Universidad de San Carlos de Guatemala
Facultad de ingeniería
Escuela de ciencias y sistemas

Manual Técnico

Fernando Arturo Pensamiento Calderón
3011-83457-0101
201602743

Introducción

Este manual fue realizado para que si algún programador en el futuro desea innovar arit software o agregar nuevo contenido sea fácil de entender y de expandir, o que alguien pueda recrear algún software parecido también lo pueda usar de guía.

Requerimientos del sistema.

- Tener instalado java versión 8
- Tener por lo menos dos gigas de RAM
- Utilizar netbeans, este fue desarrollado en netbeans 8.0.1

Patrones utilizados

Se utilizó el patrón intérprete.

Explicación de paquetes y clases.

Analizador

El paquete analizador solo funciona para darle color al texto ingresado al IDE, la única clase que tiene es ColorArit, la cual es generada por un archivo Flex.

Clases auxiliares

Nodo

Esta clase abstracta es de la cual todos los nodos heredan.

Atributos de la clase.

```
   public ArrayList<Nodo> hijos = new ArrayList<Nodo>();
    public int fila;
    public int columna;
    public boolean tieneHijos;
```

Constructores de la clase

```
public Nodo(int f, int c) {
        fila = f;
        columna = c;
    }

    public Nodo(int f, int c, Nodo hijo) {
        fila = f;
        columna = c;
        hijos = new ArrayList<Nodo>();
        hijos.add(hijo);
        this.tieneHijos = true;
    }

    public Nodo(int f, int c, Nodo hijoI, Nodo hijoD) {
        fila = f;
```

```
        columna = c;
        hijos = new ArrayList<Nodo>();
        hijos.add(hijoI);
        hijos.add(hijoD);
        this.tieneHijos = true;
    }

    public Nodo(int f, int c, ArrayList<Nodo> hj) {
        fila = f;
        columna = c;
        hijos = hj;
        this.tieneHijos = true;
    }
```

En esta clase puse bastantes constructores para hacerlo más fácil del lado de javacc y Flex y Cup

```
public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        System.out.println("TODO funcion ejecutar en " + this.getClass().get
Name() + "  no realizada aun");
        return null;
    }
```

Un método ejecutar para que todas las clases tuvieran esto, se añadió lo de imprimir si faltaba hacer para que si alguna clase no lo sobrescribiera me informara que faltaba realizar la función ejecutar.

```
 public void dibujar(Dibujador d, String padre) {
        //d.st += this.hashCode() + "[label=\"" + this.getClass().getSimpleN
ame() + " " + fila  + " " + columna + "\" ]; \n";
        d.st += this.hashCode() + "[label=\"" + this.getClass().getSimpleNam
e() + "\" ]; \n";
        d.st += padre + " ->  " + this.hashCode() + ";\n";
        dibHijos(d);
    }

    public void dibHijos(Dibujador d) {
        if (tieneHijos && hijos != null) {
            for (Nodo n : hijos) {
                if (n == null) {
                    continue;
                }
                n.dibujar(d, this.hashCode() + "");
            }
        }
    }
```

Los métodos de dibujar y dibujar hijos se encargan de dibujar el AST, debido a que ambas gramáticas utilizan los mismos nodos solo es necesario hacerlo una vez y si en algún nodo se desea cambiar la forma de imprimir se le puede sobrescribir

contenedorEnum

Contiene los enums utilizados en el programa.

```
    public enum Tipos {
        cadena, numerico, entero, booleano , nulo
    }
```

Dibujador

Esta clase solo tiene un atributo de tipo String para que se mandara por referencia y no por valor.

Comparadores

Ternario

Esta clase se encarga de realizar las funciones ternarias, hereda de nodo. No se le agrega ningún atributo y su código de ejecutar es:

```
@Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        Nodo n = hijos.get(0);
        Object o1 = n.ejecutar(ts, aux), o2 = aux.ayuda_bool(o1);
        if (!(o2 instanceof Boolean)) {
            aux.error("Se esperaba un valor booleano", n.fila, n.columna);
        }
        return (boolean) o2 ? hijos.get(1).ejecutar(ts, aux) : hijos.get(2).
ejecutar(ts, aux);

    }
```

Compi_proyecto_pkg1

Es el paquete que se crea junto con el proyecto, este solo contiene la clase principal que inicia todo.

Funciones del lenguaje

En este paquete se encuentran algunas de las funciones del lenguaje y los ciclos

Continuar

Se utiliza en los ciclos para pasar a la siguiente iteración, su código es

```
 @Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        if (ts.seencontroContinue()) {
            ts.setcontinues();
        } else {
            aux.error("no se esperaba un continue", fila, columna);
        }
        return null;
    }
```

El método seencontroContinue es para saber si este es un ciclo y si esta bien que llegue un continue en ese momento.

Do_while

Su método ejecutar tiene

```
    @Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        Nodo n = hijos.get(1), n1 = hijos.get(0);
        Object o1, o2;
        o1 = n.ejecutar(ts, aux);
        o2 = aux.ayuda_bool(o1);

        do {
            Tabla_Sim ts2 = new Tabla_Sim(ts, "Do_while", aux);
            ts2.esciclo = true;
            n1.ejecutar(ts2, aux);
            if (ts2.haybreak) {
                break;
            }
            o1 = n.ejecutar(ts, aux);
            o2 = aux.ayuda_bool(o1);
            if (!(o2 instanceof Boolean)) {
                aux.error("Se esperaba un valor booleano", n.fila, n.columna
);
                return null;
            }
        } while ((boolean) o2);

        return null;
    }
```

Realiza la función del do while del lenguaje.

For

Se le agrego un atributo de tipo String llamado st; este guarda el nombre de la variable que itera en el For

Se modificó el método dibujar para que imprimiera también la variable que itera

```
 @Override
    public void dibujar(Dibujador d, String padre) {
```

```
        d.st += this.hashCode() + "[label=\"" + this.getClass().getSimpleNam
e() + " [" + st + "]  \" ]; \n";
        d.st += padre + " ->  " + this.hashCode() + "; \n";
        dibHijos(d);
    }
```

Y el método ejecutar tiene:

```
@Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {

        Object o1 = hijos.get(0).ejecutar(ts, aux);
        Nodo n = hijos.get(1);
        if (o1 instanceof Simbolo_prim) {
            Simbolo_prim sp = (Simbolo_prim) o1;
            ejecVec(new Vector(sp, aux), ts, aux, n);
        } else if (o1 instanceof Vector) {
            ejecVec((Vector) o1, ts, aux, n);
        } else if (o1 instanceof Matriz) {
            ejecMat((Matriz) o1, ts, aux, n);
        } else if (o1 instanceof Lista) {
            ejecList((Lista) o1, ts, aux, n);
        } else if (o1 instanceof Array) {
            ejecArr((Array) o1, ts, aux, n);
        } else {
            //error :(
        }

        return null;
    }
```

En cualquiera de esos métodos se obtiene el arreglo lineal de la estructura y se llama después al siguiente método:

```
public void ejecArrOb(ArrayList<Object> arr, Tabla_Sim ts, Auxiliar aux, Nod
o n, Estructura e) {
        int c = 0;
        for (Object sp : arr) {
            Tabla_Sim ts2 = new Tabla_Sim(ts, "For", aux);
            ts2.esciclo = true;
            ts2.agregar_var(st, sp);

            ts2.esFor = true;
            ts2.nFor = c++;
            ts2.estFor = e;
            ts2.stFor = st;
```

```
            n.ejecutar(ts2, aux);
            if (ts2.haybreak) {
                break;
            }
        }
    }
```

Funciones_nativas

Esta clase contiene las funciones nativas del lenguaje, esta no hereda de la clase nodo.
Esta clase es muy grande por eso solo se explicara que atributos tiene y cada método.

```
int fila, columna;

    Auxiliar au;

    public ArrayList<Bar_chart> arrbc = new ArrayList<>();
    public ArrayList<Histogram_chart> arrhc = new ArrayList<>();
    public ArrayList<Line_chart> arrlc = new ArrayList<>();
    public ArrayList<Pie_chart> arrpc = new ArrayList<>();
```

Un auxiliar ya que es el que contiene los errores y las salidas a consola.
También es necesario la fila y columna para saber dónde reportar el error.

Y los arreglos son de las distintas graficas que se pueden realizar.

```
public Object selFunc(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos, Str
ing func, int f, int c)
```

Este método recibe una tabla de símbolos que es donde esta almacenada la información, un auxiliar que es donde se tienen los errores y las consolas, un arreglo de nodos que son los parámetros, el nombre de la función y la fila y columna de donde se llama.
En este método solo se direcciona al método que cumple con la función indicada.

```
public Object hist(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Se encarga de realizar la gráfica de histograma.

```
public Object plot(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Realiza la gráfica de líneas o la gráfica de dispersión.

```
public Object barplot(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Realiza la gráfica de barras.

```
public Object pie(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Realiza la gráfica de pie

```
public Object Arr(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Es el método que crea los arreglos.

```
public Object lista(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```

Es el método que se encarga de crear las listas

```
public Object oparr(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos, Strin
g s)
```
Se encarga de realizar las funciones de media, mediana y moda.

```
public Object opnum(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos, Strin
g s)
```
Realiza las funciones de trunk y round

```
public Object remove(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```
Realiza la funcione de remove

```
 public Object opstring(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos, S
tring s)
```
Realiza las operaciones de stringlength, tolowercase y touppercase

```
public Object nalgo(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos, boole
an ncol)
```
Realiza las operaciones de nrow y ncol

```
public Object length(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```
Mira el tamaño de una estructura

```
public Object typeof(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```
Mira el tipo de una estructura

```
public Object matrix(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```
Es el método que se encarga de crear matrices.

```
public Object c(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> hijos)
```
Es el método que se encarga de la función c del lenguaje, esta crea listas o vectores.

```
public Object Print(Tabla_Sim ts, Auxiliar aux, ArrayList<Nodo> arr)
```
Imprime en consola

If

Es la clase que se encarga de la función if.

```
 @Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        String ambito = "If";
        Nodo n = hijos.get(0);
        Object o1 = n.ejecutar(ts, aux);
        Object o2 = aux.ayuda_bool(o1);
        if (!(o2 instanceof Boolean)) {
            return aux.error("Se esperaba un valor booleano", n.fila, n.colu
mna);
        }

        if ((boolean) o2) {
            n = hijos.get(1);
```

```
        } else if (hijos.size() == 3) {

            n = hijos.get(2);
            if (n instanceof If) {
                n.ejecutar(ts, aux);
                return null;
            }
            ambito = "else";
        } else {
            return null;
        }
        Tabla_Sim ts2 = new Tabla_Sim(ts, ambito, aux);
        n.ejecutar(ts2, aux);
        return null;
    }
```

Clase parar

Hace lo mismo que el continúe pero con break;

Clase retorno

```
 @Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        if (ts.seencontroReturn()) {
            Object o = null;
            aux.aiuda++;
            if (hijos.size() == 1) {
                if (hijos.get(0) != null) {
                    o = hijos.get(0).ejecutar(ts, aux);
                    if (o == null) {
                        return aux.error("Return devolvio null", fila, colum
na);
                    }
                }
            }
            ts.setReturn(o);
            return null;
        }
        return aux.error("No se esperaba un return ya que no se esta dentro
de una funcion. ", fila, columna);
    }
```

Pregunta si se espera un retorno y se lo asigna a la tabla de símbolos de la función.

Switch

```
@Override
    public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        Object o1 = hijos.get(0).ejecutar(ts, aux);
```

```
        Tabla_Sim ts2 = new Tabla_Sim(ts, "switch", aux);
        ts2.esswitch = true;

        Simbolo_prim sp = null;
        if (o1 instanceof Simbolo_prim) {
            sp = (Simbolo_prim) o1;
        } else if (o1 instanceof Vector) {
            sp = ((Vector) o1).arr.get(0);
        } else if (o1 instanceof Matriz) {
            sp = ((Matriz) o1).arr.get(0);
        } else {
            return aux.error("En el switch se espera un simbolo primario, un
 vector o una matriz", fila, columna);
        }

        ArrayList<Nodo> arrn = hijos.get(1).hijos;

        boolean coincidioCase = false;
        int def = -1;
        for (int a = 0; a < arrn.size(); a++) {
            if (ts2.haybreak || ts2.hayreturn || ts2.haycontinue) {
                break;
            }

            Nodo n = arrn.get(a);
            if (n instanceof Default) {
                if (def != -1) {
                    return aux.error("Solo puede haber un default por swtich
", n.fila, n.columna);
                }
                def = a;
            }
            if (coincidioCase) {
                if (n instanceof Case) {
                    continue;
                }
                n.ejecutar(ts2, aux);
            } else {
                if (!(n instanceof Case)) {
                    continue;
                }
                Object o2 = n.ejecutar(ts2, aux);
                Simbolo_prim sp2 = null;
                if (o2 instanceof Simbolo_prim) {
                    sp2 = (Simbolo_prim) o2;
```

```
                } else if (o2 instanceof Matriz) {
                    sp2 = ((Matriz) o2).arr.get(0);
                } else if (o2 instanceof Vector) {
                    sp2 = ((Vector) o2).arr.get(0);
                }
                coincidioCase = comparar(sp, sp2);
            }
        }
        if (!coincidioCase) {
            for (int a = def; a < arrn.size(); a++) {
                if (ts2.haybreak || ts2.hayreturn) {
                    break;
                }
                Nodo n = arrn.get(a);
                if (n instanceof Case) {
                    continue;
                }
                n.ejecutar(ts2, aux);
            }
        }

        return null;
    }
```

Revisa si coincide con algún case y luego ejecuta hasta que se encuentre un break o se acaben los nodos, si no se encuentra un break ejecuta el default y realiza lo mismo.

While

Realiza el while del lenguaje

```
    @Override
     public Object ejecutar(Tabla_Sim ts, Auxiliar aux) {
        Nodo n = hijos.get(0), n1 = hijos.get(1);
        Object o1, o2;
        o1 = n.ejecutar(ts, aux);
        o2 = aux.ayuda_bool(o1);
        if (!(o2 instanceof Boolean)) {
            aux.error("Se esperaba un valor booleano", n.fila, n.columna);
        }

        while ((boolean) o2) {
            Tabla_Sim ts2 = new Tabla_Sim(ts, "While", aux);
            ts2.esciclo = true;
            n1.ejecutar(ts2, aux);
            if (ts2.haybreak) {
                break;
            }
```

```
            o1 = n.ejecutar(ts, aux);
            o2 = aux.ayuda_bool(o1);
            if (!(o2 instanceof Boolean)) {
                aux.error("Se esperaba un valor booleano", n.fila, n.columna
);
                return null;
            }
        }

        return null;
    }
```

Generador de graficas.

Se usa para general las gráficas, se utiliza la librería jfreechart.

Interfaz

Tiene la interfaz del proyecto y una clase llamada Campotexto que es la que ayuda a que el texto tenga colores.

Una clase para generar el reporte de errores y otra para general el de la tabla de símbolos.

Tabla de símbolos

Es la que tiene las estructuras del lenguaje y se encarga de guardar las variables

Estructura

Clase abstracta de la cual heredan todas las estructuras, tiene de atributos un auxiliar para reportar errores
Métodos:

En el constructor solicita el auxiliar.

```
    public Estructura copear()
```

Copea la estructura para que siempre se pase por valor y no por referencia.

```
public int size()
```

Devuelve el tamaño de la estructura.

Funcion

Esta clase guarda las funciones creadas por el usuario a la hora de ejecutar el programa.

```
  public String nombre;
    public Nodo inst_cuerpo = null;
    public ArrayList<Nodo> vars = new ArrayList<>();
    public int nvars = 0;
```

Guarda el nombre, los parámetros y las instrucciones del cuerpo de la función.

Auxiliar

```
public int aiuda = 0;
```

```
    public int aiuda2 = 1;
    public boolean javacc;
    public TextArea tx;
    public TextArea txterr;
    public String st = "";
    public String error = "";
    public ArrayList<MiError> arrErr = new ArrayList<>();
    public Tabla_Sim global;



    public HashMap<String, Funcion> hfun = new HashMap<String, Funcion>();

    public ArrayList<String> nats = new ArrayList<>();
    public Funciones_nativas f = new Funciones_nativas();
```

Este guarda un booleano para saber si se ejecuto desde javacc o no , las dos consolas, el texto de las consolas, un arreglo de errores, la tabla de símbolos global para pasárselo a las funciones, un hashmap de las funciones creadas por el usuario, un arraylist con el nombre de las funciones nativas y las funciones nativas.

Y funciones que puedan ayudar en cualquier parte del proyecto.

Matriz

Guarda las estructuras de tipo matriz, estas se meten en un arreglo lineal y se guarda las dimensiones, se accede al arreglo con el mapeo lexicográfico

Vector

Guarda los vectores.

Lista

Guarda los datos de la matriz en un arraylist de tipo objeto a la cual solo acepta ingresar vectores o listas.

Arreglo

Guarda las estructuras de tipo arreglo, esta tienen un arreglo de objetos y un arreglo de enteros, se accede por medio del mapeo lexicográfico.

Simbolo_prim

Es el elemento mas básico del proyecto, puede ser de tipo cadena, numérico, entero, booleano o nulo y se guarda en un atributo de tipo object.

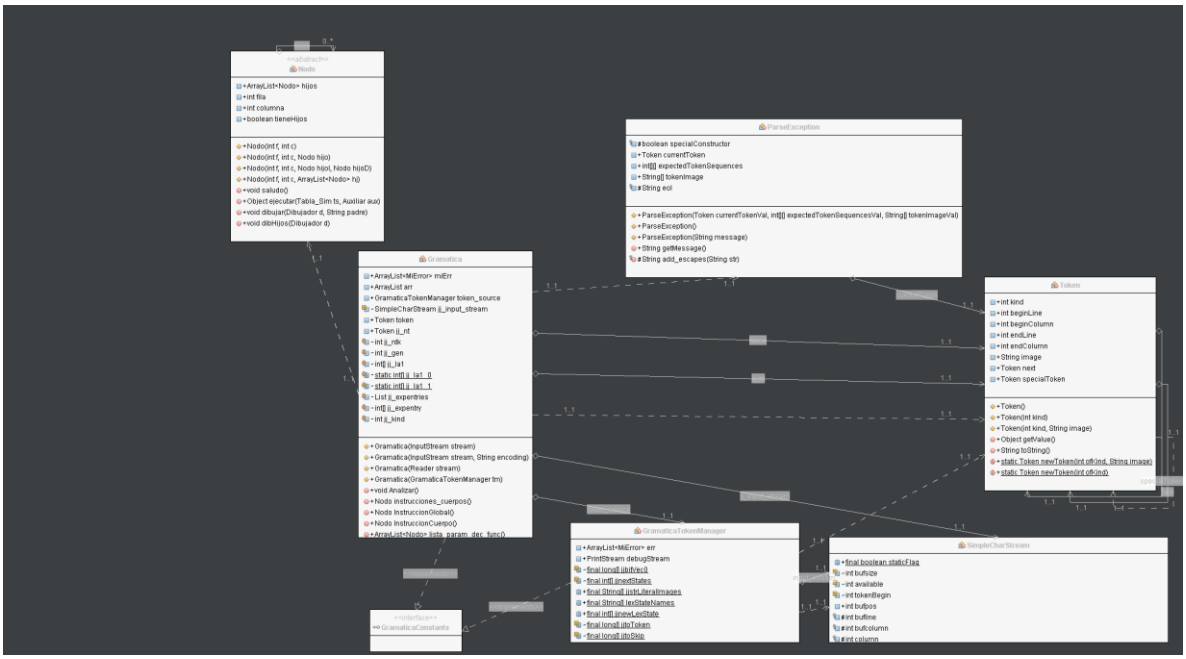
Tabla_Sim

es el que guarda todas las variables.

Tiene un hashmap de tipo estructura y un arrreglo de hijos. Para que se pueda imprimir el reporte de la tabla de símbolos, tambine tiene booleanos y un atributo de tipo object para manejar los return, break y continue.

Diagramas de clases

Debido al tamaño del proyecto y que se realizó con el patrón interprete a todos los nodos heredar de una clase abstracta llamada nodo y que en cada clase se manda una tabla de símbolos, si se muestra una imagen general del diagrama no se entendería casi nada por esa razón se realizaron varios diagramas pequeños pero entendibles.



Este es un diagrama que representa a como se ejecuta en Flex y cup, la mayoría son clases de Flex y cup con excepción de la que está en la esquina superior izquierda que es la clase nodo, esta es la clase abstracta de la cual todo hereda.

En javacc ocurre lo mismo todas las clases de este usan a nuestra clase nodo.