****

[*www.utpl.edu.ec*](http://www.utpl.edu.ec)

*Sistemas Informáticos y Computación*

***ESTRUCTURAS DE DATOS***

******

*Operación ABB con interfaz gráfica en Java*

Tabla de Contenidos

[Operación ABB con interfaz gráfica en Java 1](#_Toc46867508)

[*Introducción* 1](#_Toc46867509)

[La clase Árbol 1](#_Toc46867510)

[*Problema* de la clase Árbol 1](#_Toc46867511)

[*Análisis* para las clases empleado y prueba 1](#_Toc46867512)

[La clase Intermediario 2](#_Toc46867513)

[*Problema* de la clase Intermediario 2](#_Toc46867514)

[*Análisis* para las clases Intermediario 2](#_Toc46867515)

[La clase *Nodo* 3](#_Toc46867516)

[*Problema* Nodo 3](#_Toc46867517)

[*Análisis* Nodo 3](#_Toc46867518)

[*Codificación y documentación* Nodo 4](#_Toc46867519)

[La clase *InterfazArbol* 4](#_Toc46867520)

[*Problema* *InterfazArbol* 4](#_Toc46867521)

[*Análisis* *InterfazArbol* 5](#_Toc46867522)

[*Codificación y documentación* InterfazArbol 5](#_Toc46867523)

[La clase *Graficar* 8](#_Toc46867524)

[*Problema* Graficar 8](#_Toc46867525)

[*Análisis* Graficar 8](#_Toc46867526)

[*Codificación y documentación* Graficar 8](#_Toc46867527)

[La clase *Lienzo* 9](#_Toc46867528)

[*Problema* Lienzo 9](#_Toc46867529)

[*Análisis* Lienzo 9](#_Toc46867530)

[*Codificación y documentación* Lienzo 9](#_Toc46867531)

[Bibliography 11](#_Toc46867532)

## **Operación ABB con interfaz gráfica en Java**

## ***Introducción***

Los árboles binarios de búsqueda son las estructuras por preferencia para una organización jerárquica, ya que en si su propia estructura facilita la organización y la lectura por medio de métodos recursivos los datos que este contiene, al ser una estructura de datos, puede almacenar cualquier tipo de datos.

Es por eso por lo que es primordial poder aprender, saber y entender acerca de los arboles binarios en la materia de Estructuras de Datos.

# **La clase Árbol**

## ***Problema* de la clase Árbol**

La clase árbol es la clase primordial para el ejercicio, debido a que dentro de esta se van a encontrar los métodos principales de operación como es el método de eliminar datos, insertar datos o buscar si existe algún dato. Además también de los métodos con recorridos para poder obtener las cadenas con los recorridos.

## ***Análisis* para las clases empleado y prueba**

Cree una clase llamada **Arbol**, que incluya la raíz del árbol que será de donde se conectarán con los demás nodos que se agreguen al árbol. Su clase debe tener un ***constructor*** que inicialice la raíz en nulo. Se debe proporcionar métodos para las operaciones ABB básicas y para la creación de los recorridos, en estos métodos se van a usar variables con modificador private para poder acceder a ellos solo desde los métodos de esta clase.

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASES** | **ATRIBUTOS** | **MÉTODOS** | **REQUERIMIENTOS** |
| **Arbol** | raiz  num\_nodos  alt  preorden  inorden  postorden | constructor()  insertar()  eliminar()  recorrido()  Preorden()  Postorden()  Inorden() | Inicializar la raíz del árbol en nulo en el constructor, para esto no es necesario que reciba ningún parámetro. |

|  |  |
| --- | --- |
|  |  |
| **1** | class Arbol {  private Nodo raiz;  int num\_nodos;  int alt;  String preorden = "Recorrido PreOrden\n";  String inorden = "Recorrido EnOrden\n";  String postorden = "Recorrido PostOrden\n";  public Arbol() { // el constructor inicializa un Arbol vacío de enteros  raiz = null;  }  public void insertarNodo(int valorInsertar) {  if (raiz == null)  raiz = new Nodo(valorInsertar);  else  raiz.insertar(valorInsertar);  }    private void altura(Nodo aux, int nivel) {    if (aux != null) {  altura(aux.getIzq(), nivel + 1);  alt = nivel;  altura(aux.getDer(), nivel + 1);  }  }  //Devuleve la altura del arbol  public int getAltura() {  altura(raiz, 1);  return alt;  }    public String recorridoPostorden() {  postOrden(raiz);  return this.postorden;  }    private void postOrden(Nodo nodoAr) {  if (nodoAr == null)  return;  postOrden(nodoAr.nodoIzq);  postOrden(nodoAr.nodoDer);  this.postorden = String.format("%s%d ", this.postorden, nodoAr.datos);  }    public String recorridoEnorden() {  enOrden(raiz);  return this.inorden;  }    private void enOrden(Nodo nodoAr) {  if (nodoAr == null)  return;    enOrden(nodoAr.nodoIzq);  this.inorden = String.format("%s%d ", this.inorden, nodoAr.datos);  enOrden(nodoAr.nodoDer);  }    public String recorridoPreorden() {  preOrden(raiz);  return this.preorden;  }    private void preOrden(Nodo nodoAr) {  if (nodoAr == null)  return;  this.preorden = String.format("%s%d ", this.preorden, nodoAr.datos);  preOrden(nodoAr.nodoIzq);  preOrden(nodoAr.nodoDer);  }    public boolean eliminar(int dato) {  Nodo aux = raiz;  while (aux != null) {  if (dato == aux.getDatos()) {  if(aux.nodoDer != null || aux.nodoIzq != null){  return false;  }  aux = null;  return true;  } else if (dato > aux.getDatos()) {  aux = aux.getDer();  } else {  aux = aux.getIzq();  }  }  return false;  }    } |
|  |  |

**Fig. 2** | Codificación de la clase ***Arbol***

# **La clase Intermediario**

## ***Problema* de la clase Intermediario**

La clase intermediario debe servir como un conducto de comunicación entre dos clases, en este caso esas clases son Arbol e InterfazArbol. Es necesaria esta clase debido a que con este conducto de comunicación se reduce el amontonamiento de métodos, volviendo más legible el código además de brindarle más control al programador sobre este, ya que se puede cambiar las respuestas de los métodos, o a partir de estas dar otras respuestas e interrumpir los datos para evitar confusiones.

## ***Análisis* para las clases Intermediario**

Se crea la clase Intermediario para poder administrar las respuestas de los métodos que están en la clase Arbol, en esta clase se implementa la clase árbol, ya no es necesaria implementar en la clase InterfazArbol debido a que el único comunicador va a ser la clase Intermediario.

De esta forma se puede cortar o devolver ciertos valores en caso de recibir x o y datos de los métodos de Arbol, o incluso se pueden aplicar una sentencia try{} para poder discriminar los errores y evitar que la interfaz caiga.

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASES** | **ATRIBUTOS** | **MÉTODOS** | **REQUERIMIENTOS** |
| **Intermediario** | ABB | constructor()  insertar()  recorrido  buscar  borra()  getDibujo() | Crear un atributo tipo objeto Arbol. |

|  |  |
| --- | --- |
|  |  |
| **1** | class Intermediario {  Arbol ABB = new Arbol();  public Intermediario() {  }  public boolean insertar(Integer dato) {  return (this.ABB.agregar(dato));  }  private String recorrido(LinkedList it, String msg) {  int i = 0;  String r = msg + "\n";  while (i < it.size()) {  r += "\t" + it.get(i).toString() + "";  i++;  }  return (r);  }  //Metodo para buscar dato en el nodo  public String buscar(Integer dato) {  boolean siEsta = this.ABB.existe(dato);  String r = "El dato:" + dato.toString() + "\n";  r += siEsta ? "Si se encuentra en el arbol" : "No se encuentra en el arbol";  return (r);  }    public String borrar(int dato)  {  Integer x = this.ABB.borrar(dato);  if (x == 0)  return ("No existe el dato en el arbol");  return ("Borrado el dato: " + x.toString());  }    public JPanel getDibujo() {  return this.ABB.getdibujo();  }  } |
|  |  |

**Fig. 2** | Codificación de la clase ***Arbol***

|  |
| --- |
| run:  BUILD SUCCESSFUL (total time: 60 minutes 9 seconds) |
|  |
|  |

# **La clase *Nodo***

## ***Problema* Nodo**

El nodo es el componente principal y esencial de los árboles, ya que con estos nodos se crean las ramas y las hojas de los árboles y le dan su estructura que lo caracteriza. Por lo tanto al ser un ABB solamente debe crear un nodo con dos nodos, izquierdo y derecho.

## ***Análisis* Nodo**

La clase debe mantener las funciones básicas de una programación orientada a objetos. Por ejemplo los getters o setters para todos sus atributos, ya que con esto se puede acceder de manera más fácil y rápida directamente desde los objetos que se creen

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASES** | **ATRIBUTOS** | **MÉTODOS** | **REQUERIMIENTOS** |
| **Nodo** | Int datos  nodoIzq  nodoDer | Constructor()  Getters  Setters  Insertar() | Se debe aquí realizar el verdaero proceso de inserción al árbol binario.  El constructor empieza inicializando los valores de los nodos hijos a nulos, solamente tiene que existir el valor del nodo. |

## ***Codificación y documentación* Nodo**

|  |  |
| --- | --- |
| **17** | class Nodo {  int datos;  Nodo nodoIzq;  Nodo nodoDer;    public Nodo(int datosNodo) {  datos = datosNodo;  nodoIzq = nodoDer = null;  }  public int getDatos() {  return datos;  }  public void setDato(int dato) {  this.datos = dato;  }  public Nodo getIzq() {  return nodoIzq;  }  public void setIzq(Nodo izq) {  this.nodoIzq = izq;  }  public Nodo getDer() {  return nodoDer;  }  public void setDer(Nodo der) {  this.nodoDer = der;  }  public void insertar(int valorInsertar) {  if (valorInsertar < datos) {  if (nodoIzq == null)  nodoIzq = new Nodo(valorInsertar);  else  nodoIzq.insertar( valorInsertar );  }  else if (valorInsertar > datos) {  if (nodoDer == null)  nodoDer = new Nodo(valorInsertar);  else  nodoDer.insertar( valorInsertar );  }  }  } |

**Fig. 2** | Código de la clase Nodo

|  |
| --- |
| /\* |
| run: |
| Datos de salida por consola |
| BUILD SUCCESSFUL (total time: 0 seconds) |
| \*/ |

# **La clase** ***InterfazArbol***

## ***Problema* *InterfazArbol***

Esta clase es la cual nos permite generar la ventana de la interfaz gráfica, se necesita que esta clase mantenga una comunicación con el simulador o el controlador del árbol, debido a que mediante ese simulador se puede generar la gráfica con los valores que se vayan agregando, también se debe comunicar con la clase donde se vaya a generar la gráfica puesto que estas se generan de mejor manera en un Panel de Java.

## ***Análisis* *InterfazArbol***

Esta clase es la cual nos da la posibilidad de ver la interfaz de la aplicación, por lo tanto posee métodos los cuales son los botones y cajas de texto de la interfaz mediante los cuales recolecta la información ingresada por los usuarios.

Después de accionar estos botones procede a llamar a los métodos del intermediario para poder agregar los datos al árbol y que sean almacenados.

## ***Codificación y documentación* InterfazArbol**

|  |  |
| --- | --- |
| **17** | public class InterfazArbol extends javax.swing.JFrame {  private Intermediario simulador = new Intermediario();  /\*\*  \* Creates new form Vista  \*/  public InterfazArbol() {  initComponents();  this.inicializar(false);  }  private void inicializar(boolean enable) {  this.InOrden.setEnabled(enable);  this.PostOrden.setEnabled(enable);  this.PreOrden.setEnabled(enable);  }  /\*\*  \* This method is called from within the constructor to initialize the form.  \* WARNING: Do NOT modify this code. The content of this method is always  \* regenerated by the Form Editor.  \*/  @SuppressWarnings("unchecked")  // <editor-fold defaultstate="collapsed" desc="Generated Code">  private void initComponents() {  jPanel2 = new javax.swing.JPanel();  jDesktopPane1 = new javax.swing.JDesktopPane();  jInternalFrame2 = new javax.swing.JInternalFrame();  jScrollPane1 = new javax.swing.JScrollPane();  impresion = new javax.swing.JTextArea();  InOrden = new javax.swing.JButton();  PreOrden = new javax.swing.JButton();  txtdato = new javax.swing.JTextField();  botonInsertar = new javax.swing.JButton();  PostOrden = new javax.swing.JButton();  jLabel1 = new javax.swing.JLabel();  jLabel2 = new javax.swing.JLabel();  jLabel3 = new javax.swing.JLabel();  setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  setResizable(false);  getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());  jPanel2.setBackground(new java.awt.Color(255, 255, 255));  jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder(null, "Panel de Pruebas", javax.swing.border.TitledBorder.DEFAULT\_JUSTIFICATION, javax.swing.border.TitledBorder.DEFAULT\_POSITION, new java.awt.Font("Tahoma", 1, 10))); // NOI18N  jPanel2.setOpaque(false);  jPanel2.setLayout(new org.netbeans.lib.awtextra.AbsoluteLayout());  jDesktopPane1.setOpaque(false);  jInternalFrame2.setIconifiable(true);  jInternalFrame2.setMaximizable(true);  jInternalFrame2.setResizable(true);  jInternalFrame2.setEnabled(false);  jInternalFrame2.setFocusCycleRoot(false);  jInternalFrame2.setVisible(true);  jDesktopPane1.add(jInternalFrame2);  jInternalFrame2.setBounds(0, 0, 590, 320);  jPanel2.add(jDesktopPane1, new org.netbeans.lib.awtextra.AbsoluteConstraints(18, 28, 590, 327));  jScrollPane1.setOpaque(false);  impresion.setEditable(false);  impresion.setColumns(20);  impresion.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N  impresion.setRows(5);  impresion.setBorder(javax.swing.BorderFactory.createTitledBorder("Resultados de operaciones"));  impresion.setOpaque(false);  jScrollPane1.setViewportView(impresion);  impresion.getAccessibleContext().setAccessibleName("Recorridos");  jPanel2.add(jScrollPane1, new org.netbeans.lib.awtextra.AbsoluteConstraints(20, 360, 590, 80));  getContentPane().add(jPanel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(350, 10, 630, 460));  jPanel2.getAccessibleContext().setAccessibleName("Gráfica del Árbol Binario");  jPanel2.getAccessibleContext().setAccessibleDescription("Gráfica del Árbol Binario");  InOrden.setFont(new java.awt.Font("Tahoma", 3, 11)); // NOI18N  InOrden.setText("InOrden");  InOrden.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  InOrdenActionPerformed(evt);  }  });  getContentPane().add(InOrden, new org.netbeans.lib.awtextra.AbsoluteConstraints(110, 340, 82, 33));  PreOrden.setFont(new java.awt.Font("Tahoma", 3, 11)); // NOI18N  PreOrden.setText("PreOrden");  PreOrden.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  PreOrdenActionPerformed(evt);  }  });  getContentPane().add(PreOrden, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 340, -1, 33));  txtdato.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  txtdatoActionPerformed(evt);  }  });  getContentPane().add(txtdato, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 200, 220, 30));  botonInsertar.setFont(new java.awt.Font("Tahoma", 1, 12)); // NOI18N  botonInsertar.setText("Insertar");  botonInsertar.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  botonInsertarActionPerformed(evt);  }  });  getContentPane().add(botonInsertar, new org.netbeans.lib.awtextra.AbsoluteConstraints(10, 240, 91, 30));  PostOrden.setFont(new java.awt.Font("Tahoma", 3, 11)); // NOI18N  PostOrden.setText("PostOrden");  PostOrden.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  PostOrdenActionPerformed(evt);  }  });  getContentPane().add(PostOrden, new org.netbeans.lib.awtextra.AbsoluteConstraints(210, 340, -1, 33));  jLabel1.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  jLabel1.setText("SISTEMA DE OPERACIONES");  getContentPane().add(jLabel1, new org.netbeans.lib.awtextra.AbsoluteConstraints(50, 50, 260, 40));  jLabel2.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  jLabel2.setText("CON");  getContentPane().add(jLabel2, new org.netbeans.lib.awtextra.AbsoluteConstraints(150, 90, -1, -1));  jLabel3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N  jLabel3.setText("ÁRBOLES BINARIOS");  getContentPane().add(jLabel3, new org.netbeans.lib.awtextra.AbsoluteConstraints(90, 120, -1, -1));  pack();  }// </editor-fold>  private void botonInsertarActionPerformed(java.awt.event.ActionEvent evt) {  try {  int dato = Integer.parseInt(txtdato.getText());  if (this.simulador.insertar(dato)) {  JOptionPane.showMessageDialog(null, "El dato fue insertado correctamente", " ...", 1);  this.inicializar(true);  complementos();  }  } catch (Exception e) {  JOptionPane.showMessageDialog(null, "No se pudo insertar el dato", "Intenta de nuevo...", 0);  }  limpiar();  }  private void InOrdenActionPerformed(java.awt.event.ActionEvent evt) {  // TODO add your handling code here:  String recorrido = "";  recorrido = simulador.miArbol.recorridoEnorden();  this.impresion.setText("");  this.impresion.setText(recorrido);  this.simulador.miArbol.limpiar();  limpiar();  }  private void PreOrdenActionPerformed(java.awt.event.ActionEvent evt) {  // TODO add your handling code here:  String recorrido = "";  recorrido = simulador.miArbol.recorridoPreorden();  this.impresion.setText("");  this.impresion.setText(recorrido);  this.simulador.miArbol.limpiar();  limpiar();  }  private void PostOrdenActionPerformed(java.awt.event.ActionEvent evt) {  // TODO add your handling code here:  String recorrido = "";  recorrido = simulador.miArbol.recorridoPostorden();  this.impresion.setText("");  this.impresion.setText(recorrido);  this.simulador.miArbol.limpiar();  limpiar();  }  private void txtdatoActionPerformed(java.awt.event.ActionEvent evt) {  // TODO add your handling code here:  }  public void complementos() {  this.repintarArbol();  }  private void repintarArbol() {  this.jDesktopPane1.removeAll();  Rectangle tamaño = this.jInternalFrame2.getBounds();  this.jInternalFrame2 = null;  this.jInternalFrame2 = new JInternalFrame("Representación gráfica", true);  this.jDesktopPane1.add(this.jInternalFrame2, JLayeredPane.DEFAULT\_LAYER);  this.jInternalFrame2.setVisible(true);  this.jInternalFrame2.setBounds(tamaño);  this.jInternalFrame2.setEnabled(false);  this.jInternalFrame2.add(this.simulador.getDibujo(), BorderLayout.CENTER);  }  /\*\*  \* @param args the command line arguments  \*/  public static void main(String args[]) {  /\* Set the Nimbus look and feel \*/  //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html  \*/  try {  for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  if ("Nimbus".equals(info.getName())) {  javax.swing.UIManager.setLookAndFeel(info.getClassName());  break;  }  }  } catch (ClassNotFoundException ex) {  java.util.logging.Logger.getLogger(InterfazArbol.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (InstantiationException ex) {  java.util.logging.Logger.getLogger(InterfazArbol.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (IllegalAccessException ex) {  java.util.logging.Logger.getLogger(InterfazArbol.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (javax.swing.UnsupportedLookAndFeelException ex) {  java.util.logging.Logger.getLogger(InterfazArbol.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  }  //</editor-fold>  //</editor-fold>  //</editor-fold>  //</editor-fold>  /\* Create and display the form \*/  java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  new InterfazArbol().setVisible(true);  }  });  }  private void limpiar() {  txtdato.setText("");  }  // Variables declaration - do not modify  private javax.swing.JButton InOrden;  private javax.swing.JButton PostOrden;  private javax.swing.JButton PreOrden;  private javax.swing.JButton botonInsertar;  private javax.swing.JTextArea impresion;  private javax.swing.JDesktopPane jDesktopPane1;  private javax.swing.JInternalFrame jInternalFrame2;  private javax.swing.JLabel jLabel1;  private javax.swing.JLabel jLabel2;  private javax.swing.JLabel jLabel3;  private javax.swing.JPanel jPanel2;  private javax.swing.JScrollPane jScrollPane1;  private javax.swing.JTextField txtdato;  // End of variables declaration  } |

# 

# **La clase *Graficar***

## ***Problema* Graficar**

La clase debe aportar los valores necesarios para calcular la posición en los cuales las figuras serán retratadas, además debe devolver valores concretos para poder comunicarse con la clase principal InterfazArbol.

## ***Análisis* Graficar**

La clase recorre con sus métodos el árbol que se recibe en el constructor, de esta manera extrae todos los datos de todos los nodos de l árbol binario u por lo tanto se van acumulando los valores, después de esto se van presentando estos valores obtenidos en la gráfica y se conectan mediante líneas y grafos.

|  |  |  |  |
| --- | --- | --- | --- |
| **CLASES** | **ATRIBUTOS** | **MÉTODOS** | **REQUERIMIENTOS** |
| **Graficar** | arbol  hashmap posiciónnodos  subarbolestamano  dirty  parent2child  Dimension empty  Fontmetrics fm | Constructor()  calcularPosiciones  calcularTamañosubarbol  calcularPosicion  dibujarArbol  paint | El método Paint()  Debe ser sobreescribible, debido a que a cada inserción de nodos se va a mostrar de nuevo el árbol. |

## ***Codificación y documentación* Graficar**

|  |  |
| --- | --- |
| **17** | class Graficar extends JPanel {  private Arbol miArbol;  private HashMap posicionNodos = null;  private HashMap subArbTam = null;  private boolean dirty = true;  private int parent2child = 20, child2child = 30;  private Dimension empty = new Dimension(0, 0);  private FontMetrics fm = null;  public Graficar(Arbol miArbol) {  this.miArbol = miArbol;  this.setBackground(Color.WHITE);  posicionNodos = new HashMap();  subArbTam = new HashMap();  dirty = true;  repaint();  }  private void calcularPosiciones() {  posicionNodos.clear();  subArbTam.clear();  Nodo root = this.miArbol.getRaiz();  if (root != null) {  calcularTamañoSubarbol(root);  calcularPosicion(root, Integer.MAX\_VALUE, Integer.MAX\_VALUE, 0);  }  }  private Dimension calcularTamañoSubarbol(Nodo n) {  if (n == null) {  return new Dimension(0, 0);  }  Dimension ld = calcularTamañoSubarbol(n.getIzq());  Dimension rd = calcularTamañoSubarbol(n.getDer());  int h = fm.getHeight() + parent2child + Math.max(ld.height, rd.height);  int w = ld.width + child2child + rd.width;  Dimension d = new Dimension(w, h);  subArbTam.put(n, d);  return d;  }  private void calcularPosicion(Nodo n, int left, int right, int top) {  if (n == null) {  return;  }  Dimension ld = (Dimension) subArbTam.get(n.getIzq());  if (ld == null) {  ld = empty;  }  Dimension rd = (Dimension) subArbTam.get(n.getDer());  if (rd == null) {  rd = empty;  }  int center = 0;  if (right != Integer.MAX\_VALUE) {  center = right - rd.width - child2child / 2;  } else if (left != Integer.MAX\_VALUE) {  center = left + ld.width + child2child / 2;  }  int width = fm.stringWidth(n.getDato() + "");  posicionNodos.put(n, new Rectangle(center - width / 2 - 3, top, width + 6, fm.getHeight()));  calcularPosicion(n.getIzq(), Integer.MAX\_VALUE, center - child2child / 2, top + fm.getHeight() + parent2child);  calcularPosicion(n.getDer(), center + child2child / 2, Integer.MAX\_VALUE, top + fm.getHeight() + parent2child);  }  private void dibujarArbol(Graphics2D g, Nodo n, int puntox, int puntoy, int yoffs) {  if (n == null) {  return;  }  Rectangle r = (Rectangle) posicionNodos.get(n);  g.draw(r);  g.drawString(n.getDato() + "", r.x + 3, r.y + yoffs);  if (puntox != Integer.MAX\_VALUE) {  g.drawLine(puntox, puntoy, (int) (r.x + r.width / 2), r.y);  }  dibujarArbol(g, n.getIzq(), (int) (r.x + r.width / 2), r.y + r.height, yoffs);  dibujarArbol(g, n.getDer(), (int) (r.x + r.width / 2), r.y + r.height, yoffs);  }  @Override  public void paint(Graphics g) {  super.paint(g);  fm = g.getFontMetrics();  if (dirty) {  calcularPosiciones();  dirty = false;  }  Graphics2D g2d = (Graphics2D) g;  g2d.translate(getWidth() / 2, parent2child);  dibujarArbol(g2d, this.miArbol.getRaiz(), Integer.MAX\_VALUE, Integer.MAX\_VALUE,  fm.getLeading() + fm.getAscent());  fm = null;  }  } |

**Fig. 2** | Código de la clase Graficar

# **La clase *Lienzo***

## ***Problema* Lienzo**

Se necesita de una ventana en la cual se pueda dibujar la gráfica del árbol binario

## ***Análisis* Lienzo**

La clase recorre con sus métodos el árbol que se recibe en el constructor, de esta manera extrae todos los datos de todos los nodos del árbol binario u por lo tanto se van acumulando los valores, después de esto se van presentando estos valores obtenidos en la gráfica y se conectan mediante líneas y grafos.

## ***Codificación y documentación* Lienzo**

|  |  |
| --- | --- |
| **17** | class Lienzo extends javax.swing.JPanel {  private String ruta;  public Lienzo() {  initComponents();  this.setSize(933, 690);  this.ruta = "";  }  @Override  public void paintComponent(Graphics g) {  Dimension tam = getSize();  ImageIcon ImagenFondo = new ImageIcon(new ImageIcon(getClass().getResource(ruta)).getImage());  g.drawImage(ImagenFondo.getImage(), 0, 0, tam.width, tam.height, null);  setOpaque(false);  super.paintComponent(g);  }  @SuppressWarnings("unchecked")  // <editor-fold defaultstate="collapsed" desc="Generated Code">  private void initComponents() {  javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);  this.setLayout(layout);  layout.setHorizontalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGap(0, 462, Short.MAX\_VALUE)  );  layout.setVerticalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGap(0, 323, Short.MAX\_VALUE)  );  }// </editor-fold>  // Variables declaration - do not modify  // End of variables declaration  } |

**Fig. 2** | Código de la clase Graficar

# **Bibliography**

DEITEL, P. J., & DEITEL, H. M. (2012). *COMO PROGRAMAR EN JAVA* (9 ed.). (E. PEARSON, Ed.) Naucalpan de Juarez, México.

Kojevnikov, A. (1 de 1 de 2016). *hilite.me*. Recuperado el 26 de 8 de 2016, de Flask and Pygments: http://hilite.me

Oracle and/or its affiliates. (1 de 1 de 1993, 2016). *API Java*. Recuperado el 26 de 8 de 2016, de Java™ Platform, Standard Edition 7 API Specification: https://docs.oracle.com/javase/7/docs/api/

Oracle Corporation. (1 de 1 de 2016). *NetBeans IDE*. Recuperado el 26 de 8 de 2016, de NetBeans: https://netbeans.org

Universidad Técnica Particular. (1 de 8 de 2016). *EVA*. Recuperado el 26 de 8 de 2016, de Entorno Virtual de Aprendizaje UTPL: https://eva3.utpl.edu.ec/

Universidad Técnica Particular de Loja. (1 de 1 de 2016). *GitLab*. (UTPL, Editor) Recuperado el 29 de 8 de 2016, de GitLab Community Edition: https://git.taw.utpl.edu.ec

**RÚBRICA PARA LA EVALUACIÓN DE ENSAYOS / PROGRAMAS.**

Las rúbricas son una de las herramientas más importantes para evaluar de una manera objetiva, precisa y realista, es oportuno validar los ensayos con criterios bien definidos calificando el esfuerzo y resultados de todo un bimestre, evidenciando así el nivel de competencias adquiridas en la asignatura de **Estructuras de Datos**.

Captura de pantalla de un celular con letras

Descripción generada automáticamente