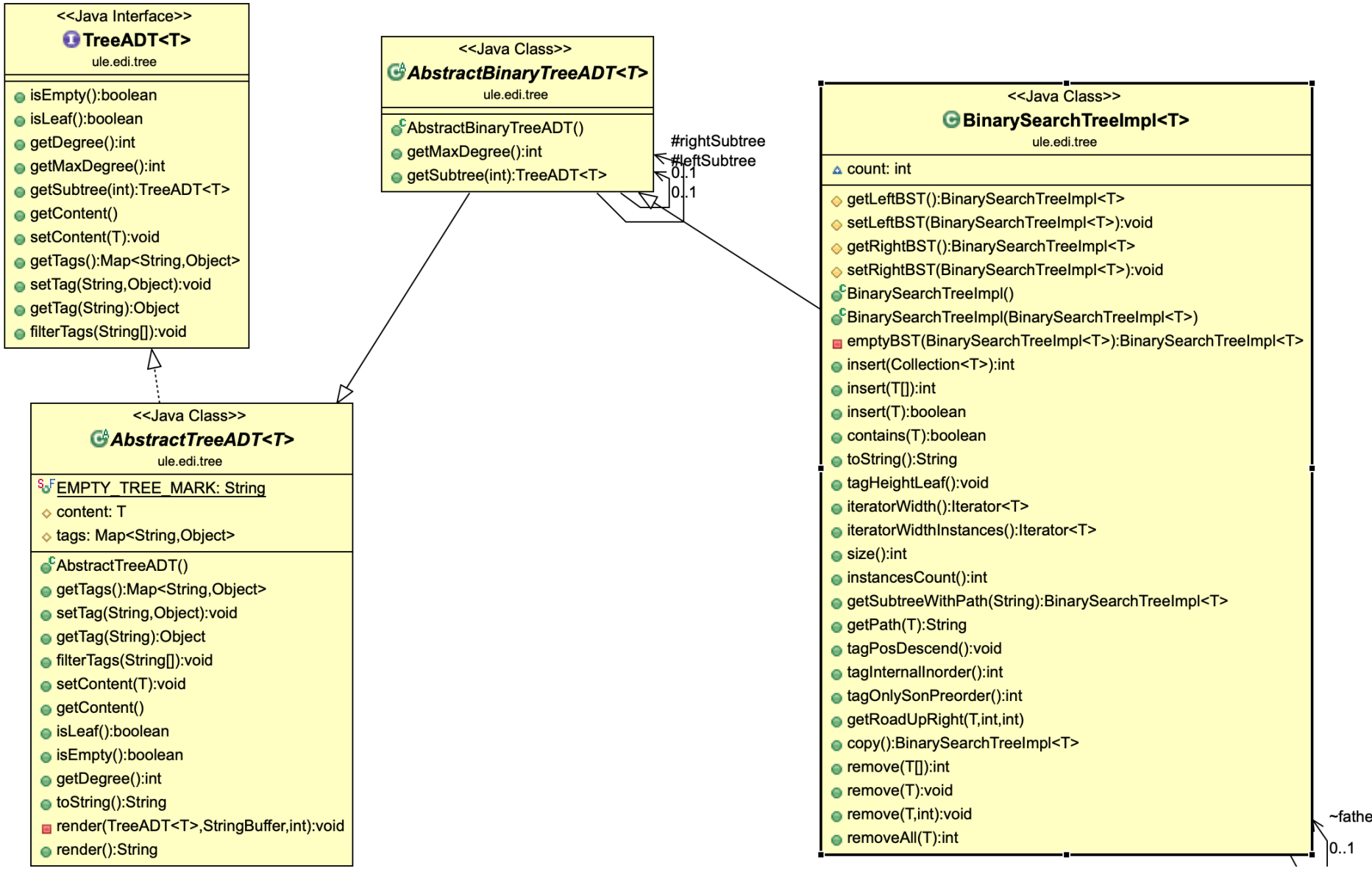
PRÁCTICA 5. - ÁRBOLES

- Árboles binarios de búsqueda

En esta práctica se seguirán las siguientes consideraciones:

- El árbol vacío es un nodo vacío (content, left y right a null)
- Si un nodo no tiene hijo izquierdo, su hijo izquierdo apuntará a un nodo vacío
- Si un nodo no tiene hijo derecho, su hijo derecho apuntará a un nodo vacío


<<Java Interface>>
TreeADT<T>
ule.edi.tree
isEmpty():boolean
isLeaf():boolean
getDegree():int
getMaxDegree():int
getSubtree(int):TreeADT<T>
getContent()
setContent(T):void
getTags():Map<String,Object>
setTag(String,Object):void
getTag(String):Object
filterTags(String[]):void
<<Java Class>>
AbstractBinaryTreeADT<T>
ule.edi.tree
AbstractBinaryTreeADT()
getMaxDegree():int
getSubtree(int):TreeADT<T>
#rightSubtree
#leftSubtree
0..1
0..1
<<Java Class>>
BinarySearchTreeImpl<T>
ule.edi.tree
count: int
getLeftBST():BinarySearchTreeImpl<T>
setLeftBST(BinarySearchTreeImpl<T>):void
getRightBST():BinarySearchTreeImpl<T>
setRightBST(BinarySearchTreeImpl<T>):void
BinarySearchTreeImpl()
BinarySearchTreeImpl(BinarySearchTreeImpl<T>)
emptyBST(BinarySearchTreeImpl<T>):BinarySearchTreeImpl<T>
insert(Collection<T>):int
insert(T[]):int
insert(T):boolean
contains(T):boolean
toString():String
tagHeightLeaf():void
iteratorWidth():Iterator<T>
iteratorWidthInstances():Iterator<T>
size():int
instancesCount():int
getSubtreeWithPath(String):BinarySearchTreeImpl<T>
getPath(T):String
tagPosDescend():void
tagInternalInorder():int
tagOnlySonPreorder():int
getRoadUpRight(T,int,int)
copy():BinarySearchTreeImpl<T>
remove(T[]):int
remove(T):void
remove(T,int):void
removeAll(T):int
~fathe
0..1
<<Java Class>>
AbstractTreeADT<T>
ule.edi.tree
EMPTY_TREE_MARK: String
content: T
tags: Map<String,Object>
AbstractTreeADT()
getTags():Map<String,Object>
setTag(String,Object):void
getTag(String):Object
filterTags(String[]):void
setContent(T):void
getContent()
isLeaf():boolean
isEmpty():boolean
getDegree():int
toString():String
render(TreeADT<T>,StringBuffer,int):void
render():String

AbstractTreeADT<T>

- **Atributos:**
  - content: T
  - **tags:Map<String,Object>**
    - (creado como un HashMap<>())
    - Map es un interface y HashMap es una clase que implementa el interfaz Map con tablas hash.
- **Métodos relevantes:**
  - **setTag(String,Object)**: etiqueta el nodo actual con la etiqueta pasada como primer parámetro. (Añade al HashMap tags el par (String,Object)
    - setTag("level", nivel)
  - **getTag(String)**: setTag(String,Object): devuelve el valor almacenado junto a la clave pasada como parámetro.
    - getTag("level")
  - **filterTags(String… labels):** deja solamente las etiquetas indicadas. Muy util para los tests (en cada método, filtrar solamente las etiquetas que se piden en ese método)
    - filterTags("level")

Interface Map

```
1  public interface Map<K,V> {
2
3  // Basic operations
4
5  V put(K key, V value);
6
7  V get(Object key);
8
9  V remove(Object key);
10
11 boolean containsKey(Object key);
12
13 boolean containsValue(Object value);
14
15 int size();
16
17 boolean isEmpty();
18
19 // Bulk operations
20
21 void putAll(Map<? extends K, ? extends V> m);
22
23 void clear();
24
25 // Collection Views
26
27 public Set<K> keySet();
28
29 public Collection<V> values();
30
```

HashMap: ED que representa pares <key,value>

- **Método put(K key, V value)**: inserta el par (key,value) en la colección. Si la key ya existe se sobreescribe, ya que no puede haber elementos con la misma key.
- **Método V get(K key)**: devuelve el valor almacenado junto a la clave pasada como parámetro.

AbstractBinaryTreeADT<T>

- Atributos:
  - AbstractBinaryTreeADT<T> leftSubtree
  - AbstractBinaryTreeADT<T> rightSubtree

BinarySearchTreeADTImpl<T extends Comparable<? super T>> extends AbstractBinaryTreeADT<T>

- Atributos:
  - BinarySearchTreeImpl<T> father;
  - int count
- El tipo de los elementos del árbol tiene que ser **comparable** (implementar el interface comparable o tener en su jerarquía de herencia algún antepasado que lo implemente).