  0 (B)
B  #  a  a  #  b  b  2  #  0  0  0 (0) 0
B  #  a  a  #  b  b  2  #  0  0 (0) 0  0
B  #  a  a  #  b  b  2  #  0 (0) 0  0  0
B  #  a  a  #  b  b  2  # (0) 0  0  0  0
B  #  a  a  #  b  b  2 (#) 0  0  0  0  0
B  #  a  a  #  b  b (2) #  0  0  0  0  0
B  #  a  a  #  b (b) 2  #  0  0  0  0  0
B  #  a  a  #  b  b (2) #  0  0  0  0  0
B  #  a  a  #  b  b  b (#) 0  0  0  0  0
B  #  a  a  #  b  b  b  # (0) 0  0  0  0
B  #  a  a  #  b  b  b  #  0 (0) 0  0  0
B  #  a  a  #  b  b  b  #  0  0 (0) 0  0
B  #  a  a  #  b  b  b  #  0  0  0 (0) 0
B  #  a  a  #  b  b  b  #  0  0  0  0 (0)
B  #  a  a  #  b  b  b  #  0  0  0  0  0 (B)
B  #  a  a  #  b  b  b  #  0  0  0  0 (0) 0
B  #  a  a  #  b  b  b  #  0  0  0 (0) 0  0
B  #  a  a  #  b  b  b  #  0  0 (0) 0  0  0
B  #  a  a  #  b  b  b  #  0 (0) 0  0  0  0
B  #  a  a  #  b  b  b  # (0) 0  0  0  0  0
B  #  a  a  #  b  b  b (#) 0  0  0  0  0  0
B  #  a  a  #  b  b (b) #  0  0  0  0  0  0
B  #  a  a  #  b  b  b (#) 0  0  0  0  0  0
B  #  a  a  #  b  b (b) #  0  0  0  0  0  0
B  #  a  a  #  b (b) 2  #  0  0  0  0  0  0
B  #  a  a  # (b) 2  2  #  0  0  0  0  0  0
B  #  a  a (#) 2  2  2  #  0  0  0  0  0  0
```

```
B  #  a (a) #  2  2  2  #  0  0  0  0  0  0
B  #  a  a (#) 2  2  2  #  0  0  0  0  0  0
B  #  a  a  # (2) 2  2  #  0  0  0  0  0  0
    ~~~ accept

Process finished with exit code 0
```

توضیح نحوه حل سوال

در کلاس Main صرفا یک object از Director ساخته میشود و عملیات شروع میشود

```java
public class Main {

    public static void main( String[] args ) {
        Director director = new Director();
        director.start();
    }
}
```

در کلاس Director فیلد console را تعریف کردیم که صرفا عملیات وردی و خروجی انجام میدهد و منطق اصلی برنامه در همین کلاس Director میباشد که ماشین تورینگ را شبیه سازی میکند

```java
public class Director {
    Console console;

    HashSet<String> alphabet;
    ArrayList<Rule> rules;
    ArrayList<String> accepts;
    ArrayList<String> rejects;

    ArrayList<String> tape;

    String state;
    int index;
```

فیلد alphabet برای حروف الفبای tape و rules میباشد یعنی کارکتر هایی که میخواند و مینویسد البته درست است که ما اینجا از کلمه کارکتر استفاده کردیم ولی فرم ذخیره آن از نوی String است که محدودیتی در length کارکتری که میخونیم و مینویسیم نداشته باشیم

فیلد rules درواقع قوانین این ماشین تورینگ است که ما به عنوان ورودی میدهیم و Director در این فیلد ذخیره میکند که با استفاده از آن ماشین تورینگ را شبیه سازی کنیم

همچنین فیلد های accepts و rejects برای ذخیره state های مربوطه است که در شبیه سازی وقتی به آن state ها برسیم ماشین متوقف شود و نشان بدهد که در حالت accept است یا reject همچنین شاین ذکر است که اگر در شبیه سازی هیچوقت به حالت های accept و reject نرسیم یعنی هیچ به halt نرسیم، برنامه stack overflow میشود :)

```java
public class Director {
    Console console;

    HashSet<String> alphabet;
    ArrayList<Rule> rules;
    ArrayList<String> accepts;
    ArrayList<String> rejects;

    ArrayList<String> tape;

    String state;
    int index;
```

برای پیاده سازی tape از یک ArrayList یعنی از یک آرایه flexible استفاده شده که اگر به انتهای یا ابتدای tape رسیدیم یک B که نماینده Blank هست قرار گیرد

برای مثلا این حالتی از tape است که head بر روی index آخر tape قرار دارد و کارتر B یعنی Blank یعنی خالی میخواند

```
B  #  a  a  #  b  b  b  #  0  0  0  0  0 (B)
```

فیلد state هم نمایانگر حالت فعلی است که در ابتدا q0 است

و index هم برای مکان فعلی head بر روی tape است

```java
public void start() {
    resetMachine();
    console.getRules( director: this);
    console.getAccepts( director: this);
    console.getRejects( director: this);
    console.getTape( director: this);
    console.computeMaxLength(alphabet, tape);

    try {
        operate();
    } catch (AcceptException e) {
        console.printTape(tape, index);
        console.printInfo("accept");
    } catch (RejectException e) {
        console.printTape(tape, index);
        console.printInfo("reject");
    }
}
```

متد start شروع کار director میباشد که به شرح آن میپردازیم

ابتدا ماشین را ریسیت میکنیم

```java
private void resetMachine() {
    setState("q0");
    index = 0;
    tape.add("B"); // B stands for
}
```

سپس قوانین را از کاربر میگیریم بعد حالت های accept بعد reject بعد محتوایت tape را یا همان ورودی و string که قرار است وضعیت آن بررسی شود

متد computeMaxLength خیلی اهمیت ندارد و صرفا برای محاسبه طول بزرگترین کارکتر برای نمایش است

منطق اصلی کار از اینجا شروع میشود که ما متد operate را صدا میزنیم و درواقع ماشین توریتگ ما شروع به کار میکند و هرجا که Accept شد یک AcceptException پرتاب میکند که ما بفهمیم ماشین با حالت Accept رفته و دیگر operate نکند برای reject هم به همین طورت

همچنین متد printTape برای چاپ محتوات فعلی و مکان head میباشد

```java
private void operate() throws AcceptException, RejectException {
    console.printTape(tape, index);
    String read = getCurrentChar();
    for (Rule rule : rules) {
        if (ruleIsTrue(rule, read, state)) {
            executeRule(rule);
            checkAcceptReject();
            operate();
        }
    }
    throw new RejectException();
}
```

در متد operate درواقع کاری که ما میکنیم این است که ابتدا کارکتری که الان head آن را میخواند میخوانیم و بعد در تمام rule ها با توجه به state فعلی میگردیم تا ببینیم کدام rule اجرا بشود

```
(q_9, 1) -> (q_9, 1, L)
(q_9, a) -> (q_1, a, R)

(q_4, 2) -> (q_4, 2, R)
(q_4, #) -> (q_5, #, R)

(q_5, 0) -> (q_5, 0, R)
(q_5, B) -> (q_6, 0, L)
```

مثلا اگر قوانینما این ها باشند و read = a باشد و state = q_9 باشد قانون دوم اجرا میشود یعنی به state q_1 میرویم و a را روی write tape میکنیم و بعد به سمت Right میرویم
درواقع یعنی قانون را execute میکنیم بعد چک میکنیم که حالت جدید آیا حالت accept است اگر بود accept میشود اگر reject بود reject میشود یعنی throw reject exception میکنیم اگر هم نبود باز به شکل بازگشتی به operate کردن ادامه میدهیم تا بلاخره halt کنیم
اگر هم هیچکدام از rule ها با شرایط فعلی یعنی read و state همخوانی نداشت reject میکنیم

بقیه قسمت های کار هم به این صورت است

```
private String getCurrentChar() {
    return tape.get(index);
}
```

```
private boolean ruleIsTrue( Rule rule,
                            String read,
                            String currentState ) {
    return rule.origin.equals(currentState) && rule.read.equals(read);
}
```

```
private void executeRule( Rule rule ) {
    setState(rule.dest);
    updateStr(rule.write);
    updateIndex(rule.direction);
    checkEndOfStr();
    checkStartOfStr();
}
```

```
private void checkEndOfStr() {
    if (index == tape.size()) {
        tape.add("B");
    }
}

private void checkStartOfStr() {
    if (index == -1) {
        tape.add( index: 0,  element: "B");
        index = 0;
    }
}
```

```
private void setState( String dest ) {
    state = dest;
}

private void updateStr( String write ) {
    tape.set(index, write);
}

private void updateIndex( Direction direction ) {
    switch (direction) {
        case L:
            index--;
            break;
        case R:
            index++;
            break;
    }
}
```

```
private void checkAcceptReject() throws AcceptException, RejectException {
    if (accepts.contains(state)) {
        throw new AcceptException();
    }
    if (rejects.contains(state)) {
        throw new RejectException();
    }
}
```

فیلد های کلاس Rule نیز به این شکل است که نیاز به توضیح ندارد

همچنین برای خواندن و استخراج ورودی از regex استفاده شده که کار ساده تر گردد

```
(origin, read) -> (dest, write, direction(LR))
```

```java
public class Rule {
    public String origin;
    public String read;

    public String dest;
    public String write;
    public Direction direction;
```

```java
public Rule( String ruleStr ) {
    compile(ruleStr);
}

private void compile( String ruleStr ) {
    String patternAsString = "\\((.+), (.+)\\) -> \\((.+), (.+), ([RL])\\)";
    Pattern pattern = Pattern.compile(patternAsString);
    Matcher matcher = pattern.matcher(ruleStr);
    if (matcher.find()) {
        origin = matcher.group(1);
        read = matcher.group(2);

        dest = matcher.group(3);
        write = matcher.group(4);
        direction = Direction.valueOf(matcher.group(5));
    } else {
        throw new RuntimeException("invalid input rule pattern");
    }
}
```