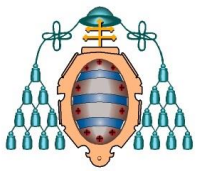
ESTRUCTURAS JERARQUICAS SESIÓN 2

Estructura de Datos

2021-2022

María del Rosario Suárez Fernández

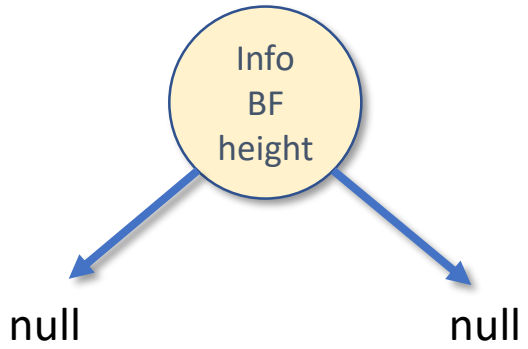


Departamento de Informática
UNIVERSIDAD DE OVIEDO

Proyecto3 - Árboles

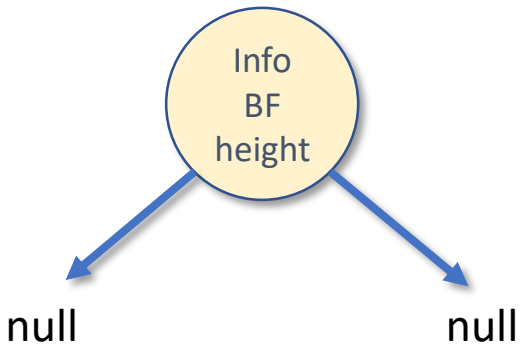
- Abrir el proyecto llamado **Arboles**
- Crear un nuevo paquete llamado **AVL**
- Dentro del paquete crear:
  - Clase **AVLNode** → implementa el nodo de un árbol AVL

```
public class AVLNode <T extends Comparable <T>>
```



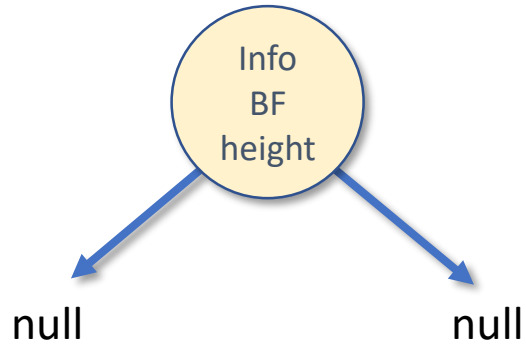
Clase AVLNode - Propiedades

Propiedad	Descripción
`private T info`	En contenido del nodo que será de tipo genérico
`private AVLNode<T> left`	Nodo hijo izquierdo
`private AVLNode<T> right`	Nodo hijo derecho
`private int balanceFactor`	Factor de balance del nodo
`private int height`	Altura del nodo



Clase AVLNode - Métodos

- `public AVLNode(T clave)`
- `public void setInfo(T clave)`
- `public T getInfo()`
- `public void setLeft(AVLNode<T> nodo)`
- `public void setRight(AVLNode<T> nodo)`
- `public AVLNode<T> getLeft()`
- `public AVLNode<T> getRight()`



Clase AVLNode - Métodos

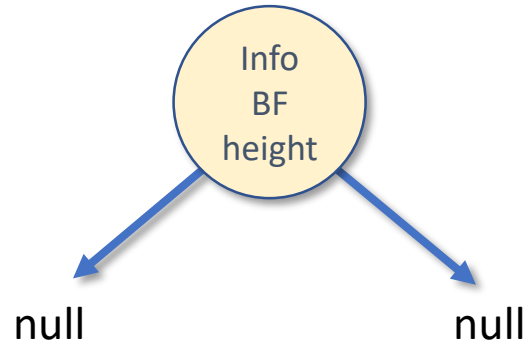
- `public int getHeight()`
- `public int getBF()`
- `public void updateBFHeight()`
- `public String toString() {`

```
    return info.toString()+":BF="+ getBF();
}
```

Ejemplos

8:BF=1

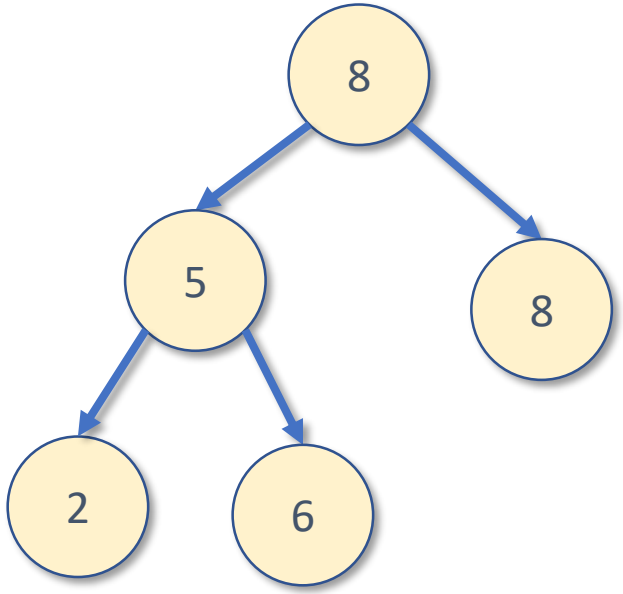
4:BF=0



Proyecto3 - Árboles

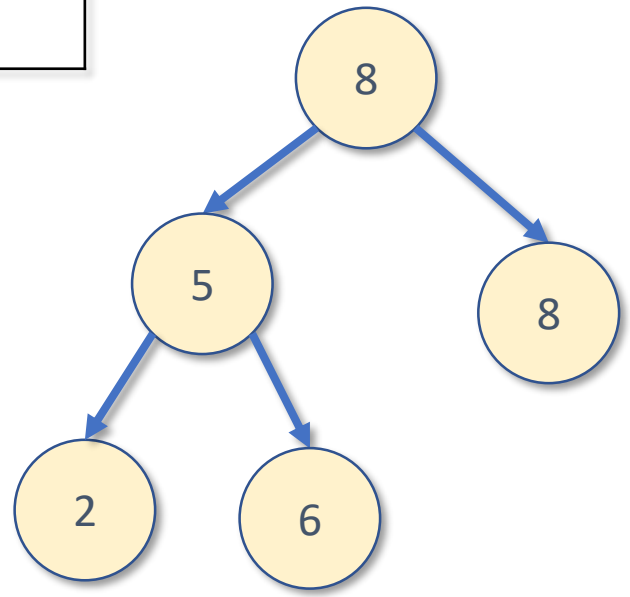
- Dentro del paquete **AVL** crear:
  - Clase **AVLTree** → implementa un árbol AVL

```
public class AVLTree<T extends Comparable<T>>
```



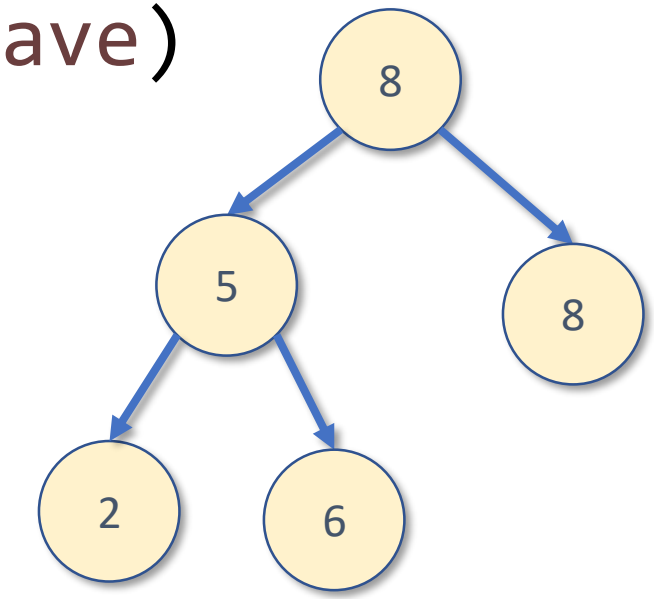
Clase AVLTree - Propiedades

Propiedad	Descripción
`private` AVLNode<T> `raiz`	Nodo raíz del árbol AVL

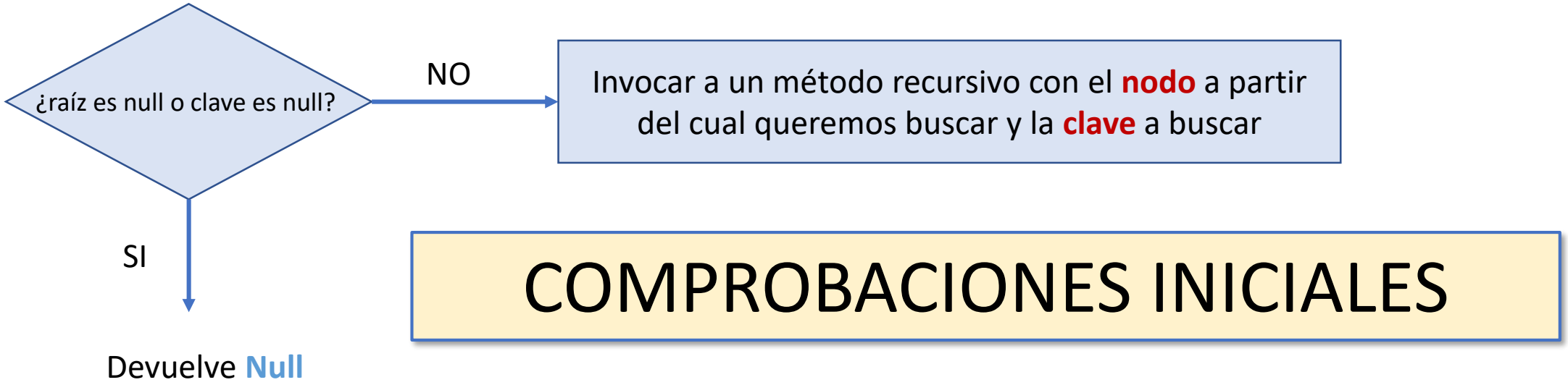


Clase AVLTree - Métodos

- `public AVLTree()`
- `public AVLNode<T> searchNode(T clave)`
- `public int addNode(T clave)`
- `public String preOrder()`
- `public String postOrder()`
- `public String inOrder()`
- `public int removeNode(T clave)`

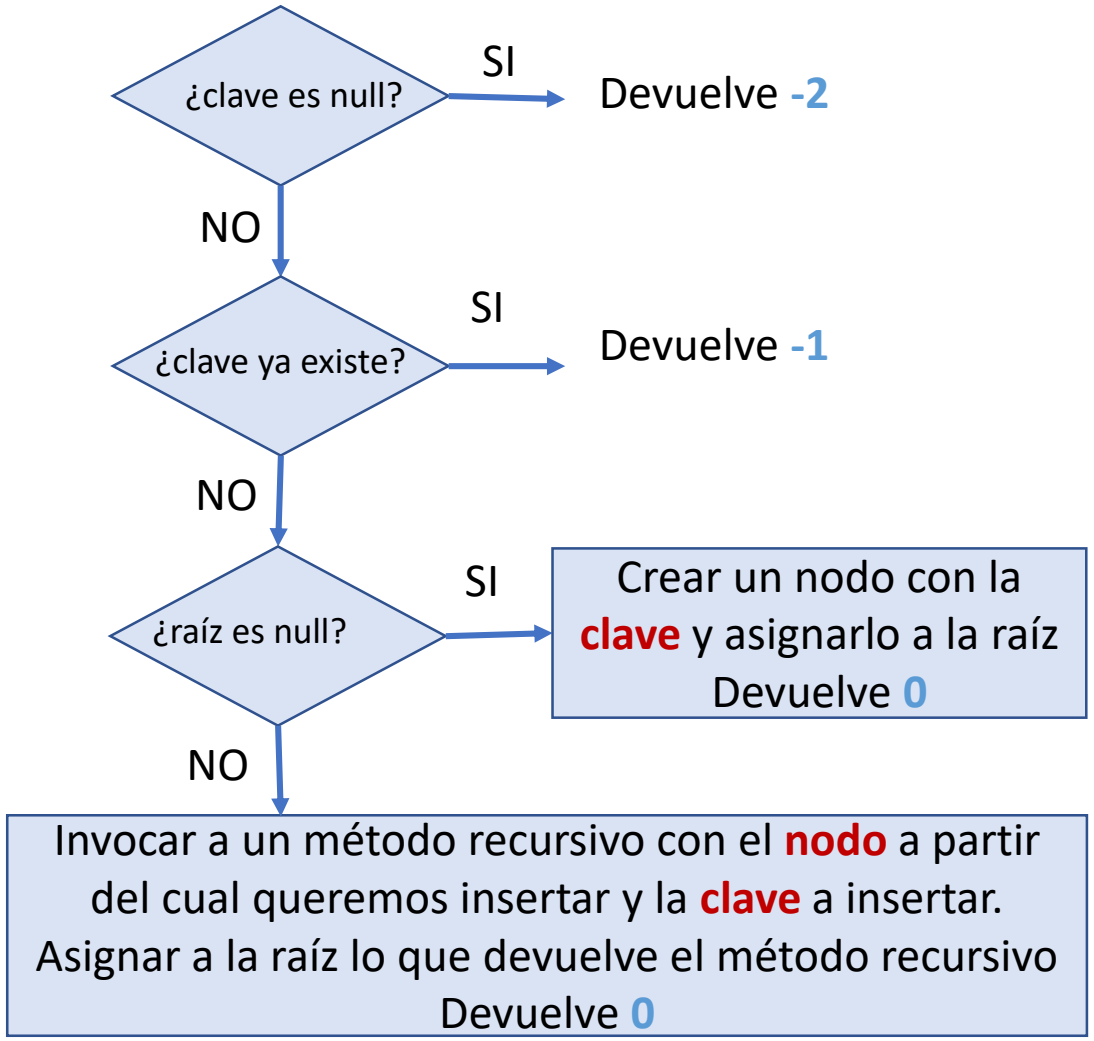


Clase BSTree - searchNode



El **método recursivo**, buscar la **clave** por la derecha o por la izquierda, dependiendo de si esta es mayor o menor que el **nodo** a partir del cual se busca. Devuelve el **nodo completo** encontrado o **Null** si no lo encuentra

Clase BSTree - addNode



COMPROBACIONES INICIALES

El **método recursivo**, busca un hueco por la derecha o por la izquierda, dependiendo de si la **clave** a insertar es mayor o menor que el **nodo** a partir del cual se quiere insertar. Cuando lo encuentra, crea un nuevo nodo con la **clave** y lo asigna por la derecha o por la izquierda. Actualiza el **nodo** y lo devuelve

Clase AVLTree – Actualizar, rotar y recorridos

- `private AVLNode<T> updateAndBalanceIfNecesary(AVLLNode<T> nodo)`
- `private AVLNode<T> singleLeftRotation(AVLLNode<T> nodo)`
- `private AVLNode<T> singleRightRotation (AVLLNode<T> nodo)`
- `private AVLNode<T> doubleLeftRotation (AVLLNode<T> nodo)`
- `private AVLNode<T> doubleRightRotation (AVLLNode<T> nodo)`

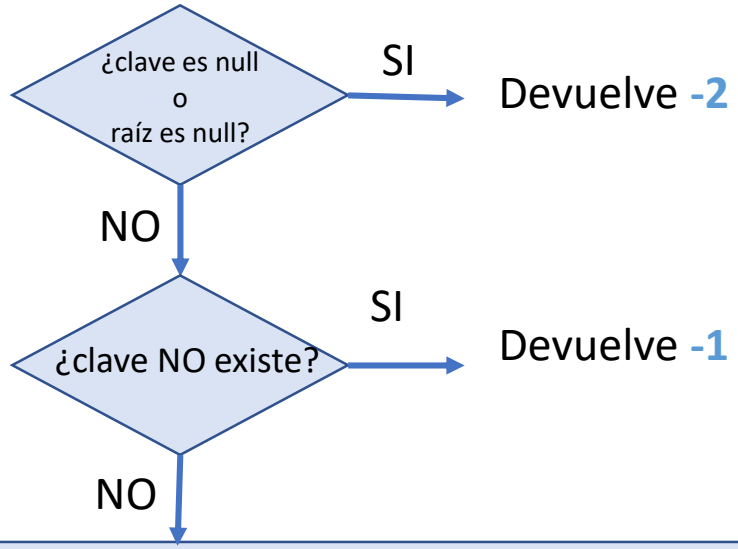
Clase AVLTree - updateAndBalanceIfNecesary

```
private AVLNode<T> updateAndBalancedIfNecesary(AVLNode<T> nodo) {
    nodo.updateBFHeight();
    if (nodo.getBF() == -2)
        if (nodo.getLeft().getBF() == 1)
            nodo = doubleLeftRotation(nodo);
        else //-1 o cero
            nodo = singleLeftRotation(nodo);
     else if (nodo.getBF() == 2)
        if (node.getRight().getBF() == -1)
            nodo = doubleRightRotation(nodo);
        else //1 o cero
            nodo = singleRightRotation(nodo);
    return nodo;
}
```

Tareas para casa (Evaluación continua)

- Terminar los métodos de la clase AVLTree funcionando
- Elaborar un conjunto de baterías de test que prueben:
  - El método añadir con todos los casos posibles de error
  - Los recorridos
- Documentar las clases con JavaDoc

Clase AVLTree - removeNode



Invocar a un método recursivo con el **nodo** a partir del cual queremos borrar y la **clave** a borrar.
Asignar a la raíz lo que devuelve el método recursivo
Devuelve **0**

COMPROBACIONES INICIALES

El **método recursivo**, busca la clave por la derecha o por la izquierda, dependiendo de si la **clave** a borrar es mayor o menor que el **nodo** a partir del cual se quiere borrar. Cuando la encuentra (puede no tener hijos, un hijo o dos) la borra con los mecanismos vistos en clase de teoría.
Devuelve el **nodo actualizado**

Tareas adicionales

- Implementar un método que devuelva el padre de un valor de tipo T que se le pasa como parámetro. Devolverá el **nodo padre completo** o **null** si no tiene padre o el elemento pasado como parámetro no existe o el elemento es o el árbol es null
  - `public AVLNode<T> padreDe(T clave)`
- Implementar un método que devuelva el número de aristas que hay desde *clave1* hasta *clave2* teniendo en cuenta que no hay ciclos y clave1 siempre es un ancestro de clave2. Devolverá el **numero de aristas** o **0** si clave1 es null o clave2 es null o el árbol es null o clave1 y clave2 son iguales
  - `public int numAristas(T clave1, T clave2)`

Tareas para casa (Evaluación continua)

- Terminar los métodos de la clase AVLTree funcionando
- Elaborar un conjunto de baterías de test que prueben:
  - El método borrar con todos los casos posibles de error
- Terminar los métodos:
  - padreDe(clave)
  - numAristas(clave1, clave2)
- Documentar las clases con JavaDoc