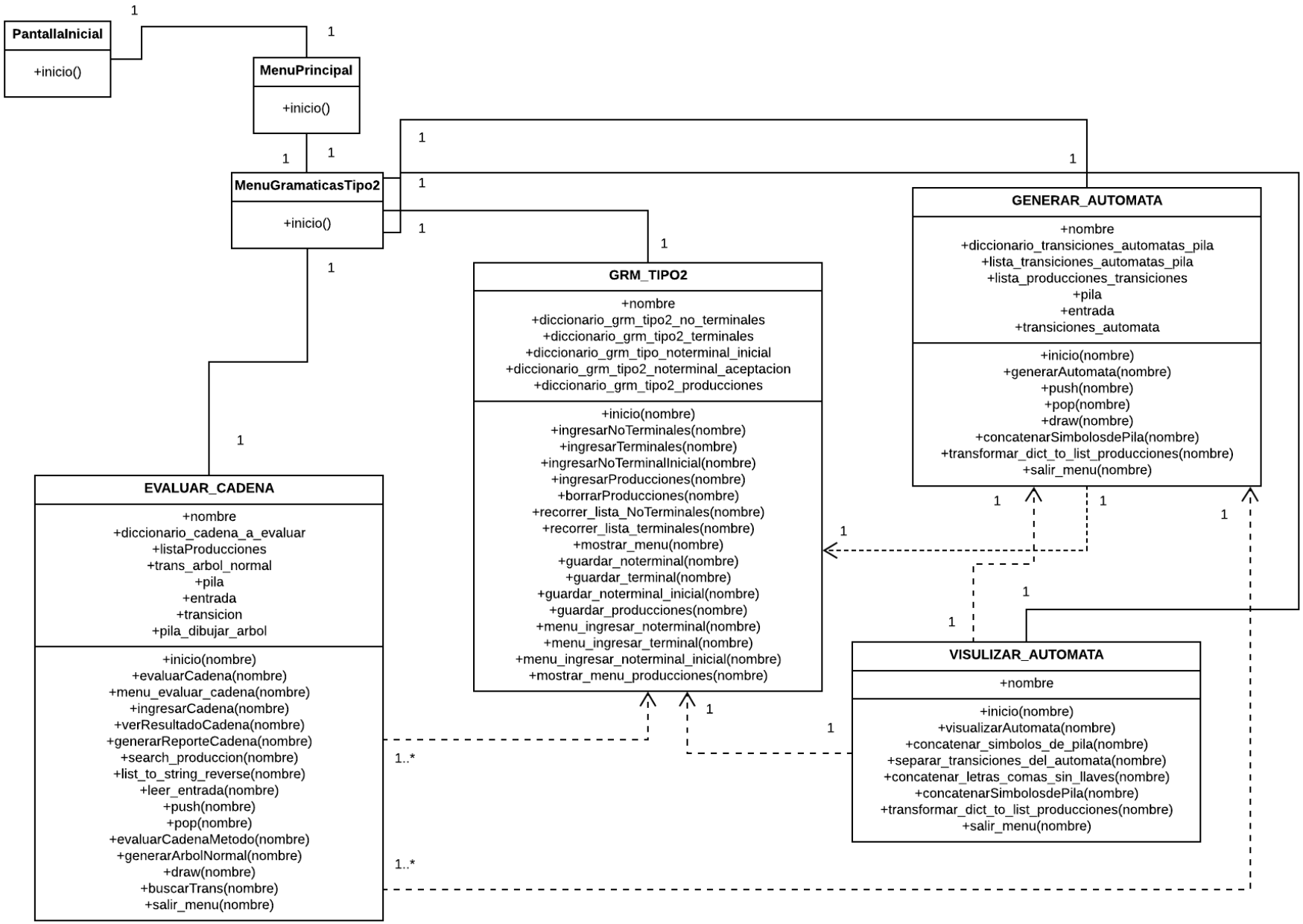
Nombre: Erick Erasmo Jiménez Palacios

Carnet: 201603171

Manual Técnico del Proyecto 2 de Lenguajes de Programación

Diagrama de clases:


PantallaInicial
+inicio()
MenuPrincipal
+inicio()
MenuGramaticasTipo2
+inicio()
GRM_TIPO2
+nombre
+diccionario_grm_tipo2_no_terminales
+diccionario_grm_tipo2_terminales
+diccionario_grm_tipo_noterminal_inicial
+diccionario_grm_tipo2_noterminal_aceptacion
+diccionario_grm_tipo2_producciones
+inicio(nombre)
+ingresarNoTerminales(nombre)
+ingresarTerminales(nombre)
+ingresarNoTerminalInicial(nombre)
+ingresarProducciones(nombre)
+borrarProducciones(nombre)
+recorrer_lista_NoTerminales(nombre)
+recorrer_lista_terminales(nombre)
+mostrar_menu(nombre)
+guardar_noterminal(nombre)
+guardar_terminal(nombre)
+guardar_noterminal_inicial(nombre)
+guardar_producciones(nombre)
+menu_ingresar_noterminal(nombre)
+menu_ingresar_terminal(nombre)
+menu_ingresar_noterminal_inicial(nombre)
+mostrar_menu_producciones(nombre)
GENERAR_AUTOMATA
+nombre
+diccionario_transiciones_automatas_pila
+lista_transiciones_automatas_pila
+lista_producciones_transiciones
+pila
+entrada
+transiciones_automata
+inicio(nombre)
+generarAutomata(nombre)
+push(nombre)
+pop(nombre)
+draw(nombre)
+concatenarSimbolosdePila(nombre)
+transformar_dict_to_list_producciones(nombre)
+salir_menu(nombre)
EVALUAR_CADENA
+nombre
+diccionario_cadena_a_evaluar
+listaProducciones
+trans_arbol_normal
+pila
+entrada
+transicion
+pila_dibujar_arbol
+inicio(nombre)
+evaluarCadena(nombre)
+menu_evaluar_cadena(nombre)
+ingresarCadena(nombre)
+verResultadoCadena(nombre)
+generarReporteCadena(nombre)
+search_produccion(nombre)
+list_to_string_reverse(nombre)
+leer_entrada(nombre)
+push(nombre)
+pop(nombre)
+evaluarCadenaMetodo(nombre)
+generarArbolNormal(nombre)
+draw(nombre)
+buscarTrans(nombre)
+salir_menu(nombre)
VISULIZAR_AUTOMATA
+nombre
+inicio(nombre)
+visualizarAutomata(nombre)
+concatenar_simbolos_de_pila(nombre)
+separar_transiciones_del_automata(nombre)
+concatenar_letras_comas_sin_llaves(nombre)
+concatenarSimbolosdePila(nombre)
+transformar_dict_to_list_producciones(nombre)
+salir_menu(nombre)
1
1..*

Paradigmas utilizados:

Se utilizó el paradigma orientado a objetos para guardar la información correspondiente a la gramática y al autómata de pila. Tales se guardaron en objetos de Gramática Tipo 2 y Generar Autómata de Pila. Se instancian los objetos para guardarlos en la clase con su respectivo nombre.

```python
class GRM_TIPO2:
    nombre = '' # Declaramos el atributo nombre

    diccionario_grmtipo2_noterminales = dict() #Declaramos diccionario que contiene los nombres de AFDs y su conjunto de estados pertenecientes
    diccionario_grmtipo2_terminales = dict() #Declaramos diccionario que contiene los nombres de AFDs y su conjunto de alfabeto pertenecientes a
    diccionario_grmtipo2_noterminal_inicial = dict() #Declaramos diccionario que contiene los estados iniciales del AFD
    diccionario_grmtipo2_noterminal_aceptacion = dict() #Declaramos diccionario que contiene los estados de aceptacion del AFD
    diccionario_grmtipo2_producciones = dict() #Declaramos diccionario que contiene los estados de aceptacion del AFD
    diccionario_grmtipo2_producciones_metodo2 = dict() #Declaramos diccionario el metodo 2 para reportes
    diccionario_grmtipo2_producciones_metodo2_original = dict() #Declaramos diccionario el metodo 2 transformado a metodo 1 para usarlos en las

    listaNoTerminales = []
    listaTerminales = []
    listaNoTerminalInicial = []
    listaNoTerminalesAceptacion = []
    listaProducciones = []
    listaProduccionesMetodo2 = []

    def __init__(self, nombre): #Inicializamos la gramática pasando por parámetros el nombre y self.
        self.nombre = nombre

    def inicio(self):
        # INSTANCIAMOS LA CLASE DENTRO DE SÍ MISMA y creamos el objeto xy que referencia al objeto gramática
        xy = GRM_TIPO2 (self.nombre) # Pasamos en la instancia el parámetro del nombre de la gramática que se utiliza en ese momento
        print ("#######################  PROYECTO 2  ########################")
        print ("####################### MENÚ INGRESAR O MODIFICAR GRAMÁTICA ########################")
        xy.mostrar_menu(self.listaNoTerminales, self.listaTerminales, self.listaNoTerminalInicial, self.listaNoTerminalesAceptacion, self.listaP

    # Declaramos más MÉTODOS que llevan el parámetro self, que se refiere al objeto del método que se está siendo llamado, es decir self
    def ingresarTerminales(self):
        xy = GRM_TIPO2 (self.nombre)
        print ("#######################  PROYECTO 2  ########################")
```

```python
class GENERAR_AUTOMATA:
    nombre = '' # Declaramos el atributo nombre

    #Declaramos las variables, listas y diccionarios a usar
    diccionario_transiciones_automatas_pila = dict() #Declaramos diccionario que contiene los nombres de AFDs y su conjunto de estados perteneci
    lista_transiciones_automatas_pila = [] # Servirá para guardarlo en los diccionarios
    lista_producciones_transiciones = [] # Servirá para graficar el paso 5
    lista_estados_transiciones = [] # Servirá para graficar el paso 5

    pila = []
    entrada = []
    entrada_string = ""
    transiciones_automata = []
    transiciones_impresas = []

    estados = ["i","p","q","f"]
    #trans = [("A","B", 1),("A","E",0),("A","E",1),("A","A",1),("A","D",1),("F","F",1),("D","C",1),("B","A",0), ("E","C",0),("F","D",0), ("C","A
    trans = []
    inicial = ["i"]
    alf = []
    terminal = ()
    l = list(terminal)
    l.append("f")
    terminal = tuple(l)

    x_contador_pila = 0

    def __init__(self, nombre): #Inicializamos el autómata pasando por parámetros el nombre y self.
        self.nombre = nombre

    def inicio(self):
        # INSTANCIAMOS LA CLASE DENTRO DE SÍ MISMA y creamos el objeto xy que referencia al objeto Generar Autómata
        xy = GENERAR_AUTOMATA (self.nombre) # Pasamos en la instancia el parámetro del nombre de la gramática que se utiliza en ese momento
```

Código para generar autómata de q a q

```
#Paso 5 de q a q
xy.transformar_dict_to_list_producciones()
mi_cont = 0
for x in xy.lista_producciones_transiciones:
    x1 = "q" # Declaramos el estado actual
    x2 = xy.leer_entrada(xy.entrada, xy.entrada_string) # Leemos el símbolo del alfabeto que lee la entrada
    if mi_cont == 0:
        if xxx.diccionario_grmtipo2_noterminal_inicial[self.nombre] == xy.lista_estados_transiciones[0]:
            #xsdd = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería
            x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            xsdd = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x4 = "q" # Declaramos el estado a pasar
            xy.push(xy.pila, xy.lista_estados_transiciones[xy.pendejos(mi_cont)], "abcdef") #Insertamos símbolo en la
        else:
            x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería
            x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x4 = "q" # Declaramos el estado a pasar
    else:
        x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería el E
        x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
        x4 = "q" # Declaramos el estado a pasar
    xy.push_transiciones_automata(5, x1, x2, x3, x4, x5)
    xy.x_contador_pila = 2
    self.lista_transiciones_automatas_pila.append(x1+", "+x2+", "+x3+"; "+x4+", "+x5+"*")
    xy.diccionario_transiciones_automatas_pila.update({self.nombre:xy.recorrer_lista_transiciones_metodo1_afd()})
    mi_cont = mi_cont + 1
```

Se utilizó el paradigma funcional para realizar el autómata de pila, utilizando una pila propia con las instrucciones de push y pop correspondientes.

```python
def push(self, pila, simbolo_insertar_pila, simbolo_a_devolver):
    x = simbolo_insertar_pila
    z = simbolo_a_devolver
    #x = x[0:1]
    pila.append(x)
    return z

def pop(self, pila, contador):
    xy = GENERAR_AUTOMATA (self.nombre)
    x = ''
    if contador == 0:
        x = 'λ'
    elif contador == 1:
        x = pila.pop()
    elif contador == 2:
        x = pila.pop()
    return x
```

Tales se utilizaron en:

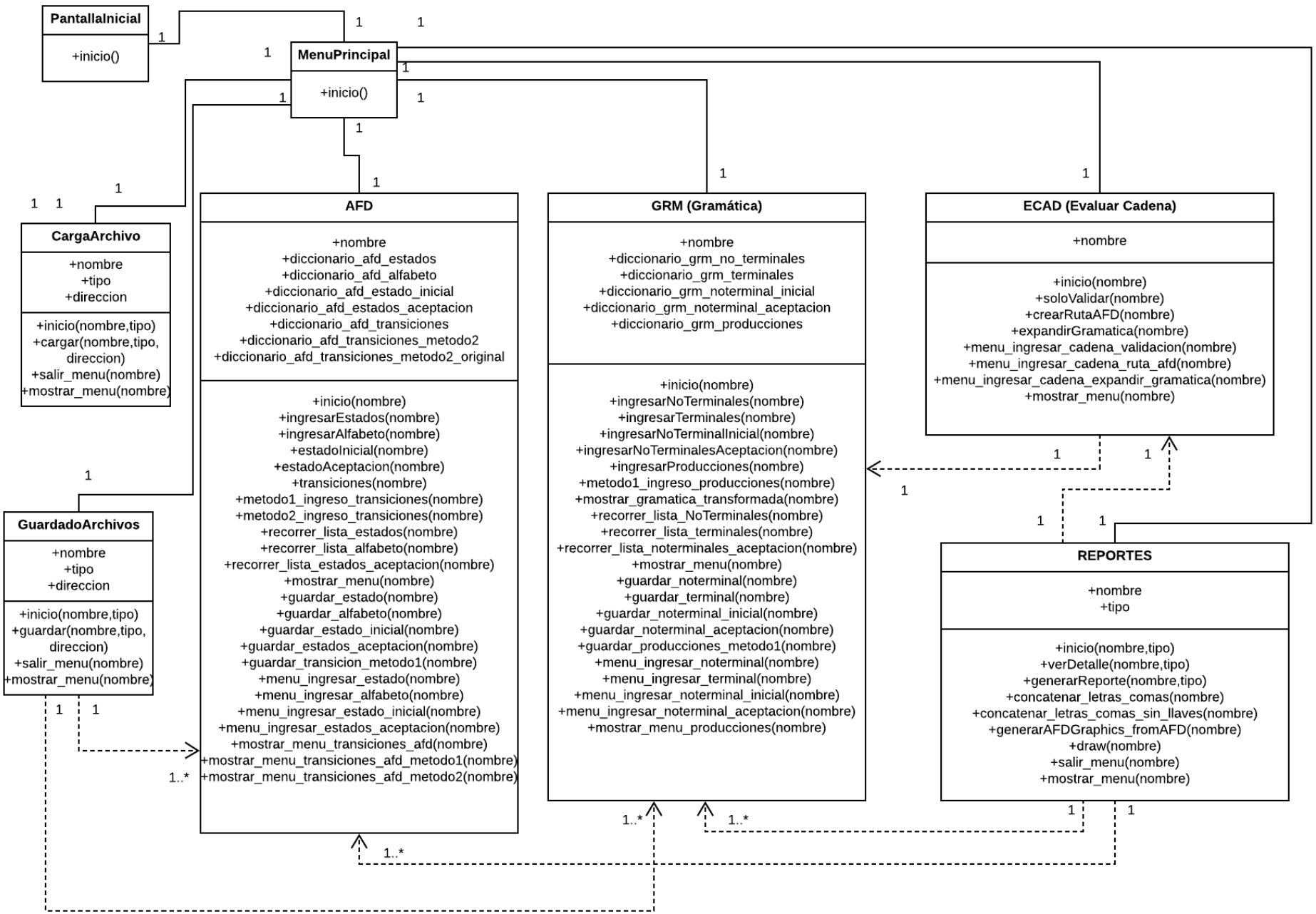
```
#Paso 5 de q a q
xy.transformar_dict_to_list_producciones()
mi_cont = 0
for x in xy.lista_producciones_transiciones:
    x1 = "q" # Declaramos el estado actual
    x2 = xy.leer_entrada(xy.entrada, xy.entrada_string) # Leemos el símbolo del alfabeto que lee la entrada
    if mi_cont == 0:
        if xxx.diccionario_grmtipo2_noterminal_inicial[self.nombre] == xy.lista_estados_transiciones[0]:
            #xsdd = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería
            x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            xsdd = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x4 = "q" # Declaramos el estado a pasar
            xy.push(xy.pila, xy.lista_estados_transiciones[xy.pendejos(mi_cont)], "abcdef") #Insertamos símbolo en la
        else:
            x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería
            x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
            x4 = "q" # Declaramos el estado a pasar
    else:
        x5 = xy.push(xy.pila, xy.lista_estados_transiciones[mi_cont], x) #Insertamos símbolo en la pila que sería el
        x3 = xy.pop(xy.pila, xy.x_contador_pila) #Extraemos símbolo de la pila
        x4 = "q" # Declaramos el estado a pasar
    xy.push_transiciones_automata(5, x1, x2, x3, x4, x5)
    xy.x_contador_pila = 2
    self.lista_transiciones_automatas_pila.append(x1+", "+x2+", "+x3+"; "+x4+", "+x5+"*")
    xy.diccionario_transiciones_automatas_pila.update({self.nombre:xy.recorrer_lista_transiciones_metodo1_afd()})
    mi_cont = mi_cont + 1
```

Así como también se utilizaron para leer la cadena del autómata:

```
cccd = 0
for x in range(1,50):
    if cccd == 0:
        if len(xy.pila)>1 and len(xy.entrada)>0:
            xy.pila_imprimir.append(xy.list_to_string_reverse(xy.pila)) #Resolvimos el problema a mostrar en pila
            xy.entrada_imprimir.append(xy.list_to_string(xy.entrada)) #Resolvimos el problema de la entrada
            pila_for = xy.pop(xy.pila,1) #z
            if pila_for == xy.entrada[0]:
                t_imp = "(q, "+pila_for+", "+pila_for+"; "+ "q, "+ "λ)"
                xy.transicion_imprimir.append(t_imp) #Resolvimos el problema de transiciones
                del xy.entrada[0]
            else:
                loli = xy.search_produccion(pila_for, xy.entrada[0], xy.transicion, xy.entrada)
                if loli != '':
                    t_imp = "(q, λ, "+pila_for+"; q, "+loli+")"
                    xy.transicion_imprimir.append(t_imp) #Resolvimos el problema de transiciones
                    xy.push(xy.pila, loli, loli, 5)
                else:
                    #xy.pila_imprimir.pop()
                    #xy.pila_imprimir.append("-------------") #Resolvimos el problema a mostrar en pila
                    #xy.entrada_imprimir.pop()
                    #xy.entrada_imprimir.append("-------------") #Resolvimos el problema de la entrada
                    xy.transicion_imprimir.append("CADENA INVALIDA")
                    cccd = 1
        elif len(xy.pila)==1 and xy.pila[0]=="#" and len(xy.entrada)==0:
            xy.pila_imprimir.append(xy.list_to_string_reverse(xy.pila)) #Resolvimos el problema a mostrar en pila
            xy.entrada_imprimir.append("-------------") #Resolvimos el problema de la entrada
            pila_for = xy.pop(xy.pila,1) #z
            t_imp = "(q, λ, "+pila_for+"; f, λ)"
            xy.transicion_imprimir.append(t_imp)
        elif len(xy.pila)==0 and len(xy.entrada)==0:
```

Manual Técnico del Proyecto 1 de Lenguajes de Programación

Diagrama de clases:


PantallaInicial
+inicio()
MenuPrincipal
+inicio()
CargaArchivo
+nombre
+tipo
+direccion
+inicio(nombre,tipo)
+cargar(nombre,tipo, direccion)
+salir_menu(nombre)
+mostrar_menu(nombre)
AFD
+nombre
+diccionario_afd_estados
+diccionario_afd_alfabeto
+diccionario_afd_estado_inicial
+diccionario_afd_estados_aceptacion
+diccionario_afd_transiciones
+diccionario_afd_transiciones_metodo2
+diccionario_afd_transiciones_metodo2_original
+inicio(nombre)
+ingresarEstados(nombre)
+ingresarAlfabeto(nombre)
+estadoInicial(nombre)
+estadoAceptacion(nombre)
+transiciones(nombre)
+metodo1_ingreso_transiciones(nombre)
+metodo2_ingreso_transiciones(nombre)
+recorrer_lista_estados(nombre)
+recorrer_lista_alfabeto(nombre)
+recorrer_lista_estados_aceptacion(nombre)
+mostrar_menu(nombre)
+guardar_estado(nombre)
+guardar_alfabeto(nombre)
+guardar_estado_inicial(nombre)
+guardar_estados_aceptacion(nombre)
+guardar_transicion_metodo1(nombre)
+menu_ingresar_estado(nombre)
+menu_ingresar_alfabeto(nombre)
+menu_ingresar_estado_inicial(nombre)
+menu_ingresar_estados_aceptacion(nombre)
+mostrar_menu_transiciones_afd(nombre)
+mostrar_menu_transiciones_afd_metodo1(nombre)
+mostrar_menu_transiciones_afd_metodo2(nombre)
GRM (Gramática)
+nombre
+diccionario_grm_no_terminales
+diccionario_grm_terminales
+diccionario_grm_noterminal_inicial
+diccionario_grm_noterminal_aceptacion
+diccionario_grm_producciones
+inicio(nombre)
+ingresarNoTerminales(nombre)
+ingresarTerminales(nombre)
+ingresarNoTerminalInicial(nombre)
+ingresarNoTerminalesAceptacion(nombre)
+ingresarProducciones(nombre)
+metodo1_ingreso_producciones(nombre)
+mostrar_gramatica_transformada(nombre)
+recorrer_lista_NoTerminales(nombre)
+recorrer_lista_terminales(nombre)
+recorrer_lista_noterminales_aceptacion(nombre)
+mostrar_menu(nombre)
+guardar_noterminal(nombre)
+guardar_terminal(nombre)
+guardar_noterminal_inicial(nombre)
+guardar_noterminal_aceptacion(nombre)
+guardar_producciones_metodo1(nombre)
+menu_ingresar_noterminal(nombre)
+menu_ingresar_terminal(nombre)
+menu_ingresar_noterminal_inicial(nombre)
+menu_ingresar_noterminal_aceptacion(nombre)
+mostrar_menu_producciones(nombre)
ECAD (Evaluar Cadena)
+nombre
+inicio(nombre)
+soloValidar(nombre)
+crearRutaAFD(nombre)
+expandirGramatica(nombre)
+menu_ingresar_cadena_validacion(nombre)
+menu_ingresar_cadena_ruta_afd(nombre)
+menu_ingresar_cadena_expandir_gramatica(nombre)
+mostrar_menu(nombre)
GuardadoArchivos
+nombre
+tipo
+direccion
+inicio(nombre,tipo)
+guardar(nombre,tipo, direccion)
+salir_menu(nombre)
+mostrar_menu(nombre)
REPORTES
+nombre
+tipo
+inicio(nombre,tipo)
+verDetalle(nombre,tipo)
+generarReporte(nombre,tipo)
+concatenar_letras_comas(nombre)
+concatenar_letras_comas_sin_llaves(nombre)
+generarAFDGraphics_fromAFD(nombre)
+draw(nombre)
+salir_menu(nombre)
+mostrar_menu(nombre)
1
1..*

Paradigmas utilizados:

Se utilizó el paradigma orientado a objetos, ya que se usó diferentes clases tales como el objeto Gramática y el objeto AFD, que se instancian y reciben como parámetro el nombre de la misma. Tales clases poseen métodos y parámetros, como los diccionarios y métodos para ejecutar los menús respectivos.

Además de la clase AFD y GRM (Gramática) se tienen otras clases utilizadas como la clase reportes, la clase evaluar cadenas, la clase cargar archivo y guardar archivo. Cada una con sus métodos respectivos y funciones para guardar datos, buscar datos, y recuperar datos.

```
class AFD:
    nombre = ''

    diccionario_afd_estados = dict() #Declaramos diccionario que contiene los nombres de AFDs y su conjunto de estados pertenecientes a cada uno
    diccionario_afd_alfabeto = dict() #Declaramos diccionario que contiene los nombres de AFDs y su conjunto de alfabeto pertenecientes a cada u
    diccionario_afd_estado_inicial = dict() #Declaramos diccionario que contiene los estados iniciales del AFD
    diccionario_afd_estados_aceptacion = dict() #Declaramos diccionario que contiene los estados de aceptacion del AFD
    diccionario_afd_transiciones = dict() #Declaramos diccionario que contiene los estados de aceptacion del AFD
    diccionario_afd_transiciones_metodo2 = dict() #Declaramos diccionario el metodo 2 para reportes
    diccionario_afd_transiciones_metodo2_original = dict() #Declaramos diccionario el metodo 2 transformado a metodo 1 para usarlos en las valid
```

```
def __init__(self, nombre):
    self.nombre = nombre

def inicio(self):
    xy = AFD (self.nombre)
    print ("#######################  PROYECTO 1  ########################")
    print ("####################### MENÚ AFD ########################")
    xy.mostrar_menu(self.listaEstadosAFD, self.listaAlfabetoAFD, self.listaEstadoInicialAFD, self.listaEstadosAceptacionAFD,

def ingresarEstado(self):
    xy = AFD (self.nombre)
    print ("#######################  PROYECTO 1  ########################")
    print ("####################### MENÚ AFD >> Ingresar Estados de "+ self.nombre +"  ########################")
    print ("")
    xy.menu_ingresar_estado()

def ingresarAlfabeto(self):
    xy = AFD (self.nombre)
    print ("#######################  PROYECTO 1  ########################")
    print ("####################### MENÚ AFD >> Ingresar símbolos del Alfabeto de "+ self.nombre +"  ########################
    print ("")
    xy.menu_ingresar_alfabeto()

def estadoInicial(self):
    xy = AFD (self.nombre)
    print ("#######################  PROYECTO 1  ########################")
    print ("####################### MENÚ AFD >> Ingresar Estado Inicial de "+ self.nombre +"  ########################")
    print ("")
    xy.menu_ingresar_estado_inicial()

def estadoAceptacion(self):
```

Otros objetos como la clase Reportes recibe como parámetro el nombre y el tipo, que luego se instancian en los demás métodos:

```python
class REPORTES:
    nombre = ''
    tipo = ''

    def __init__(self, nombre, tipo):
        self.nombre = nombre
        self.tipo = tipo

    def inicio(self):
        xy = REPORTES (self.nombre, self.tipo)
        print ("#######################  PROYECTO 1  ########################")
        if self.tipo == 'grm':
            print ("####################### MENÚ REPORTES >> Gramática "+ self.nombre +"  ########################")
        elif self.tipo == 'afd':
            print ("####################### MENÚ REPORTES >> AFD "+ self.nombre +"  ########################")
        xy.mostrar_menu() #Pasamos los estados al menú con el propósito de borrarlos despúes

    def verDetalle(self):
        xy = REPORTES (self.nombre, self.tipo)
        xxx = AFD(self.nombre)
        yyy = GRM(self.nombre)
        print ("#######################  PROYECTO 1  ########################")
        if self.tipo == 'grm':
            print ("####################### MENÚ REPORTE >> Ver detalle Gramática "+ self.nombre +"  ########################")
        elif self.tipo == 'afd':
            print ("####################### MENÚ REPORTES >> Ve detalle AFD "+ self.nombre +"  ########################")
        print ("")
```

Clase cargar Archivos:

```python
class CA:
    nombre = ''
    tipo = ''
    direccion = ''

    def __init__(self, nombre, tipo, direccion):
        self.nombre = nombre
        self.tipo = tipo
        self.direccion = direccion

    def inicio(self):
        xy = CA (self.nombre, self.tipo, self.direccion)
        print ("#######################  PROYECTO 1  ########################")
        if self.tipo == 'grm':
            print ("####################### MENÚ CARGAR ARCHIVOS >> Gramática "+ self.nombre +"  ########################")
        elif self.tipo == 'afd':
            print ("####################### CARGAR ARCHIVOS >> AFD "+ self.nombre +"  ########################")
        xy.verDetalle() #Pasamos los estados al menú con el propósito de borrarlos despúes

    def verDetalle(self):
        xy = CA (self.nombre, self.tipo, self.direccion)
        xxx = AFD(self.nombre)
        yyy = GRM(self.nombre)
        if self.tipo == 'afd':
            #data_file = 'C:\\Users\\vuyisile\\Desktop\\Test\\data.txt'
            #C:\Users\USUARIO\Desktop\
            #binariasConAlmenosDos0
```

También se utilizó el paradigma Funcional, ya que también se definieron diferentes procedimientos o funciones que retornan valor tales como:

Pedir el nombre del AFD o la Gramática a guardar y luego pasar por parámetros a las clases respectivas.

```
def pedir_nombre_afd():
        print ("######################  PROYECTO 1  ########################")
        print ("###################### MENÚ AFD ########################")
        print ("")
        cadena = input("Introduce el nombre del AFD o escribe salir para regresar al menú principal: ")
        if cadena == 'salir':
            os.system(var) #Limpiamos o cambiamos pantalla
            MenuPrincipal.inicio()
        else:
            if(len(cadena)==0):
                print ("El nombre no puede estar vacío. Por favor, ingresa un nombre.")
                pedir_nombre_afd()
            elif (cadena.isspace()):
                print ("El nombre no puede estar vacío. Por favor, ingresa un nombre.")
                pedir_nombre_afd()
        return cadena
```

O pedir el nombre de un archivo a cargar:

```
def pedir_nombre_archivo():
    os.system(var)
    cadena = input("Introduce el nombre del archivo alojado en la dirección antes ingresada: ")
    if cadena == 'salir':
        os.system(var) #Limpiamos o cambiamos pantalla
        pedir_nombre_archivo()
    else:
        cadena = cadena
    return cadena
```