

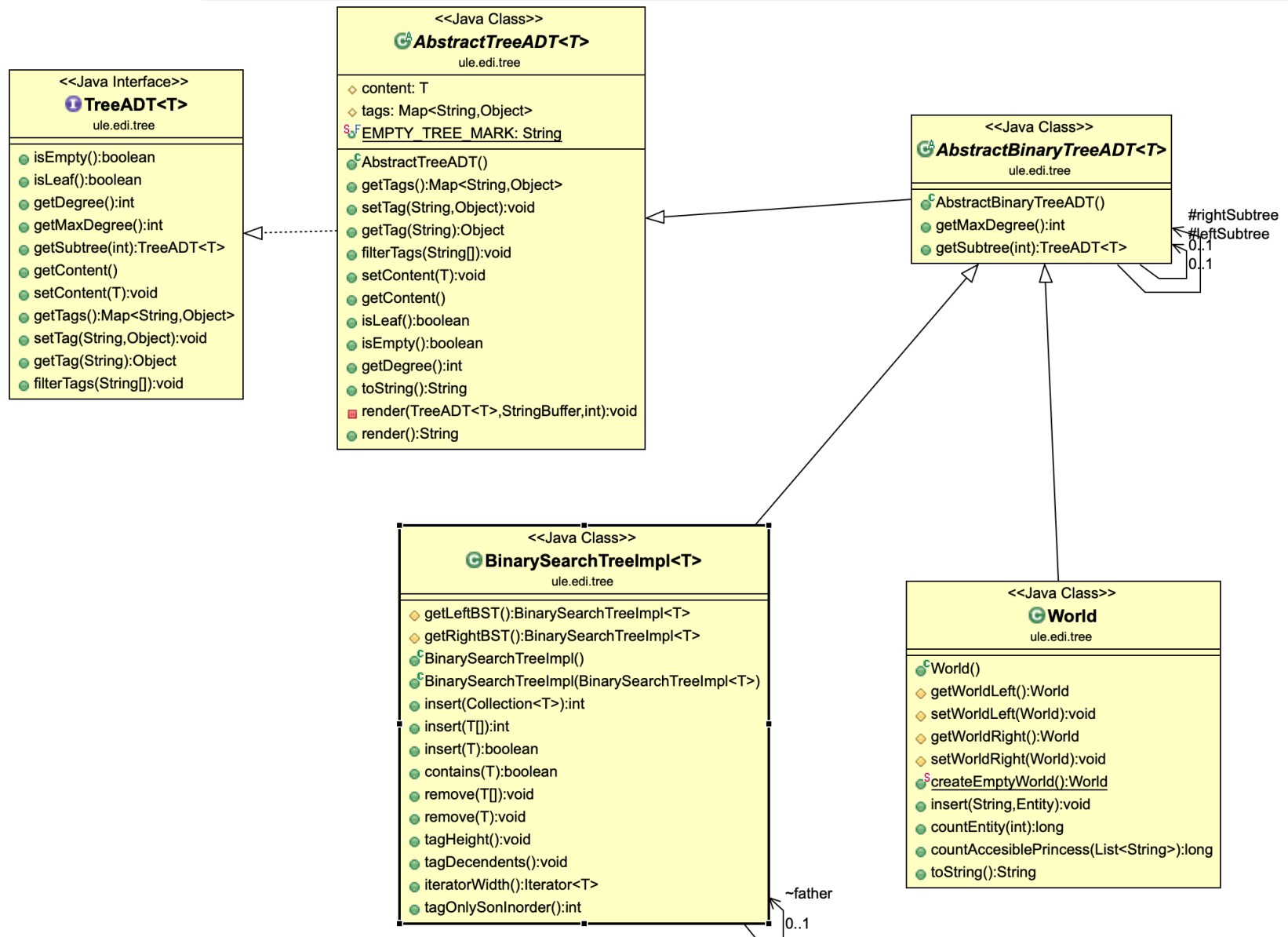
PRÁCTICA 5. - ÁRBOLES

- Árboles binarios de búsqueda
- Árboles que representan mundos

En los dos tipos de árboles 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

CLASES UTILIZADAS EN EL PROYECTO



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 a utilizar en la práctica:**
 - **setTag(String,Object):** etiqueta el nodo actual con la etiqueta pasada como primer parámetro. (Añade al **HashMap tags** el par (String,Object)
 - setTag(“onlySon”, num)
 - **getTag(String):** devuelve el valor almacenado junto a la clave pasada como parámetro.
 - getTag(“onlySon”)
 - **filterTags(String... labels):** deja solamente las etiquetas indicadas. Muy útil 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

- Es una colección donde el acceso a los elementos es constante (tarda lo mismo en acceder a cada elemento): $O(1)$
- 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 la clave almacenada junto al key pasado como parámetro.

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

- No define nuevos atributos
- 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).
- Implementar los métodos comentados con TODO

BinarySearchTreeImpl<T>

- Atributos de los árboles binarios de búsqueda:
 - BinarySearchTreeImpl<T> father
 - Heredados de AbstractTreeADT<T>:
 - T content
 - Map<String, Object> tags
 - Heredados de AbstractTreeADT<T>:
 - AbstractBinaryTreeADT<T> leftSubtree
 - AbstractBinaryTreeADT<T> rightSubtree

World extends

AbstractBinaryTreeADT<LinkedList<Entity>>

- Es un árbol binario (no de búsqueda).
- Los elementos (content) de los nodos del árbol de tipo World contienen **listas de entidades**.
- Cada entidad tiene un **tipo** y una **cardinalidad**.
- Implementar los métodos comentados con TODO
- Por ejemplo el mundo:

~~{[F(1)], {[F(1)], {[D(2), P(1)], \emptyset , \emptyset }, {[C(1)], \emptyset , \emptyset }}, \emptyset }~~

```
[F(1)]
|  [F(1)]
|  |  [D(2), P(1)]
|  |  |   $\emptyset$ 
|  |  |   $\emptyset$ 
|  |  [C(1)]
|  |  |   $\emptyset$ 
|  |  |   $\emptyset$ 
|   $\emptyset$ 
```


Entity

- Implementa el interface Comparable<Entity>
- Atributos:
 - int **type**: tipo definido por constantes:
 - long **count**: nº de instancias de ese tipo

UNKNOWN = 0;

DRAGON = 1;

PRINCESS = 3;

WARRIOR = 5;

CASTLE = 7;

FOREST = 9;