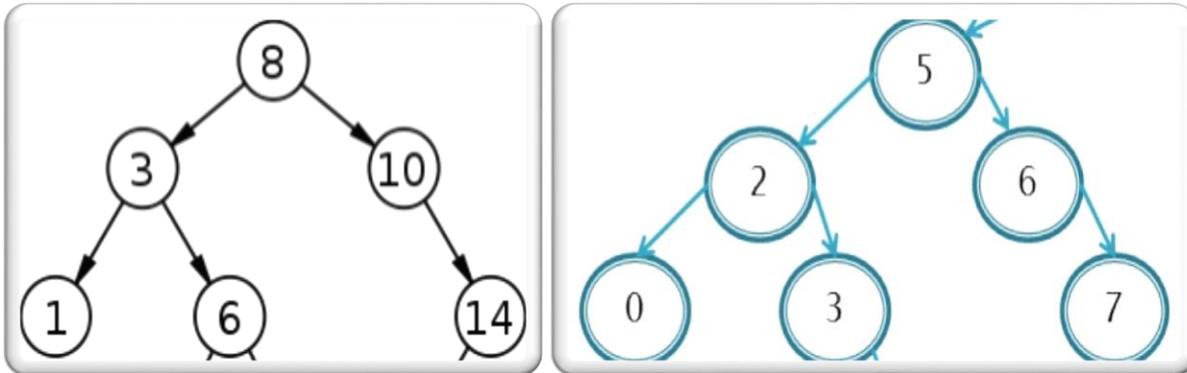


[Estructuras de Datos]



ARBOLES BINARIOS.

Copyright

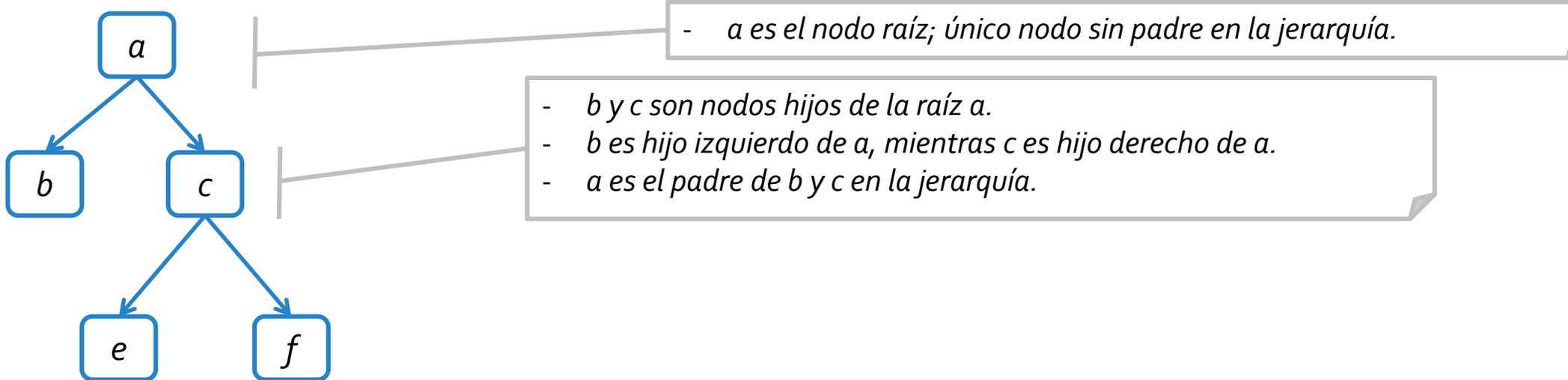
- Copyright © 2019 Ing. Federico Joaquín (federico.joaquin@cs.uns.edu.ar)
- El uso total o parcial de este material está permitido siempre que se haga mención explícita de su fuente: **“Notas de Clase. Estructuras de Datos.” Federico Joaquín. Universidad Nacional del Sur. (c) 2019.**
- Las presentes transparencias constituyen una guía acotada y simplificada de la temática abordada, y deben utilizarse únicamente como material adicional o de apoyo a la bibliografía indicada en el programa de la materia.

ÁRBOLES BINARIOS

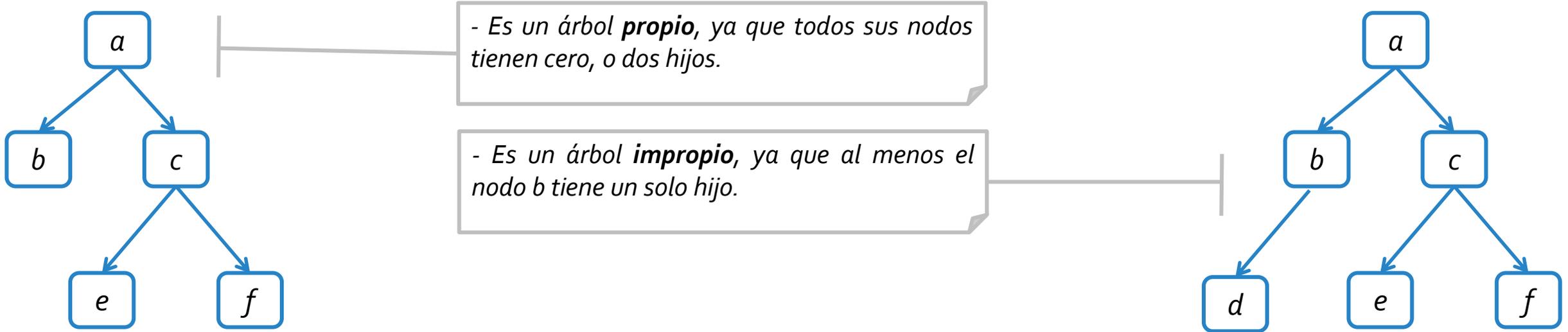
Introducción: ¿qué es un árbol binario?

- Un árbol binario es un TDA que almacena una colección de elementos de forma jerárquica.
- Los elementos de un árbol son representados mediante nodos.
- Cada nodo n, a excepción de uno que se distingue como raíz, mantiene una única referencia a un nodo m que se considera padre de n en la jerarquía.
- Todos los nodos pueden tener cero, uno o dos nodos que se consideran hijos en la jerarquía.
- Luego, un árbol binario es un árbol donde cada nodo tiene a lo sumo dos hijos y:
 - Un nodo hijo es o bien hijo izquierdo, o bien hijo derecho de su padre.
 - El hijo izquierdo precede al hijo derecho en el orden de los hijos de un nodo.

Introducción: ¿qué es un árbol binario?

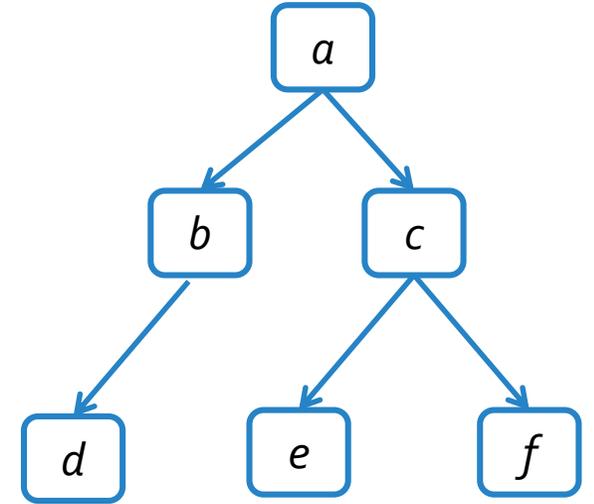
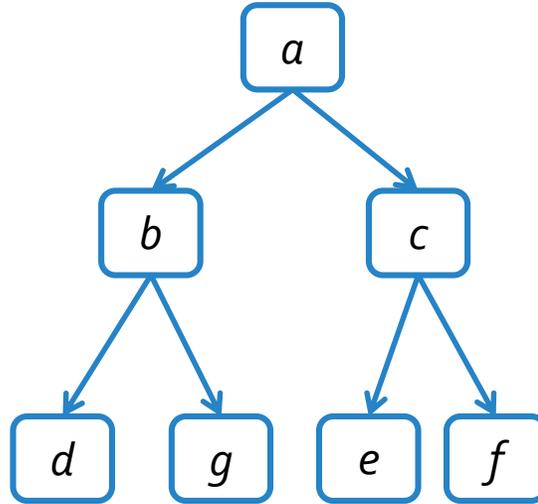
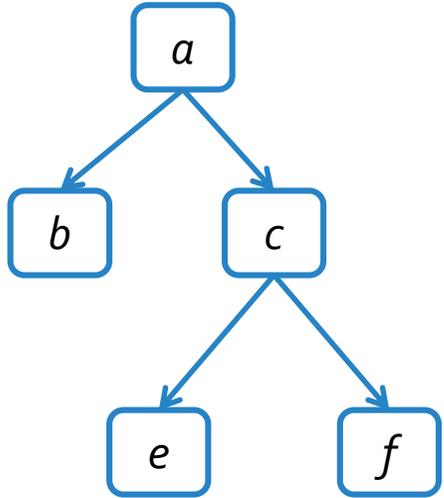


Introducción: ¿qué es un árbol binario?



- Un árbol binario se dice propio si cada nodo tiene cero o dos hijos.
- Un árbol binario se dice impropio si no es un árbol binario propio.

Introducción: ¿qué es un árbol binario?



- Un árbol binario se dice propio si cada nodo tiene cero o dos hijos.
- Un árbol binario se dice impropio si no es un árbol binario propio.
- Un árbol binario se dice lleno si todo nodo (excepto las hojas) tiene dos hijos.

TDA ARBOL BINARIO

TDA Arbol Binario

- Se puede definir un tipo de dato [Arbol Binario](#) que indique qué métodos lo define.
- Al igual que en el TDA Lista, se utilizará el concepto de [Position](#) para referenciar elementos del árbol.

```
public interface BinaryTree<E> extends Iterable<E>{
    //Operaciones de consulta y manipulación.
    public int size();
    public boolean isEmpty();
    public Iterator<E> iterator();
    public Iterable<Position<E>> positions();
    public E replace(Position<E> v, E e) throws InvalidPositionException;
    public Position<E> root() throws EmptyTreeException;
    public Position<E> parent(Position<E> v) throws InvalidPositionException, BoundaryViolationException;
    public boolean isInternal(Position<E> v) throws InvalidPositionException;
    public boolean isExternal(Position<E> v) throws InvalidPositionException;
    public boolean isRoot(Position<E> v) throws InvalidPositionException;
    ...
}
```

TDA Arbol Binario

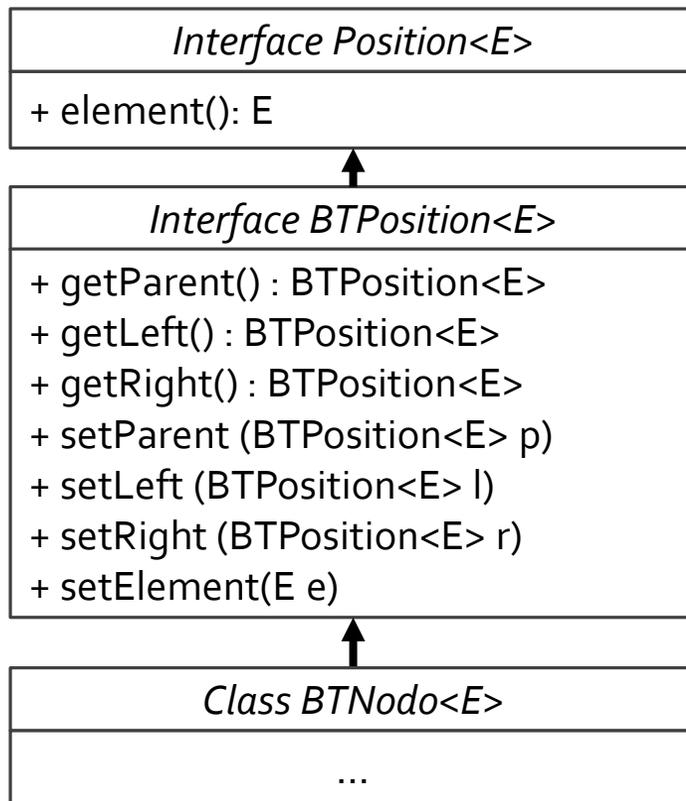
- Se puede definir un tipo de dato [Arbol Binario](#) que indique qué métodos lo define.
- Al igual que en el TDA Lista, se utilizará el concepto de [Position](#) para referenciar elementos del árbol.

```
public interface BinaryTree<E> extends Iterable<E>{
    //Operaciones para creación y modificación.
    public Position<E> left(Position<E> v) throws InvalidPositionException, BoundaryViolationException;
    public Position<E> right(Position<E> v) throws InvalidPositionException, BoundaryViolationException;
    public boolean hasLeft(Position<E> v) throws InvalidPositionException;
    public boolean hasRight(Position<E> v) throws InvalidPositionException;
    public Position<E> createRoot(E r) throws InvalidOperationException;
    public Position<E> addLeft(Position<E> v, E r) throws InvalidOperationException, InvalidPositionException;
    public Position<E> addRight(Position<E> v, E r) throws InvalidOperationException, InvalidPositionException;
    public E remove(Position<E> v) throws InvalidOperationException, InvalidPositionException;
    public void attach(Position<E> r, BinaryTree<E> T1, BinaryTree<E> T2) throws InvalidPositionException;
}
```

TDA ARBOL BINARIO:: IMPLEMENTACIÓN MEDIANTE NODOS CON REF. AL PADRE e HIJOS

ED BTNodo

- Bajo esta implementación, se define una ED **BTNodo** que mantiene un rótulo, así como una referencia a tres nodos que representan los nodos **padre** e **hijo izquierdo** y **derecho**, respectivamente, del nodo modelado.

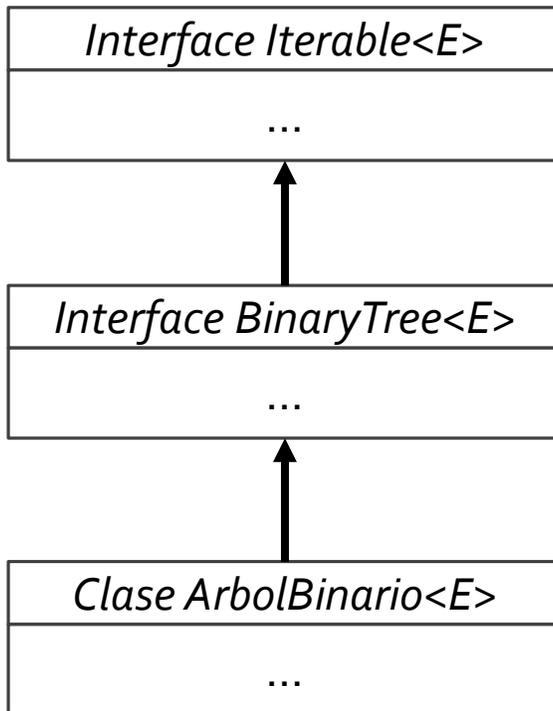


```
public class BTNodo<E> implements BTPosition<E> {
    protected E elemento;
    protected BTPosition<E> padre, hi, hd;

    public BTNodo(E el, BTPosition<E> p){
        elemento = el;
        padre = p;
    }
    public BTPosition<E> getParent(){ return padre; }
    public BTPosition<E> getLeft(){ return hi; }
    public BTPosition<E> getRight(){ return hd; }
    public E element() { return elemento; }
    public void setParent(BTPosition<E> p){ padre = p; }
    public void setLeft(BTPosition<E> l){ hi = l; }
    public void setRight(BTPosition<E> r){ hd = r; }
    public void setElement(E e){ elemento = e; }
}
```

ED Arbol Binario

- Bajo esta implementación, un árbol binario se define manteniendo referencia a una BTPosition que representa la raíz. A partir de este, luego se puede acceder a todos los restantes nodos.



```
public class ArbolBinario<E> implements BinaryTree<E> {
    protected BTPosition<E> raiz;
    protected int tamaño;

    public ArbolBinario(){
        tamaño = 0;
        raiz = null;
    }

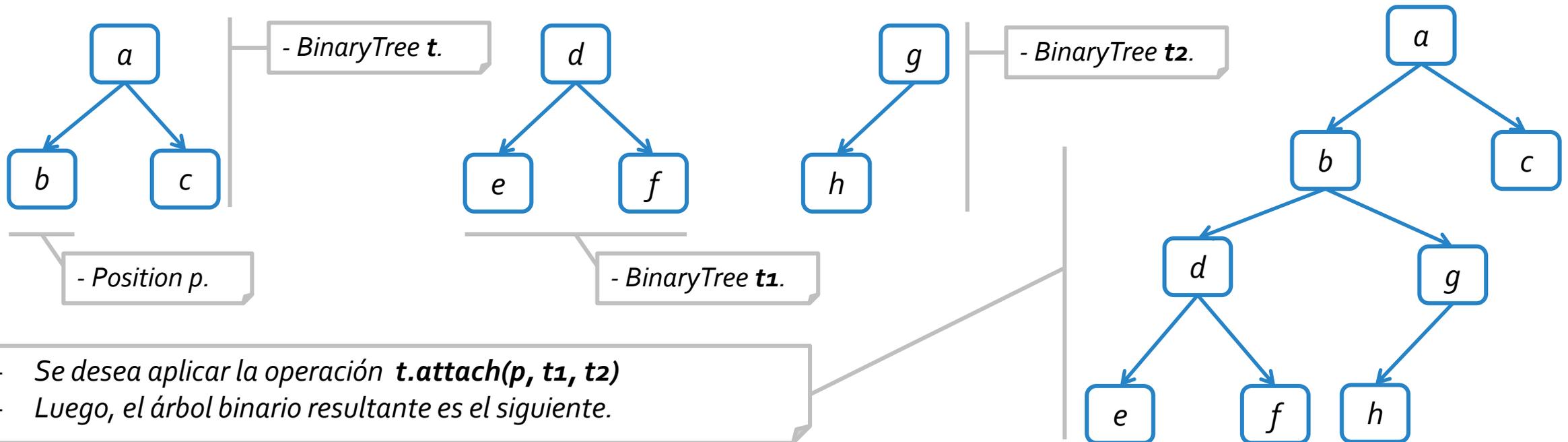
    public ArbolBinario(E rot){
        tamaño = 1;
        raiz = new BTNodo<E>(rot, null);
    }
    ...
}
```

TDA ARBOL BINARIO :: IMPLEMENTACIÓN DE ATTACH

Implementación operación attach

- El TDA Árbol Binario considera una operación que permiten insertar como subárboles de un dado nodo sin hijos, a dos árboles binarios parametrizados.

```
public interface Tree<E> extends Iterable<E>{  
    //Operaciones para creación y modificación.  
    public void attach(Position<E> p, BinaryTree<E> t1, BinaryTree<E> t2) throws InvalidPositionException;  
}
```



Implementación operación attach

- En *Goodrich & Tamassia* se considera una implementación que en la materia consideramos **incorrecta** de la operación **attach**.

```
protected void attach_mal(Position<E> p, BinaryTree<E> t1, BinaryTree<E> t2) throws InvalidPositionException {
    BTPosition<E> raiz_local = checkPosition(p);
    if (raiz_local.getLeft() != null || raiz_local.getRight() != null)
        throw new InvalidPositionException("La posicion no corresponde a un nodo hoja");

    BTPosition<E> raiz_t1 = null, raiz_t2 = null;

    try {
        if (!t1.isEmpty())
            raiz_t1 = checkPosition(t1.root());
        if (!t2.isEmpty())
            raiz_t2 = checkPosition(t2.root());

        raiz_local.setLeft(raiz_t1);
        raiz_t1.setParent(raiz_local);
        raiz_local.setRight(raiz_t2);
        raiz_t2.setParent(raiz_local);
        tamaño += t1.size() + t2.size();
    } catch (EmptyTreeException e) {}
}
```

- Mediante el método `checkPosition`, se produce un **acceso a la estructura** de `t1` y `t2` que es **totalmente incorrecto**.
- No hay **ninguna forma** de asegurar que `t1` y `t2` están implementados en términos de **BTPositions**, por lo que puede que la operación falle.
- Más aún: ¿qué sucede si en `t1` y `t2` se **realizan cambios**? ¿Estos **afectan** al árbol local que acaba de realizar el **attach**?
- Por estas razones, esta implementación es considerada **inválida** ya que corrompe todas las reglas de **encapsulamiento** y utilización de TDAs que estudiamos a lo largo del cuatrimestre.

Implementación operación attach

```
public void attach(Position<E> p, BinaryTree<E> t1, BinaryTree<E> t2) throws InvalidPositionException {
    BTPosition<E> raiz_local = checkPosition(p);
    if (raiz_local.getLeft() != null || raiz_local.getRight() != null)
        throw new InvalidPositionException("La posicion no corresponde a un nodo hoja");
    try {
        //Clonación de T1 como subárbol izquierdo
        if (!t1.isEmpty()) {
            Position<E> raiz_t1 = t1.root();
            BTPosition<E> hi_raiz_local = new BTNodo<E>(raiz_t1.element(), raiz_local);
            raiz_local.setLeft(hi_raiz_local);
            clonar(raiz_local.getLeft(), raiz_t1, t1);
        }

        //Clonación de T2 como subárbol derecho
        if (!t2.isEmpty()) {
            Position<E> raiz_t2 = t2.root();
            BTPosition<E> hd_raiz_local = new BTNodo<E>(raiz_t2.element(), raiz_local);
            raiz_local.setRight(hd_raiz_local);
            clonar(raiz_local.getRight(), raiz_t2, t2);
        }
        tamaño += T1.size() + t2.size();
    } catch (EmptyTreeException e) { raiz_local.setLeft(null); raiz_local.setRight(null); }
}
```

Implementación operación attach

```
protected void clonar(BTPosition<E> padre_local, Position<E> padre_t, BinaryTree<E> t) {
    try {
        //Si existe hijo izquierdo en T de padre_t, se clona este y el subárbol a partir del hijo izquierdo de padre_t.
        if (t.hasLeft(padre_t)) {
            Position<E> hi_padre_t = t.left(padre_t);
            BTPosition<E> hi_padre_local = new BTNodo<E>(hi_padre_t.element(), padre_local);
            padre_local.setLeft(hi_padre_local);
            clonar(hi_padre_local, hi_padre_t, t);
        }

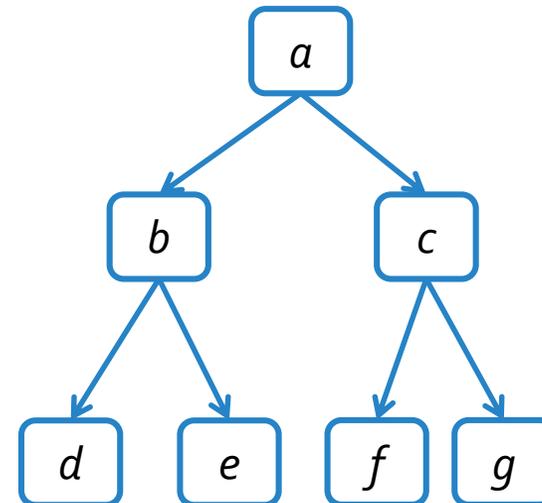
        //Si existe hijo derecho en T de padre_t, se clona este y el subárbol a partir del hijo derecho de padre_t.
        if (t.hasRight(padre_t)) {
            Position<E> hd_padre_t = t.right(padre_t);
            BTPosition<E> hd_padre_local = new BTNodo<E>(hd_padre_t.element(), padre_local);
            padre_local.setRight(hd_padre_local);
            clonar(hd_padre_local, hd_padre_t, t);
        }
    } catch (InvalidPositionException | BoundaryViolationException e) {
        padre_local.setLeft(null);
        padre_local.setRight(null);
    }
}
```

Problema propuesto.



Clonación de árboles binarios

- Desarrolle un método que, recibiendo la altura A y un iterador de elementos correspondiente a un árbol binario lleno T, **retorne un nuevo árbol binario** cuyos elementos deben ser la clonación de T. Tenga en cuenta que el iterador de elementos recibidos realiza un recorrido en preorden en T.
 - Árbol binario T.
 - Altura (T) = 2
 - Iterador en pre-orden: a, b, d, e, c, f, g.



Creando árbol clonado a partir de iterador

```
public static <E> BinaryTree<E> crear_arbol_completo(Iterator<E> it, int altura) {
    BinaryTree<E> toReturn = new ArbolBinario<E>();
    try {
        if (it.hasNext()) {
            E rotulo = it.next();
            toReturn.createRoot(rotulo);
            if (altura>0)
                clonar_arbol_completo(toReturn, toReturn.root(), it, altura-1);
        }
    } catch(InvalidOperationException | EmptyTreeException e) {
        toReturn = null;
    }
    return toReturn;
}
```

//Sigue en la siguiente slide

Creando árbol clonado a partir de iterador

```
protected static <E> void clonar_arbol_completo(BinaryTree<E> arbol, Position<E> padre, Iterator<E> it, int altura) {
    E rotulo;
    Position<E> nuevoPadre;
    try {
        //Clonación de las hojas de árbol.
        if (altura == 0) {
            rotulo = it.next();
            arbol.addLeft(padre, rotulo);
            rotulo = it.next();
            arbol.addRight(padre, rotulo);
        }else{
            //Clonación de nodos internos del árbol.
            rotulo = it.next();
            nuevoPadre = arbol.addLeft(padre, rotulo);
            clonar_arbol_completo(arbol, nuevoPadre, it, altura-1);
            rotulo = it.next();
            nuevoPadre = arbol.addRight(padre, rotulo);
            clonar_arbol_completo(arbol, nuevoPadre, it, altura-1);
        }
    }catch(InvalidOperationException | InvalidPositionException e){}
}
```

Problemas propuestos.



Clonación de árboles binarios

- Agregue a la clase *ArbolBinario* un constructor que, recibiendo la altura A y un iterador de elementos correspondiente a un árbol binario lleno T , inicialice el árbol binario cuyos elementos deben ser la clonación de T . Tenga en cuenta que el iterador de elementos recibido realiza un recorrido en preorden en T .
- Agregue a la clase *ArbolBinario* un método `clonar()`, que retorne un nuevo *BinaryTree* cuyos elementos son la clonación del árbol binario que recibe el mensaje. Recuerde que está agregando una operación sobre el árbol binario, por lo que tiene total acceso a la ED del mismo, pero no tiene acceso a la ED del nuevo árbol binario a retornar.
- Desarrolle un método que dado un árbol binario T_1 , retorne un nuevo árbol binario T_2 cuyos elementos son la clonación del árbol binario T_1 . Recuerde que está desarrollando una operación que opera por sobre dos árboles binarios, por lo que no puede acceder a su ED en ningún momento y deberá operar a través de las operaciones ofrecidas para tal fin.



Fin de la presentación.