

Documentación de código

INGENIERA DE SOFTWARE II

CARLOS DANIEL HERRERA TOLENTINO

```
/*
```

```
* To change this license header, choose License Headers in Project Properties.
```

```
* To change this template file, choose Tools | Templates
```

```
* and open the template in the editor.
```

```
*/
```

```
package proyecto_estructura;
```

```
import java.text.DateFormat;
```

```
import java.text.SimpleDateFormat;
```

```
import java.util.ArrayList;
```

```
import java.util.Collections;
```

```
import java.util.Date;
```

```
/**
```

```
*
```

```
* @author Tolentino
```

```
* 04/06/2019
```

```
* proyecto de estructura de datos
```

```
* el proyecto hace un banco con arboles binarios, donde tenemos 2 tipos de retiros que es con
```

```
* y sin tarjeta en este caso la opcion 1 es sin tarjeta y la 2 con tarjeta.
```

```
* en esta clase tenemos lo funcional del arbol.
```

```
* agregar: agregamos o insertamos la sucursal al arbol.
```

```
* quitar: eliminar la sucursal si ya no tienen dinero.
```

```
* postOrden: ordenamiento para eliminar ya que queremos eliminar de la izq,der,raiz.
```

```
* borrarCola2: es para eliminar la sucursal una vez que no tiene dinero.
```

```
* eliminarNodo: es para el nodo apunte a otro que no sea el que se va eliminar.
```

```
* obtenerReemplazo: obtenemos el siguiente nodo a donde se va a apuntar.
```

```
* ColaLista_movimientos: imprimimos los movimientos de esa sucursal.
```

```
* mover_movimientos: obtiene la informacion de la sucursal y sus movimientos.
```

- * imprimirNivel: obtenemos la izq y der del arbol.
- * imprimirNivel imprimimos los niveles del arbol.
- * retornarAltura: le damos la altura a los lados del arbol.
- * retornarAltura: imprime la altura de los lados del arbol y su raiz.
- * imp_ordenados imprimimos ordenados las sucursales ingresadas.
- * preOrden obtenemos la raiz,izq,der.
- * InoOrden obtenemos der,raiz,izq.
- * vacio: declaramos que la raiz esta en 0.
- */

```
public class ArbolBinario {
```

```
    DateFormat hourFormat = new SimpleDateFormat("HH:mm:ss");//tomar el formato de
    hora,minutos,segundos
```

```
    NodoArbol raiz;
```

```
    int cant, altura;
```

```
    String h;
```

```
    String[] niveles;
```

```
    String[] ar;
```

```
    ColaLista x = new ColaLista();
```

```
    ColaLista ordenados = new ColaLista();
```

```
    ColaLista_movimientos c1 = new ColaLista_movimientos();
```

```
    ColaLista_movimientos mov_locacion = new ColaLista_movimientos();
```

```
    ArrayList<persona> test = new ArrayList<>();//arreglo tipo persona
```

```
    public ArbolBinario() {
```

```
        raiz = null;
```

```
    }
```

```
    /*agregar
```

```
        este metodo va agregar la cantidad y nombre al arbol junto con sus hojas
```

lo que utilizaremos un nodo para que existan las conexiones, donde nos pide el tipo, nom, cantidad.

nuevo es el arbol con los datos necesarios como el tipo, nom, cantidad.

raiz de esta forma vamos a representar la raiz del arbol

padre el inicio del arbol

auxiliar para diferenciar para donde va si izq o derecha

*/

```
public void agregar(int tipo, String nom, int cantidad) {  
    // creamos un nuevo objeto tipo NodoArbol, para insertar  
    NodoArbol nuevo = new NodoArbol(tipo, nom, cantidad);  
    // si raiz es nulo, el nuevo valor sera lo que se inserto, esto se debe a que el arbol esta vacio  
    if (raiz == null) {  
        raiz = nuevo;  
    } else {  
        //  
        NodoArbol auxiliar = raiz;  
        NodoArbol padre;  
        // hacemos un ciclo para que apunte ya sea izq o der.  
        while (true) {  
            //nos ayuda apuntar hacia el lado necesario para insertar  
            padre = auxiliar;  
            // si la cantidad introducida es menor a lo que tiene raiz, el puntero apunta a la izq  
            if (cantidad < auxiliar.cantidad) {  
                //el puntero apunta a la izq  
                auxiliar = auxiliar.izquierda;  
                // si izquierda es nulo, se podra insertar  
                if (auxiliar == null) {  
                    // hacemos un nuevo nodo que apunte a derecha
```

```

        padre.izquierda = nuevo;

        return;
    }
} else {
    // si la cantidad es mayor , el punto se mueve a derecha
    auxiliar = auxiliar.derecha;
    // se verifica que en dercha este nulo
    if (auxiliar == null) {
        // hacemos un nuevo nodo que apunte a derecha
        padre.derecha = nuevo;
        return;
    }
}
}
}
}
}
}
}

```

```

public boolean vacio() {
    return raiz == null;
}

```

```

public void quitar(NodoArbol x3, int opcion, int cantidad1) {
    //nos ayuda para que no se repita el resultado
    boolean c = true;
    while (c != false) {
        // objeto para la insercion de fechas
        Date date = new Date();
        // mandamos llamar la funcion frenteCola_retuern la cual obtiene el tope del una cola
    }
}

```

```

String[] ar = x.frenteCola_retuern();

// si el usuario elije 1

if (opcion == 1) {

    // usamos ar y preguntamos si arr[0] es igual a opcion y a su vez ar[2] es mayor a la cantidad a
restar

    if (Integer.parseInt(ar[0]) == opcion && Integer.parseInt(ar[2]) >= cantidad1) {

        // creamos un while que busca en el arbol, el nombre que tiene ar[1]

        while (!x3.nombre.equals(ar[1])) {

            // si lo encuentra pregunta ar[2] es menor a la cantidad del nodo del alrbol

            if (Integer.parseInt(ar[2]) < x3.cantidad) {

                // apuntamos a izq

                x3 = x3.izquierda;

            } else {

                // apuntamos a derecha

                x3 = x3.derecha;

            }

        }

        //realizamos la resta del nodo y la cantidad

        x3.cantidad -= cantidad1;

        // insertamos en una lista la info del nodo y lo que se le resto

        c1.insertar(x3.dato, x3.nombre, x3.cantidad, "Retiro:" + cantidad1+" pesos",
hourFormat.format(date));

        //// prueba acomodo de nodos cada vez que se reste , se elimina el nodo y se vuelve a
insertar///

        // todo esto para evitar que numeros menores queden mal acomodados///

        eliminarNodo(ar);

        // agrega el mismo nodo que se borro al mismo arbol

        agregar(x3.dato, x3.nombre, x3.cantidad);

        ///preguntamos si el nodo.cantidad es 0//

        if (x3.cantidad == 0) {

```

```

        // si lo es elimina el nodo, y a su vez inserta la informacion del nodo a una lista
        eliminarNodo(ar);

        c1.insertar(x3.dato, x3.nombre, x3.cantidad, "Nodo Eliminado", hourFormat.format(date));

        // muestra la informacion de ese nodo que se borro en especifico
        mover_movimientos(x3.nombre, c1);

        mov_locacion.imprimir();
    }

    //rompe el ciclo ya que se resto la cantidad
    c = false;
} else {

    // si ar[0] no es igual a la opcion del usuario , eliminamos el tope de la cola y entra otra vez

    x.quitar();
}

} else {

    //// dos , lo mismo de arriba pero esta vez para la opcion 2
    if (Integer.parseInt(ar[0]) == opcion && Integer.parseInt(ar[2]) >= cantidad1) {
        while (!x3.nombre.equals(ar[1])) {
            if (Integer.parseInt(ar[2]) < x3.cantidad) {
                //apuntamos a la izq
                x3 = x3.izquierda;
            } else {
                //apuntamos a la derecha
                x3 = x3.derecha;
            }
        }
    }

    //le restamos la cantidad
    x3.cantidad -= cantidad1;

```



```

    if (r != null) {
        PostOrden(r.izquierda); //nos dirigimos a la izquierda
        PostOrden(r.derecha); //nos dirigimos a la derecha
        x.insertar(r.dato, r.nombre, r.cantidad); //insertamos la raíz con el dato,nombre,cantidad
        ordenados.insertar(r.dato, r.nombre, r.cantidad); //ordenamos lo insertado
        System.out.println "[" + "Nombre: " + r.nombre + " " + "Dinero: " + r.cantidad + "]; //imprimos el
nombre y cantidad
    }
}

```

```

public void PreOrden(NodoArbol r) {
    if (r != null) {

        System.out.print "[" + "Nombre: " + r.nombre + " " + "Dinero: " + r.cantidad + "];
        PreOrden(r.izquierda);
        PreOrden(r.derecha);
    }
}

```

```

public void borrar_cola() {
    if (!x.colaVacia()) {
        while (!x.colaVacia()) {
            x.quitar();
        }
    }
}

```

/*borrar_cola2

borrar lo insertado en la cola

ordenados hace referencia a la cola

colaVacia es el metodo para saber que no tiene nada insertado

usamos el metodo quitar para borrar de la cola

```
*/
```

```
public void borrar_cola2() {  
    if (!ordenados.colaVacia()) {  
        while (!ordenados.colaVacia()) {  
            ordenados.quitar();//borramos de la cola  
        }  
    }  
}
```

```
public boolean eliminarNodo(String[] ar) {  
    NodoArbol aux = raiz;  
    NodoArbol padre = raiz;  
    boolean esHijoIzq = true;//utilizamos para saber si se elimino correctamente  
    while (!aux.nombre.equals(ar[1])) {  
        padre = aux;  
        if (Integer.parseInt(ar[2]) < aux.cantidad) {  
            esHijoIzq = true;//identificamos si es hijo izquierdo  
            aux = aux.izquierda;//apuntamos hacia la izquierda  
        } else {  
            esHijoIzq = false;// si no es hijo izquierdo  
            aux = aux.derecha;//apuntamos a la derecha  
        }  
        if (aux == null) {  
            return false;  
        }  
    }  
    /// fin while
```

```

if (aux.izquierda == null && aux.derecha == null) {
    if (aux == raiz) {
        raiz = null;//raiz del arbol hacemos null
    } else if (esHijoIzq) {
        padre.izquierda = null;// la izquierda del padre se hace null
    } else {
        padre.derecha = null;// la derecha del padre se hace null
    }
} else if (aux.derecha == null) {
    if (aux == raiz) {
        raiz = aux.izquierda; //nos dirigimos a la izquierda
    } else if (esHijoIzq) {
        padre.izquierda = aux.izquierda;
    } else {
        padre.izquierda = aux.derecha;
    }
} else if (aux.izquierda == null) {
    if (aux == raiz) {
        raiz = aux.derecha;
    } else if (esHijoIzq) {
        padre.izquierda = aux.derecha;//toma el valor de derecha
    } else {
        padre.derecha = aux.derecha;//toma el valor de derecha
    }
} else {
    NodoArbol reemplazo = obtenerNodoReemplazo(aux);
    if (aux == raiz) {
        raiz = reemplazo;//reemplazamos la raiz
    }
}

```

```

    } else if (esHijoIzq) {
        padre.izquierda = reemplazo; //eliminamos el padre y apunte a izquierda
    } else {
        padre.derecha = reemplazo; //eliminamos el padre y apunte a derecha
    }
    reemplazo.izquierda = aux.izquierda; //se convierte en nodo padre la izquierda
}
return true; //si esto se cumple devolvemos el nodo eliminado
}

```

/*obtenerNodoReemplazo

se utiliza para poder sustituir el padre y poder apuntar a otro nodo

reemplazarPadre lo usamos para que aux lo reemplace

Reemplazo hacia donde debe apuntar

aux lo usamos para que tome el nodo padre

*/

```

public NodoArbol obtenerNodoReemplazo(NodoArbol NodoReemplazo) {
    NodoArbol ReemplazarPadre = NodoReemplazo;
    NodoArbol Reemplazo = NodoReemplazo;
    NodoArbol aux = NodoReemplazo.derecha;
    while (aux != null) {
        ReemplazarPadre = Reemplazo; //reemplazamos al nodo padre
        Reemplazo = aux; //obtenemos el reemplazo
        aux = aux.izquierda; // nos vamos a la izq para subir el reemplazo
    }
    if (Reemplazo != NodoReemplazo.derecha) {
        ReemplazarPadre.izquierda = Reemplazo.derecha; //indica hacia donde debe apuntar este caso izq
        Reemplazo.derecha = NodoReemplazo.derecha; // indica hacia donde debe apuntar este caso
derecha
    }
}

```

```

    }

    return Reemplazo;//lo reemplazamos
}

/*
se utiliza para imprimir los movientos realizados
c1 toma el metodo para imprimir la colalista
*/
public ColaLista_movimientos mov() {
    c1.imprimir();//imprimos la lista ordenada
    return c1;
}

/*movimientos
se utiliza para poder ver los movimientos que hizo el cliente
movimientos tomara el valor de x1.frente que sera el frente de la cola
mov_locacion llama al metodo insertar para tomar los datos necesarios
x2 tomara el siguiente de la cola
*/
public void mover_movimientos(String x, ColaLista_movimientos x1) {
    //tomamos los movimientos del primero
    movimientos x2 = x1.frente;
    while (x2 != null) {
        if (x2.getNombre().equals(x)) {
            //tomamos los valores como el tipo,nombre,dinero,transaccion y hora
            mov_locacion.insertar(x2.getTipo(), x2.getNombre(), x2.getDinero(), x2.getTransaccion(),
x2.getHora());
            //nos vamos al siguiente de la cola
            x2 = x2.getNext();
        } else {
            //nos vamos al siguiente de la cola

```

```

        x2 = x2.getNext();
    }

}

}

}

/*InOrden
usamos el ordenamiento para tomar los valores primero de la izquierda a la hora de retirar dinero
r hacia donde apuntar izq o derecha
*/
public void InOrden(NodoArbol r) {
    if (r != null) {
        InOrden(r.izquierda);//apunta a la izquierda

        System.out.println "[" + "Nombre: " + r.nombre + " " + "Dinero: " + r.cantidad + "");//imprime el
nombre y cantidad que tiene

        InOrden(r.derecha);//apunta a la derecha
    }
}

/*imprimirNivel
imprimimos el arbol por niveles apartir de la izquierda o derecha
niveles te indica apartir del nodo padre para abajo
hacemos un ciclo para que vaya tomando el tamaño de niveles
*/
public void imprimirNivel() {
    niveles = new String[altura + 1];//toma la altura del padre y le aumenta uno

    imprimirNivel(raiz, 0);

    System.out.print("          ");

    for (int i = 0; i < niveles.length; i++) {

```

```

        //toma el valor de los niveles
        System.out.println(niveles[i]); //imprimimos por nivel

    }
}

/*imprimirNivel
    imprimir el arbol con su derecha y izquierda
    niveles tomamos la informacion del banco como nombre y cantidad
    imprimirNivel lo tomamos del metodo, tomamos la izquierda y lo vamos sumando uno
    imprimirNivel lo tomamos del metodo, tomamos la derecha y lo vamos sumando uno
*/
private void imprimirNivel(NodoArbol pivote, int nivel2) {
    if (pivote != null) {
        niveles[nivel2] = "[" + pivote.nombre + " " + pivote.cantidad + "]" + " " +
            ((niveles[nivel2] != null) ? niveles[nivel2] : ""); //tomamos el nombre y la cantidad del banco
        imprimirNivel(pivote.izquierda, nivel2 + 1); //imprimos la izquierda y le va aumentando uno
        imprimirNivel(pivote.derecha, nivel2 + 1); //imprimimos la derecha y le va aumentando uno
    }
}

/*retornarAltura
    aqui indicamos desde donde empiece el arbol
    altura es la altura del arbol
*/
public int retornarAltura() {
    altura = 0;

    retornarAltura(raiz, 1); //la raiz la mostraremos en el nivel 1

    return altura; //mostraremos la altura
}

```

```

/*retornarAltura
mostramos la izquierda y derecha del arbol de acuerdo a su nivel
retornarAltura tomaremos el nivel dependiendo a donde va izquierda o derecha
reco es el nodo raiz del arbol
*/
private void retornarAltura(NodoArbol reco, int nivel) {
    if (reco != null) {
        retornarAltura(reco.izquierda, nivel + 1);//nos dirigimos a la izquierda y le vamos ir aumentando 1
        if (nivel > altura) {
            altura = nivel;//estara la altura en el nivel indicado
        }
        retornarAltura(reco.derecha, nivel + 1);//nos dirigimos a la derecha y le vamos ir aumentando 1
    }
}

/*imp_ordenados
en este metodo imprimiremos la cola ordenada
lol es un string donde ordenamos los que estan enfrente de la cola
persona o p tomamos el valor de donde retirara la persona
add agregamos una persona
ordenados.quitar lo sacamos de la cola
*/
public void imp_ordenados() {
    while (!ordenados.colaVacia()) {
        String[] lol = ordenados.frenteCola_retuern();//la cola para ordenarlos
        persona p = new persona(Integer.parseInt(lol[0]), lol[1], Integer.parseInt(lol[2]));//diferenciar por
cual metodo retira
        test.add(p);// agregamos a la persona
        ordenados.quitar();// sacamos a la persona
    }
}

```



```
Collections.sort(test);  
for (persona elemento : test) {  
    System.out.println(elemento);//los imprime ordenados  
}  
test.clear();//limpiamos los ordenados  
}  
}
```

Bibliografía

☞ [Can92] Cannon, L. W. et al. Recommended C Style and Coding Standards

<ftp://ftp.csi.ull.es/pub/assignas/AUTOMALF/doc/cstyle.ps.gz>

☞ [Dox00] Doxygen: a documentation system for C++, IDL (Corba and Microsoft flavors) and C. Dimitri van Heesch.

<http://www.stack.nl/~dimitri/doxygen/index.html>

☞ [Dox00B] Instrucciones básicas para trabajar con Doxygen. F. de Sande

<http://nereida.deioc.ull.es/~sande/doxygen/doxygen.html>

☞ [Dye99] Dyer, D. The Top 10 Ways to get screwed by the "C" programming language.

<http://www.andromeda.com/people/ddyer/topten.html>

☞ [Kar99] Karplus, K. In-Program Documentation.

http://www.cse.ucsc.edu/~karplus/185/w99/reader/7_In_program_Documentation.html

☞ [Oua97] Oualline, S. Practical C Programming. O'Reilly & Associates, Inc., 1997 ISBN: 1-56592-306-5

☞ [Van78] Van Tassel, D. Program Style, Design, Efficiency, Debugging and Testing. Prentice-Hall, 1978. ISBN: 0-13-729947-8