Árboles

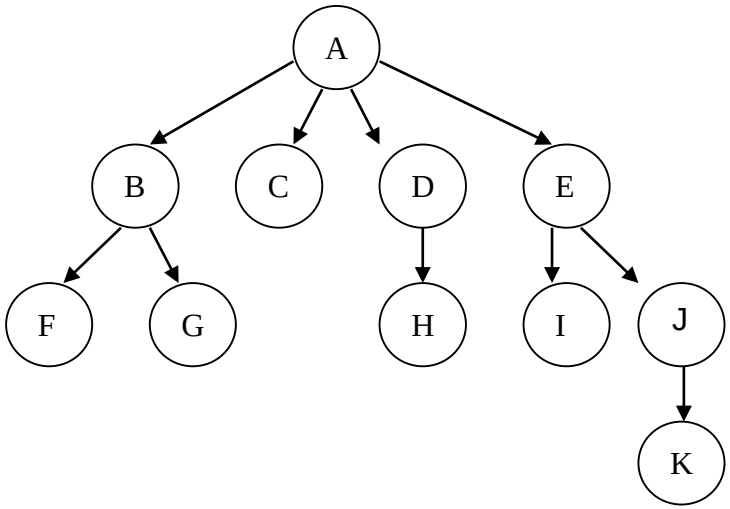
- Presentar la estructura no lineal más importante en computación
- Mostrar la especificación e implementación de varios tipos de árboles
- Algoritmos de manipulación de árboles

Árboles

Contenido

Árbol General

- Un árbol general es un árbol ordenado en el que cada nodo tiene un número arbitrario de hijos.
- También se les llama árboles multicamino.



TAD Árbol General

- **Especificación**

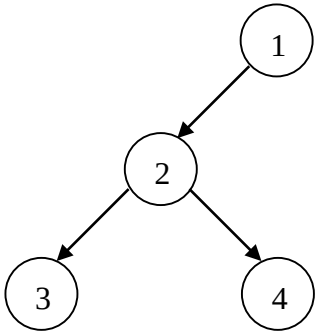
```
public class ArbolGeneral<E>{
    // Declaración de tipos: ArbolGeneral
    // Características: Un árbol general es un árbol ordenado donde cada nodo tiene un número
    //                  arbitrario de hijos. Los objetos son modificables
    public ArbolGeneral();
        // Produce:Un árbol vacío
    public ArbolGeneral(E elemRaiz, ArbolGeneral<E>… hijos) throws NullPointerException;
        // Produce:Si hijos es null, lanza la excepción NullPointerException.
        //         En caso contrario, construye el árbol de raíz elemRaiz, con los subárboles
        //         contenidos en hijos.
    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 elemento almacenado en la raíz
```

TAD Árbol General

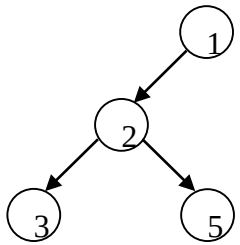
```
public ArbolGeneral<E> hijoMasIzq() throws ArbolVacioExcepcion;
        // Produce:  Si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino devuelve el subárbol más a la izquierda
public ArbolGeneral<E> hermanoDer() throws ArbolVacioExcepcion;
        // Produce:  Si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino devuelve el hermano derecho
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 elemRaíz a la raíz del árbol this
public void setHijo(ArbolGeneral<E> hijo) throws ArbolVacioExcepcion, NullPointerException;
        // Modifica: this
        // Produce:  Si hijo es null, lanza la excepción NullPointerException.
        //           En caso contrario, si this está vacío lanza la excepción ArbolVacioExcepcion,
        //           sino añade hijo como subárbol más a la derecha de this
public void suprimir();
        // Modifica: this
        // Produce:  Elimina el árbol general
}
```

TAD Árbol General

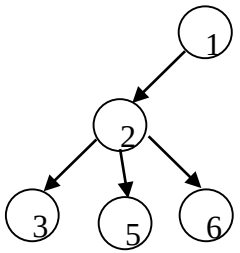
Ejemplo de uso:



- ArbolGeneral<Integer> tres = new ArbolGeneral<>(3);
- ArbolGeneral<Integer> cuatro = new ArbolGeneral<>(4);
- ArbolGeneral<Integer> dos = new ArbolGeneral<>(2, tres, cuatro);
- ArbolGeneral<Integer> uno = new ArbolGeneral<>(1, dos);
- uno.esVacio() → devolvería false
- tres.raiz() → devolvería el entero 3
- dos.hijoMasIzq() → devolvería el árbol general tres
- tres.hermanoDer() → devolvería el árbol general cuatro
- dos.esta(1) → devolvería false
- cuatro.setRaiz(5) → modificaría el árbol pasando a ser →



- dos.setHijo(new ArbolBinario<>(6)) → modificaría el árbol pasando a ser →



TAD Árbol General

Implementación

- Paso 1: Definición interfaz

```
public interface ArbolGeneral<E>{
        public boolean esVacio();
        public E raiz() throws ArbolVacioExcepcion;
        public ArbolGeneral<E> hijoMasIzq() throws ArbolVacioExcepcion ;
        public ArbolGeneral<E> hermanoDer() throws ArbolVacioExcepcion;
        public boolean esta(E elemento);
        public void setRaiz(E elemRaiz) throws ArbolVacioExcepcion;
        public void setHijo(ArbolGeneral<E> hijo) throws ArbolVacioExcepcion,
                                                         NullPointerException;
        public void suprimir();
 }
```

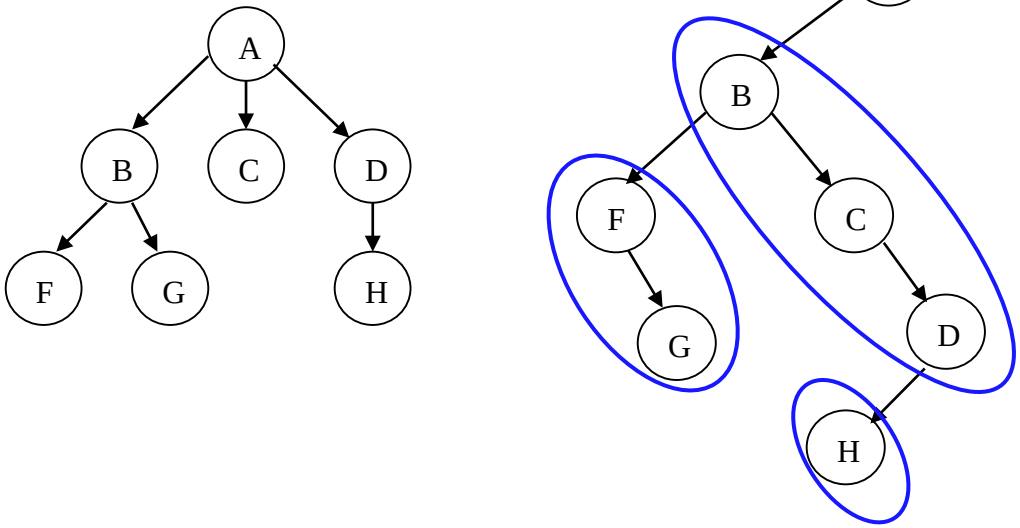
- Paso 2: Clase implemente la interfaz
  - Mediante estructuras enlazadas genéricas
    - **public class** EnlazadoArbolGeneral<E> **implements** ArbolGeneral<E>

TAD Árbol General

- Representación
  - La forma más común de implementación de árboles generales es mediante árboles binarios.
  - El hijo izquierdo es el hijo más a la izquierda del árbol general y el hijo derecho es el siguiente hermano hacia la derecha.
  - Ventaja de esta representación: Simplicidad.

TAD Árbol General

- Representación



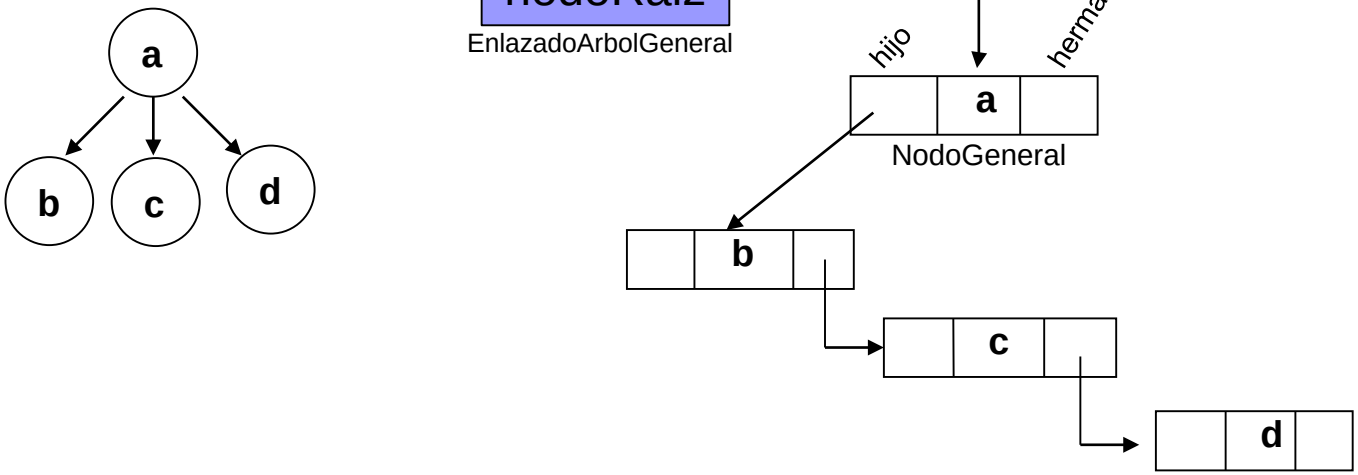
  - Se define una clase NodoGeneral<E>, idéntica a la clase NodoBinario<E>, en la que se han cambiado el nombre a las referencias:
    - la referencia al hijo izquierdo en NodoBinario se llama *hijo* en la clase NodoGeneral,
    - la referencia al hijo derecho en NodoBinario se llama *hermano* en NodoGeneral.

Nodo General

```
public class NodoGeneral<E>{
        private E elemento;                     // referencia al elemento del nodo
        private NodoGeneral<E> hijo;            // referencia al nodo hijo mas a la izquierda
        private NodoGeneral<E> hermano;         // referencia al nodo hermano derecha

        public NodoGeneral (E e, NodoGeneral<E> hi, NodoGeneral<E> her){
                elemento = e;
                hijo = hi;
                hermano= her;
        }
        public E getElemento() {
                return elemento;
        }
        public NodoGeneral<E> getHijo() {
                return hijo;
        }
        public NodoGeneral<E> getHer() {
                return hermano;
        }
        public void setElemento(E e) {
                elemento = e;
        }
        public void setIzq(NodoGeneral<E> hi) {
                hijo = hi;
        }
        public void setDer(NodoGeneral<E> her) {
                hermano = her;
        }
}
```

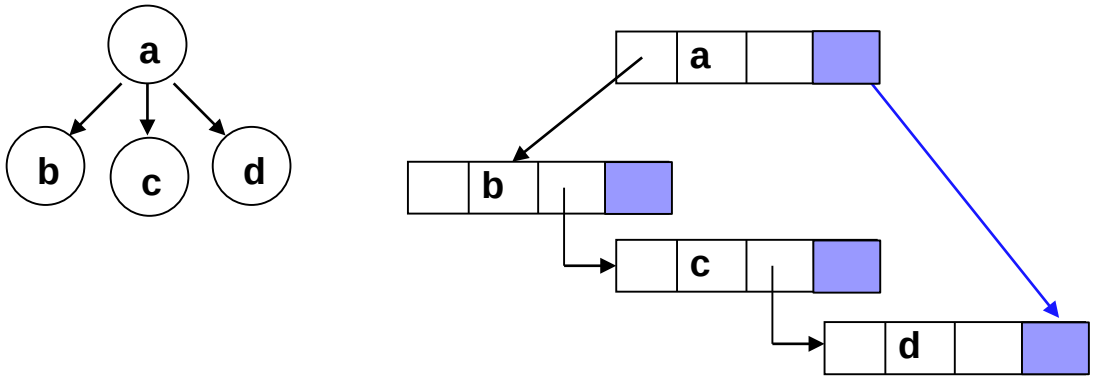
Implementación árbol general




```
public class EnlazadoArbolGeneral<E> implements ArbolGeneral<E> {
        private NodoGeneral<E> nodoRaiz;
```

Otras implementaciones

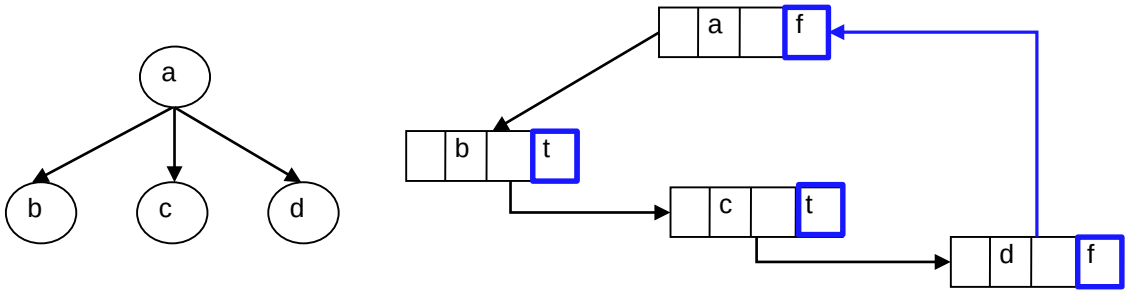
- Si se necesita conocer con frecuencia cuál es el último hijo (hijo más a la derecha) de un determinado elemento, una solución consiste en usar una referencia extra en cada nodo para apuntar a su hijo más a la derecha.




```
public class NuevoNodoGeneral<E>{
    private E elemento;                          // referencia al elemento del nodo
    private NuevoNodoGeneral<E> hijoMasIzq;      // referencia al nodo hijo mas a la izquierda
    private NuevoNodoGeneral<E> hermano;         // referencia al nodo hermano derecha
    private NuevoNodoGeneral<E> hijoMasDer;      // referencia al nodo hijo mas a la derecha
```

Otras implementaciones

- Si se necesita conocer con frecuencia cuál es el padre de un determinado elemento, una solución consiste en usar la referencia hermano que está a *null* en el último hijo para apuntar al nodo padre. En este caso también necesitaríamos un atributo extra (boolean) en cada nodo para indicar si hemos llegado al final en la lista de hijos o no.



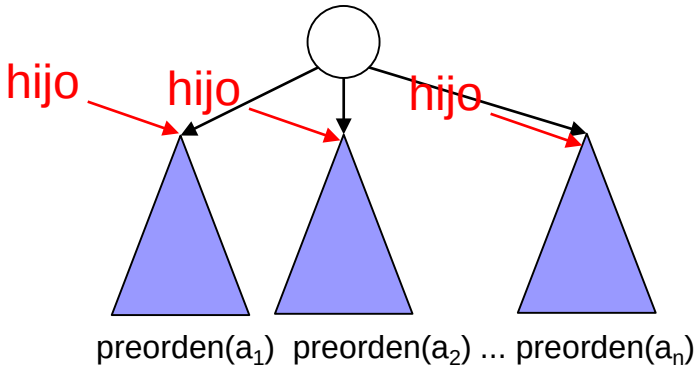

```
public class NuevoNodoGeneral2<E>{
    private E elemento;                                  // referencia al elemento del nodo
    private NuevoNodoGeneral2<E> hijoMasIzq;             // referencia al nodo hijo mas a la izquierda
    private NuevoNodoGeneral2<E> hermano;                // referencia al nodo hermano derecha
    private boolean esHermano;                           // true si el nodo por la derecha es hermano;
                                                         // false en caso contrario
```

Uso TAD Árbol General

Recorridos en profundidad:

Árbol vacío

Árbol no vacío




```
public static <E> void preorden(ArbolGeneral <E> a){
        if (!a.esVacio()) {
                System.out.print(a.raiz() + " ");
                ArbolGeneral<E> hijo = a.hijoMasIzq();
                while (!hijo.esVacio()){
                        preorden(hijo);
                        hijo =hijo.hermanoDer();
                }
        }
}
```

Uso TAD Árbol General

- Cuenta el número de nodos de un árbol general:

```
public static int contarNodos (ArbolGeneral<Integer> a){
        if (a.esVacio())
           return 0;
        else {
            int cont = 1;
            ArbolGeneral<Integer> hijo = a.hijoMasIzq();
            while ( ! hijo.esVacio()){
                cont+= contarNodos(hijo);
                hijo = hijo.hermanoDer();
            }
            return cont;
        }
    }
```

Uso TAD Árbol General

- Cuenta el número de hojas de un árbol general:

```
public static <E> int numHojas(ArbolGeneral<E> a){
        if (a.esVacio()) return 0;
        else if (a.hijoMasIzq().esVacio()) return 1;
                else {
                    int cont = 0;
                    ArbolGeneral<E> hijo = a.hijoMasIzq();
                    while ( ! hijo.esVacio() )  {
                        cont += numHojas(hijo);
                        hijo = hijo.hermanoDer();
                     }
                     return cont;
                }
}
```

Uso TAD Árbol General

- Determina si dos árboles generales son idénticos:

```
public static <E> boolean identicos(ArbolGeneral<E> a, ArbolGeneral<E> b){

    if (a.esVacio() && b.esVacio())
        return true;
    if (!a.esVacio() && !b.esVacio())
        if (!a.raiz().equals(b.raiz())) return false;
        else {
            ArbolGeneral<E> ha = a.hijoMasIzq();
            ArbolGeneral<E> hb = b.hijoMasIzq();
            while ( ! ha.esVacio() && ! hb.esVacio() && identicos(ha,hb))
            {
                    ha = ha.hermanoDer();
                    hb = hb.hermanoDer();
            }
            return ha.esVacio() && hb.esVacio();
        }
    return false;
}
```