Classe Main:

```
package Minimizacao;

import Classes.*;
import java.util.*;

public class Main {

	// Variaveis declaradas aqui, inicialmente criamos um estado de erro sem informacao.

	public static Estado EstadoDeErro = new Estado("", 0, false, true);
	public static int QtdAlfabeto;
	public static int QtdEstadosFinais;
	public static char[] Alfabeto;
	public static int EstadoInicial;
	public static Estado[][] TabelaTransicao;
	public static String[][] TabelaTriangular;
	public static Scanner sc = new Scanner(System.in);
	public static ArrayList<Estado> ListaEstados = new ArrayList<Estado>();
	public static int QuantidadeEstados;

	public static void main(String[] args) {
		Iniciar();
	}

	public static void Iniciar() {
		Inicio();
```

```
		// Entrada de informacoes para a criacao de tabelas e armazenamento de
dados.

		System.out.println("Digite a quantidade de estados: ");
		QuantidadeEstados = sc.nextInt();

		// Metodo que cria estados e armazena em uma lista.
		CriarEstados(QuantidadeEstados);

		System.out.println("Digite o Estado Inicial: ");
		EstadoInicial = sc.nextInt();

		while (EstadoInicial > QuantidadeEstados) {
			System.out.println("Estado Inicial invalido digite novamente: ");
			EstadoInicial = sc.nextInt();
		}

		System.out.println("Digite a quantidade de letras do alfabeto: ");
		QtdAlfabeto = sc.nextInt();

		Alfabeto = new char[QtdAlfabeto];
		TabelaTransicao = new Estado[999][999];
		TabelaTriangular = new String[999][999];

		// Continua armazenar informacoes em arrays.

		for (int i = 0; i < QtdAlfabeto; i++) {
			System.out.println("Digite a " + i + " letra do alfabeto: ");
			Alfabeto[i] = sc.next().charAt(0);
		}
```

```
        System.out.println("Digite a quantidade de estados finais: ");
        QtdEstadosFinais = sc.nextInt();

        while (QtdEstadosFinais > QuantidadeEstados) {
                System.out.println("Estado final invalido digite novamente: ");
                QtdEstadosFinais = sc.nextInt();
        }

        for (int i = 0; i < QtdEstadosFinais; i++) {
                System.out.println("Digite o " + i + " estado final: ");
                ListaEstados.get(sc.nextInt()).setFinal(true);
        }

        // Metodos explicado mais abaixo.

        ImprimirTabela();
        PreencherTabela();
        VerificarPreRequisitosAlcancavel(ListaEstados.get(EstadoInicial));
        if (VerificarPreRequisitosAlcancavel() == false) {
                VerificarPreRequisitosTransicaoTotal();
                ImprimirTabelaPreenchida();
                ImprimirTabelaTriangular();
                TrivialmenteNaoEquivalentes();
                AnalisarParesNaoMarcados();
                JuntarEstadosNaoMarcados();
                TabelaFinal();
                ImprimirDescricaoFormal();
        }
        Final();
}
```

```
// Esse metodo cria estados e armazena em uma lista com padroes "virgem" e seta o estado inicial, nao e possivel escolher os numeros do estados.
public static void CriarEstados(int QtdEstados) {
	for (int i = 0; i < QtdEstados; i++) {
		ListaEstados.add(new Estado("S" + i, i, false, true));
	}
	ListaEstados.get(EstadoInicial).setInicial(true);
}


// A tabela guia e impressa usando as informacoes armazenadas acima.
public static void ImprimirTabela() {
	System.out.println("************************");
	System.out.println("*   Tabela guia!    *");
	System.out.println("************************");
	for (int i = -1; i < ListaEstados.size(); i++) {
		if (i != -1) {
			System.out.print(ListaEstados.get(i).getNome());
		}
		for (int j = 0; j < QtdAlfabeto; j++) {
			if (i == -1) {
				System.out.print("\t" + Alfabeto[j]);
			} else {
				System.out.print("\t" + i + "," + j);
			}
		}
		System.out.println(" ");
	}
}
```

```
	// Nesse metodo preenchemos uma matriz com as informacoes das transicoes.
	public static void PreencherTabela() {
		for (int i = 0; i < ListaEstados.size(); i++) {
			for (int j = 0; j < QtdAlfabeto; j++) {
				System.out.println("Digite o estado para " + i + "," + j + "
:");
				try {
					TabelaTransicao[i][j] =
ListaEstados.get(sc.nextInt());
				} catch (Exception e) {
					TabelaTransicao[i][j] = null;
				}
			}
		}
	}

	// Criamos um estado de erro caso tenha alguma transicao nula na tabela
preenchida e setamos todas as transicoes nulas para esse estado.
	public static void VerificarPreRequisitosTransicaoTotal() {
		boolean FlagEstadoDeErro = false;
		for (int i = 0; i < ListaEstados.size(); i++) {
			for (int j = 0; j < QtdAlfabeto; j++) {
				if (TabelaTransicao[i][j] == null) {
					if (FlagEstadoDeErro == false) {
						CriarEstadoDeErro();
					}
					TabelaTransicao[i][j] = EstadoDeErro;
					FlagEstadoDeErro = true;
				}
			}
		}
	}
```

```
        // Criamos um estado e colocamos na lista, sempre vai receber o numero de
estados.
        public static void CriarEstadoDeErro() {
                System.out.println("**************************");
                System.out.println("* Criando Estado de Erro! *");
                System.out.println("**************************");
                EstadoDeErro.setNome("S" + ListaEstados.size());
                EstadoDeErro.setNumero(ListaEstados.size());
                ListaEstados.add(EstadoDeErro);
                for (int i = 0; i < 0; i++) {
                        TabelaTransicao[ListaEstados.size()][i] = EstadoDeErro;
                }
        }

        // Imprime a tabela preenchida com os dados armazenadas em uma matriz.
        public static void ImprimirTabelaPreenchida() {
                System.out.println("************************");
                System.out.println("*  Tabela Preenchida! *");
                System.out.println("************************");
                for (int i = -1; i < ListaEstados.size(); i++) {
                        if (i != -1) {
                                System.out.print(ListaEstados.get(i).getNome());
                        }
                        for (int j = 0; j < QtdAlfabeto; j++) {
                                if (i == -1) {
                                        System.out.print("\t" + Alfabeto[j]);
                                } else {
                                        System.out.print("\t" +
TabelaTransicao[i][j].getNome());
                                }
```

```
                }
                System.out.println(" ");
            }
        }

        // Utilizada a busca por profundidade para verificar se os estados sao
alcancaveis.
        public static void VerificarPreRequisitosAlcancavel(Estado EstadoInicial) {
            EstadoInicial.setAcessado(true);
            for (int i = 0; i < QtdAlfabeto; i++) {
                if (TabelaTransicao[EstadoInicial.getNumero()][i] != null
                                    &&
TabelaTransicao[EstadoInicial.getNumero()][i].isAcessado() == false) {

VerificarPreRequisitosAlcancavel(TabelaTransicao[EstadoInicial.getNumero()][
i]);
                }
            }
        }

        // Verifica se todos os estados estao com a flag visitado ativa, caso nao esteja ele
para a execucao.
        public static boolean VerificarPreRequisitosAlcancavel() {

        System.out.println("**********************************************
**********");
            System.out.println("*          Verificacao alcancavel!          *");

        System.out.println("**********************************************
**********");
            for (int i = 0; i < ListaEstados.size(); i++) {
                System.out.println(ListaEstados.get(i).getNome() + ":" +
ListaEstados.get(i).isAcessado());
                if (ListaEstados.get(i).isAcessado() == false) {
```

```
        System.out.println("**********************************************************
**********");
                                System.out.println("* Existe estado inalcancavel! Nao eh
possivel Minimizar! *");

        System.out.println("**********************************************************
**********");
                                return true;
                        }
                }
                return false;
        }

        // Impressao da tabela triangular atraves de matriz.
        public static void ImprimirTabelaTriangular() {
                System.out.println("************************");
                System.out.println("*  Tabela Triangular! *");
                System.out.println("************************");
                for (int i = 1; i <= ListaEstados.size(); i++) {
                        for (int j = 0; j < ListaEstados.size() && j < (i + 1); j++) {
                                if (j == 0 && i != ListaEstados.size()) {
                                        System.out.print(ListaEstados.get(i).getNome());
                                } else if (i == ListaEstados.size()) {
                                        if (j != ListaEstados.size() - 1) {
                                                System.out.print("\t" +
ListaEstados.get(j).getNome());
                                        }
                                } else {
                                        System.out.print("\t" + i + "," + (j - 1));
                                }

                        }
```

```
                System.out.println(" ");
            }
        }

        // Percorre a matriz triangular verificando se os campos sao finais ou nao para fazer a juncao.
        public static void TrivialmenteNaoEquivalentes() {
            for (int i = 1; i < ListaEstados.size(); i++) {
                for (int j = 0; j < (ListaEstados.size() - 1) && j < i; j++) {
                    if (ListaEstados.get(i).isFinal() != ListaEstados.get(j).isFinal()) {
                        TabelaTriangular[i][j] = "X";
                    }
                }
            }

            System.out.println("****************************************");
            System.out.println("* Pares Trivialmente nao equivalentes *");
            System.out.println("****************************************");
            ImprimirTabelaTriangularComX();
        }

        // Impressao da tabela triangular com os dados armazenados em uma matriz.
        public static void ImprimirTabelaTriangularComX() {
            for (int i = 1; i <= ListaEstados.size(); i++) {
                for (int j = 0; j < ListaEstados.size() && j < (i + 1); j++) {
                    if (j == 0 && i != ListaEstados.size()) {
                        System.out.print(ListaEstados.get(i).getNome());
                    } else if (i == ListaEstados.size()) {
                        if (j != ListaEstados.size() - 1) {
                            System.out.print("\t" + ListaEstados.get(j).getNome());
```

```
                }
            } else {
                System.out.print("\t" + TabelaTriangular[i][j - 1]);
            }

        }
        System.out.println(" ");
    }
}

public static void AnalisarParesNaoMarcados() {
    boolean FlagTotal = true;
    // Percorre toda a matriz triangular ate que nao ocorra mais trocas.
    while (FlagTotal) {
        FlagTotal = false;
        // Percorre a tabela triangular indice por indice procurando um
campo nulo.
        for (int i = 1; i < ListaEstados.size(); i++) {
            for (int j = 0; j < (ListaEstados.size() - 1) && j < i; j++) {
                if (TabelaTriangular[i][j] == null) {
                    for (int k = 0; k < QtdAlfabeto; k++) {
                        // Verifica na tabela de transicao os
valores a ser verificadas
                        // se forem iguais ele nao pode ser
marcado um X caso contrario entra no if.
                        if
(TabelaTransicao[i][k].getNumero() != TabelaTransicao[j][k].getNumero()) {
                            try {
                                // Caso ele encontre
um X na tabela triangular na posicao em que procura ele anula o campo.
                                if
(TabelaTriangular[TabelaTransicao[i][k].getNumero()][TabelaTransicao[j][k]
```

```
.getNumero()] == "X") {

TabelaTriangular[i][j] = "X";

                                                                FlagTotal =
true;

System.out.println("Anulando os campos: " + i + " " + j);

                                                                    }
                                                            } catch (Exception e) {
                                                                // Para que ele
procure tanto na posicao 2,1 ou posicao 1,2 da matriz foi usado o try/catch mas a ideia
se mantem.
                                                                if
(TabelaTriangular[TabelaTransicao[j][k].getNumero()][TabelaTransicao[i][k]

.getNumero()] == "X") {

TabelaTriangular[i][j] = "X";

                                                                FlagTotal =
true;

System.out.println("Anulando os campos: " + i + " " + j);
                                                                    }
                                                                }
                                                            }
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                                System.out.println("************************");
                                System.out.println("*  Pares nao marcados *");
                                System.out.println("************************");
```

```
            ImprimirTabelaTriangularComX();
        }


        public static void JuntarEstadosNaoMarcados() {
            System.out.println("************************");
            System.out.println("*  Juntando estados   *");
            System.out.println("************************");

            // Variaveis auxiliares para a criacao de novos estados.

            int TamanhoListaAntiga = ListaEstados.size();
            int AlturaEstados = ListaEstados.size();
            for (int i = 0; i < TamanhoListaAntiga; i++) {
                for (int j = 0; j < (ListaEstados.size() - 1) && j < i; j++) {
                    // Percorre toda a matriz triangular procurando um campo nulo, nesse momento sao campos que ja podem ser juntados.
                    if (TabelaTriangular[i][j] == null) {
                        // Se esses estados nao possuirem novos estados, ou seja estados que ainda nao foram juntados ele entra nesse if,
                        // nesse if ele cria um novo estados com as mesmas propriedades dos estados anteriores (se eram finais vao continuar finais)
                        // e assim por diante, nao esquecendo de alterar as flags de controle como se o estado e ativo ou nao (se ele ja nao foi juntado
                        // com outro) e seus novos estados no qual trabalha como um ponteiro para esse novo estado, importante lembrar de aumentar
                        // a quantidade de estados com alturaEstados++, esse novo estado recebe a mesma transicao dos estados anteriores.
                        if (ListaEstados.get(i).getNovoEstado() == null && ListaEstados.get(j).getNovoEstado() == null) {
                            System.out.println("Criando um novo estado!");
                            if (ListaEstados.get(i).isInicial()) {
```

```
                                                ListaEstados.add(new Estado("S" +
ListaEstados.size(), ListaEstados.size(), false, true));
                                                ListaEstados.get(ListaEstados.size()
- 1).setInicial(true);
                                        } else if (ListaEstados.get(i).isFinal()) {
                                                ListaEstados.add(new Estado("S" +
ListaEstados.size(), ListaEstados.size(), true, true));
                                        } else {
                                                ListaEstados.add(new Estado("S" +
ListaEstados.size(), ListaEstados.size(), false, true));
                                        }
                                        ListaEstados.get(i).setAtivo(false);
                                        ListaEstados.get(j).setAtivo(false);

        ListaEstados.get(i).setNovoEstado(ListaEstados.get(ListaEstados.size() - 1));

        ListaEstados.get(j).setNovoEstado(ListaEstados.get(ListaEstados.size() - 1));
                                        for (int a = 0; a < QtdAlfabeto; a++) {
                                                TabelaTransicao[AlturaEstados][a]
= TabelaTransicao[i][a];
                                        }
                                        AlturaEstados++;
                                } else {
                                        // Caso o estado ja foi juntado com outro ele
pega a referencia do novo estado do estado antigo
                                        // e seta como um novo estado do estado
atual sempre inativando os estados antigos.
                                        if (ListaEstados.get(i).getNovoEstado() !=
null) {
                                                ListaEstados.get(j).setAtivo(false);

        ListaEstados.get(j).setNovoEstado(ListaEstados.get(i).getNovoEstado());
                                        } else {
                                                ListaEstados.get(i).setAtivo(false);
```

```
                ListaEstados.get(i).setNovoEstado(ListaEstados.get(j).getNovoEstado());
                                        }
                                }
                        }
                }
        }

        // Troca os estados inativos para o atributo novo estado do estado antigo.
        public static void TabelaFinal() {
                for (int i = 0; i < ListaEstados.size(); i++) {
                        for (int j = 0; j < QtdAlfabeto; j++) {
                                try {
                                        if (!TabelaTransicao[i][j].isAtivo()) {
                                                TabelaTransicao[i][j] =
TabelaTransicao[i][j].getNovoEstado();
                                        }
                                } catch (Exception e) {

                                }
                        }
                }
        }

        // Impressoes de toda a descricao formal ja alteradas.
        public static void ImprimirDescricaoFormal() {
                System.out.println("************************");
                System.out.println("*  Tabela minimizada! *");
                System.out.println("************************");
                for (int i = -1; i < ListaEstados.size(); i++) {
```

```
                if (i != -1 && ListaEstados.get(i).isAtivo()) {
                        System.out.print(ListaEstados.get(i).getNome());
                }
                if (i == -1 || ListaEstados.get(i).isAtivo()) {
                        for (int j = 0; j < QtdAlfabeto; j++) {
                                if (i == -1) {
                                        System.out.print("\t" + Alfabeto[j]);
                                } else {
                                        System.out.print("\t" +
TabelaTransicao[i][j].getNome());
                                }
                        }
                        System.out.println(" ");
                }
        }

        System.out.println("************************");
        System.out.println("*  Lista de estados!  *");
        System.out.println("************************");
        for (Estado e : ListaEstados) {
                if (e.isAtivo()) {
                        System.out.println(e.getNome());
                }
        }
        System.out.println("*****************************");
        System.out.println("* Lista de estados finais! *");
        System.out.println("*****************************");
        for (Estado e : ListaEstados) {
                if (e.isAtivo() && e.isFinal()) {
                        System.out.println(e.getNome());
                }
```

```
                }
                System.out.println("**********************");
                System.out.println("*    Estado Inicial!  *");
                System.out.println("**********************");
                System.out.println(ListaEstados.get(EstadoInicial).getNome());
                System.out.println("**********************");
                System.out.println("*      Alfabeto!      *");
                System.out.println("**********************");
                for (char a : Alfabeto) {
                        System.out.println(a);
                }

        }

        // Informacao sobre o trabalho.
        public static void Inicio() {

        System.out.println("**************************************************
****");
                System.out.println("* Minimizacao de automatos finitos deterministicos
*");
                System.out.println("*       Autor: Fernando Goncalves Hansen         *");
                System.out.println("*       E-mail: fernandohansen@outlook.com
*");

        System.out.println("**************************************************
****");
        }

        // Parte Final.
        public static void Final() {

        System.out.println("******************************************");
```

```
            while (!sc.nextLine().equals("X")) {
                System.out.println("Digite X para sair...");
            }
            System.out.println("Obrigado!");

        System.out.println("*********************************************");
        }

}
```

Classe Estado:

```
package Classes;

public class Estado {
	private String Nome;
	private int Numero;
	private boolean Acessado;
	private boolean Final;
	private boolean Inicial;
	private boolean Ativo;
	private Estado NovoEstado;

	public boolean isAtivo() {
		return Ativo;
	}

	public void setAtivo(boolean ativo) {
		Ativo = ativo;
	}

	public Estado getNovoEstado() {
		return NovoEstado;
	}

	public void setNovoEstado(Estado novoEstado) {
		NovoEstado = novoEstado;
	}

	public boolean isInicial() {
		return Inicial;
	}

	public void setInicial(boolean inicial) {
		Inicial = inicial;
	}

	public boolean isFinal() {
		return Final;
	}

	public void setFinal(boolean final1) {
		Final = final1;
	}

	public int getNumero() {
		return Numero;
	}
```

```
	public boolean isAcessado() {
		return Acessado;
	}

	public void setAcessado(boolean acessado) {
		Acessado = acessado;
	}

	public void setNumero(int numero) {
		Numero = numero;
	}

	public Estado(String Nome, int Numero, boolean Final, boolean Ativo) {
		this.Nome = Nome;
		this.Numero = Numero;
		this.Final = Final;
		this.Ativo = Ativo;
	}

	public String getNome() {
		return Nome;
	}

	public void setNome(String nome) {
		Nome = nome;
	}
}
```