Ejercicio U6_2C_E1:

Lo más fácil, sobrecargando recorridoPreorden(), realmente se trata de cambiar el nombre ayudantePreorden por recorridoPreoden()

```
 public void recorridoPreorden() {
        recorridoPreorden(raiz);
    }

    private void recorridoPreorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }

        System.out.print(nodo.datos+ " ");
        recorridoPreorden(nodo.nodoIzq);
        recorridoPreorden(nodo.nodoDer);
    }
```

Otra forma, escribir un getRaiz() y que lo use el programador del main.

De esta manera, para el programador del main es un método más difícil de entender ya que si lo invoca sobre el árbol puede preguntarse ...

```
 arbol.preordenSinAyudante(arbol.getRaiz());
```

¿Porque no accede el propio método a la raíz de su árbol?¿Porqué se la tengo que indicar yo?

```
class NodoArbol {
    NodoArbol nodoIzq;
    int datos;
    NodoArbol nodoDer;

    public NodoArbol(int datosNodo) {
        datos = datosNodo;
        nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
    }
}

class Arbol {
    private NodoArbol raiz;
    public Arbol() {
        raiz = null;
    }
    public NodoArbol getRaiz() {
        return raiz;
    }

    // si el valor ya existe en el arbol, no inserta nada
    public void insertar(int valorInsertar) {
        if (raiz == null) {
            raiz = new NodoArbol(valorInsertar);
        } else {
            ayudanteInsertarNodo(raiz,valorInsertar);
        }
    }
    private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
        // inserta en el subárbol izquierdo
        if (valorInsertar < a.datos) {
            // inserta nuevo NodoArbol
            if (a.nodoIzq == null) {
                a.nodoIzq = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
            }
        }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
            if (a.nodoDer == null) {
                a.nodoDer = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoDer,valorInsertar);
            }
        }
    }


    public void recorridoPreorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }
```

```
        System.out.print(nodo.datos+ " ");
        recorridoPreorden(nodo.nodoIzq);
        recorridoPreorden(nodo.nodoDer);
    }

}

public class App {
    public static void main(String args[]) {
        Arbol arbol = new Arbol();
        arbol.insertar(10);arbol.insertar(20);arbol.insertar(8);arbol.insertar(15);
        arbol.insertar(26);arbol.insertar(22);arbol.insertar(17);arbol.insertar(4);
        arbol.recorridoPreorden(arbol.getRaiz());

    }
}
```

Ejercicio U6_2C_E2:

```
class NodoArbol {
    NodoArbol nodoIzq;
    int datos;
    NodoArbol nodoDer;

    public NodoArbol(int datosNodo) {
        datos = datosNodo;
        nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
    }
}

class Arbol {
    private NodoArbol raiz;
    public Arbol() {
        raiz = null;
    }

    public void insertar(int valorInsertar) {
        if (raiz == null) {
            raiz = new NodoArbol(valorInsertar);
        } else {
            ayudanteInsertarNodo(raiz,valorInsertar);
        }
    }
    private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
        // inserta en el subárbol izquierdo
        if (valorInsertar < a.datos) {
            // inserta nuevo NodoArbol
            if (a.nodoIzq == null) {
                a.nodoIzq = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
            }
        }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
            if (a.nodoDer == null) {
                a.nodoDer = new NodoArbol(valorInsertar);
            } else {
                ayudanteInsertarNodo(a.nodoDer,valorInsertar);
            }
        }
    }

    public boolean existe(int valor){
        return ayudanteExiste(valor,raiz);
    }
    private boolean ayudanteExiste(int valor, NodoArbol nodo){
        if(nodo==null){
            return false;
        }else if(nodo.datos==valor){
            return true;
        }else if(valor<nodo.datos){
            return ayudanteExiste(valor,nodo.nodoIzq);
        }else{
            return ayudanteExiste(valor,nodo.nodoDer);
        }
    }

}

public class App {
    public static void main(String args[]) {
        Arbol arbol = new Arbol();
        arbol.insertar(10);arbol.insertar(20);arbol.insertar(8);arbol.insertar(15);
        arbol.insertar(26);arbol.insertar(22);arbol.insertar(17);arbol.insertar(4);
```

```
      System.out.println("\nExiste el valor "+ 3 + "?: "+ arbol.existe(3));
      System.out.println("\nExiste el valor "+ 22 + "?: "+ arbol.existe(22));

   }
}
```

el if de ayudanteExiste() se podrían simplificar con

```
return ayudanteExiste(nodo.nodoIzq,dato) || ayudanteExiste(nodo.nodoDer,dato);
```

el || es un operador perezoso así que si por la izquierda da true ya no se mira la derecha y por tanto sería eficiente como el if-else

Ejercicio U6_2C_E3:

```
class NodoArbol {

   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }

}

class Arbol {

   private NodoArbol raiz;

   public Arbol() {
      raiz = null;
   }

   // si el valor ya existe en el arbol, no inserta nada
   public void insertar(int valorInsertar) {
      if (raiz == null) {
         raiz = new NodoArbol(valorInsertar);
      } else {
         ayudanteInsertarNodo(raiz,valorInsertar);
      }
   }
   private void ayudanteInsertarNodo(NodoArbol a, int valorInsertar){
      // inserta en el subárbol izquierdo
      if (valorInsertar < a.datos) {
         // inserta nuevo NodoArbol
         if (a.nodoIzq == null) {
            a.nodoIzq = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoIzq,valorInsertar);
         }
      }else if (valorInsertar > a.datos) { // inserta en el subárbol derecho
         if (a.nodoDer == null) {
            a.nodoDer = new NodoArbol(valorInsertar);
         } else {
            ayudanteInsertarNodo(a.nodoDer,valorInsertar);
         }
      }
   }
   public void recorridoPreorden() {
      ayudantePreorden(raiz);
```

```
    }

     private void ayudantePreorden(NodoArbol nodo) {
        if (nodo == null) {
            return;
        }

        System.out.print(nodo.datos+ " ");
        ayudantePreorden(nodo.nodoIzq);
        ayudantePreorden(nodo.nodoDer);
    }
    public void recorridoPreordenConTAB() {
        ayudantePreordenConTAB(raiz,"");
    }
 private void ayudantePreordenConTAB(NodoArbol nodo,String tab) {
        if (nodo == null) {
            return;
        }

        System.out.println(tab+nodo.datos);
        tab=tab+"\t";
        ayudantePreordenConTAB(nodo.nodoIzq,tab);
        ayudantePreordenConTAB(nodo.nodoDer,tab);
    }
 }

public class App {

    public static void main(String args[]) {
        Arbol arbol = new Arbol();

        arbol.insertar(8);arbol.insertar(4);arbol.insertar(10);
        arbol.insertar(2);arbol.insertar(5);arbol.insertar(9);
        arbol.insertar(12);

        System.out.println("\nRecorrido preorden .....");
        arbol.recorridoPreorden();
        System.out.println("\nRecorrido preorden con indentaciones.....");
        arbol.recorridoPreordenConTAB();
    }
}
```

Ejercicio U6_2C_E4:

TAMBIÉN PUEDO IMPRIMIR NULL Al LLEGAR A ARBOL VACÍO para ver la forma del árbol**.**

Simplemente al caso nodo==null añado un println() pero esto hace que todavía visualice mejor el árbol de memoria

```
 private void ayudantePreordenConTAB(NodoArbol nodo,String tab) {
        if (nodo == null) {
            System.out.println(tab+"null");
            return;
        }

        System.out.println(tab+nodo.datos);
        tab=tab+"\t";
        ayudantePreordenConTAB(nodo.nodoIzq,tab);
        ayudantePreordenConTAB(nodo.nodoDer,tab);
    }
 }
```

Ejercicio U6_2C_E5: Altura de un AB

```
import java.util.Scanner;

class NodoArbol {
   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }
}

class Arbol {
   public NodoArbol raiz;
   String[] arbolString;//el arbol como una línea de
   int posArray=0;

   public Arbol(String[] arbolString) {
      raiz = null;
      this.arbolString=arbolString;
      this.crearArbol();
   }

   public void recorridoPreorden() {
      ayudantePreorden(raiz, "");
   }

   private void ayudantePreorden(NodoArbol nodo, String tab) {
      if (nodo == null) {
         System.out.println(tab + "null");
         return;
      }

      System.out.println(tab + nodo.datos);
      tab = tab + "\t";
      ayudantePreorden(nodo.nodoIzq, tab);
      ayudantePreorden(nodo.nodoDer, tab);
   }
   private  void crearArbol() {
      //la raiz está en arbolString[0]
      int dato=Integer.parseInt(arbolString[0]);

      if (dato == -1) {//árbol vacío
         raiz = null;

      } else {
         raiz = new NodoArbol(dato);
         ayudanteCrearArbol(raiz);


      }
   }

   private void ayudanteCrearArbol(NodoArbol padre) {
         //subarbol izquierdo del padre
         posArray++;
         int  dato=Integer.parseInt(arbolString[posArray]);
```

```
            if(dato!=-1){//si es -1 entonces nodoIzq queda con  null
                padre.nodoIzq=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoIzq);
            }

            //subarbol derecho del padre
            posArray++;
            dato=Integer.parseInt(arbolString[posArray]);
            if(dato!=-1){
                padre.nodoDer=new NodoArbol(dato);
                ayudanteCrearArbol(padre.nodoDer);
            }

    }
    int altura(){
        return ayudanteAltura(raiz);

    }
    int ayudanteAltura(NodoArbol raiz){
        if(raiz==null){
            return 0;
        }else{
            return 1 + (int)Math.max(ayudanteAltura(raiz.nodoDer),ayudanteAltura(raiz.nodoIzq));
        }
    }

}

public class App {

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        String[] arbolString = sc.nextLine().split(" ");
        Arbol arbol = new Arbol(arbolString);
        System.out.println("recorrido preorden con TABS");
        arbol.recorridoPreorden();
        System.out.println("\nAltura: "+arbol.altura());

    }
}
```

Ejercicio U6_2C_E6: Máximo y mínimo de un AB

```
import java.util.Scanner;

class NodoArbol {
   NodoArbol nodoIzq;
   int datos;
   NodoArbol nodoDer;

   public NodoArbol(int datosNodo) {
      datos = datosNodo;
      nodoIzq = nodoDer = null; //recien creado un nodo, no tiene hijos
   }
}

class Arbol {
```

```
public NodoArbol raiz;
String[] arbolString;//el arbol como una línea de
int posArray=0;

public Arbol(String[] arbolString) {
    raiz = null;
    this.arbolString=arbolString;
    this.crearArbol();
}

private  void crearArbol() {
    //la raiz está en arbolString[0]
    int dato=Integer.parseInt(arbolString[0]);

    if (dato == -1) {//árbol vacío
        raiz = null;

    } else {
        raiz = new NodoArbol(dato);
        ayudanteCrearArbol(raiz);


    }
}

private void ayudanteCrearArbol(NodoArbol padre) {
        //subarbol izquierdo del padre
        posArray++;
        int  dato=Integer.parseInt(arbolString[posArray]);
        if(dato!=-1){//si es -1 entonces nodoIzq queda con  null
            padre.nodoIzq=new NodoArbol(dato);
            ayudanteCrearArbol(padre.nodoIzq);
        }

        //subarbol derecho del padre
        posArray++;
        dato=Integer.parseInt(arbolString[posArray]);
        if(dato!=-1){
            padre.nodoDer=new NodoArbol(dato);
            ayudanteCrearArbol(padre.nodoDer);
        }

}
int max(int a, int b, int c){
    if(a>=b && a>=c){
            return a;
    }
    if(b>=a && b>=c){
            return b;
    }
    return c;


}
int maximo(){
    return maximo(raiz);
}
int maximo(NodoArbol nodo){
    if(nodo==null){
        return Integer.MIN_VALUE;
    }
    return max(nodo.datos,maximo(nodo.nodoIzq),maximo(nodo.nodoDer));
}

int min(int a, int b, int c){
    if(a<=b && a<=c){
```

```
            return a;
        }
        if(b<=a && b<=c){
            return b;
        }
        return c;


    }
    int minimo(){
        return minimo(raiz);
    }
    int minimo(NodoArbol nodo){
        if(nodo==null){
            return Integer.MAX_VALUE;
        }
        return min(nodo.datos,minimo(nodo.nodoIzq),minimo(nodo.nodoDer));
    }


}

public class App {

    public static void main(String args[]) {
        Scanner sc = new Scanner(System.in);

        String[] arbolString = sc.nextLine().split(" ");
        Arbol arbol = new Arbol(arbolString);

        System.out.println("el mayor: " + arbol.maximo());
        System.out.println("el menor: " + arbol.minimo());



    }
}
```