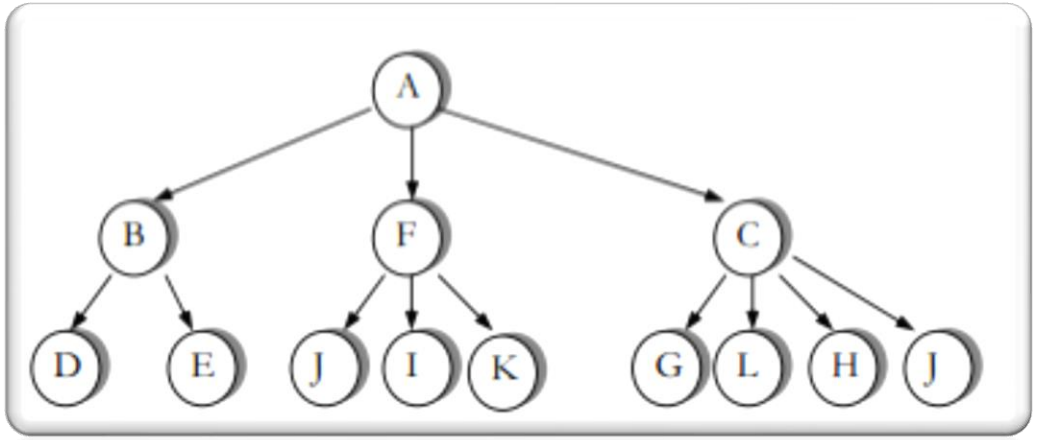
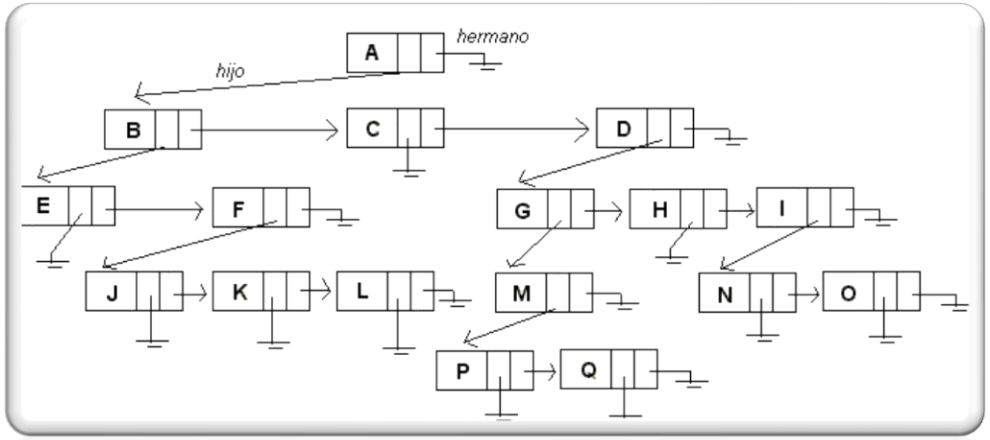
[Estructuras de Datos]





ARBOLES GENERALES.

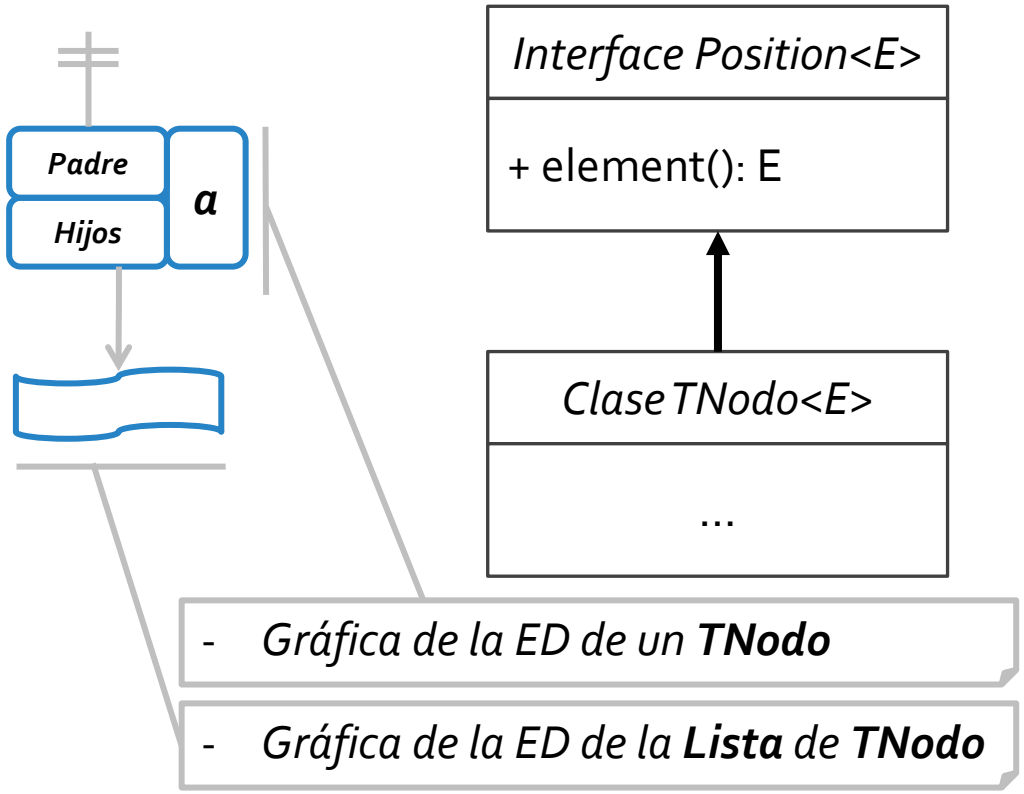
Copyright

IMPLEMENTACIÓN DEL TDA ARBOL MEDIANTE NODOS CON REF. AL PADRE y LISTA DE HIJOS

Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

ED TNodo

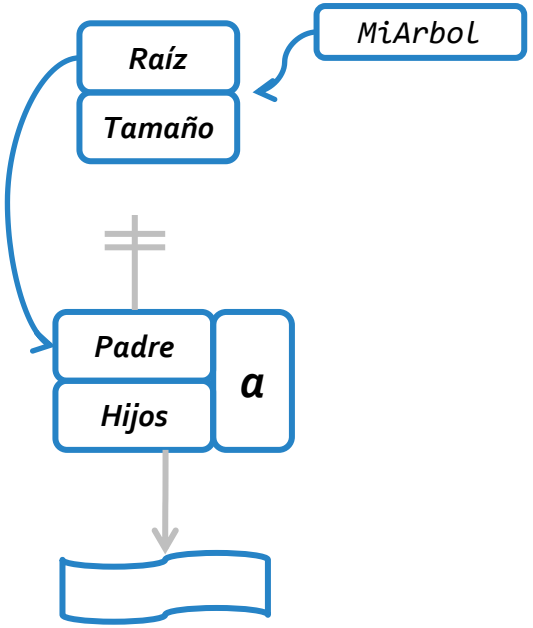
- Bajo esta implementación, se define una ED **TNodo** que mantiene un rótulo, así como una **referencia a un nodo** y una **lista de nodos** que representan los nodos **padre** e **hijos**, respectivamente, del nodo modelado.

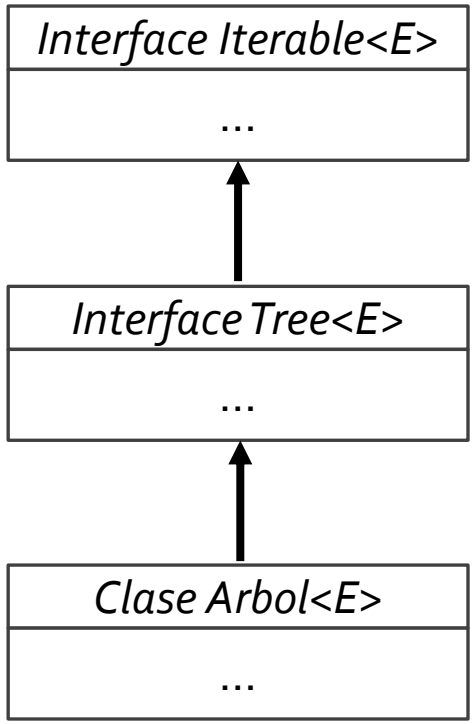



```
public class TNodo<E> implements Position<E> {
  protected E elemento;
  protected TNodo<E> padre;
  protected PositionList<TNodo<E>> hijos;

  public TNodo(E el, TNodo<E> p){
    elemento = el;
    padre = p;
    hijos = new DoubleLinkedList<TNodo<E>>();
  }

  public TNodo<E> getPadre(){ return padre; }
  public PositionList<TNodo<E>> getHijos(){ return hijos; }
  public E element() { return elemento; }
  public void setPadre(TNodo<E> p){ padre = p; }
  public void setElement(E e){ elemento = e; }
}
```

ED Arbol

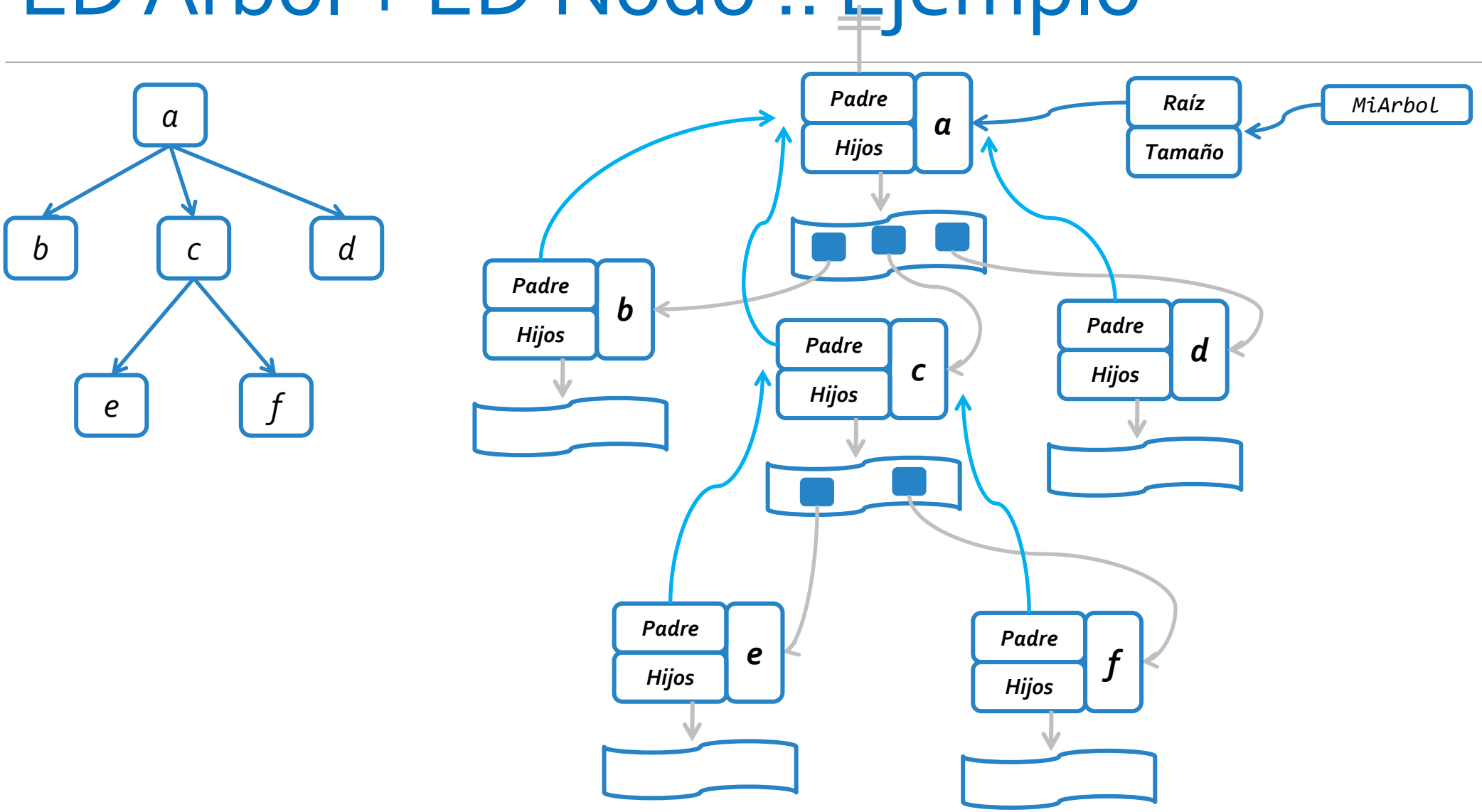
- Bajo esta implementación, un **árbol** se define manteniendo referencia a un **TNodo** que representa la **raíz**. A partir de este, luego se puede acceder a todos los restantes nodos.






```
public class Arbol<E> implements Tree<E> {
  protected TNodo<E> raiz;
  protected int tamaño;

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

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

ED Árbol + ED Nodo :: Ejemplo

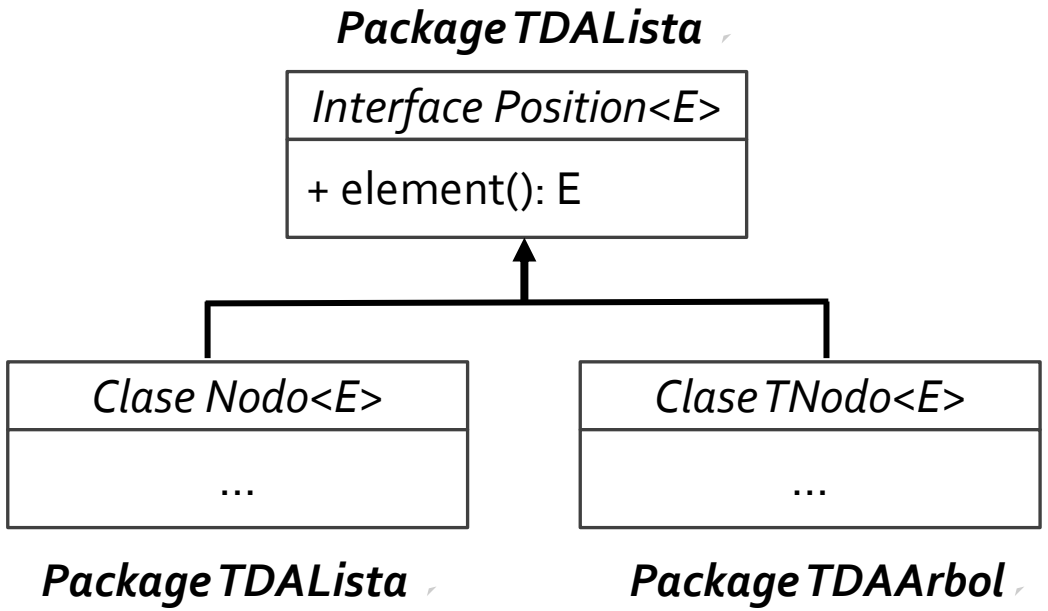


POSITION de LISTA
vs.
POSITION de ÁRBOL

Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

Diferencia entre Positions de árbol y lista

- Una de los **errores** más **frecuentes** en la materia es confundir las **Positions** de **árbol** y las de **lista**.
- En **Estructuras de Datos**, **antes** del 1º cuatrimestre 2020 (***deprecated***)



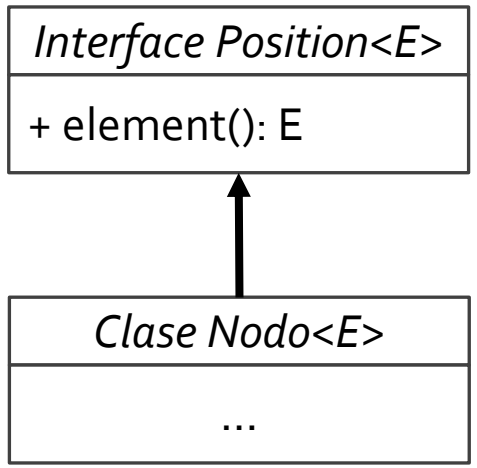

```
...
PositionList<Integer> lista = new LinkedList<Integer>();
Tree<Integer> arbol = new Arbol<Integer>();

lista.addFirst(15);
arbol.createRoot(10);

Position<Integer> pLista = lista.first();
Position<Integer> pArbol = arbol.root();

lista.remove(pArbol);
arbol.addLastChild(pLista, 63);
...
```

- *El compilador **no indicaba** error de compilación, los tipo de datos coinciden.*

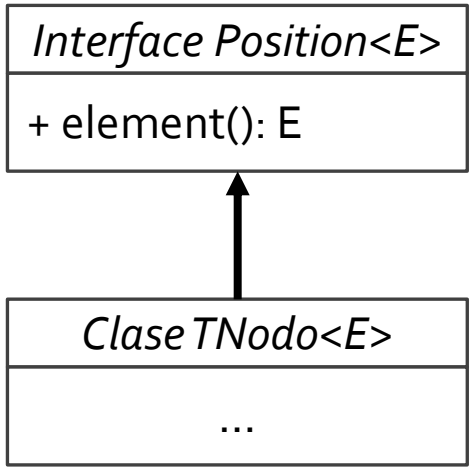
Diferencia entre Positions de árbol y lista

- Una de los **errores** más **frecuentes** en la materia es confundir las **Positions** de **árbol** y las de **lista**.
- En **Estructuras de Datos**, **a partir** del 1º cuatrimestre 2020 (***release***)

Package TDALista



Package TDAArbol

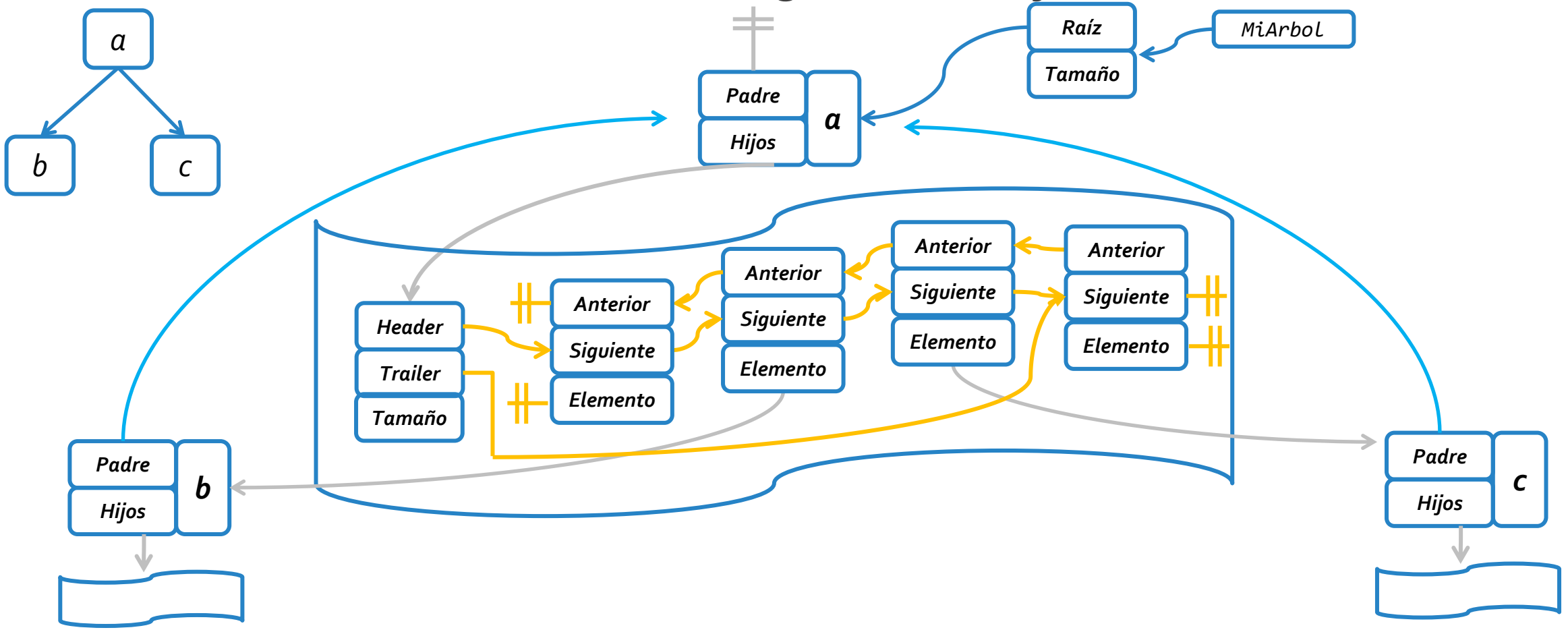



```
...
PositionList<Integer> lista = new LinkedList<Integer>();
Tree<Integer> arbol = new Arbol<Integer>();

lista.addFirst(15);
arbol.createRoot(10);

TDALista.Position<Integer> pLista = lista.first();
TDAArbol.Position<Integer> pArbol = arbol.root();

lista.remove(pArbol);
arbol.addLastChild(pLista, 63);
...
```

- *El compilador **indica** error de compilación, los tipo de datos **no** coinciden.*

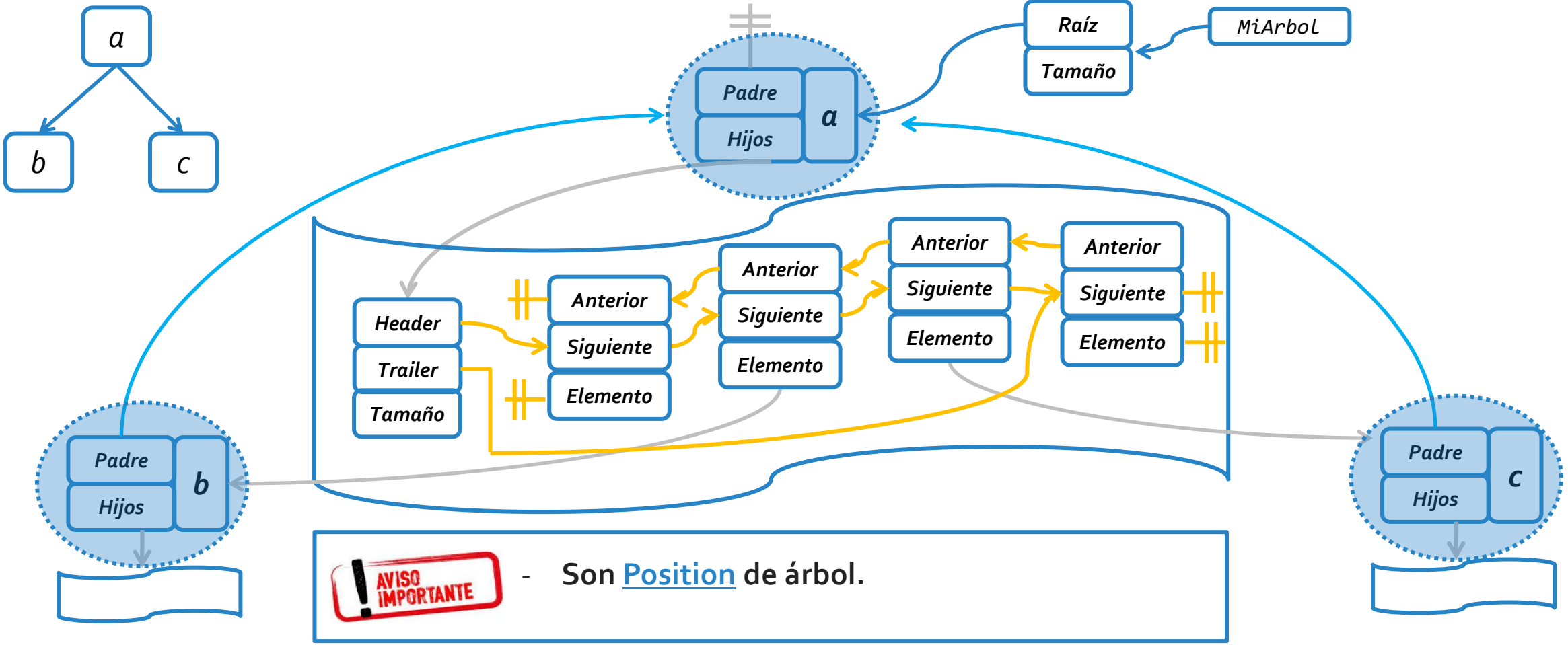
Diferencia entre Positions de árbol y lista

- Analicemos la diferencia en un diagrama de objetos.



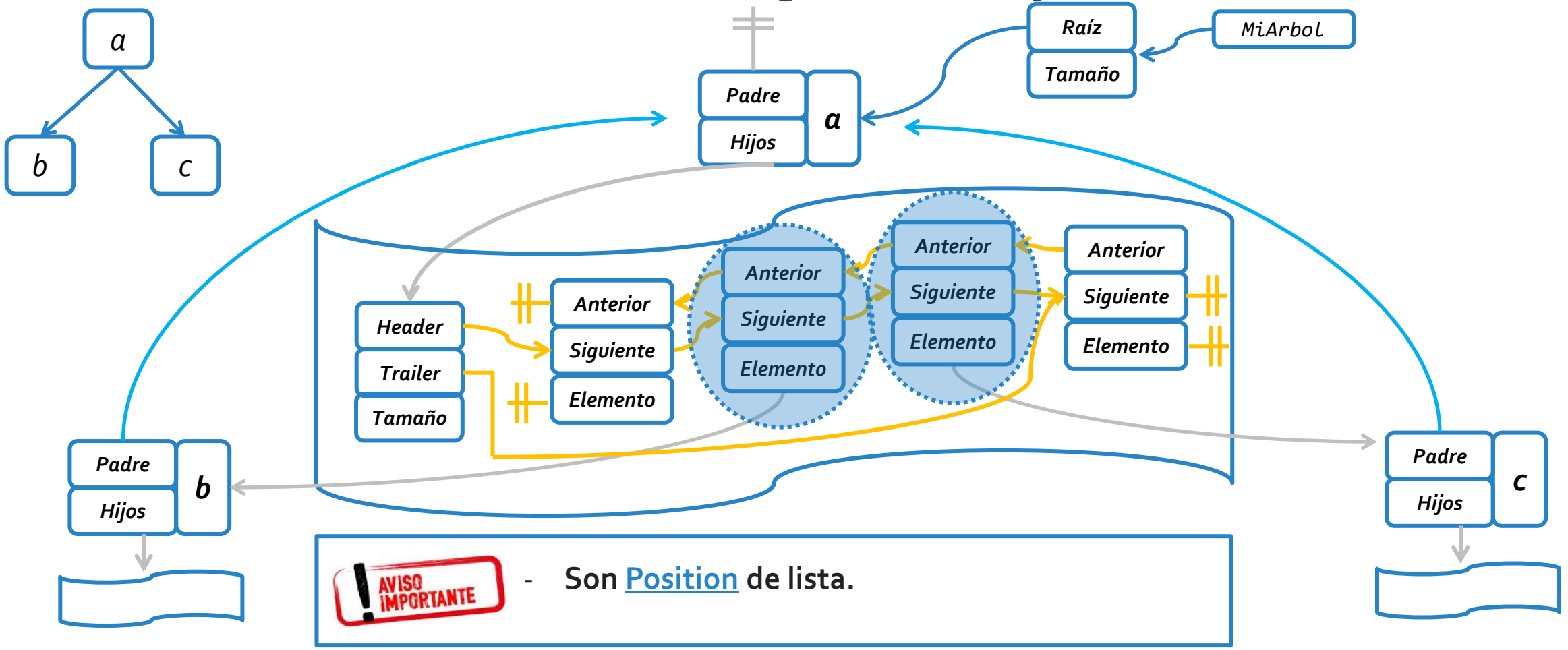
Diferencia entre Positions de árbol y lista

- Analicemos la diferencia en un diagrama de objetos.



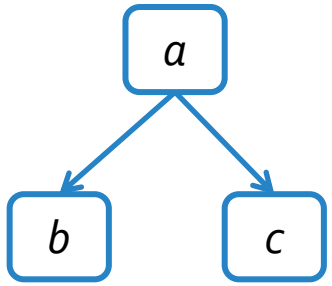
Diferencia entre Positions de árbol y lista

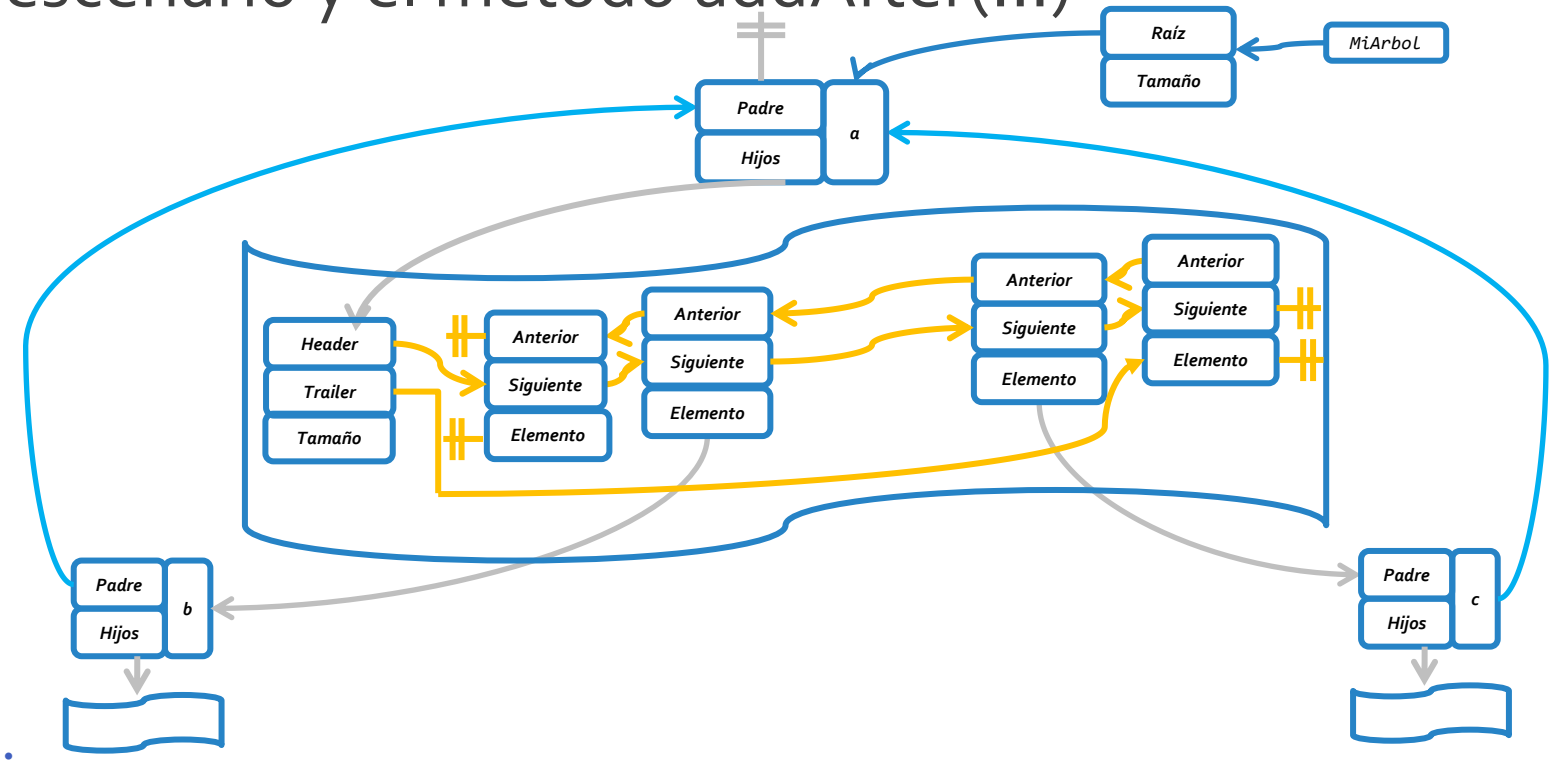
- Analicemos la diferencia en un diagrama de objetos.



Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)

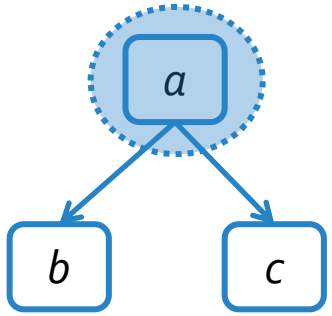


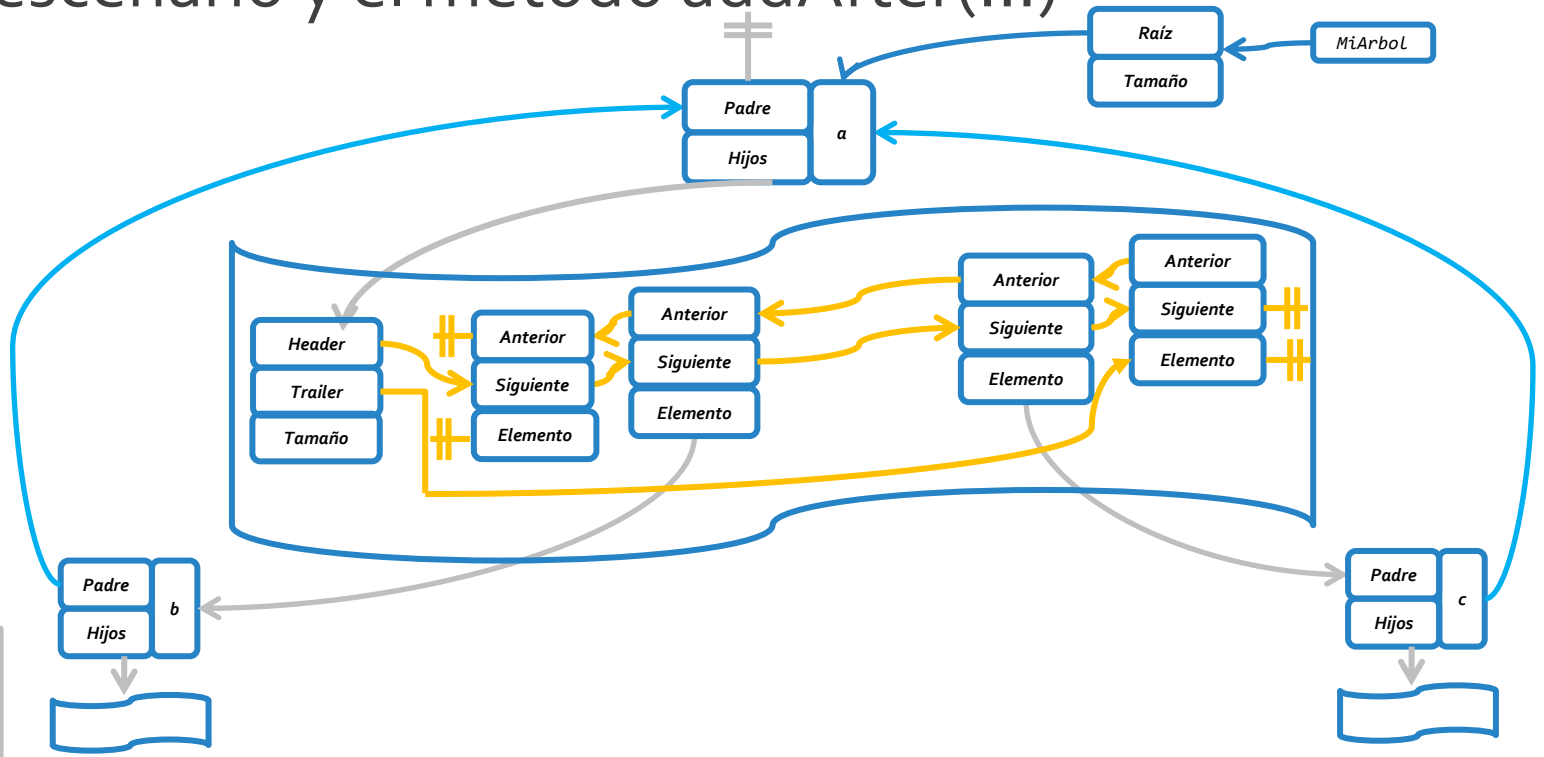



```
/**
 * Agrega un nodo con rótulo e como hijo de
 * un nodo padre dado. El nuevo nodo se
 * agregará a continuación de otro nodo
 * también dado.
 * @param e Rótulo del nuevo nodo.
 * @param p Posición del nodo padre.
 * @param lb Posición del nodo que será el
 * hermano izquierdo del nuevo nodo.
 * @return La posición del nuevo nodo creado.
 * @throws InvalidPositionException si la posición pasada por parámetro es inválida, o el árbol está vacío, o la
 * posición lb no corresponde a un nodo hijo del nodo referenciado por p.
 */
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)





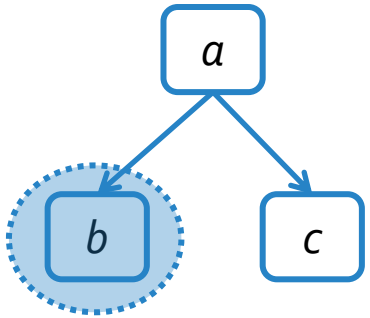
- *Utilizaremos la operación addAfter(p,lb,e).*
- *p será el nodo a.*

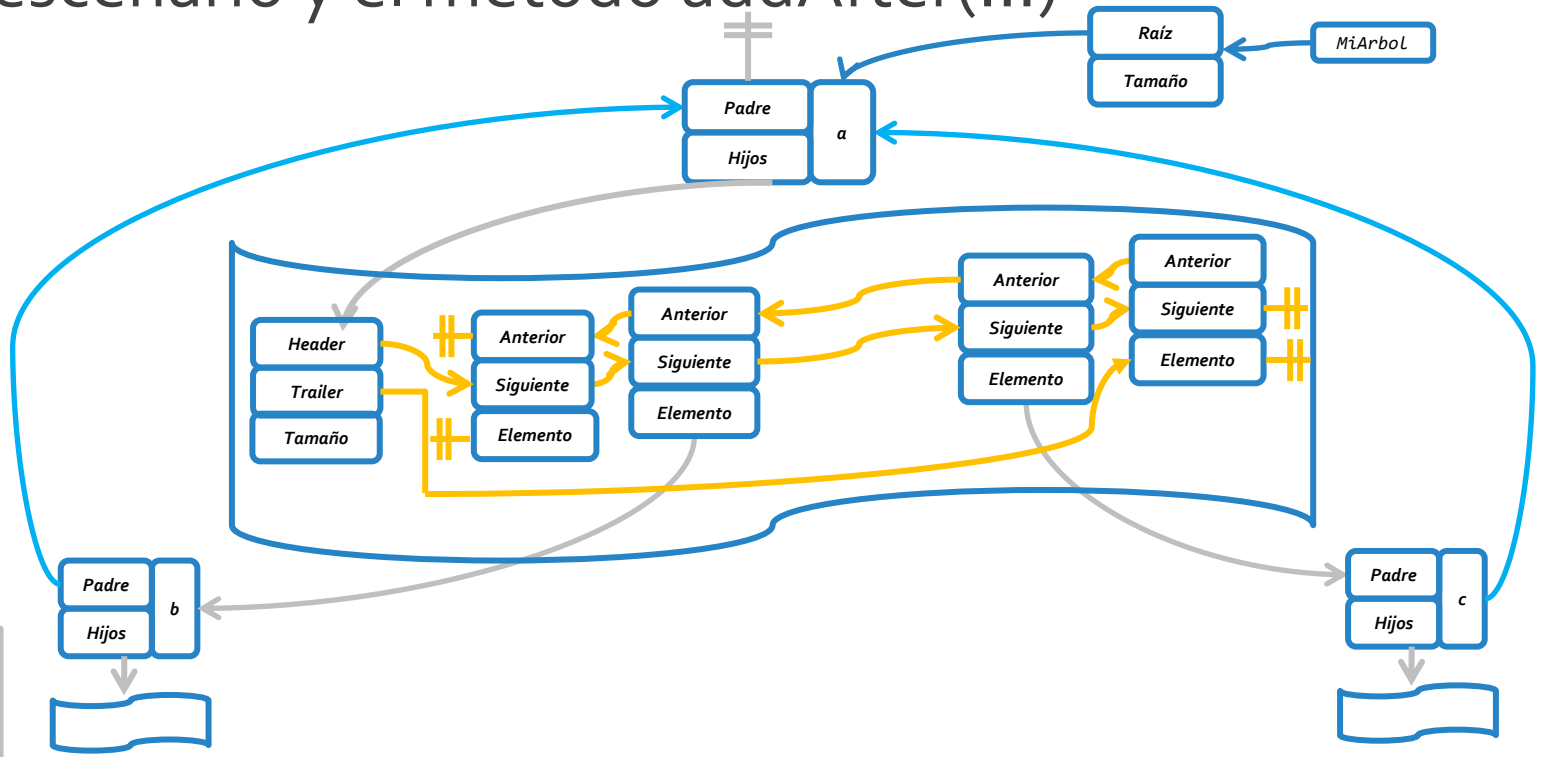
```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)





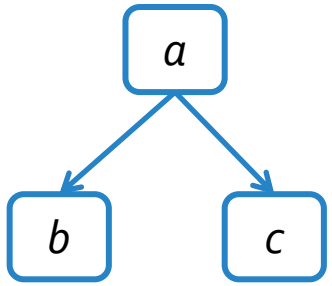
- *Utilizaremos la operación addAfter(p,lb,e).*
- *p será el nodo a.*
- *lb será el nodo b.*

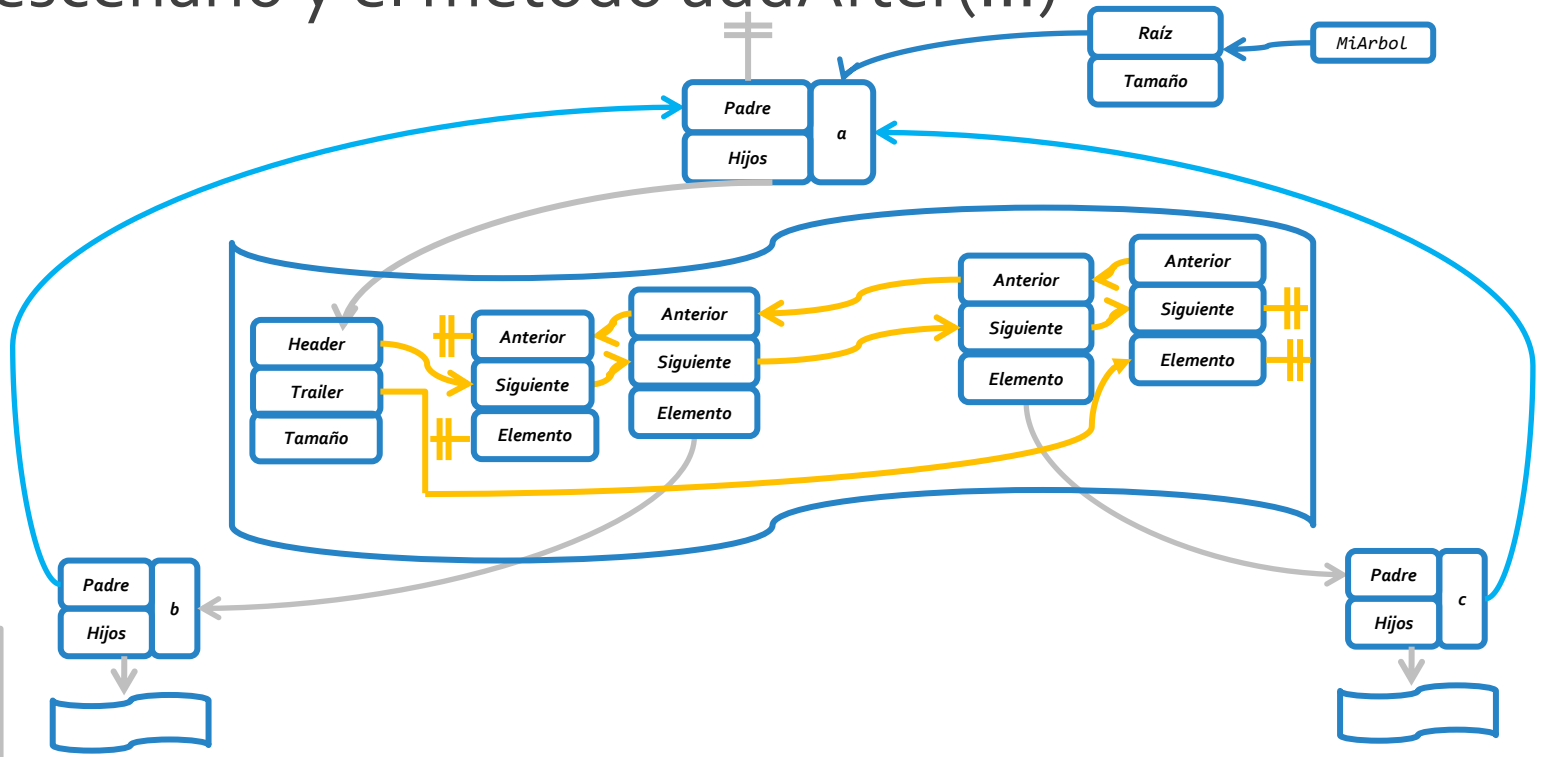
```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)





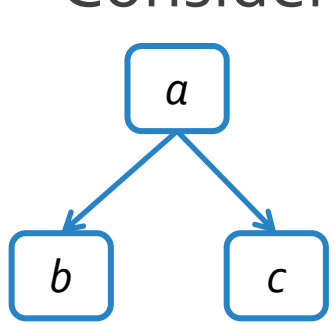
- *Utilizaremos la operación addAfter(p,lb,e).*
- *p será el nodo a.*
- *lb será el nodo b.*
- *e será el rótulo d.*

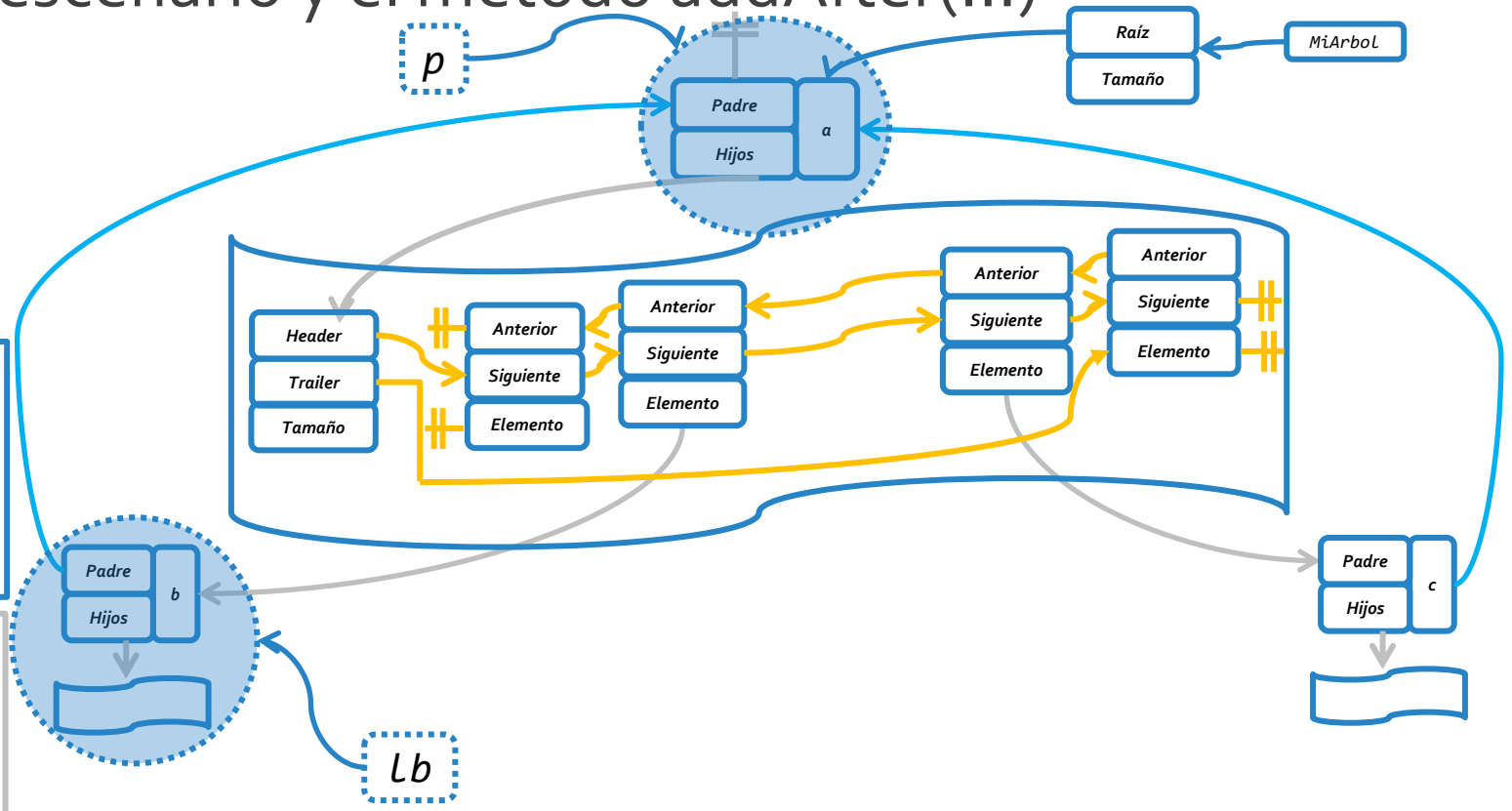
```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)





Asumimos por simplicidad que las posiciones son válidas. En la implementación, esto debe validarse.

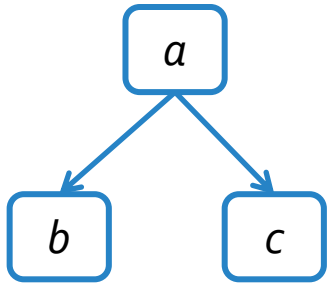
- *Utilizaremos la operación addAfter(p,lb,e).*
- *p será el nodo a.*
- *lb será el nodo b.*
- *e será el rótulo d.*

```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

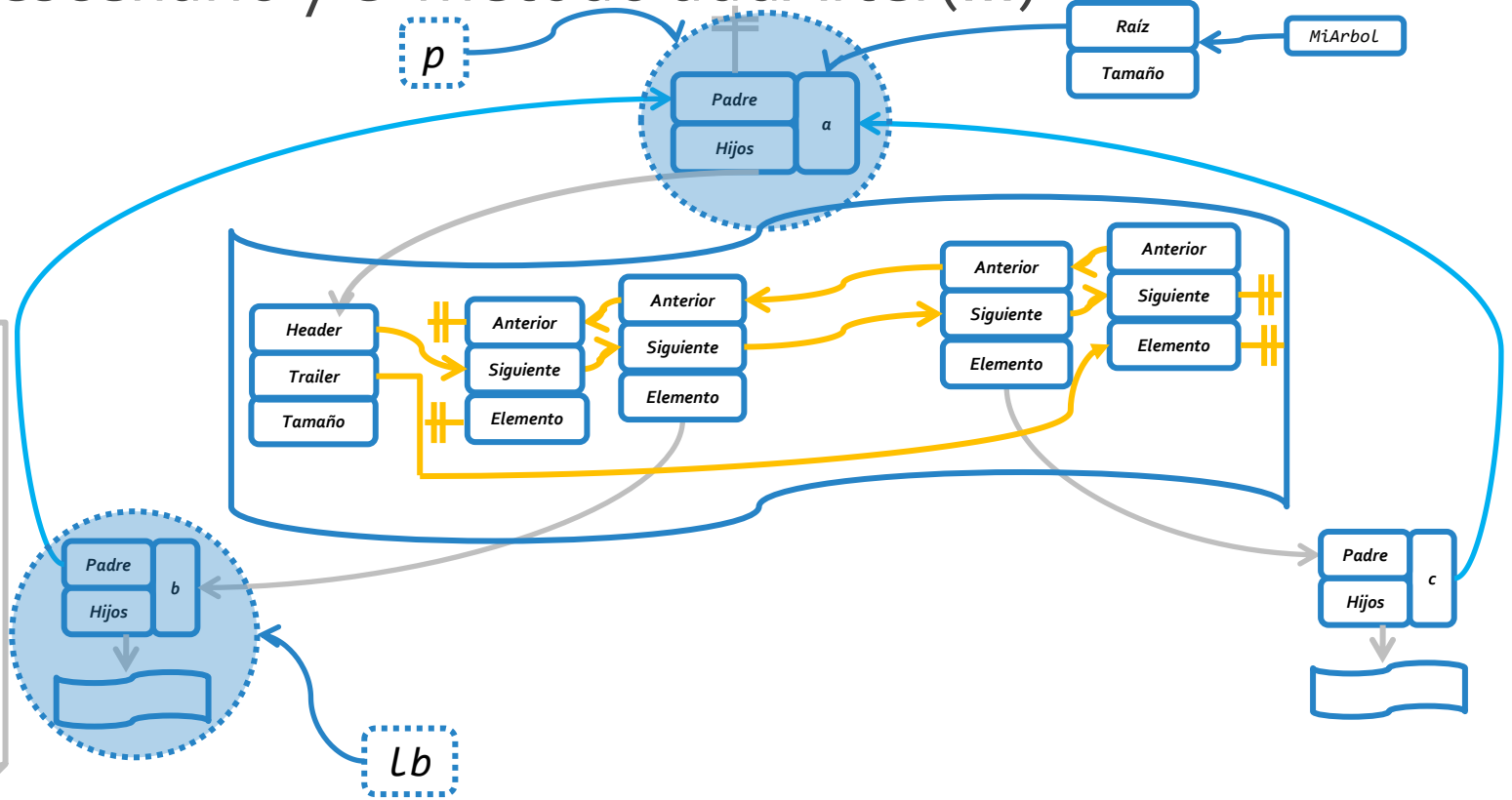
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*

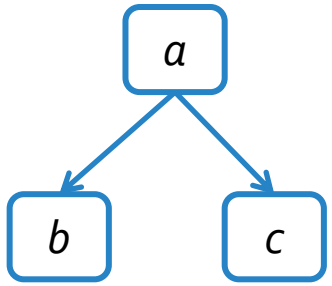



```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

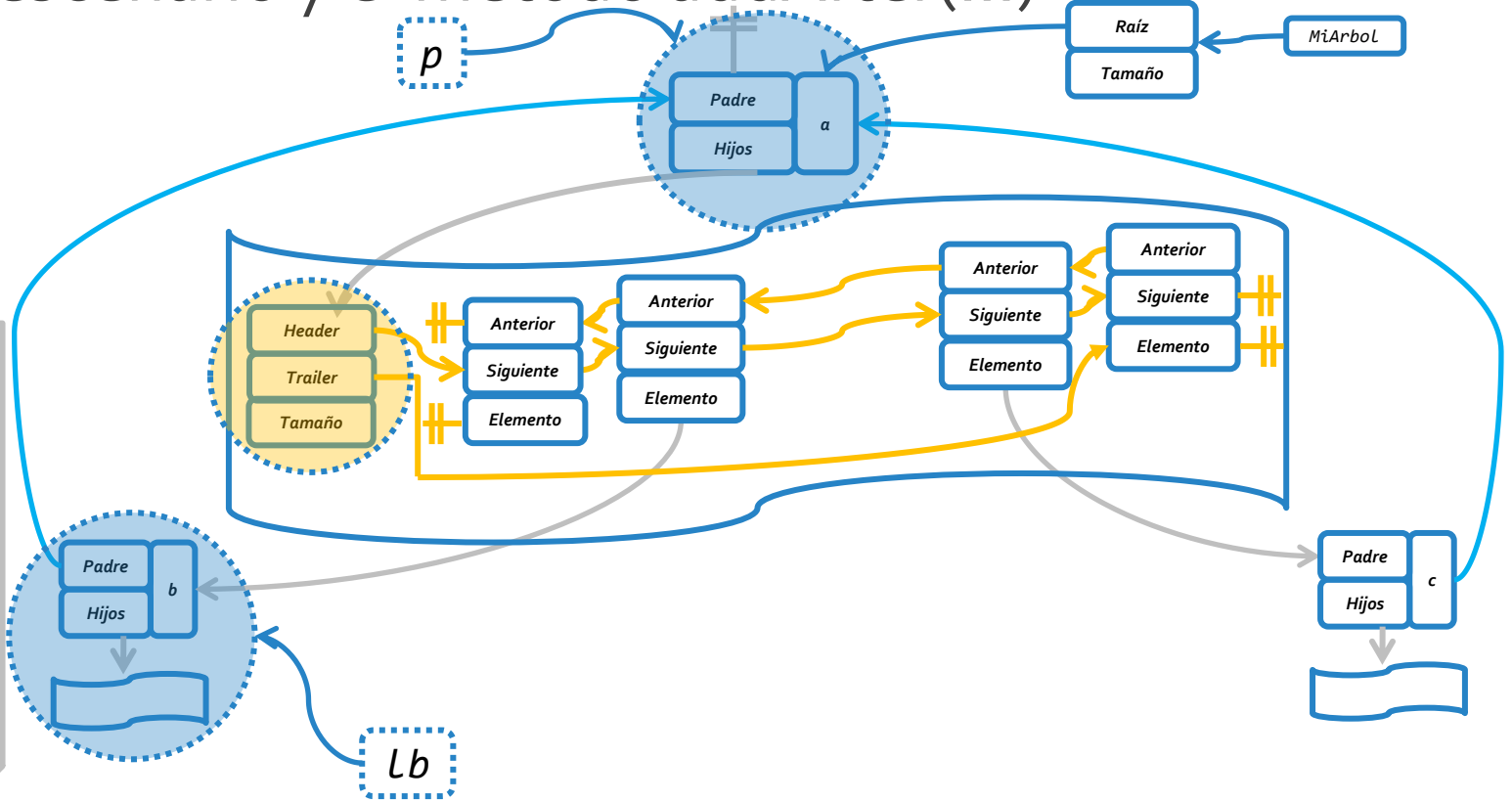
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*

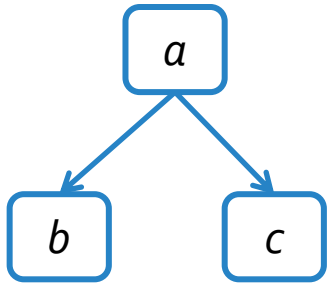



```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

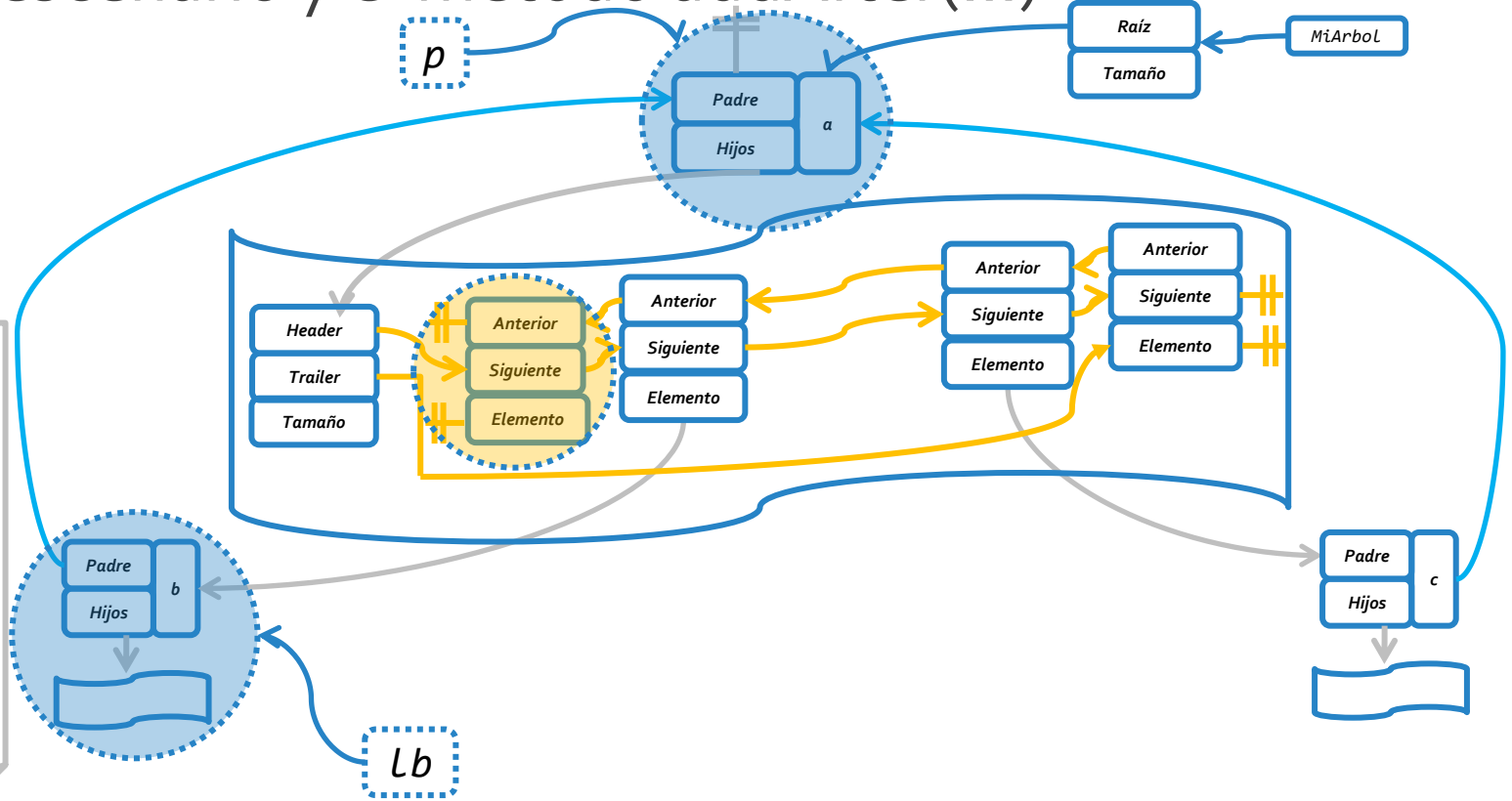
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*

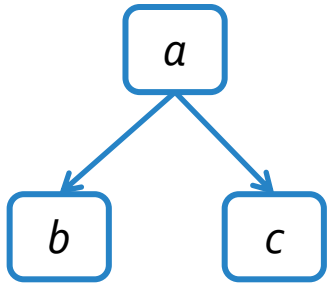



```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

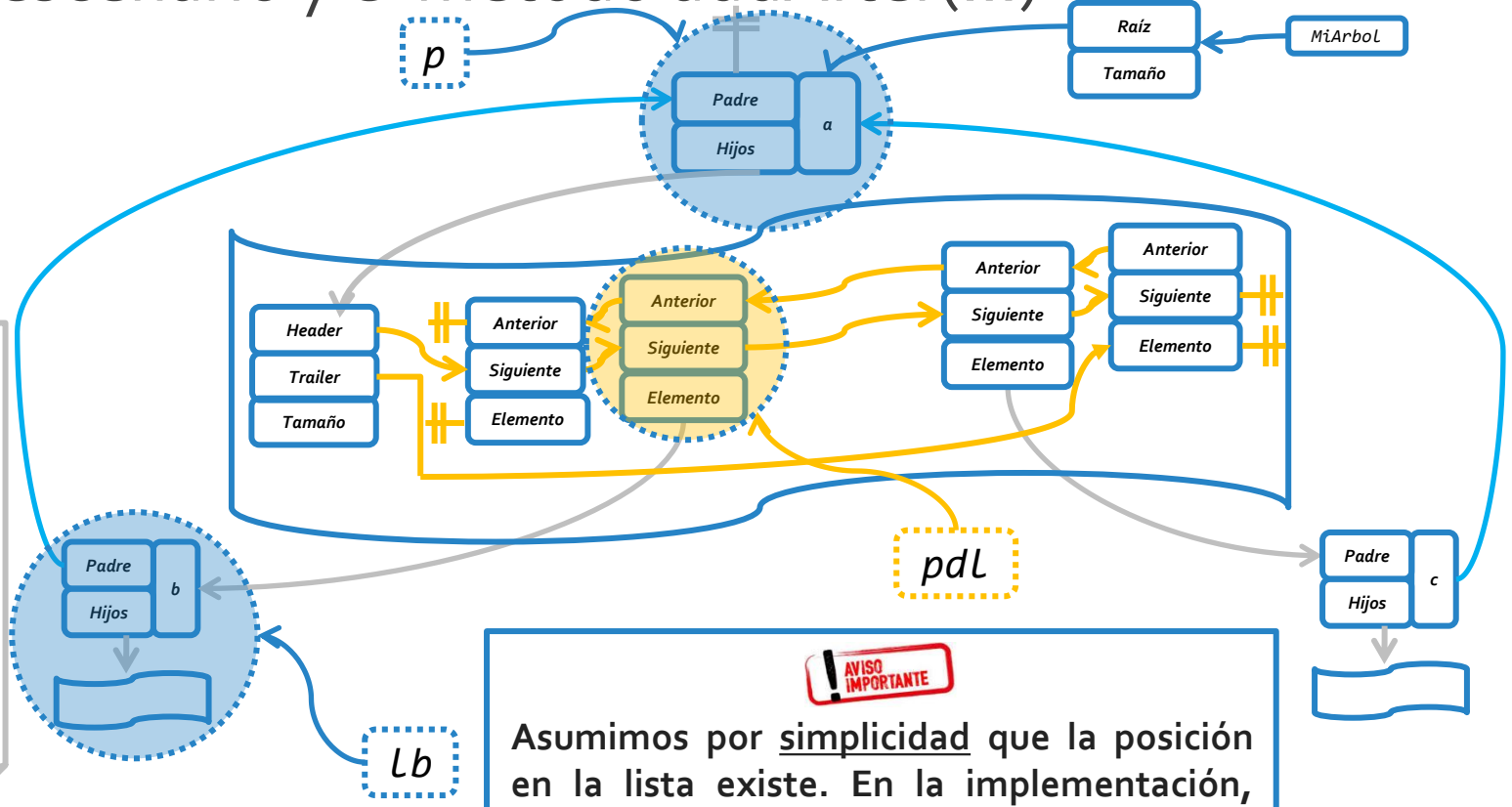
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*



AVISO IMPORTANTE

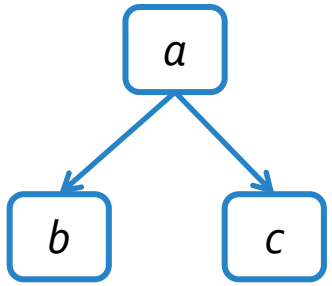
Asumimos por simplicidad que la posición en la lista existe. En la implementación, esto debe validarse.

```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

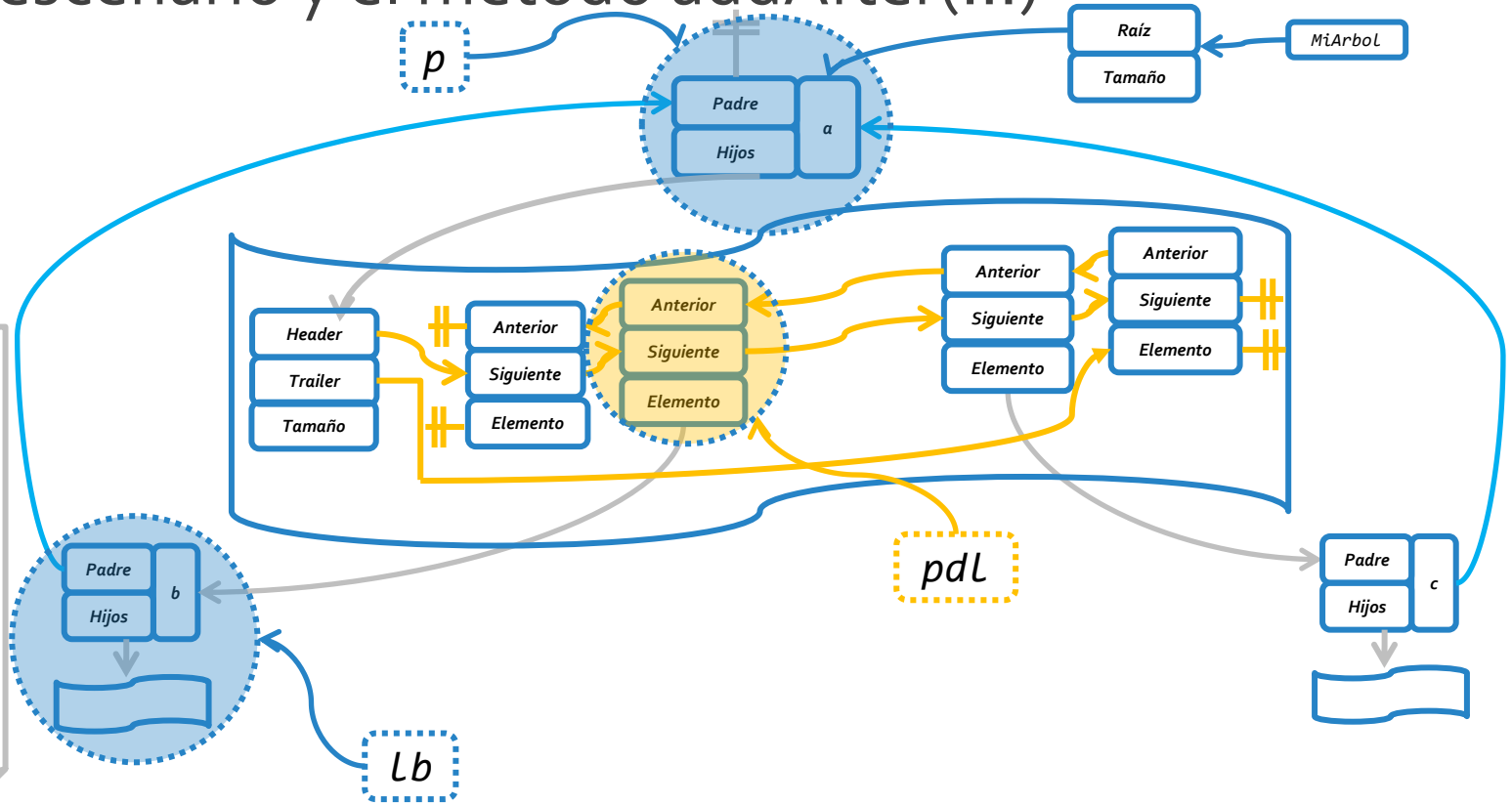
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*
- *Finalmente, en la **lista** se **agrega** luego de pdl, un nuevo nodo con rótulo d y padre a.*

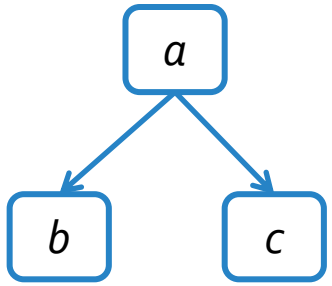



```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

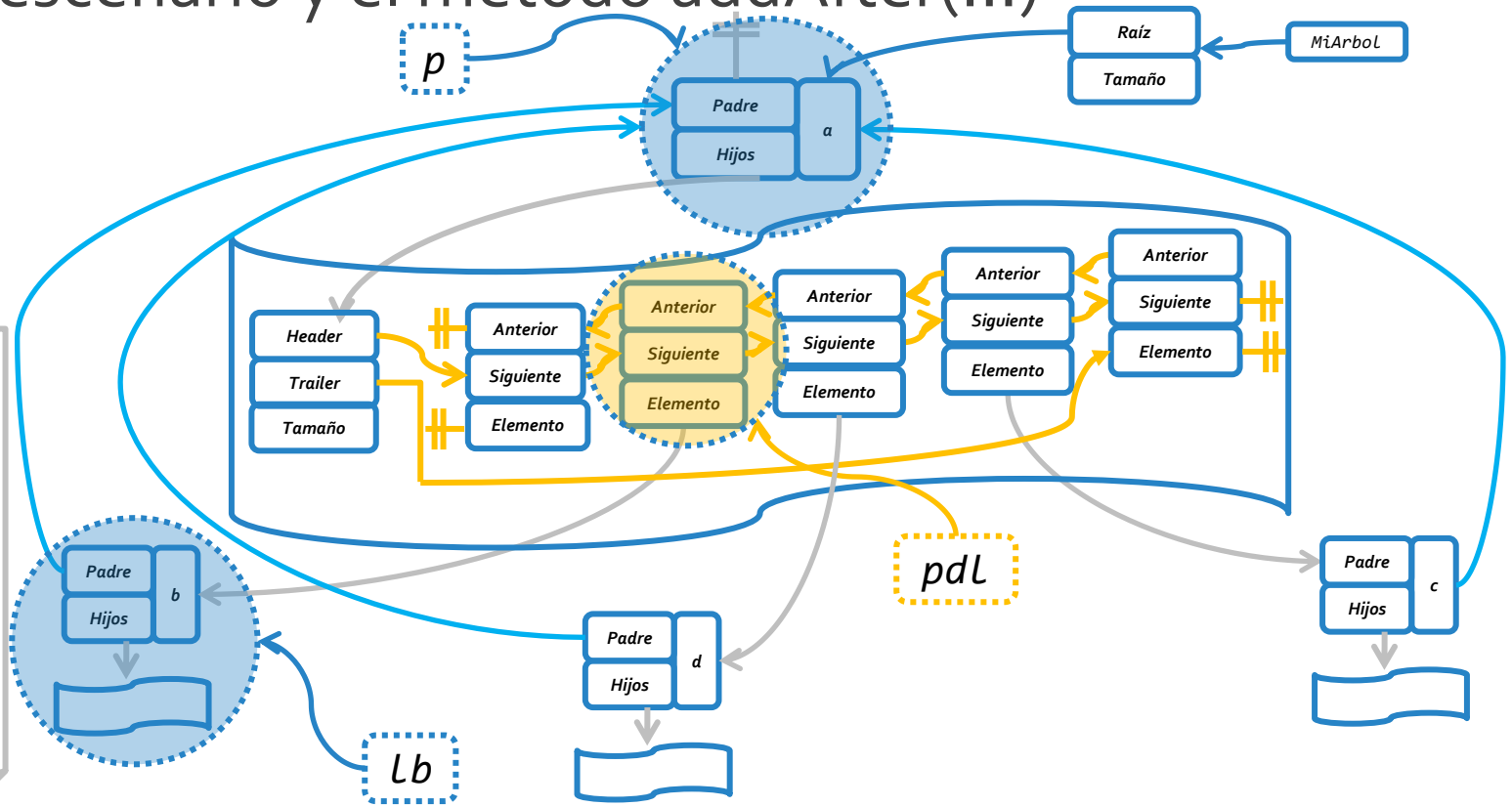
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*
- *Finalmente, en la **lista** se **agrega** luego de pdl, un nuevo nodo con rótulo d y padre a.*

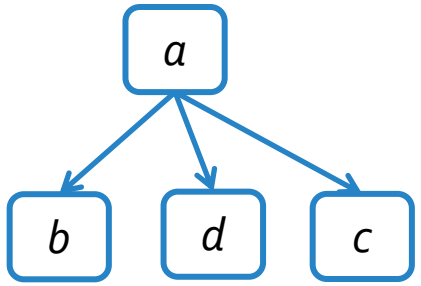



```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

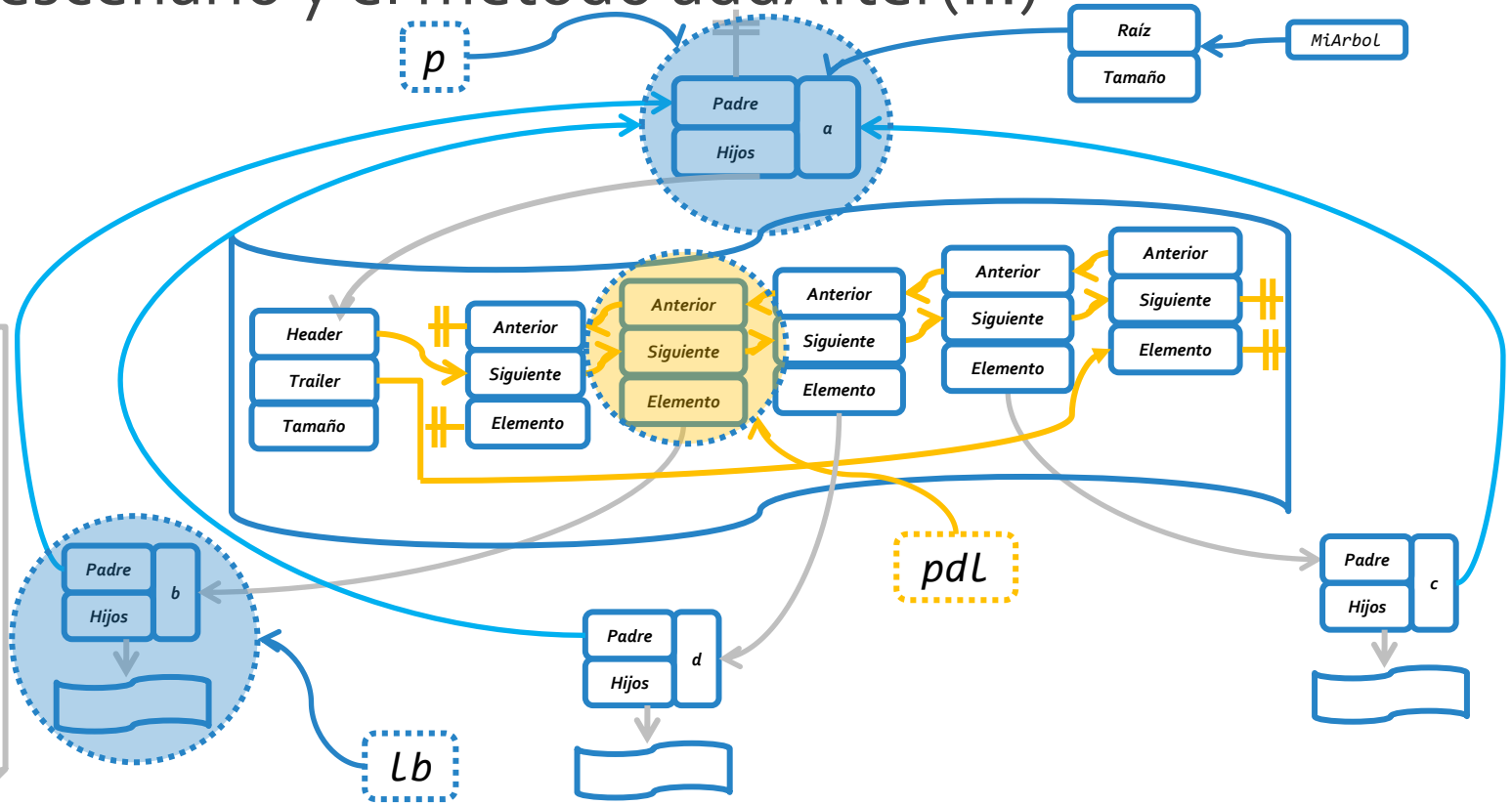
Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

Positions de árbol y lista :: Ejemplo addAfter()

- Consideremos el siguiente escenario y el método addAfter(…)



- *Utilizaremos la operación addAfter(p,lb,e).*
- *Como se agregará un nuevo nodo como hijo del **nodo p**, se debe **modificar** su lista de **nodos hijos**.*
- *Se deberá **buscar** la posición de lista (pdl), cuyo elemento sea el nodo de árbol lb.*
- *Finalmente, en la **lista** se **agrega** luego de pdl, un nuevo nodo con rótulo d y padre a.*




```
public Position<E> addAfter (Position<E> p, Position<E> lb, E e) throws InvalidPositionException;
```

Supongamos que queremos agregar un nodo con rótulo d, como hijo de a, a continuación de b.

IMPLEMENTACIÓN DE REMOVEs

Se recomienda que todo lo documentado en la siguiente sección sea complementado a través de otras herramientas multimedia dispuestas en la siguiente explicación online.

Implementación operaciones remove

- El TDA Árbol considera **tres** operaciones que permiten **eliminar** elementos del árbol.

```
public interface Tree<E> extends Iterable<E>{
  //Operaciones para creación y modificación.
  public void removeExternalNode (Position<E> p) throws InvalidPositionException;
  public void removeInternalNode (Position<E> p) throws InvalidPositionException;
  public void removeNode (Position<E> p) throws InvalidPositionException;
}
```

- En particular, ***removeExternalNode()*** y ***removeInternalNode()*** son casos particulares del caso general de eliminación que define ***removeNode()***.
- Por dicho motivo, es adecuado definir completamente el método ***removeNode()***, y re-utilizar este método cuando se requiera la ejecución de ***removeExternalNode()*** y ***removeInternalNode()***.

Implementación operaciones remove

```
public void removeExternalNode(Position<E> p) throws InvalidPositionException {
  TNodo<E> n = checkPosition(p);
  if(!n.getHijos().isEmpty())
    throw new InvalidPositionException("No es un nodo externo");
  removeNode(n);
}

public void removeInternalNode(Position<E> p) throws InvalidPositionException {
  TNodo<E> n = checkPosition(p);
  if(n.getHijos().isEmpty())
    throw new InvalidPositionException("No es un nodo interno");
  removeNode(n);
}

protected TNodo<E> checkPosition(Position<E> p) throws InvalidPositionException {
  TNodo<E> toReturn = null;
  try{
    toReturn = (TNodo<E>) p;
    if (toReturn == null || toReturn.element() == null)
      throw new InvalidPositionException("");
  }catch(ClassCastException e){ throw new InvalidPositionException(); }
  return toReturn;
}
```

Implementación operaciones remove

```
@Override
public void removeNode(Position<E> p) throws InvalidPositionException {
  TNodo<E> nEliminar = checkPosition(p);
  TNodo<E> padre = nEliminar.getPadre();
  PositionList<TNodo<E>> hijos = nEliminar.getHijos();

  try{
    if (nEliminar == raiz){
      if (hijos.size() == 0){
        raiz = null;
      }else{
        if (hijos.size() == 1){
          TNodo<E> hijo = hijos.remove(hijos.first());
          hijo.setPadre(null);
          raiz = hijo;
        }else
          throw new InvalidPositionException("No se puede eliminar raíz con hijos > 1");
      }
    }else{
      //Sigue en próxima slide.
```

Implementación operaciones remove

```
      //Si no es raíz, nEliminar tiene un padre y por lo tanto una lista de hermanos.
      PositionList<TNodo<E>> hermanos = padre.getHijos();

      //Se debe hallar la posición de lista (en hermanos) que almacene (si existe) nEliminar.
      TDALista.Position<TNodo<E>> posListaHermanos = hermanos.isEmpty() ? null : hermanos.first();
      while(posListaHermanos != null && posListaHermanos.element() != nEliminar){
        posListaHermanos = (hermanos.last() == posListaHermanos) ? null : hermanos.next(posListaHermanos);
      }

      //Si no existe posición de lista (en hermanos) que almacene a nEliminar, la posición parametrizada p es inválida.
      if (posListaHermanos == null)
        throw new InvalidPositionException("La posición p no se encuentra en la lista del padre");

      //Se agregan en la lista de hermanos de nEliminar, todos los hijos nEliminar (respetando el orden).
      while(!hijos.isEmpty()){
        TNodo<E> hijo = hijos.remove(hijos.first());
        hijo.setPadre(padre);
        hermanos.addBefore(posListaHermanos, hijo);
      }
      hermanos.remove(posListaHermanos);
    }//Else (nEliminar == raíz).

    nEliminar.setPadre(null);
    nEliminar.setElement(null);
    tamaño--;
  }catch(EmptyListException | TDALista.BoundaryViolationException e){}
}
```

Fin de la presentación.