Universidad de Chile
Facultad de Ciencias Físicas y Matemáticas
Departamento de Ciencias de la Computación

Algoritmos y Estructuras de Datos

CC3001

Tarea 4

Cálculo de derivadas usando árboles binarios

Autor: Ilana Mergudich Thal
E-Mail: ilanamergudich@gmail.com

Fecha de entrega: 5 de junio de 2017

Índice de Contenidos

1. Introducción

En esta tarea, el problema consiste en derivar una expresión polinomial mediante el uso de árboles binarios. Para ello, se ingresará una expresión en notación polaca inversa y se creará un árbol que represente la expresión. Luego se derivará el árbol, se simplificará, se agregarán los paréntesis necesarios y se leerá de manera que se entregue la expresión derivada en notación infija.

Para realizar esto, se crearon las clases **Arbol** y **Pila**. La clase árbol tiene un elemento raíz (que es un string), un hijo izquierdo y un hijo derecho (árboles). La Pila es una pila de árboles, con las funciones apilar, desapilar y estaVacía(). La Pila es utilizada para crear el árbol que representa la expresión.

2. Análisis del problema

El problema consiste en derivar expresiones polinomiales entregadas en notación polaca inversa. Inicialmente se debe crear un árbol que represente la expresión. Para ello, se crea una pila, se lee cada caracter y se realiza lo siguiente: Si el caracter es un número o una variable, se crea un árbol con el caracter en la raíz e hijos null y se apila, si el caracter es una operación (+, -, * o /), se crea un árbol con el caracter en la raiz, el último elemento de la pila como hijo derecho y el penúltimo como izquierdo. Al poner estos elementos como hijos, son desapilados. Se apila ese árbol.

En la siguiente figura se observa un ejemplo de este procedimiento:

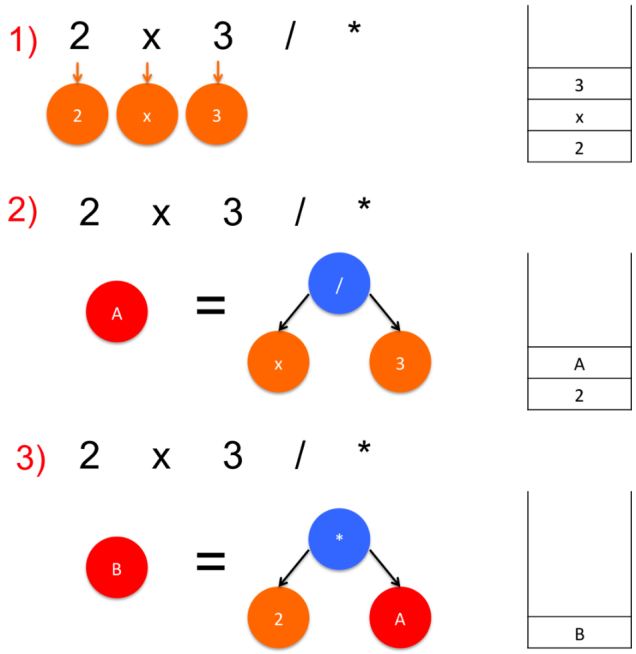

Figura 2.1: Ejemplo de árbol

Luego se debe derivar el árbol de manera recursiva utilizando las siguientes reglas:

- Si la raíz contiene un número o una variable que no es respecto de la cual se está derivando, se cambia la raíz por un 0 y ambos hiios null.
- Si la raíz contiene a la variable respecto de la cual se está derivando, se cambia la raíz por un 1 e hijos null.
- Si la raíz contiene una operación, se retorna el árbol correspondiente a la expresión que corresponda según las reglas de la derivación conocidas.

Finalmente se crea un programa que pide al usuario ingresar la expresión por derivar en notación polaca inversa y la variable respecto a la cual se desea derivar y entrega la expresión derivada en notación in-fijo.

Además se debe simplificar la operación para los siguientes casos:

- Las multiplicaciones por 0 dan 0.
- Las divisiones de un elemento por 1 dan el elemento.
- Las sumas de un elemento con 0 dan el elemento.
- Las multiplicaciones de un elemento con 1 dan el elemento.

Por otro lado, se deben agregar los paréntesis necesarios. Se consideraron necesarios en los siguientes casos:

- Si en la raíz hay una multiplicación o división y el algún hijo es una suma o una resta.
- Si en la raíz hay una resta y el hijo derecho es una suma o una resta.
- Si en la raíz hay una resta y el hijo derecho es una multiplicación.

3. Solución del problema

3.1. Clases

3.1.1. Arbol

Esta clase define un árbol según el *String* que está en su raíz ("raiz") y los árboles que están a la izquierda y derecha de la raíz ('ìzq" y "der" respectivamente). Además se define el constructor **Arbol**(String s, Arbol izq, Arbol der) que crea un árbol con el *String s* en la raíz y los árboles izquierdo y derecho indicados. En el Anexo 6.1 se muestra el código de esta clase.

3.1.2. Pila

Esta clase define una pila según el tamaño ("size") y último elemento de una lista enlazada. Para ello se creó una clase privada dentro de **Pila** llamada **Nodo**, donde se define un "elemento" del tipo **Arbol**) y un "siguiente" del tipo **Nodo**. Luego, "size" es un **int** que representa la cantidad de elementos de la lista y "ultimo" es un puntero al último **Nodo** de la lista. Además se definen diversas operaciones que se explicarán a continuación, todas ellas se rigen según el principio "LIFO" (last in first out).

3.1.2.1 Pila()

Esta operación es un constructor, se define el último como *null* y el tamaño como 0.

3.1.2.2 boolean estaVacia()

Esta operación verifica si la pila esta vacía, para ello se revisa el tamaño de la pila: si es 0, se retorna *true*, en caso contrario se retorna *false*.

3.1.2.3 void apilar(Arbol arb)

Esta operación agrega el árbol *arb* a la pila. Para ello, existen dos casos:

1. Si la pila esta vacía, se crea un nuevo **Nodo** con donde el "elemento" es el árbol *arb* y "siguiente" es *null*. Este **Nodo** será el "último".
2. Se crea un **Nodo** auxiliar, que es igual al "último" actual. Luego se modifica el último de manera que el "elemento" sea el árbol *arb* y "siguiente" sea el **Nodo** auxiliar.

Sin importar el caso, se suma uno al tamaño.

3.1.2.4 int[ ] desapilar()

Esta operación quita el último elemento de la pila y retorna ese árbol. Si la pila esta vacía, simplemente retorna *null*. En caso contrario se define un árbol x como el último elemento de la pila. Nuevamente tenemos dos casos:

1. Si la pila tenía sólo un elemento (size = 1), se define "último" como *null*.
2. Si la pila tenía más de un elemento, el nuevo "último" es el siguiente del "último", es decir, el penúltimo.

En ambos casos se resta uno al tamaño de la pila.

En el Anexo 6.2 se encuentra el código de esta clase.

3.1.3. Derivar

En esta clase se definen todos los métodos necesarios para derivar una expresión en notación polaca inversa. En el Anexo 6.3 se encuentra el código completo de esta clase.

3.1.3.1 Pila armar(String exp)

Este método recibe una expresión matemática polinomial en notación polaca inversa y retorna una Pila con un sólo elemento, que es el árbol que representa esta expresión. Para cada operación, se crea un nodo donde los hijos son los elementos que se operan. Para armar este árbol se siguen los siguientes pasos:

1. Se crea una pila inicialmente vacía.
2. Se separa el *String* en cada espacio, creándose una lista con cada símbolo del *String*. Luego se itera sobre esta lista.
3. Si el símbolo no es una operación (+, -, *, /), se crea un árbol con el símbolo en la raíz e hijos *null* y se apila.
4. Si el símbolo es una operación, se desapilan dos elementos de la pila y se crea un nuevo árbol con el símbolo en la raíz, el primer elemento en ser desapilado a la derecha y el segundo a la izquierda. Se apila este árbol.
5. Se retorna la pila resultante, que contiene un sólo árbol, que representa la expresión completa.

En la figura 2.1 se observa un ejemplo de este procedimiento para la expresión 2 x 3 / *.

3.1.3.2 Arbol simplificar(Arbol arb)

Este método recibe un árbol y simplifica las siguientes operaciones:

- Las multiplicaciones de un término por 0 dan 0.
- Las multiplicaciones de un término por 1 dan el término.
- Las sumas de un término con 0 dan el término.
- Las restas de 0 a un término dan el término.
- Las divisiones de 0 por un término dan 0.
- Las divisiones de un término por 1 dan el término.

Para ello, se revisan los siguientes casos:

1. Si el árbol es null, se retorna null.
2. Si en la raíz hay una multiplicación, existen 3 casos:

- Si alguno de los hijos es 0, se retorna un árbol con un 0 en la raíz y ambos hijos null.
- Si uno de los hijos es 1, se retorna el otro hijo simplificado.
- En cualquier otro caso, se retorna un árbol con la raíz actual y ambos hijos simplificados.

A continuación se muestra el código que evalúa estos casos:

```
if (arb.raiz.charAt(0) == '*') {
      if ((arb.der != null && arb.der.raiz.charAt(0) == '0') || (arb.izq != null &&
          arb.izq.raiz.charAt(0) == '0')) {
          return new Arbol("0", null, null);
      }
      else if (arb.der != null && arb.der.raiz.charAt(0) == '1') {
          return simplificar(arb.izq);
      }
      else if (arb.izq != null && arb.izq.raiz.charAt(0) == '1') {
          return simplificar(arb.der);
      }
      else {
          return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
      }
}
```

3. Si en la raíz hay una división, nuevamente existen 3 casos:

   - Si el hijo izquierdo es 0 (es decir, se tiene algo de la forma 0/x), se retorna un árbol con un 0 en la raíz y ambos hijos null.
   - Si el hijo derecho es 1 (es decir, se tiene algo de la forma x/1) se retorna el hijo izquierdo simplificado.
   - En cualquier otro caso se retorna un árbol con la raíz actual y ambos hijos simplificados.

   A continuación se muestra el código que evalúa estos casos:

```
else if (arb.raiz.charAt(0) == '/') {
      if (arb.izq != null && arb.izq.raiz.charAt(0) == '0') {
          return new Arbol("0", null, null);
      }
      else if (arb.der != null && arb.der.raiz.charAt(0) == '1') {
          return simplificar(arb.izq);
      }
      else {
          return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
      }
}
```

4. Si en la raíz hay una suma, existen 3 casos:

   - Si ambos hijos son 0, se retorna un árbol con un 0 en la raíz y ambos hijos null.
   - Si uno de los dos hijos es 0, se retorna el otro hijo simplificado.
   - En cualquier otro caso se retorna un árbol con la raíz actual y ambos hijos simplificados.

A continuación se muestra el código que evalúa estos casos:

```
else if (arb.raiz.charAt(0) == '+') {
      if ((arb.izq != null && arb.izq.raiz.charAt(0) == '0') && (arb.der != null &&
          arb.der.raiz.charAt(0) == '0')) {
          return new Arbol("0", null, null);
      }
      else if (arb.izq != null && arb.izq.raiz.charAt(0) == '0') {
          return simplificar(arb.der);
      }
      else if (arb.der != null && arb.der.raiz.charAt(0) == '0') {
          return simplificar(arb.izq);
      }
      else {
          return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
      }
}
```

5. Si en la raíz hay una resta, existen sólo 2 casos:
   - Si el hijo derecho es un 0, se retorna el árbol izquierdo simplificado.
   - En cualquier otro caso se retorna un árbol con la raíz actual y ambos hijos simplificados.

   A continuación se muestra el código que evalúa estos casos:

```
else if (arb.raiz.charAt(0) == '-') {
      if (arb.der != null && arb.der.raiz.charAt(0) == '0') {
          return simplificar(arb.izq);
      }
      else {
          return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
      }
}
```

Se puede observar que este método funciona de manera recursiva, logrando retornar la simplificación de la expresión entregada (representada como un árbol).

3.1.3.3 Arbol derivar(Arbol arb, String var)

Este método recibe un árbol que representa una expresión matemática y retorna el árbol que representa su derivada. Para ello, se utilizan las siguientes reglas de derivación:

- La derivada de una constante es 0.
- La derivada de la variable es 1.
- La derivada de una suma o resta las expresiones f y g es $f' \pm g'$.
- La derivada de una multiplicación las expresiones f y g es $f' * g + f * g'$.
- La derivada de una división de las expresiones f y g es $(f' * g - f * g')/(g * g)$

Al igual que el método anterior, este funciona de manera recursiva y se evalúan los siguientes casos:

1. Si la raíz es la variable, se cambia la raíz por un 1.
2. Si la raíz es un + o un -, se reemplazan los hijos por ellos mismos derivados y simplificados.
3. Si la raíz es *, la figura siguiente representa la modificación del árbol:

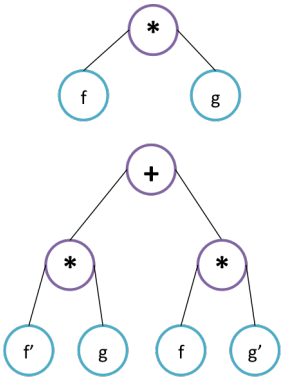

Figura 3.1: Derivada de la multiplicación

Se cambia la raíz por una suma, se derivan y simplifican ambos hijos. Luego se crean árboles con una multiplicación en la raíz, un hijo sin derivar y el otro con y se simplifican. Finalmente se definen estos nuevos árboles como hijos de la suma. Se retorna la simplificación de este árbol.

4. Si la raíz es /, la siguiente figura representa la modificación del árbol:

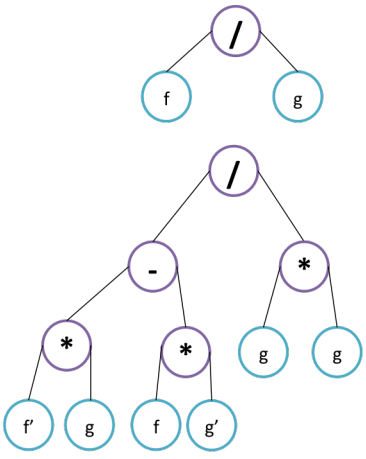

Figura 3.2: Derivada de la división

Se mantiene la raíz, se derivan y simplifican ambos hijos. Se cambia el hijo izquierdo por un menos y el derecho por una multiplicación. Los hijos del menos son dos multiplicaciones, que

a su vez, tienen un hijo original derivado y el otro sin derivar. Los subhijos del hijo derecho son el hijo derecho original. Se retorna la simplificación de este árbol.

3.1.3.4 void parDer(Arbol a)

Este método agrega un ")" lo más a la derecha posible del árbol. Para ello, si el hijo derecho es null, agrega el parentesis, sino revisa si el hijo derecho del hijo derecho es null recursivamente.

3.1.3.5 void parIzq(Arbol a)

Este método agrega un "(" lo más a la izquierda posible del árbol. Para ello, si el hijo izquierdo es null, agrega el parentesis, sino revisa si el hijo izquierdo del hijo izquierdo es null recursivamente.

A continuación se muestran ejemplos de parDer() y parIzq():

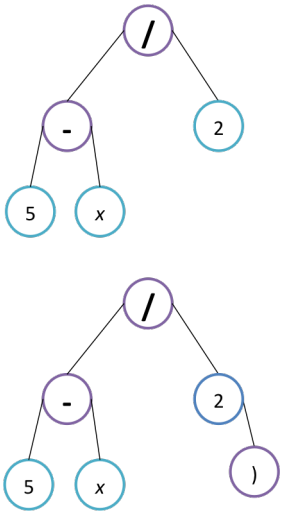

Figura 3.3: Ejemplo parDer()

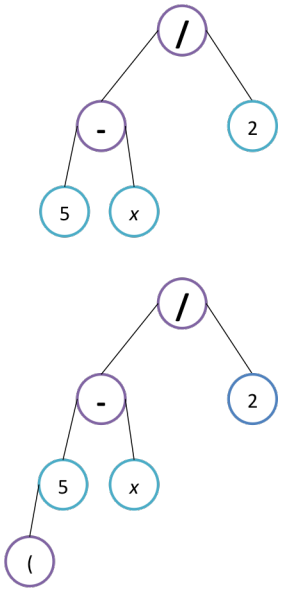

Figura 3.4: Ejemplo parIzq()

3.1.3.6 void parentesis(Arbol arb)

Este método agrega paréntesis a una expresión entregada como un árbol en los siguientes casos:

1. Si en la raíz hay una multiplicación o división y el algún hijo es una suma o una resta. Por ejemplo $3 * (y + 8)$ o $(5 - x)/2$.
2. Si en la raíz hay una resta y el hijo derecho es una suma o una resta. Por ejemplo $2 - (4 + x)$ o $5 - (y - 7)$.

Para ello, se evalúan los casos mencionados y se aplica parIzq() y parDer() a los árboles hijos que tienen en su raíz a la expresión que requiere el paréntesis. En la siguiente figura se observa un ejemplo.

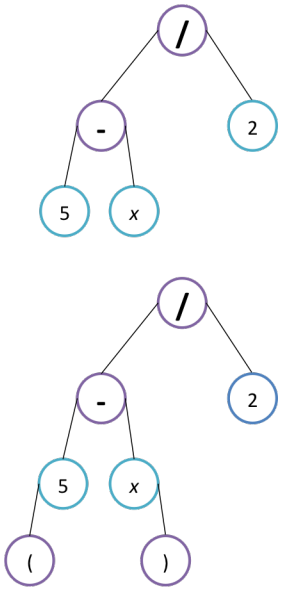

Figura 3.5: Ejemplo de parentesis(arb a)

Cabe mencionar que existe un caso adicional en que se requiere paréntesis: cuando la raíz es una división y el hijo derecho es una multiplicación. No fue posible agregar este caso en este método, por lo que se solucionó directamente en el método "leer" que se explicará a continuación.

3.1.3.7 void leer(Arbol arb)

Este método recibe un árbol e imprime la expresión que está representada. Además, como se mencionó anteriormente, imprime paréntesis en caso de que haya una raíz con una división y su hijo derecho sea una multiplicación.

Para ello, se pone en los siguientes casos y opera de forma recursiva:

1. Si ambos hijos son null, simplemente se imprime el string de la raíz.

A continuación se muestra el código para este caso:

```
if(arb.izq == null && arb.der == null) {
                System.out.print(arb.raiz);
                System.out.print(" ");
}
```

2. Si sólo el árbol derecho es null, se lee el hijo izquierdo de manera recursiva, luego de imprime la raíz seguida de un espacio.

   A continuación se muestra el código para este caso:

```
else if(arb.izq != null && arb.der == null) {
                leer(arb.izq);
                System.out.print(arb.raiz);
                System.out.print(" ");
}
```

3. Si sólo el hijo izquierdo es null, se revisa si se cumple la condición mencionada anteriormente para poner paréntesis. Si se cumple, se imprime la raíz y luego se lee el hijo derecho entre paréntesis. Si no se cumple, simplemente se imprime la raíz y luego el hijo derecho de manera recursiva.

   A continuación se muestra el código para este caso:

```
else if(arb.izq == null) {
                if(arb.raiz.charAt(0) == '/' && (arb.der.raiz.charAt(0) == '/' ||
                    arb.der.raiz.charAt(0) == '*')){
                    System.out.print(arb.raiz);
                    System.out.print(" ( ");
                    leer(arb.der);
                    System.out.print(") ");
                }
                else{
                    System.out.print(arb.raiz);
                    System.out.print(" ");
                    leer(arb.der);
                }

}
```

4. Si ninguno de los hijos es null, se revisa si el árbol cumple la condición de paréntesis. En caso de hacerlo, se lee el hijo izquierdo, se imprime la raíz y luego se lee el hijo derecho entre paréntesis. En caso contrario, la única diferencia es que no se ponen los paréntesis para el hijo derecho.

   A continuación se muestra el código para este caso:

```
else {
                if(arb.raiz.charAt(0) == '/' && (arb.der.raiz.charAt(0) == '/' ||
                    arb.der.raiz.charAt(0) == '*')){
                    leer(arb.izq);
```

```
4                   System.out.print(arb.raiz);
5                   System.out.print(" ( ");
6                   leer(arb.der);
7                   System.out.print(") ");
8               }
9
10              else{
11                  leer(arb.izq);
12                  System.out.print(arb.raiz);
13                  System.out.print(" ");
14                  leer(arb.der);
15              }
16
17  }
```

3.1.3.8 void main(String args[])

Esta función permite utilizar todas las anteriores para resolver el problema propuesto en la tarea. Inicialmente pide al usuario ingresa la expresión que desea derivar y luego con respecto a qué variable. Se guardan los datos ingresados. Luego, mediante el uso de armar(), se crea una Pila que contiene al árbol que representa la expresión entregada. Se ponen paréntesis a la expresión original usando parentesis() y se guarda. Se imprime esta expresión usando leer(). Después de deriva el árbol, se le ponen parentesis y se imprime la expresión de la derivada utilizando leer().

4. Modo de uso

Para utilizar este programa, se deben guardar los archivos **Pila.java**, **Arbol.java** y **Derivar.java** en una misma carpeta. A continuación, se deben compilar las tres clases y luego ejecutar **Derivar.java**. Al hacer esto, se solicitará al usuario ingresar la expresión que desea derivar y a continuación la variable respecto a cual desea derivar. Es fundamental que la expresión sea escrita en notación polaca inversa y que exista un espacio entre cada caracter. Entonces el programa imprimirá la expresión ingresada en notación "normal" y luego su derivada, simplificada según lo explicado anteriormente.

5. Resultados

5.1. Ejemplos de entradas y salidas

1. Escriba la expresion que desea derivar 2 x 3 / * y x - +
   ¿Con respecto a que variable desea derivar? x
   La expresion corresponde a:
   2 * x / 3 + y - x
   Su derivada con respecto a x es:
   2 * 3 / (3 * 3) + 0 - 1

2. Escriba la expresion que desea derivar 2 x 3 / * y x - +
   ¿Con respecto a que variable desea derivar? y
   La expresion corresponde a:
   2 * x / 3 + y - x
   Su derivada con respecto a y es:
   1

3. Escriba la expresion que desea derivar 2 y + 3 * 5 /
   ¿Con respecto a que variable desea derivar? y
   La expresion corresponde a:
   (2 + y) * 3 / 5
   Su derivada con respecto a y es:
   3 * 5 / (5 * 5)

4. Escriba la expresion que desea derivar 2 y + 3 * 5 /
   ¿Con respecto a que variable desea derivar? x
   La expresion corresponde a:
   (2 + y) * 3 / 5
   Su derivada con respecto a x es:
   0

5. Escriba la expresion que desea derivar x y / 5 /
   ¿Con respecto a que variable desea derivar? x
   La expresion corresponde a:
   x / y / 5 Su derivada con respecto a x es:
   y / (y * y) * 5 / (5 * 5)

6. Escriba la expresion que desea derivar x y / 5 /
   ¿Con respecto a que variable desea derivar? y
   La expresion corresponde a:
   x / y / 5 Su derivada con respecto a y es:
   (0 - x) / (y * y) * 5 / (5 * 5)

7. Escriba la expresion que desea derivar 3 2 + x y + /
   ¿Con respecto a que variable desea derivar? x
   La expresion corresponde a:
   (3 + 2) / (x + y)

Su derivada con respecto a x es:
(0 - ((3 + 2))) / (((x + y)) * ((x + y)))

8. Escriba la expresion que desea derivar 3 2 + x y + /
¿Con respecto a que variable desea derivar? y
La expresion corresponde a:
(3 + 2) / (x + y)
Su derivada con respecto a y es:
(0 - ((3 + 2))) / (((x + y)) * ((x + y)))

5.2. Conclusión

A grandes rasgos, se resolvió de manera satisfactoria el problema. Fue posible derivar la expresión, simplificarla y escribirla en notación in-fijo. No obstante, como se observa en los casos 7 y 8, para algunas expresiones, se duplican los paréntesis necesarios. Por otro lado, para un futuro desarrollo de este programa, se recomienda unificar la solución al problema de los paréntesis ya que la función parentesis() hace algunos casos y la función leer() otros, por lo que se pueden producir confusiones.

A pesar de esto, se resolvió el problema propuesto en la tarea.

6. Anexos

6.1. Arbol

```
public class Arbol {

    String raiz;
    Arbol izq;
    Arbol der;

    public Arbol(String s, Arbol izq, Arbol der) {
        this.raiz = s;
        this.izq = izq;
        this.der = der;
    }


}
```

6.2. Pila

```
public class Pila {

    private Nodo ultimo;
    private int size;

    private class Nodo{
        Arbol elemento;
        Nodo siguiente;

    }

    public Pila() {
        ultimo = null;
        size = 0; //inicialmente la pila está vacía

    }


    public boolean estaVacia() {
        return size == 0;

    }


    public void apilar(Arbol a) {
        if(estaVacia()){
            ultimo = new Nodo();
            ultimo.elemento = a;
            ultimo.siguiente = null;

```

```
        }
        else{
            Nodo aux = ultimo;
            ultimo = new Nodo();
            ultimo.elemento = a;
            ultimo.siguiente = aux;
        }

        size = size+1;
    }

    public Arbol desapilar() {
        if (!estaVacia()) {
            Arbol x = ultimo.elemento;
            if(size == 1){
                ultimo = null;
            }
            else{
                ultimo = ultimo.siguiente;
            }

            size = size -1;
            return x;

        }
        else{
            return null;
        }
    }

}
```

6.3. Derivar

```
import java.util.*;

public class Derivar {

    public static Pila armar(String exp){
        Pila pila = new Pila();
        for (String simbolo : exp.split(" ")){
            char sim = simbolo.charAt(0);

            if(sim == '*' || sim == '/' || sim == '+' || sim == '-'){
                Arbol der = pila.desapilar();
                Arbol izq = pila.desapilar();
                Arbol a = new Arbol(simbolo, izq, der);

                pila.apilar(a);

            }

```

```
19              else{
20                  Arbol a = new Arbol(simbolo, null, null);
21                  pila.apilar(a);
22              }
23          }
24
25          return pila;
26      }
27
28      public static Arbol simplificar(Arbol arb){
29          if(arb != null) {
30              if (arb.raiz.charAt(0) == '*') {
31                  if ((arb.der != null && arb.der.raiz.charAt(0) == '0') || (arb.izq != null
                        && arb.izq.raiz.charAt(0) == '0')) {
32                      return new Arbol("0", null, null);
33                  }
34                  else if (arb.der != null && arb.der.raiz.charAt(0) == '1') {
35                      return simplificar(arb.izq);
36
37                  }
38                  else if (arb.izq != null && arb.izq.raiz.charAt(0) == '1') {
39                      return simplificar(arb.der);
40
41                  }
42                  else {
43                      return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
44
45                  }
46              }
47              else if (arb.raiz.charAt(0) == '/') {
48                  if (arb.izq != null && arb.izq.raiz.charAt(0) == '0') {
49                      return new Arbol("0", null, null);
50                  }
51                  else if (arb.der != null && arb.der.raiz.charAt(0) == '1') {
52                      return simplificar(arb.izq);
53                  }
54                  else {
55                      return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
56
57                  }
58              }
59              else if (arb.raiz.charAt(0) == '+') {
60
61                  if ((arb.izq != null && arb.izq.raiz.charAt(0) == '0') && (arb.der != null
                        && arb.der.raiz.charAt(0) == '0')) {
62                      return new Arbol("0", null, null);
63                  }
64                  else if (arb.izq != null && arb.izq.raiz.charAt(0) == '0') {
65                      return simplificar(arb.der);
66                  }
67                  else if (arb.der != null && arb.der.raiz.charAt(0) == '0') {
68                      return simplificar(arb.izq);
```

```
69              }
70              else {
71                  return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
72
73              }
74
75          }
76          else if (arb.raiz.charAt(0) == '-') {
77              if (arb.der != null && arb.der.raiz.charAt(0) == '0') {
78                  return simplificar(arb.izq);
79
80              }
81              else {
82                  return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
83
84              }
85          }
86          else{
87              return new Arbol(arb.raiz, simplificar(arb.izq), simplificar(arb.der));
88          }
89
90      }
91      return null;
92  }
93
94
95
96  public static Arbol derivar(Arbol arb, String var){
97
98      char e = arb.raiz.charAt(0);
99      char v = var.charAt(0);
100
101
102     if(e == v){
103         arb.raiz = "1";
104     }
105
106     else if(e == '+' || e == '-'){
107         Arbol arbiz = new Arbol(arb.izq.raiz, arb.izq.izq, arb.izq.der);
108         Arbol arbde = new Arbol(arb.der.raiz, arb.der.izq, arb.der.der);
109
110         Arbol a = derivar(arbiz, var);
111         Arbol b = derivar(arbde, var);
112
113         arb.izq = simplificar(a);
114         arb.der = simplificar(b);
115
116
117
118     }
119
120     else if(e == '*'){
```

```
121
122             arb.raiz = "+";
123
124             Arbol arbiz11 = new Arbol(arb.izq.raiz, arb.izq.izq, arb.izq.der);
125             Arbol arbde11 = new Arbol(arb.der.raiz, arb.der.izq, arb.der.der);
126
127             Arbol arbiz = simplificar(arbiz11);
128             Arbol arbde = simplificar(arbde11);
129
130             Arbol arbiz1 = simplificar(arbiz11);
131             Arbol arbde1 = simplificar(arbde11);
132
133             Arbol a1 = derivar(arbiz, var);
134             Arbol b1 = derivar(arbde, var);
135
136             Arbol a = simplificar(a1);
137             Arbol b = simplificar(b1);
138
139             Arbol izq = new Arbol("*", a, arbde1);
140             Arbol der = new Arbol("*", arbiz1, b);
141
142             arb.izq = simplificar(izq);
143             arb.der = simplificar(der);
144
145
146         }
147
148         else if(e == '/') {
149             Arbol arbiz11 = new Arbol(arb.izq.raiz, arb.izq.izq, arb.izq.der);
150             Arbol arbde11 = new Arbol(arb.der.raiz, arb.der.izq, arb.der.der);
151
152             Arbol arbiz = simplificar(arbiz11);
153             Arbol arbde = simplificar(arbde11);
154
155             Arbol arbiz1 = simplificar(arbiz11);
156             Arbol arbde1 = simplificar(arbde11);
157
158             Arbol a1 = derivar(arbiz, var);
159             Arbol b1 = derivar(arbde, var);
160
161             Arbol a = simplificar(a1);
162             Arbol b = simplificar(b1);
163
164             Arbol izqq1 = new Arbol("*", a, arbde1);
165             Arbol izqd1 = new Arbol ("*", arbiz1, b);
166
167             Arbol izqq = simplificar(izqq1);
168             Arbol izqd = simplificar(izqd1);
169
170             Arbol izq = new Arbol("-", izqq, izqd);
171             Arbol der = new Arbol("*", arbde1, arbde1);
172
```

```
173             arb.izq = simplificar(izq);
174             arb.der = simplificar(der);
175 
176         }
177 
178         else{
179             arb.raiz = "0";
180         }
181 
182         return simplificar(arb);
183     }
184 
185     public static void parIzq(Arbol a){
186         if(a.izq == null){
187             a.izq = new Arbol("(", null, null);
188         }
189         else{
190             parIzq(a.izq);
191         }
192 
193     }
194 
195     public static void parDer(Arbol a){
196         if(a.der == null){
197             a.der = new Arbol(")", null, null);
198         }
199         else{
200             parDer(a.der);
201         }
202 
203     }
204 
205     public static void parentesis(Arbol arb){
206         if(arb.raiz.charAt(0) == '*' || arb.raiz.charAt(0) == '/') {
207             if (arb.izq.raiz.charAt(0) == '+' || arb.izq.raiz.charAt(0) == '-') {
208                 parIzq(arb.izq);
209                 parDer(arb.izq);
210             }
211 
212             if (arb.der.raiz.charAt(0) == '+' || arb.der.raiz.charAt(0) == '-') {
213                 parIzq(arb.der);
214                 parDer(arb.der);
215             }
216 
217         }
218 
219         if(arb.raiz.charAt(0) == '-' && arb.der != null && (arb.der.raiz.charAt(0) == '+'
                || arb.der.raiz.charAt(0) == '-')){
220             parIzq(arb.der);
221             parDer(arb.der);
222         }
223 
```

```
224         if(arb.izq != null) parentesis(arb.izq);
225         if(arb.der != null) parentesis(arb.der);
226 
227     }
228 
229     public static void leer(Arbol arb) {
230         if (arb != null) {
231 
232             if(arb.izq == null && arb.der == null) {
233                 System.out.print(arb.raiz);
234                 System.out.print(" ");
235             }
236 
237             else if(arb.izq != null && arb.der == null) {
238                 leer(arb.izq);
239                 System.out.print(arb.raiz);
240                 System.out.print(" ");
241             }
242 
243             else if(arb.izq == null) {
244                 if(arb.raiz.charAt(0) == '/' && (arb.der.raiz.charAt(0) == '/' ||
                        arb.der.raiz.charAt(0) == '*')){
245                     System.out.print(arb.raiz);
246                     System.out.print(" ( ");
247                     leer(arb.der);
248                     System.out.print(") ");
249                 }
250                 else{
251                     System.out.print(arb.raiz);
252                     System.out.print(" ");
253                     leer(arb.der);
254                 }
255 
256             }
257 
258             else {
259                 if(arb.raiz.charAt(0) == '/' && (arb.der.raiz.charAt(0) == '/' ||
                        arb.der.raiz.charAt(0) == '*')){
260                     leer(arb.izq);
261                     System.out.print(arb.raiz);
262                     System.out.print(" ( ");
263                     leer(arb.der);
264                     System.out.print(") ");
265                 }
266 
267                 else{
268                     leer(arb.izq);
269                     System.out.print(arb.raiz);
270                     System.out.print(" ");
271                     leer(arb.der);
272                 }
273 
```

```
            }
        }
    }



    public static void main(String[] args){
        Scanner s = new Scanner(System.in);
        System.out.print("Escriba la expresion que desea derivar ");
        String exp = s.nextLine();
        System.out.print("Con respecto a que variable desea derivar? ");
        String var = s.nextLine();

        Pila pila = armar(exp);

        Arbol inicial = pila.desapilar();


        Arbol inicial1 = inicial;

        parentesis(inicial1);

        System.out.println("La expresion corresponde a:");
        leer(inicial1);
        System.out.println("");

        Arbol derivado = derivar(inicial, var);

        parentesis(derivado);

        System.out.print("Su derivada con respecto a ");
        System.out.print(var);
        System.out.println(" es:");
        leer(derivado);

    }
}
```