Instituto Politécnico Nacional

Escuela Superior de Cómputo

Compiladores

Tecla Parra Roberto

Práctica 02 - Graficador de figuras en AWT Java

Alumno

Rafael Landa Aguirre

Graficador de figuras usando java.awt y YACC.

Grupo: 3CM8

Ciudad de México, lunes 23 de abril 2018

1. Introducción

Para esta práctica se desarrollará una aplicación de interfaz gráfica en que un usuario podrá dibujar lineas, circulos, rectángulos, e inclusive colorear el contorno de alguna de tales curvas en algún color asignado de acuerdo a un conjunto de instrucciones determinadas por un conjunto de reglas gramaticales.
Para el desarrollo de la interfaz gráfica se cuenta con la API de JAVA: AWT, para aplicaciones de escritorio en Windows/Linux.

2. Conceptos básicos

2.1. Java AWT

AWT (por sus siglas en inglés Abstract Window Toolkit) es una API para el desarrollo de aplicaciones en java.
Los componentes de AWT son de plataforma dependiente, es decir, que de acuerdo al sistema operativo es como mostrará cada unos de los componentes con el estilo propio, debido a que los componentes usan los recursos que ofrece el sistema operativo.
La librería java.awt provee paquetes de clases, tales como, Label, TextField, TextArea, CheckBox, Choice, List, etc.

2.2. Lista de instrucciones

- **LINE**: Instrucción para dibujar líneas.
- **CIRCLE**: Instrucción para dibujar círculos.
- **RECTANGLE**: Instrucción para dibujar rectángulos.
- **IMAGE**: Instrucción para dibujar imágenes.

- **COLOR**: Instrucción para colorear contornos.

2.3. Maquina virtual de Pila

Para la incorporación de nuevas instrucciones en un lenguaje determinado se necesita generar código de forma sencilla, de forma estructurada. Por otro lado, también se necesita la eliminación de la recursividad ya que en tiempo de procesamiento no es recomendable hacer uso de una pila de llamadas a función a nuvel sistema operativo.
La siguiente figura nos muestra cómo se deben de guardar las instrucciones en una RAM virtual o una pila en términos de software.

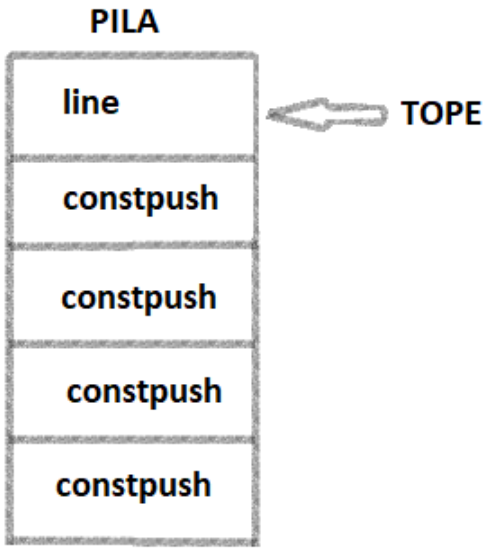

Figura 1: Estructura para generación de código para la instrucción LINE.

3. Desarrollo de la aplicación

3.1. Gramática del vector

A continiación se presenta la gramática que describe la estructura del vector cuando el usuario introduzca información desde teclado:

- instruccion:

  → NUMERO

  → LINE NUMERO NUMERO NUMERO NUMERO

  → CIRCLE NUMERO NUMERO NUMERO

  → RECTANGLE NUMERO NUMERO NUMERO NUMERO

  → IMAGE NUMERO NUMERO

  → COLOR NUMERO

3.2. Símbolos terminales usados en YACC

Para poder describir la estructura de la gramática y traducirla a lenguaje real en caracteres, tenenmos la siguiente información en el archivo $vector.cal.y$:

```
%token NUMBER LINE CIRCLE RECTANGLE COLOR IMAGE PRINT
%start list
```

3.3. Reglas gramaticales en YACC

En el código siguiente se muestra las reglas gramaticales mostradas en la sección 3.1, entonces, solo traducimos a código para YACC, no cambia mucho, solo hay que poner atención en los caracteres.

```
%%
instr: NUMBER { /* constantes */
          ((Algo)$$.obj).inst = maq.code("constpush");
               maq.code(((Algo)$1.obj).simb);
     }
     | LINE NUMBER NUMBER NUMBER NUMBER {
       /* instrccion para dibujar lineas */
       maq.code("constpush");
```

```
            maq.code(((Algo)$2.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$5.obj).simb);
      maq.code("line");
    }
    | CIRCLE NUMBER NUMBER NUMBER {
      /* instrccion para dibujar circulos */
      maq.code("constpush");
            maq.code(((Algo)$2.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
      maq.code("circle");
    }
    | RECTANGLE NUMBER NUMBER NUMBER NUMBER {
    /* instrccion para dibujar rectangulos */
      maq.code("constpush");
            maq.code(((Algo)$2.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$3.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$4.obj).simb);
      maq.code("constpush");
            maq.code(((Algo)$5.obj).simb);
      maq.code("rectangulo");
    }
```

```
    | IMAGE NUMBER NUMBER {
      /* instrccion para colocar imagenes */
      Simbolo simbolo = new Simbolo();
      simbolo.setDibujable(new Imagen(
        cargarImagen("dalmata.jpg"),
        (int)((Algo)$2.obj).simb.val,
        (int)((Algo)$3.obj).simb.val, f));
      maq.code("objectpush");
      maq.code(simbolo);
      maq.code("drawImage");
    }
    | COLOR NUMBER {
      /* instrccion para colorear contorno de las
        figuras previas */
      maq.code("constpush");
              maq.code(((Algo)$2.obj).simb);
      maq.code("color");
    }
    ;
%%
```

3.4. Analizador léxico en YACC

El código siguiente se muestra cómo se analiza caractér por caractér cuando el usuario introduce información desde teclado. Primero, para poder ejecutar la función `yylex()`, se tiene que haber invocado la función `yyparse()`.

```
int yylex() {
  String s; //Cadena del usuario
  int tok;
  Double d;
```

```
  Simbolo simbo; //Cimbolos gramaticales
  if (!st.hasMoreTokens()) //Si no existen mas tokens
       if (!newline) {
        newline = true;
     return ';'; //retornar el token
       }
  else
       return 0;
  s = st.nextToken();
  try { //Agregar a la tabla de simbolos la constante
    d = Double.valueOf(s);
    yylval = new ParserVal(
      new Algo(tabla.install("",
        NUMBER, d.doubleValue()),0)
    );
    tok = NUMBER; //retornar el lexema
  } catch (Exception e) {
     /*Agregar a la tabla de simbolos el token
    LINE, CIRCLE, RECTANGLE, IMAGE, COLOR*/
    if (Character.isLetter(s.charAt(0))) {
      if((simbo = tabla.lookup(s)) == null)
           yylval = new ParserVal(new Algo(simbo, 0));
           tok = simbo.tipo;
    } else {
      tok = s.charAt(0);
    }
  }
  return tok; // retornar el token detectado
}
```

3.5. Almacenamiento simbolos gramaticales

Para el almacenamiento de símbolos gramaticales se tiene planeada la elaboración de un objeto llamado Simbolo que permite definir una instrucción, el nombre, de que tipo es, la dirección de la primera instrucción a la función que apunta. Además, la función a la que apunta deberá ser polifórmica de tipo Dibujable.

```
class Simbolo {
  String nombre; //nombre del simbolo o variable
  short tipo; // tipo de token
  double val; //valor en caso de ser constante
  Dibujable dibujable; // Objeto polimorfico. lineas, circulo, etc.
  String metodo; // Nombre del metodo a ejecutar
  int defn; //
  Simbolo sig; //Apuntador al siguiente elemento de la lista
  Simbolo() { }
  Simbolo(String s, short t, double d) {
    nombre = s;
    tipo = t;
    val = d;
  }
   public Simbolo obtenSig() {
    return sig;
  }
   public void ponSig(Simbolo s) {
    sig = s;
  }
   public String obtenNombre() {
    return nombre;
  }
  public void setDibujable(Dibujable dibujable) {
    this.dibujable = dibujable;
```

```
  }
  public Dibujable getDibujable() {
    return dibujable;
  }
}
```

Para organizar la tabla de símbolos se tiene una lista simplemente enlazada de objetos de tipo Simbolo.

```
public class Tabla {
  Simbolo listaSimbolo;
  Tabla() {
    listaSimbolo = null;
  }
  Simbolo install(String s, short t, double d) {
    Simbolo simb = new Simbolo(s, t, d);
    simb.ponSig(listaSimbolo);
    listaSimbolo = simb;
    return simb;
  }
  Simbolo lookup(String s){
    for(Simbolo sp = listaSimbolo;
        sp != null; sp = sp.obtenSig())
      if((sp.obtenNombre()).equals(s))
        return sp;
    return null;
  }
}
```

3.6. Implementación de operaciones vectoriales

Se agregaron funciones para cada llamada a función tales como:

- `color()`.

```
void color() {
  //Necesario para colorear contornos con AWT
    Color colors[] = { Color.red, Color.green, Color.blue };
  double d1 = ((Double)pila.pop()).doubleValue();
  if (g != null)
    g.setColor(colors[(int) d1]);
  return;
}
```

- `line()`.

```
void line() {
  //Necesario para insertar lineas con AWT
  double y_2 = ((Double)pila.pop()).doubleValue();
  double x_2 = ((Double)pila.pop()).doubleValue();
  double y_1 = ((Double)pila.pop()).doubleValue();
  double x_1 = ((Double)pila.pop()).doubleValue();
  if (g != null) {
         (new Linea((int) x_1,
         (int) y_1,
         (int) x_2,
         (int) y_2))
       .dibujar(g);
  }
  return;
}
```

- `circle()`.

```
void circle() {
  //Necesario para insertar circulos con AWT
  double r = ((Double)pila.pop()).doubleValue();
  double yc = ((Double)pila.pop()).doubleValue();
  double xc = ((Double)pila.pop()).doubleValue();
  Circulo circulo = new Circulo((int) xc, (int) yc, (int) r);
  circulo.transladar((-1)*((int) r), (-1)*((int) r));
  if (g != null) {
        circulo.dibujar(g);
  }
  return;
}
```

- `drawImage()`.

```
void drawImage() {
  //Necesario para insertar imagenes con AWT
  Simbolo simbolo = (Simbolo)pila.pop();
  Dibujable dibujable = simbolo.getDibujable();
  if ((g != null) && (dibujable != null))
    dibujable.dibujar(g);
  return;
}
```

- `rectangulo()`.

```
void rectangulo() {
  //Necesario para insertar rectangulos con AWT
  double y_2 = ((Double)pila.pop()).doubleValue();
  double x_2 = ((Double)pila.pop()).doubleValue();
```

```
    double y_1 = ((Double)pila.pop()).doubleValue();
    double x_1 = ((Double)pila.pop()).doubleValue();
    if (g != null) {
      (new Rectangulo((int) x_1,
            (int) y_1,
            (int) x_2,
            (int) y_2))
      .dibujar(g);
    }
    return;
  }
```

- objectpush().

```
  void objectpush() {
    //Necesario para insertar objetos en pila
      Simbolo s;
      s = (Simbolo) prog.elementAt(pc);
      pc = pc + 1;
      pila.push(s);
      return;
    }
```

4. Resultados

A continuación se presentan los resultados obtenidos despues de haber implementado las gramaticales y las funciones que se agregaron para dibujar las figuras solicitadas

```
/**
 * Script para dibujar una casa y un carro simple.
 */
//Instrucciones para dibujar un carro
color 1;
rectangle 70 120 200 90;
rectangle 106 90 130 30;
color 2;
circle 156 230 50;
circle 236 230 50;

//Instrucciones para dibujar una casa
color 2;
rectangle 390 110 170 120;
rectangle 430 140 90 90;
color 3;
line 350 140 480 50;
line 480 50 600 140;
color 1;
circle 520 200 10;
```

La siguiente figura nos muestra el resultado de ejecutar las instrucciones mostradas previamente.

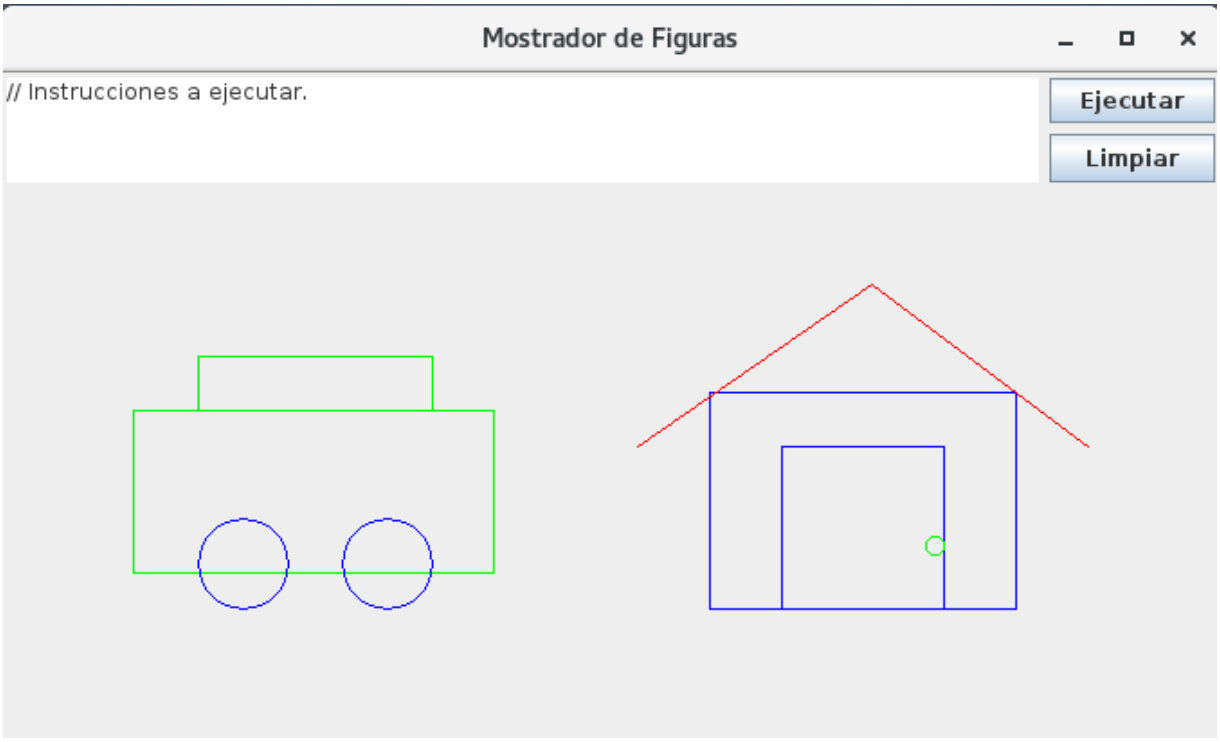

Figura 2: Operación de asignación de un vector a una variable de un sólo caractér.

5. Conclusiones

El desarrollo de esta práctica fue interesante debido al manejo de interfaces gráficas en combinación con YACC para interpretar instrucciones simples como dibujar figuras.
Al inicio me costó trabajo entender el funcionamiento usando máquina de pila pero después empezé a entenderlo mejor cuando debuggeo el código y de alguna forma me fuí guiando por las reglas semánticas.
YACC puede tener potencial para el desarrollo de otras aplicaciones que requieren que un usuario escriba instrucciones específicas, como búsquedas y se pueden procesar de mejor forma usando este tipo de programas.