Tarea Semana 11

Integrantes: Diego A. Vera S.
Gilio H. Linfati G.

Profesor: Pablo A. Valenzuela T.

Asignatura: Estructura de Datos

Fecha: 22/08/2020

Problema 1:

- **Problema 1:** este problema se encuentra disponible en el proyecto IntelliJ ED_Problema1.zip. Este consiste en implementar el código y pruebas unitarias de 6 métodos: 1) agregarNodo() y agregar(); 2) buscarNodo() y buscar(); y 3) eliminarNodo() y eliminar(). Para desarrollar este problema se debe utilizar programación en parejas, donde se espera pueda desempeñar ambos roles, es decir, conductor y observador.

```java
package ed.tarea.s11;

public class Nodo {

    int valor;
    Nodo nodoDerecho;
    Nodo nodoIzquierdo;
    Nodo(int valor) {
        this.valor = valor;
    }

}
```

```java
private Nodo agregarNodo(Nodo actual, int value) {
    Nodo nuevo;
    nuevo = new Nodo (value);
    nuevo.valor = value;
    nuevo.nodoIzquierdo = null;
    nuevo.nodoDerecho = null;
    if (root == null)
        root = nuevo;
    else {
        Nodo anterior = null, reco;
        reco = root;
        while (reco != null) {
            anterior = reco;
            if (value < reco.valor)
                reco = reco.nodoIzquierdo;
            else
                reco = reco.nodoDerecho;
        }
        if (value < anterior.valor)
            anterior.nodoIzquierdo = nuevo;
        else
            anterior.nodoDerecho = nuevo;
    }
    return actual;
}

/*
    método público, para agregar nodo
 */
public void agregar(int value) {
    // completar método
    agregarNodo(root, value);
}
```

```java
/*
    método privado, para buscar nodo con cierto valor
 */
private boolean buscarNodo(Nodo actual, int valor) {
    // completar método
    Nodo auxiliar = actual;
    if (auxiliar == null) return false;
    while (auxiliar.valor != valor) {
        if (valor < auxiliar.valor) {
            auxiliar = auxiliar.nodoIzquierdo;
        } else {
            auxiliar = auxiliar.nodoDerecho;
        }
        if (auxiliar == null) {
            return false;
        }
    }
    return true;
}

/*
    método público, para buscar nodo con cierto valor
 */
public boolean buscar(int valor) {
    // completar método
    return buscarNodo(root,valor);
}
```

```java
private Nodo eliminarNodo(Nodo actual, int valor) {
    // completar método
    try {
        if (actual == null) { return actual; }
        if (valor < actual.valor)
            actual.nodoIzquierdo = eliminarNodo(actual.nodoIzquierdo, valor);
        else if (valor > actual.valor) {
            actual.nodoDerecho = eliminarNodo(actual.nodoDerecho, valor);
        } else {
            if (actual.nodoIzquierdo == null && actual.nodoDerecho == null) {
                return null;
            } else if (actual.nodoIzquierdo == null) {
                //actual = actual.nodoDerecho;
                return actual.nodoDerecho;
            } else if (actual.nodoDerecho == null) {
                //actual = actual.nodoIzquierdo;
                return actual.nodoIzquierdo;
            } else {
                Integer valorMin = minValor(actual.nodoDerecho);
                actual.valor = valorMin;
                actual.nodoDerecho = eliminarNodo(actual.nodoDerecho,valorMin);
            }
        }
    } catch (Exception e) {
        return actual;
    }

    return actual;
}
```

```java
private Integer minValor(Nodo nodo) {
    if(nodo.nodoDerecho != null) {
        return minValor(nodo.nodoIzquierdo);
    }
    return nodo.valor;
}

/*
    método público, para eliminar nodo
 */
public void eliminar(int valor) {
    // completar método
    eliminarNodo(root,valor);
}
```

Problema 2:

- **Problema 2:** en este problema se deberá crear e implementar dos ejemplos relacionados con el tópico de estudio actual (árboles), donde se ilustre claramente buenas prácticas expuestas en la lectura “Nombres con sentido”. Los ejemplos, debenestar relacionados a un problema o contexto particular, implementar (al menos) clases, variables, métodos y respectivas pruebas unitarias. También, el código desarrollado debe incorporar comentarios javadoc y/o de línea.

Problema 2.1

```java
public class Juego {

    private String nombreJuego;
    private int anioEstrenoJuego;
    private String consolaEstreno;

    public Juego(String nombreJuego, int anioEstrenoJuego, String consolaEstreno) {
        this.nombreJuego = nombreJuego;
        this.anioEstrenoJuego = anioEstrenoJuego;
        this.consolaEstreno = consolaEstreno;
    }
```

```java
public String getNombreJuego() {
    return nombreJuego;
}

public void setNombreJuego(String nombreJuego) {
    this.nombreJuego = nombreJuego;
}

public int getAnioEstrenoJuego() {
    return anioEstrenoJuego;
}

public void setAnioEstrenoJuego(int anioEstrenoJuego) {
    this.anioEstrenoJuego = anioEstrenoJuego;
}

public String getConsolaEstreno() {
    return consolaEstreno;
}

public void setConsolaEstreno(String consolaEstreno) {
    this.consolaEstreno = consolaEstreno;
}

public String toString() {
    return getNombreJuego();
}
```

```java
public class Nodo {

    Juego juego;
    Nodo nodoDerecho;
    Nodo nodoIzquierdo;


    public Nodo(Juego juego) {
        this.juego = juego;
    }
}
```

```java
/**
 *  Metodo privado para agregar juegos.
 * @param actual
 * @param juego
 */
private Nodo agregarNodo(Nodo actual, Juego juego) {
    actual = new Nodo (juego);
    actual.juego = juego;
    actual.nodoIzquierdo = null;
    actual.nodoDerecho = null;
    if (root == null)
        root = actual;
    else {
        Nodo anterior = null, reco;
        reco = root;
        while (reco != null) {
            anterior = reco;
            if (juego.getAnioEstrenoJuego() < reco.juego.getAnioEstrenoJuego())
                reco = reco.nodoIzquierdo;
            else
                reco = reco.nodoDerecho;
        }
        if (juego.getAnioEstrenoJuego() < anterior.juego.getAnioEstrenoJuego())
            anterior.nodoIzquierdo = actual;
        else
            anterior.nodoDerecho = actual;
    }
    return actual;
}
```

```java
/**
 *  Metodo público para agregar juegos.
 * @param juego
 */
public void agregar(Juego juego) {
    agregarNodo(root, juego);
}
```

```java
/**
 *  Metodo privado para buscar juegos.
 * @param actual
 * @param juego
 */
private boolean buscarNodo(Nodo actual, Juego juego) {
    Nodo auxiliar = actual;
    if (auxiliar == null) {
        return false;
    }
    if (juego == null) return false;
    while (auxiliar.juego != juego) {
        if (juego.getAnioEstrenoJuego() < auxiliar.juego.getAnioEstrenoJuego()) {
            auxiliar = auxiliar.nodoIzquierdo;
        } else {
            auxiliar = auxiliar.nodoDerecho;
        }
        if (auxiliar == null) {
            return false;
        }
    }

    return true;
}



/**
 *  Metodo público para buscar juegos.
 * @param juego
 */
public boolean buscar(Juego juego) {
    return buscarNodo(root, juego);
}
```

```java
/**
 *  Metodo privado para mostrar juegos.
 * @param nodo
 */
private void imprimirEnOrden(Nodo nodo) {
    if (nodo == null)
        return;
    imprimirEnOrden(nodo.nodoIzquierdo); // primero el nodo izquierdo
    System.out.print(nodo.juego + " ; "); // luego imprimir el valor del nodo
    imprimirEnOrden(nodo.nodoDerecho);// luego el nodo  derecho
}

/**
 *  Metodo público para mostrar juegos.
 */
public void imprimirEnOrden() {
    imprimirEnOrden(root);
}
```

Problema 2.2

```java
public class Nodo {

    int valor;
    String dato;
    Nodo nodoDerecho;
    Nodo nodoIzquierdo;
    Nodo(int valor) { this.valor = valor; }

    public String getDato(int valor) { return dato; }

    public void setDato(String dato) { this.dato = dato; }

    @Override
    public String toString() {
        return "Nodo{" +
                "valor=" + valor +
                ", dato='" + dato + '\'' +
                '}';
    }
}
```

```java
public class arbolBinario {

    protected Nodo root = null;

    arbolBinario(int valor) { root = new Nodo(valor); }

    arbolBinario() { root = null; }

    /**
     * Método privado, para agregar nodo un valor al nodo ademas de un dato escrito.
     * @param actual variable del tipo Nodo al que se le agregara un valor ademas de un dato.
     * @param valor variable del tipo int que indica el valor que se le va a dar al Nodo actual.
     * @return retorna el objeto nodo con los valores agregados.
     **/
    private Nodo agregarNode(Nodo actual, int valor, String dato) {
        if (actual == null) {
            Nodo nuevo = new Nodo(valor);
            nuevo.setDato(dato);
            return nuevo;
        }
        if (valor < actual.valor) {
            actual.nodoIzquierdo = agregarNode(actual.nodoIzquierdo, valor, dato);
        } else if (valor > actual.valor) {
            actual.nodoDerecho = agregarNode(actual.nodoDerecho, valor, dato);
        } else {
            // value already exists
            return actual;
        }
        return actual;
    }
```

```java
/**
 * Método privado, que retorna un String que es escrito con Scanner.
 * @return la escritura del dato.
 **/
private String agregarDato() {
    System.out.println("Agregue el texto que desee");
    Scanner teclado = new Scanner(System.in);
    return teclado.nextLine();
}

/**
 *Método público, para agregar nodo.
 * @param valor variable del tipo int que indica el valor que se le va a dar al Metodo que se esta llamando.
 **/
public void agregar(int valor,String dato) {
    // completar método
    if (dato == null){
        root = agregarNode(root, valor, agregarDato());
    } else {
        root = agregarNode(root, valor, dato);
    }

}
```

```java
/**
 * Método privado, para buscar un nodo con cierto valor, ademas de devolver el dato asignado a esta.
 * @param actual variable del tipo Nodo en la que sera buscado el valor.
 * @param valor variable del tipo int que indica el valor que se esta buscando.
 * @return retorna un booleano dependiendo de si se encontro el valor a no.
 **/
private boolean buscarNodo(Nodo actual, int valor) {
    // completar método
    Nodo auxiliar = actual;
    boolean existeValor = true;
    if (auxiliar == null) return false;
    while (auxiliar.valor != valor) {
        if (valor < auxiliar.valor) {
            auxiliar = auxiliar.nodoIzquierdo;
        } else {
            auxiliar = auxiliar.nodoDerecho;
        }
        if (auxiliar == null) {
            System.out.println("No existe el nodo escrito");
            existeValor = false;
            return existeValor;
        }
    }
    mostrarDato(auxiliar,existeValor);
    return existeValor;
}
/**
 * Método privado, para mostrar el dato con el se encuentra asociado el nodo.
 * @param auxiliar variable del tipo Nodo en la que se obtiene el dato.
 * @param existeValor variable del tipo booleano que indica si existe el valor muestre el dato.
 **/
private void mostrarDato(Nodo auxiliar, boolean existeValor) {
    if (existeValor == true) System.out.println(auxiliar.toString());
}
```

```java
/**
 * Método publico, para buscar un nodo.
 * @param valor variable del tipo int que indica el valor que se esta buscando.
 * @return retorna un metodo de tipo booleano.
 **/
public boolean buscar(int valor) {
    // completar método
    return buscarNodo(root,valor);
}
```