TEMA 4

"Árboles"

- Diseño e Implementación del TAD Árbol Binario (AB)

Clase Teoría

TAD Árbol Binario

Especificación

```
public class ArbolBinario<E> {
    // Declaración de tipos: ArbolBinario
    // Características: Un árbol binario es un árbol vacío o un nodo con dos hijos (izquierdo y derecho)
    //                  que a su vez son árboles binarios. Los objetos son modificables
    public ArbolBinario();
        // Produce:  Un árbol vacío
    public ArbolBinario(E elemRaiz, ArbolBinario<E> hi, ArbolBinario<E> hd) throws NullPointerException;
        // Produce:  Si hi o hd son null, lanza la excepción NullPointerException. En caso contrario,
        //           construye un árbol de raíz elemRaiz, hijo izquierdo hi e hijo derecho hd

    public boolean esVacio();
        // Produce:  Cierto si this está vacío. Falso en caso contrario.
    public E raiz() throws ArbolVacioExcepcion;
        // Produce:  Si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino devuelve el objeto almacenado en la raíz
    public ArbolBinario<E> hijoIzq() throws ArbolVacioExcepcion;
        // Produce:  Si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino devuelve el subárbol izquierdo
    public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion;
        // Produce:  Si this está vacío lanza la excepciónArbolVacioExcepcion,
        //           sino devuelve el subárbol derecho
```

Especificación Lógica
(Nivel Lógico)

TAD Árbol Binario

```
public boolean esta (E elemento);
        // Produce:  Cierto si elemento está en this, falso, en caso contrario
public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;
        // Modifica:  this
        // Produce:  Si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino asigna el objeto elemRaiz a la raíz del árbol this
public void setHijoIzq(ArbolBinario<E> hi) throws ArbolVacioExcepcion, NullPointerException;
        // Modifica:  this
        // Produce:  Si hi es null, lanza la excepción NullPointerException.
        //           En caso contrario, si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino asigna el árbol hi como subárbol izquierdo de this
public void setHijoDer(ArbolBinario<E> hd) throws ArbolVacioExcepcion, NullPointerException;
        // Modifica:  this
        // Produce:  Si hd es null, lanza la excepción NullPointerException.
        //           En caso contrario, si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino asigna el árbol hd como subárbol derecho de this
public void suprimir ();
        // Modifica:  this
        // Produce:  El árbol binario vacío
```

TAD Árbol Binario

Ejemplo de uso del TAD Árbol binario

```
public static <E> void preorden(ArbolBinario<E> a){
        if (!a.esVacio()) {
                System.out.print(a.raiz() + " ");
                preorden(a.hijoIzq());
                preorden(a.hijoDer());
        }
}

public static <E> void anchura(ArbolBinario<E> a){
        Cola<ArbolBinario<E>> c = new EnlazadaCola<E>();
        c.insertar(a);
        do {
                a = c.suprimir();
                if (!a.esVacio()){
                        System.out.print(a.raiz() + " ");
                        c.insertar(a.hijoIzq());
                        c.insertar(a.hijoDer());
                }
        } while (!c.esVacio());
}
```

Uso
(Nivel de Aplicación)

TAD Árbol Binario

Implementación del TAD
(Nivel Físico)

Implementación

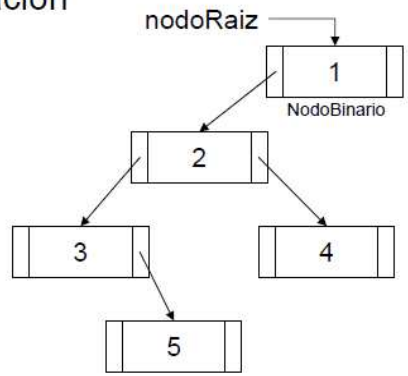
- Paso 1: Definición interfaz

```
public interface ArbolBinario<E>        {
        public boolean esVacio();
        public E raiz() throws ArbolVacioExcepcion;
        public ArbolBinario<E> hijoIzq() throws ArbolVacioExcepcion ;
        public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion;
        public boolean esta (E elemento);
        public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;
        public void setHijoIzq(ArbolBinario<E> hi)
                throws ArbolVacioExcepcion, NullPointerException;
        public void setHijoDer(ArbolBinario<E> hd)
                throws ArbolVacioExcepcion, NullPointerException;
        public void suprimir();
}
```

- Paso 2: Clase implemente la interfaz
  - Mediante estructuras enlazadas genéricas
    - public class EnlazadoArbolBinario<E> implements ArbolBinario<E>

TAD Árbol Binario

- Representación




```
public class EnlazadoArbolBinario<E> implements ArbolBinario<E>{
   private NodoBinario<E> nodoRaiz;
```

```
public class NodoBinario<E>{
        private E elemento;                 // referencia al elemento del nodo
        private NodoBinario<E> izq;         // referencia al nodo izquierdo
        private NodoBinario<E> der;         // referencia al nodo derecho

        public NodoBinario(E e, NodoBinario<E> hi, NodoBinario<E> hd){
                elemento = e;
                izq = hi;
                der = hd;
        }
        public E getElemento() {
                return elemento;
        }
        public NodoBinario<E> getIzq() {
                return izq;
        }
        public NodoBinario<E> getDer() {
                return der;
        }
        public void setElemento(E e) {
                elemento = e;
        }
        public void setIzq(NodoBinario<E> hi) {
                izq = hi;
        }
        public void setDer(NodoBinario<E> hd) {
                der = hd;
        }
}
```

TAD Árbol Binario

```
public class EnlazadoArbolBinario<E> implements ArbolBinario<E> {
   private NodoBinario<E> nodoRaiz;
   public EnlazadoArbolBinario() {
        nodoRaiz = null;
   }
   public EnlazadoArbolBinario(E elemRaiz, ArbolBinario<E> hi,
        ArbolBinario<E> hd) throws NullPointerException {

        if (hi==null || hd ==null) throw new NullPointerException();
        nodoRaiz = new NodoBinario<E>(elemRaiz,
                        ((EnlazadoArbolBinario<E>) hi).nodoRaiz,
                        ((EnlazadoArbolBinario<E>) hd).nodoRaiz);
   }
   private EnlazadoArbolBinario<E>(NodoBinario<E> raiz) {
        nodoRaiz = raiz;
   }
```

TAD Árbol Binario

```
public boolean esVacio()
{
        return nodoRaiz == null;
}
public E raiz() throws ArbolVacioExcepcion
{
        if (esVacio())
                throw new ArbolVacioExcepcion("raiz: Árbol vacío");
        return nodoRaiz.getElemento();
}
public ArbolBinario<E> hijoIzq() throws ArbolVacioExcepcion
{
        if (esVacio())
                throw new ArbolVacioExcepcion("hijoIzq: Árbol vacío");
        return new EnlazadoArbolBinario<E>(nodoRaiz.getIzq());
}
public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion
{
        if (esVacio())
                throw new ArbolVacioExcepcion("hijoDer: Árbol vacío");
        return new EnlazadoArbolBinario<E>(nodoRaiz.getDer());
}
```

TAD Árbol Binario

```
public boolean esta(E elemento)
{
        return esta(nodoRaiz,elemento);
}
private boolean esta(NodoBinario<E> raiz, E elemento)
{
        if (raiz==null)
                return false;
        if (raiz.getElemento().equals(elemento))
                return true;
        boolean temp = esta(raiz.getIzq(),elemento);
        if (!temp)
                return esta(raiz.getDer(),elemento);
        return temp;
}

public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion
{
        if (esVacio())
                throw new ArbolVacioExcepcion("raiz: Árbol vacío");
        nodoRaiz.setElemento(elemRaiz);
}
```

TAD Árbol Binario

```
public void setHijoIzq(ArbolBinario<E> hi) throws ArbolVacioExcepcion, NullPointerException
{
        if (hi==null) throw new NullPointerException();
        if (esVacio())
                throw new ArbolVacioExcepcion("setHijoIzq: Árbol vacío");
        nodoRaiz.setIzq(((EnlazadoArbolBinario<E>) hi).nodoRaiz);
}
public void setHijoDer(ArbolBinario<E> hd) throws ArbolVacioExcepcion,NullPointerException
{
        if (hd==null) throw new NullPointerException();
        if (esVacio())
                throw new ArbolVacioExcepcion("setHijoIzq: Árbol vacío");
        nodoRaiz.setDer(((EnlazadoArbolBinario<E>) hd).nodoRaiz);
}
public void suprimir()
{
        nodoRaiz = null;
}
}
```

TAD Árbol Binario

```
public interface ArbolBinario<E> {
  public boolean esVacio();
  public E raiz() throws ArbolVacioExcepcion;
  public ArbolBinario<E> hijoIzq() throws ArbolVacioExcepcion ;
  public ArbolBinario<E> hijoDer() throws ArbolVacioExcepcion;
  public boolean esta (E elemento);
  public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;
  public void setHijoIzq(ArbolBinario<E> hi)
          throws ArbolVacioExcepcion, NullPointerException;
  public void setHijoDer(ArbolBinario<E> hd)
          throws ArbolVacioExcepcion, NullPointerException;
  public void suprimir();
    }
```

```
public class ArbolVacioExcepcion extends RuntimeException
{
  public ArbolVacioExcepcion (){
    ...
  }
  public ArbolVacioExcepcion (String mensaje){
    ...
  }
}
```

```
public class EnlazadoArbolBinario<E> implements ArbolBinario<E> {
  private NodoBinario<E> nodoRaiz;

  public EnlazadoArbolBinario() {
          nodoRaiz = null;
  }
  // implementación operaciones de la interfaz
}
```

```
public class NodoBinario<E> {
  private E elemento;
  private NodoBinario<E> izq, der;

  public NodoBinario(){
  ...
  }
  // resto operaciones
}
```

TEMA 4

"Árboles"

- Diseño e Implementación del TAD Árbol Binario de Búsqueda (ABB)

Clase Teoría

TAD Árbol Binario de Búsqueda

Especificación

```
public class ArbolBusqueda<E> {
    // Declaración de tipos: ArbolBusqueda
    // Características: Es un árbol binario donde para cada nodo se cumple la propiedad
    //                  de que todos los nodos con clave menor que la suya están en
    //                  su subárbol izquierdo y todos los nodos con clave mayor o igual
    //                  se encuentran en el subárbol derecho.
    //                  Los objetos son modificables

    public ArbolBusqueda();
        // Produce: Un árbol vacío
    public boolean esVacio();
        // Produce:         Cierto si el árbol está vacío. Falso en caso contrario.
    public E raiz() throws ArbolVacioExcepcion;
        // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,
        //          sino devuelve el objeto almacenado en la raíz
    public ArbolBusqueda<E> hijoIzq() throws ArbolVacioExcepcion;
        // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,
        //          sino devuelve el subárbol izquierdo
```

Especificación Lógica
(Nivel Lógico)

TAD Árbol Binario de Búsqueda

Especificación

```
public ArbolBusqueda<E> hijoDer() throws ArbolVacioExcepcion;
        // Produce: Si el árbol está vacío lanza la excepción ArbolVacioExcepcion,
        //          sino devuelve el subárbol derecho

public void insertar(E o);
        // Modifica:        this
        // Produce:         Añade el objeto o a this

public void eliminar(E o) throws ElementoIncorrecto;
        // Modifica:        this
        // Produce:         Si o no existe en el árbol, lanza la excepción
        //                  ElementoIncorrecto sino elimina el objeto de this.

public boolean buscar(E o) ;
        // Produce:         Devuelve cierto si el objeto está en el árbol y
        //                  falso en otro caso
}
```

TAD Árbol Binario de Búsqueda

Implementación

- ¿Cómo comparar los elementos de los nodos?
  Solución: los elementos deben ser instancias de una clase que implemente la interface Comparable<E> existente en java.

```
public interface Comparable<E>{
        public int compareTo(E e);
}
```

Implementación
(Nivel Físico)

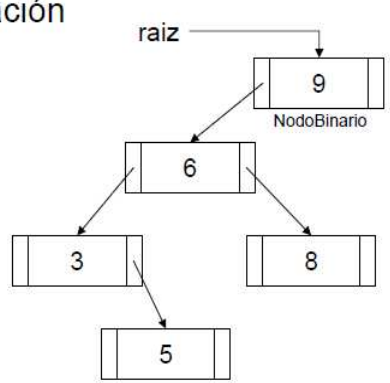
- Paso 1: Definición interfaz

```
public interface ArbolBusqueda <E extends Comparable<E>> {
    public boolean esVacio();
    public E raiz() throws ArbolVacioExcepcion;
    public ArbolBusqueda<E> hijoIzq() throws ArbolVacioExcepcion;
    public ArbolBusqueda<E> hijoDer() throws ArbolVacioExcepcion;
    public void insertar(E elemento);
    public void eliminar(E elemento) throws ElementoIncorrecto;
    public boolean buscar(E elemento);
    }
```

- Paso 2: Clase implemente la interfaz
  - Mediante estructuras enlazadas genéricas
    - `public class ArbolBinarioBusqueda<E extends Comparable<E>> implements ArbolBusqueda<E>`

TAD Árbol Binario de Búsqueda

- Representación




```
public class ArbolBinarioBusqueda<E extends Comparable<E>>
                        implements ArbolBusqueda<E>
    private NodoBinario<E> raiz;
```

TAD Árbol Binario de Búsqueda

```
public class ArbolBinarioBusqueda<E extends Comparable<E>> implements ArbolBusqueda<E> {
    private NodoBinario<E> raiz;
    public ArbolBinarioBusqueda(){
        raiz=null;
    }
    private ArbolBinarioBusqueda(NodoBinario<E> r){
        raiz=r;
     }
    public boolean esVacio(){
        return raiz==null;
    }
    public E raiz() throws ArbolVacioExcepcion{
        if (esVacio()) throw new ArbolVacioExcepcion("raiz: Árbol vacío");
        return raiz.getElemento();
    }
    public ArbolBusqueda<E> hijoIzq() throws ArbolVacioExcepcion {
        if (esVacio()) throw new ArbolVacioExcepcion("hijoIzq: Árbol vacío");
        return new ArbolBinarioBusqueda<E>(raiz.getIzq());
    }

    public ArbolBusqueda<E> hijoDer() throws ArbolVacioExcepcion {
        if (esVacio()) throw new ArbolVacioExcepcion("hijoDer: Árbol vacío");
        return new ArbolBinarioBusqueda<E>(raiz.getDer());
    }
```

TAD Árbol Binario de Búsqueda

```
public void insertar(E elemento){
        raiz=insertar(raiz, elemento);
    }

private NodoBinario<E> insertar(NodoBinario<E> r, E elemento){
        if(r==null)
                return new NodoBinario<E>(elemento,null,null);
        else if (elemento.compareTo(r.getElemento())<0)
                r.setIzq(insertar(r.getIzq(),elemento));
            else r.setDer(insertar(r.getDer(),elemento));
        return r;
    }
```

TAD Árbol Binario de Búsqueda

```
public boolean buscar(E elemento){
        return buscar(raiz, elemento);
    }

private boolean buscar (NodoBinario<E> r, E elemento){
        if (r==null)
                return false;
        else if (elemento.compareTo(r.getElemento())==0)
                return true;
            else if (elemento.compareTo(r.getElemento())<0)
                        return buscar(r.getIzq(),elemento);
                else return buscar(r.getDer(),elemento);
    }
```

TAD Árbol Binario de Búsqueda

```
public void eliminar(E elemento) throws ElementoIncorrecto{
        raiz=eliminar(raiz, elemento);
    }

private NodoBinario<E> eliminar (NodoBinario<E> r, E elemento) throws ElementoIncorrecto{
        if (r==null) throw new ElementoIncorrecto("eliminar: elemento no existe");
        else if (elemento.compareTo(r.getElemento())<0)
                r.setIzq(eliminar(r.getIzq(),elemento));
            else if (elemento.compareTo(r.getElemento())>0)
                        r.setDer(eliminar(r.getDer(),elemento));
                        else if (r.getIzq()!=null && r.getDer()!=null){
                        r.setElemento(getMasIzq(r.getDer()));
                        r.setDer(eliminarMasIzq(r.getDer()));
                    }
                    else return(r.getIzq()!=null)?r.getIzq():r.getDer();
        return r;
    }
```

TAD Árbol Binario de Búsqueda

```
private E getMasIzq(NodoBinario<E> r){
        while (r.getIzq()!=null)
                r=r.getIzq();
        return r.getElemento();
}
private NodoBinario<E> eliminarMasIzq(NodoBinario<E> r){
        if (r.getIzq()==null)
                return r.getDer();
        else r.setIzq(eliminarMasIzq(r.getIzq()));
        return r;
}
```

- Los métodos buscar, insertar y eliminar llaman a un método **private** que recibe como parámetro el nodo raíz del árbol y una referencia al elemento.
- Para simplificar el código se emplea recursión, pero sería posible una implementación no recursiva.

TAD Árbol Binario de Búsqueda