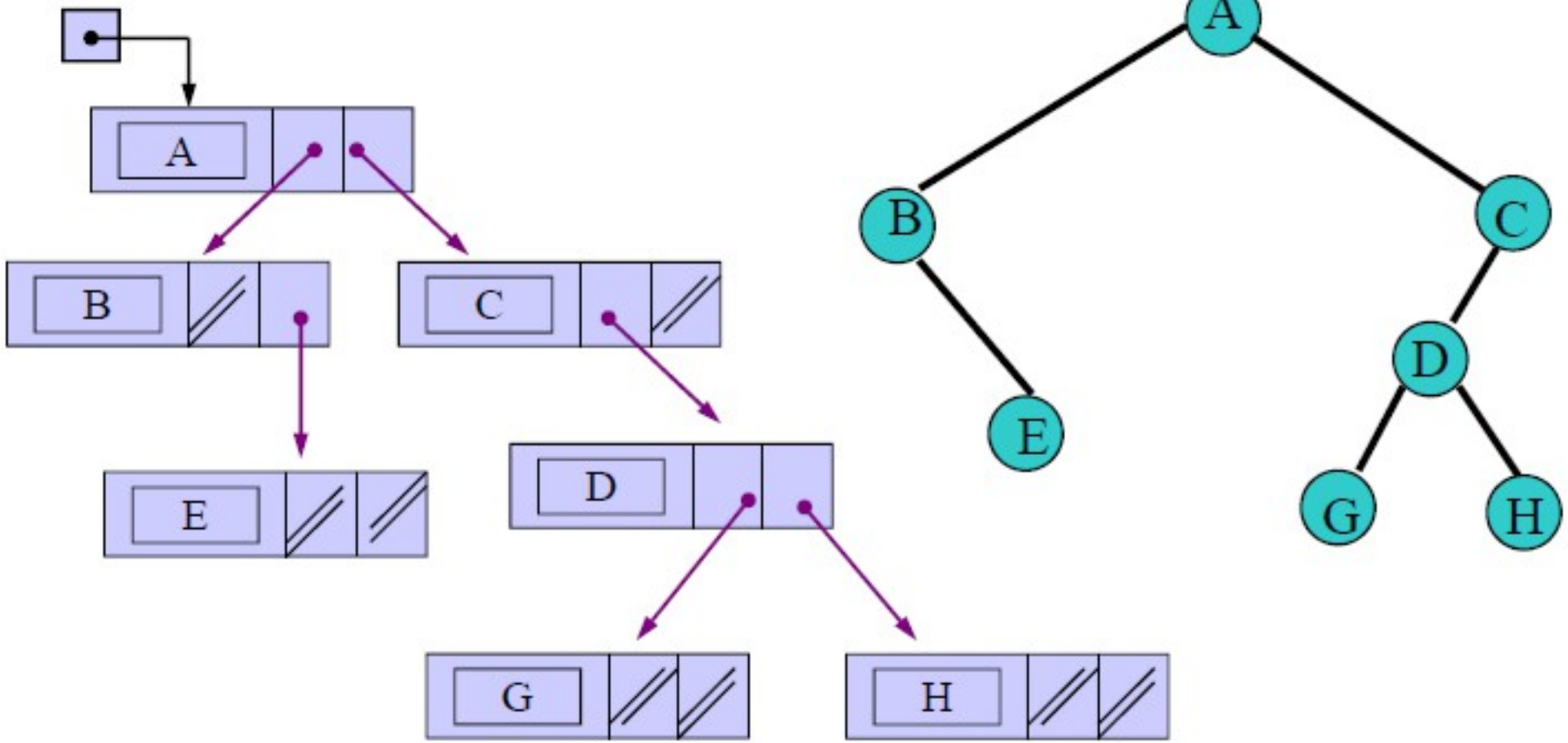
Programación 3

Práctica 3:
Árboles binarios y Generales

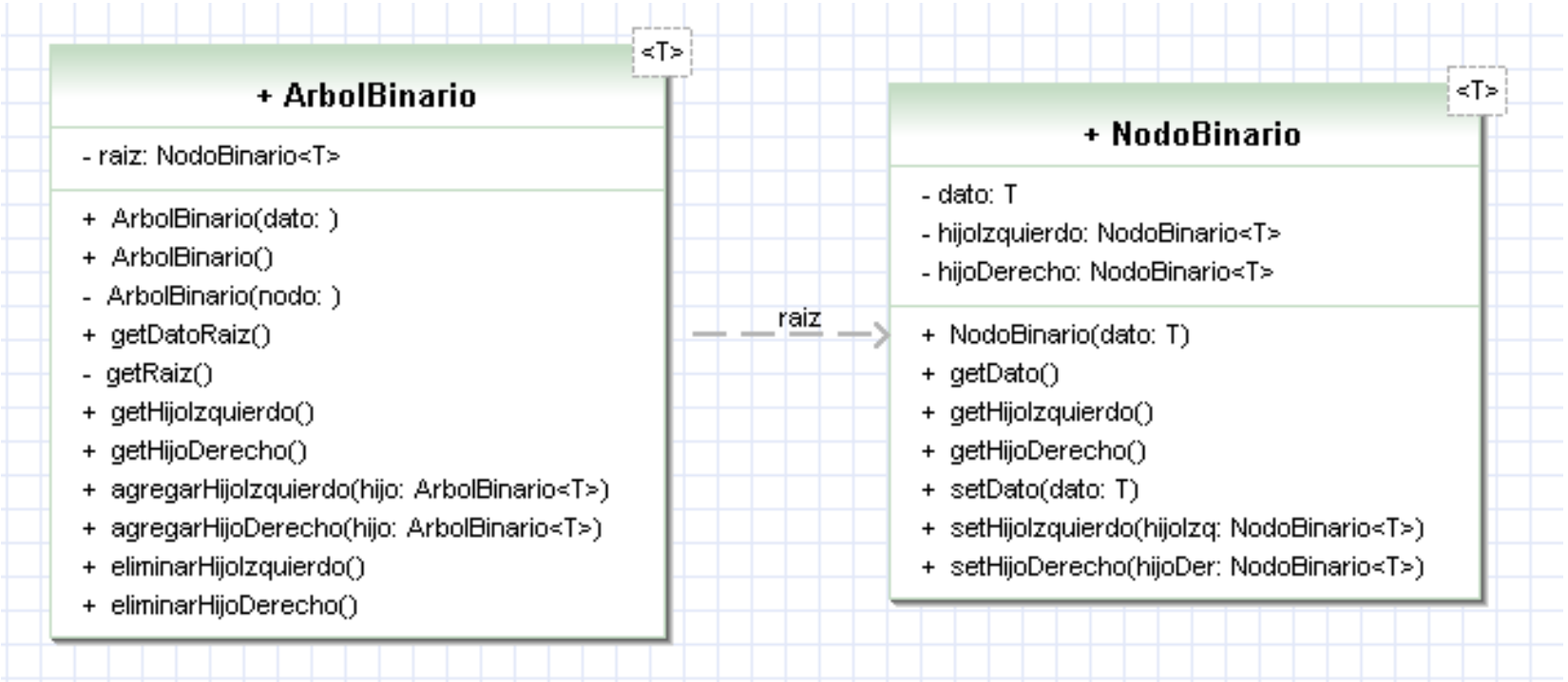
Práctica 3

- Arbol binario: hijo izquierdo - hijo derecho.
- Tener en cuenta que no es un árbol ordenado
  - Insertar un hijo es simplemente asignar referencias de nodos.

Práctica 3



Práctica 3


+ ArbolBinario
<T>
- raiz: NodoBinario<T>
+ ArbolBinario(dato:)
+ ArbolBinario()
- ArbolBinario(nodo:)
+ getDatoRaiz()
- getRaiz()
+ getHijoIzquierdo()
+ getHijoDerecho()
+ agregarHijoIzquierdo(hijo: ArbolBinario<T>)
+ agregarHijoDerecho(hijo: ArbolBinario<T>)
+ eliminarHijoIzquierdo()
+ eliminarHijoDerecho()
raiz
+ NodoBinario
<T>
- dato: T
- hijoIzquierdo: NodoBinario<T>
- hijoDerecho: NodoBinario<T>
+ NodoBinario(dato: T)
+ getDato()
+ getHijoIzquierdo()
+ getHijoDerecho()
+ setDato(dato: T)
+ setHijoIzquierdo(hijoIzq: NodoBinario<T>)
+ setHijoDerecho(hijoDer: NodoBinario<T>)

Práctica 3

- Prestar atención a la diferencia entre Árbol y Nodo.
- El ArbolBinario tiene una raíz, que al pedirle el hijo izquierdo devuelve un nuevo árbol con el nodo como raíz.
- Implementar
  - getHijoIzquierdo,
  - agregarHijoIzquierdo.

Práctica 3

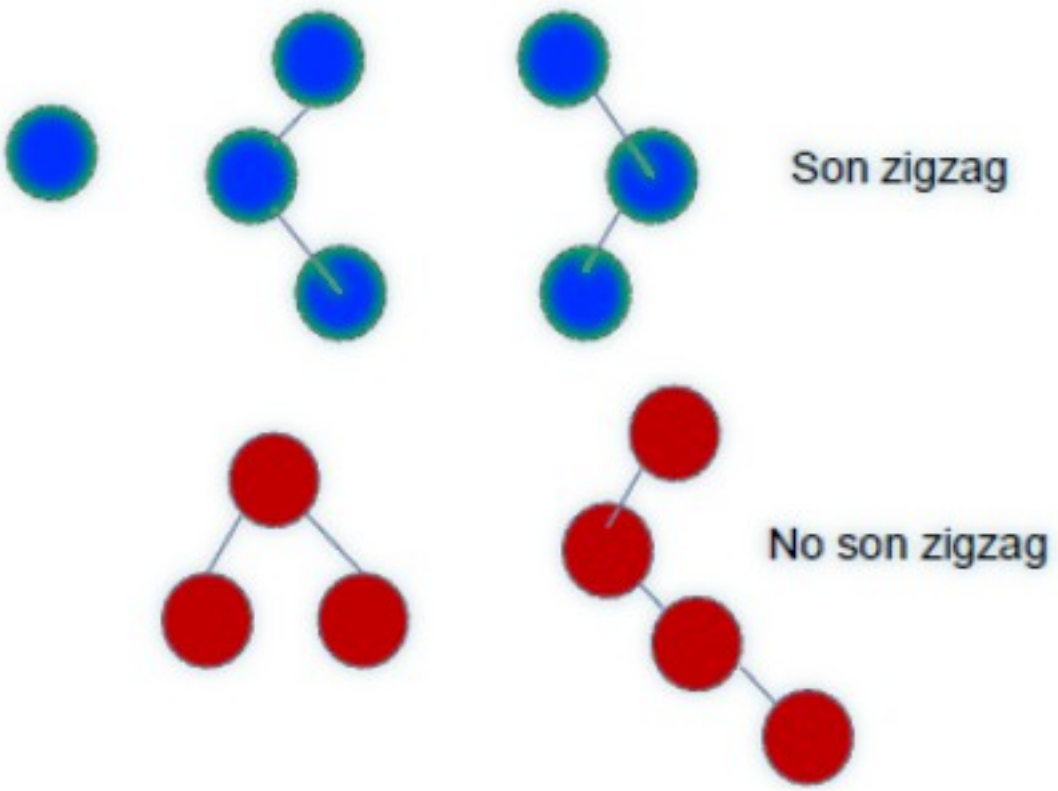
```
// ¿Es correcto?
class ArbolBinario<T> {
  ArbolBinario<T> getHijoIzquierdo() {
    return getRaiz().getHijoIzquierdo();
  }
}
```

```
// Lo anterior NO es correcto.
class ArbolBinario<T> {
  ArbolBinario<T> getHijoIzquierdo() {
    return new ArbolBinario<T>(
      getRaiz().getHijoIzquierdo());
  }
}
```

Práctica 3

- Ejercicio 2-C. zigzag(). Devuelve true si el árbol es degenerado con direcciones alternadas, esto es, si en lugar de ser una lista donde todos los hijos están en el lado izquierdo o derecho, se van alternando.

Práctica 3



Práctica 3

- ¿Cómo se determina si un árbol es zig-zag?
- ¿Cuántos hijos debe tener cada nodo?
- ¿Qué “forma” debe tener el árbol?
- ¿Qué tipo de recorrido puede solucionar el problema?
  - Iterativo, recursivo?

Práctica 3

- Para cada nodo, es zigzag si tiene a lo sumo un hijo, y cada uno está de forma alternada.
- Hace falta llevar cuenta de cómo es la posición del nodo actual con respecto al padre.
- Una forma es hacer un recorrido recursivo.

Práctica 3

```
public boolean esZigZag() {
  if (esVacio() || esHoja())
    return true;
  if (tieneHijoIzquierdo() && tieneHijoDerecho())
    return false;
  else if (tieneHijoIzquierdo())
    return getHijoIzquierdo().esZigZagDerecho();
  else if (tieneHijoDerecho())
    return getHijoDerecho().esZigZagIzquierdo();
  return true;
}
```

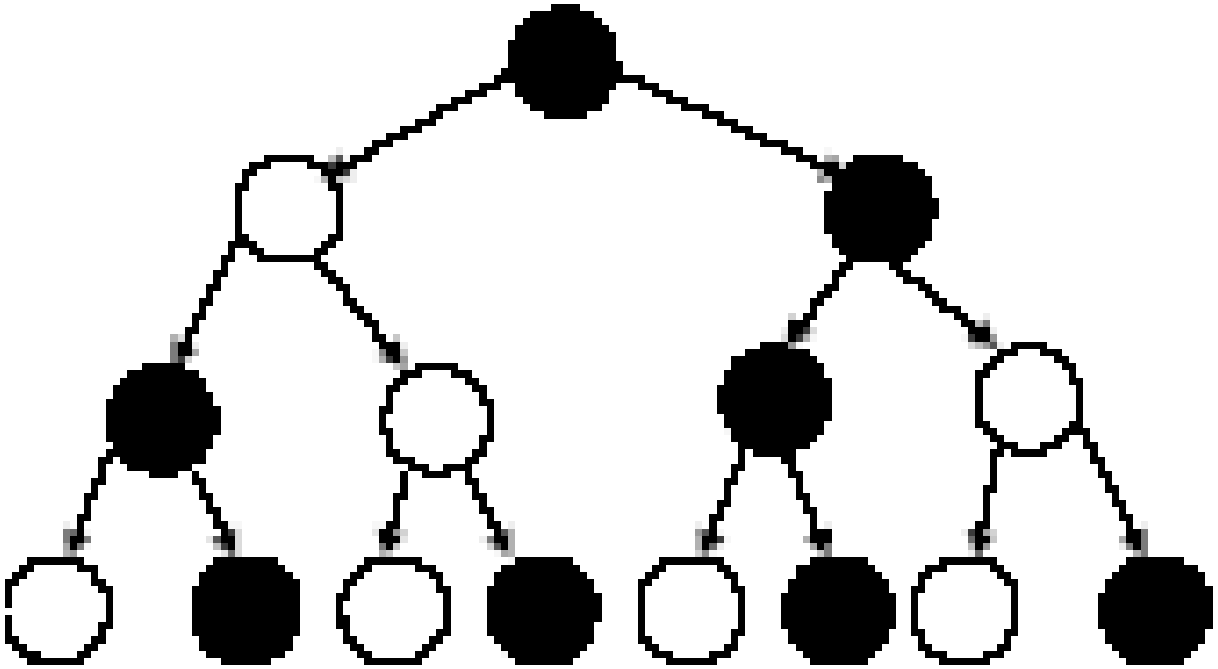
Práctica 3

```
private boolean esZigZagDerecho() {
  if (esVacio() || esHoja())
    return true;
  return tieneHijoDerecho() && !tieneHijoIzquierdo() &&
    getHijoDerecho().esZigZagIzquierdo();
  }
private boolean esZigZagIzquierdo() {
  if (esVacio() || esHoja())
    return true;
  return tieneHijoIzquierdo() && !tieneHijoDerecho() &&
    getHijoIzquierdo().esZigZagDerecho();
}
```

Práctica 3

- Ejercicio 2-F: colorear(). Un árbol binario coloreado es un árbol binario lleno cuyos nodos tienen uno de dos colores posibles: negro o blanco. El color para la raíz del árbol es negro. Y para cada nivel los colores de los nodos se intercalan, como así también al comenzar cada nivel.

Práctica 3



Práctica 3

- ¿Qué característica debe tener el árbol?
  - ¿Cualquier árbol se puede colorear de esta forma?
- ¿De qué forma se puede “colorear” el árbol?

Práctica 3

- La forma más directa es hacer un recorrido por niveles en el árbol. Se pinta la raíz de negro, y cada nodo que se visita se va alternando de color (con respecto al anterior).
- El recorrido por niveles permite visitar los nodos de un árbol completo de izquierda a derecha.

Práctica 3

- El primer nodo de cada nivel se va alternando de color. En el nivel de la raíz es negro.
- Hace falta contemplat esto en el recorrido, al momento de detectar el nivel.

Práctica 3

```
public void colorear() {
  ColaGenerica<ArbolBinario<T>>
cola = new
ColaGenerica<ArbolBinario<T>>();
  int color = 0, nivel = 0;
  cola.poner(this); cola.poner(null);
  while (!cola.esVacia()) {
    ArbolBinario<T> e =
cola.sacar();
    if (e != null) {
      e.setDato(color); color = 1 -
color;
      if (e.tieneHijoIzquierdo())
cola.poner(e.getHijoIzquierdo());
      if (e.tieneHijoDerecho())
cola.poner(e.getHijoDerecho());
    } else {
      if (!cola.esVacia())
        cola.poner(null);
      nivel++;
      if ((nivel % 2) == 0) color = 0;
      else color = 1;
    }
  }
}
```

Práctica 3

- Tarea: Hacer la versión recursiva de colorear().