```java
package afnaafd;
// @author LuisR
import java.util.ArrayList;
import java.util.HashMap;
import java.util.InputMismatchException;
import java.util.Iterator;
import java.util.LinkedHashSet;
import java.util.Scanner;
import java.util.Set;
import java.util.TreeSet;
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(source: System.in);
        int cantQ = 0;
        int cantA = 0;
        int valQ0 = -1;
        String q0 = null;
        String q = null;
        AFNaAFD conv;
        while (cantQ < 1 || cantQ > 10) {
            try {
                System.out.print(s: "Ingrese la cantidad del conjunto finito de estados (1-10): ");
                cantQ = sc.nextInt();
                if (cantQ < 1 || cantQ > 10) {
                    System.out.println(x: "Solo valores de 1 a 10.\n");
                }
            } catch (InputMismatchException e) {
                System.out.println(x: "Debe ingresar un número entero válido.\n");
                sc.next();
            }
        }

        while (cantA < 1 || cantA > 3) {
            try {
                System.out.print(s: "Ingrese la cantidad de simbolos del alfabeto de entrada (1-3): ");
                cantA = sc.nextInt();
                if (cantA < 1 || cantA > 3) {
                    System.out.println(x: "Solo valores de 1 a 3.\n");
                }
            } catch (InputMismatchException e) {
                System.out.println(x: "Debe ingresar un número entero válido.\n");
                sc.next();
            }
        }

        conv = new AFNaAFD(cantQ, cantA);

        System.out.println(x: conv.getQ());
        System.out.println(x: conv.getE());
```

```
50
51          char qLetra = 0;
52
53          while (qLetra != 'q' || valQ0 < 0 || valQ0 >= cantQ) {
54              try {
55                  System.out.print(s: "Ingrese el estado inicial: ");
56                  q0 = sc.next();
57                  qLetra = q0.charAt(index: 0);
58                  valQ0 = Integer.parseInt(s: q0.substring(beginIndex: 1));
59                  if (qLetra != 'q' && (valQ0 < 0 || valQ0 >= cantQ)) {
60                      System.out.println(x: "Redaccion incorrecta.\n");
61                  } else if (qLetra != 'q') {
62                      System.out.println(x: """
63                                            El estado inicial debe iniciar con la letra q,
64                                            y esta, posteriormente, tener un valor valido
65                                            dentro del conjunto de estados.\n""");
66                  } else if (valQ0 < 0 || valQ0 >= cantQ) {
67                      System.out.println("El conjunto de estados abarca de q0 a q" + (cantQ - 1));
68                      System.out.println();
69                  }
70              } catch (InputMismatchException | NumberFormatException e) {
71                  System.out.println(x: "Debe ingresar un estado válido dentro del conjunto finito de estados.\n");
72              }
73          }
74
75          conv.setEstInic(q0);
76
77          System.out.println("q\u2080 = " + conv.getEstInic());
78
79          boolean band = true;
80          Set<String> conjTemp = new TreeSet<>();
81
82          while (band) {
83              try {
84                  qLetra = 0;
85                  valQ0 = -1;
86                  boolean val = false;
87                  while (qLetra != 'q' || valQ0 < 0 || valQ0 >= cantQ || val) {
88                      try {
89                          System.out.print(s: "Ingrese el estado para el conjunto de estados finales: ");
90                          q = sc.next();
91                          qLetra = q.charAt(index: 0);
92                          valQ0 = Integer.parseInt(s: q.substring(beginIndex: 1));
93                          val = conjTemp.contains(o: q);
94                          if (qLetra != 'q' && (valQ0 < 0 || valQ0 >= cantQ)) {
95                              System.out.println(x: "Redaccion incorrecta.\n");
96                          } else if (qLetra != 'q') {
97                              System.out.println(x: """
98                                                    El estado debe iniciar con la letra q,
```

```
                                    y esta, posteriormente, tener un valor valido
                                    dentro del conjunto de estados.\n""");
                        } else if (valQ0 < 0 || valQ0 >= cantQ) {
                            System.out.println("El conjunto de estados abarca de q0 a q" + (cantQ - 1));
                            System.out.println();
                        } else if (val) {
                            System.out.println(x: "El estado final que acaba de ingresar ya fue ingresado con anterioridad.\n");
                        }
                    } catch (InputMismatchException | NumberFormatException e) {
                        System.out.println(x: "Debe ingresar un estado válido dentro del conjunto finito de estados.\n");
                    }
                }
                conjTemp.add(e: q);
                if (conjTemp.size() == cantQ) {
                    band = false;
                } else {
                    System.out.println(x: "Desea seguir ingresando mas estados al mismo conjunto? (true/false)");
                    band = sc.nextBoolean();
                }
            } catch (InputMismatchException e) {
                System.out.println(x: """
                                    Solamente se permite ingresar true o false.
                                    Si se desea seguir ingresando mas estados, debe ingresar true.
                                    Si no se desea seguir ingresando mas estados, debe ingresar false.\n""");
                band = true;
            }
        }

        conv.setConjEstFin(estFin: conjTemp);
        System.out.println(x: conv.getF());

        for (int i = 0; i < cantQ; i++) {
            for (int j = 0; j < cantA; j++) {
                System.out.println("δ(q" + i + ", " + j + ")");
                band = true;
                conjTemp = new TreeSet<>();
                while (band) {
                    try {
                        if (conjTemp.isEmpty()) {
                            System.out.println(x: "Desea ingresar un conjunto vacio? (true/false)");
                            band = sc.nextBoolean();
                        } else {
                            band = false;
                        }
                        if (band) {
                            q = "";
                        } else {
                            qLetra = 0;
```

```
                        -
                        valQ0 = -1;
                        boolean val = false;
                        while (qLetra != 'q' || valQ0 < 0 || valQ0 >= cantQ || val) {
                            try {
                                System.out.print("Ingrese el estado para la funcion de transicion de q" + i + " con " + j + ": ");
                                q = sc.next();
                                qLetra = q.charAt(index: 0);
                                valQ0 = Integer.parseInt(s: q.substring(beginIndex: 1));
                                val = conjTemp.contains(o: q);
                                if (qLetra != 'q' && (valQ0 < 0 || valQ0 >= cantQ)) {
                                    System.out.println(x: "Redaccion incorrecta.\n");
                                } else if (qLetra != 'q') {
                                    System.out.println(x: """
                                    El estado debe iniciar con la letra q,
                                    y esta, posteriormente, tener un valor valido
                                    dentro del conjunto de estados.\n""");
                                } else if (valQ0 < 0 || valQ0 >= cantQ) {
                                    System.out.println("El conjunto de estados abarca de q0 a q" + (cantQ - 1));
                                    System.out.println();
                                } else if (val) {
                                    System.out.println(x: "El estado final ingresado ya fue ingresado con anterioridad.\n");
                                }
                            } catch (InputMismatchException | NumberFormatException e) {
                                System.out.println(x: "Debe ingresar un estado válido dentro del conjunto finito de estados.\n");
                            }
                        }
                    }
                    conjTemp.add(e: q);
                    if (q.equals(anObject: "") || conjTemp.size() == cantQ) {
                        band = false;
                    } else {
                        System.out.println(x: "Desea seguir ingresando mas estados al mismo conjunto? (true/false)");
                        band = sc.nextBoolean();
                    }
                } catch (InputMismatchException e) {
                    System.out.println(x: """
                                Solamente se permite ingresar true o false.
                                Si se desea ingresar un conjunto vacio, debe ingresar true.
                                Si no se desea ingresar un conjunto vacio, debe ingresar false.\n""");
                    band = true;
                    sc.next();
                }
            }
            conv.setD(conjuntoD: conjTemp);
        }
    }

    System.out.println(x: conv.getD());
```

```
196             Set<String>[][] listaB = conv.getFunTransicion();
197
198             Set<String> t = new TreeSet<>();
199             t.add(e: conv.getEstInic());
200
201             Set<String> delta = new LinkedHashSet<>();
202
203             ArrayList<String>[] conjDelta = new ArrayList[listaB[0].length];
204             for (int i = 0; i < conjDelta.length; i++) {
205                 conjDelta[i] = new ArrayList<>();
206             }
207
208             Set<Set<String>> d = new LinkedHashSet<>();
209             d.add(e: t);
210             delta.add(e: t.toString());
211
212             Iterator<Set<String>> iterador = d.iterator();
213
214             Set<String> c = new LinkedHashSet<>();
215             int n1 = 0;
216             int n2 = 0;
217             int n3 = d.size();
218
219             ArrayList<Set<String>>[] conjD = new ArrayList[listaB[0].length];
220             for (int i = 0; i < conjD.length; i++) {
221                 conjD[i] = new ArrayList<>();
222             }
223
224             while (n2 < n3) {
225                 for (int i = 0; i <= n1; i++) {
226                     c = iterador.next();
227                 }
228                 n1++;
229                 if (c.size() > 1) {
230                     for (int i = 0; i < conjD.length; i++) {
231                         Set<String> temp = new TreeSet<>();
232                         boolean bandl = true;
233                         for (String s : c) {
234                             if (s.equals(anObject: "{ }") || s.equals(anObject: "")) {
235                                 t = conv.getVacio();
236                             } else {
237                                 t = listaB[Integer.parseInt(s: s.substring(beginIndex: 1))][i];
238                             }
239                             temp = t;
240                             bandl = bandl && t.toString().equals(anObject: "[{ }]");
241                         }
242                         if (!bandl) {
```

```
                        temp = new TreeSet<>();
                        Object[] arrayT = c.toArray();
                        for (Object o : arrayT) {
                            String s = (String) o;
                            if (s.equals(anObject: "{ }") || s.equals(anObject: "")) {
                                t = conv.getVacio();
                            } else {
                                t = listaB[Integer.parseInt(s: s.substring(beginIndex: 1))][i];
                            }
                            if (!(t.toString().equals(anObject: "[{ }]") || t.toString().equals(anObject: "[]"))) {
                                for (String sl : t) {
                                    temp.add(e: sl);
                                }
                            }
                        }
                    }
                    conjD[i].add(e: temp);
                    conjDelta[i].add(e: temp.toString());
                    d.add(e: temp);
                    delta.add(e: temp.toString());
                }
            } else {
                for (String s : c) {
                    for (int i = 0; i < conjD.length; i++) {
                        if (s.equals(anObject: "{ }") || s.equals(anObject: "")) {
                            t = conv.getVacio();
                        } else {
                            t = listaB[Integer.parseInt(s: s.substring(beginIndex: 1))][i];
                        }
                        conjD[i].add(e: t);
                        conjDelta[i].add(e: t.toString());
                        d.add(e: t);
                        delta.add(e: t.toString());
                    }
                }
            }
            iterador = d.iterator();
            n2++;
            n3 = d.size();
        }

        Object[] array = delta.toArray();
        String[][] afdC = new String[array.length][conjDelta.length + 1];

        for (int i = 0; i < array.length; i++) {
            afdC[i][0] = (String) array[i];
        }
        for (int i = 0; i < conjDelta.length; i++) {
```

```
                for (int j = 0; j < array.length; j++) {
                    afdC[j][i + 1] = conjDelta[i].get(index: j);
                }
            }
            for (int i = 0; i < afdC.length; i++) {
                for (int j = 0; j < afdC[0].length; j++) {
                    System.out.printf(format: "%2ls", afdC[i][j]);
                }
                System.out.println();
            }
            System.out.println();
            HashMap<String, String> renombre = new HashMap<>();
            for (int i = 0; i < afdC.length; i++) {
                renombre.put(afdC[i][0], "r" + i);
            }
            String[][] afd = new String[afdC.length][afdC[0].length];
            for (int i = 0; i < afd.length; i++) {
                for (int j = 0; j < afd[0].length; j++) {
                    afd[i][j] = renombre.get(afdC[i][j]);
                }
            }
            for (int i = 0; i < afd.length; i++) {
                for (int j = 0; j < afd[0].length; j++) {
                    System.out.printf(format: "%5s", afd[i][j]);
                }
                System.out.println();
            }
        }
    }

```

```java
package afnaafd;
// @author LuisR
import java.util.Set;
import java.util.TreeSet;
public class AFNaAFD {
    private final String[] conjEst;
    private final String[] alfabeto;
    private String q0;
    private String[] conjEstFin;
    private final Set<String>[][] funTransicion;
    private int fila;
    private int columna;
    private final Set<String> vacio = new TreeSet<>();
    public Set<String> getVacio() {
        return vacio;
    }
    public AFNaAFD(){
        vacio.add(e: "{ }");
        conjEst = new String[3];
        alfabeto = new String[2];
        funTransicion = new TreeSet[3][2];
        conjEstFin = new String[2];
        conjEstFin[0] = "q1";
        conjEstFin[1] = "q3";
        fila = columna = 0;
        llenar();
        this.q0 = "q0";
        funTransicion[0][0] = new TreeSet<>();
        funTransicion[0][0].add(e: "q2");
        funTransicion[0][1] = new TreeSet<>();
        funTransicion[0][1].add(e: "q1");
        funTransicion[1][0] = new TreeSet<>();
        funTransicion[1][0].add(e: "q0");
        funTransicion[1][1] = new TreeSet<>();
        funTransicion[1][1].add(e: "q2");
        funTransicion[2][0] = new TreeSet<>();
        funTransicion[2][0].add(e: "q1");
        funTransicion[2][0].add(e: "q2");
        funTransicion[2][1] = new TreeSet<>();
        funTransicion[2][1].add(e: "q0");
        funTransicion[2][1].add(e: "q2");
    }
    public AFNaAFD(int cantQ, int cantA){
        vacio.add(e: "");
        conjEst = new String[cantQ];
        alfabeto = new String[cantA];
        funTransicion = new TreeSet[cantQ][cantA];
        fila = columna = 0;
        llenar();
```

```
50      }
51      private void llenar(){
52          for (int i = 0; i < conjEst.length; i++) {
53              conjEst[i] = "q"+i;
54          }
55          for (int i = 0; i < alfabeto.length; i++) {
56              alfabeto[i] = i+"";
57          }
58      }
59      private String getConjunto(String simbolo, String[] conjunto){
60          String s = simbolo + " = {";
61          for (int i = 0; i < conjunto.length; i++) {
62              s += conjunto[i];
63              if (i != conjunto.length-1) {
64                  s += ", ";
65              } else {
66                  s += "}";
67              }
68          }
69          return s;
70      }
71      public String getQ(){
72          return getConjunto(simbolo:"Q", conjunto: conjEst);
73      }
74      public String getE(){
75          return getConjunto(simbolo:"Σ", conjunto: alfabeto);
76      }
77      public void setEstInic(String q0){
78          this.q0 = q0;
79      }
80      public String getEstInic(){
81          return q0;
82      }
83      public void setConjEstFin(Set<String> estFin){
84          Object[] temp = estFin.toArray();
85          conjEstFin = new String[estFin.size()];
86          for (int i = 0; i < estFin.size(); i++) {
87              conjEstFin[i] = (String)temp[i];
88          }
89      }
90      public String getF(){
91          return getConjunto(simbolo:"F", conjunto: conjEstFin);
92      }
93      public void setD(Set<String> conjuntoD){
94          funTransicion[fila][columna] = conjuntoD;
95          columna++;
96          valIncremento();
97      }
98      private void valIncremento(){
```

```
98      private void valIncremento(){
99          if (columna == alfabeto.length) {
100             fila++;
101             columna = 0;
102         }
103     }
104     public Set<String>[][] getFunTransicion(){
105         return funTransicion;
106     }
107     public String getD(){
108         String matriz = "";
109         for (int i = 0; i < funTransicion.length; i++) {
110             for (int j = 0; j < funTransicion[0].length; j++) {
111                 Set temp = funTransicion[i][j];
112                 if (!temp.contains(o: "")) {
113                     String s = temp.toString();
114                     String sl = "";
115                     for (int k = 1; k < s.length()-1; k++) {
116                         sl += s.charAt(index: k);
117                     }
118                     sl = "{"+sl+"}";
119                     matriz += sl;
120                 } else {
121                     matriz += "{ }";
122                 }
123             }
124             matriz += "\n";
125         }
126         return matriz;
127     }
128 }
129
```