2° PARCIAL DE ALGORITMICA Y PROGRAMACION II - 2023

1) Agregar a la clase **ArrayQueue.java** el siguiente método:

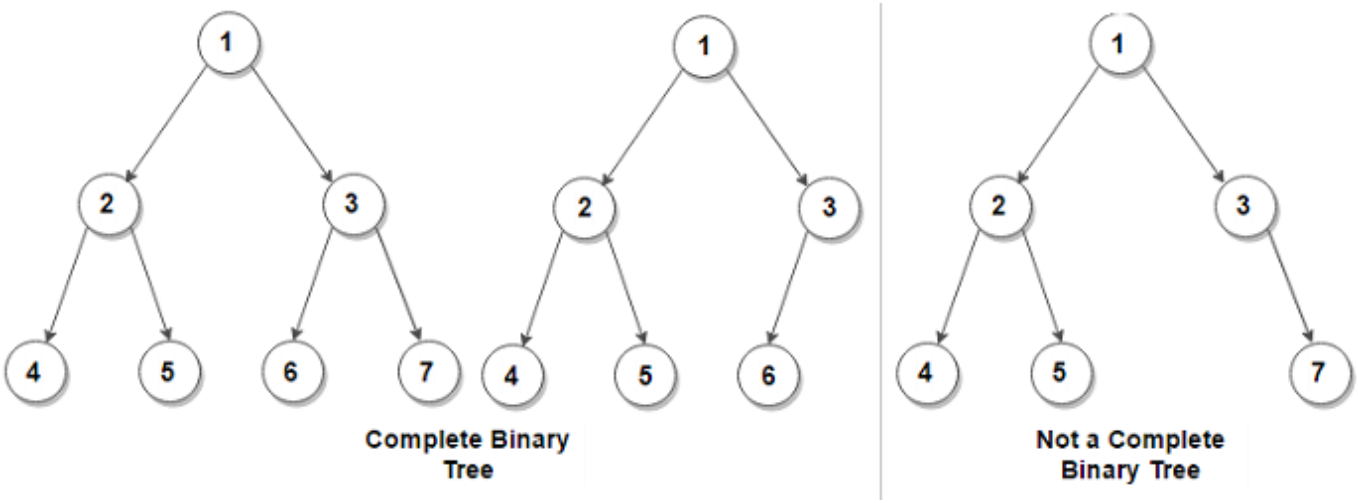
```
/**
 * Verifica si una cola es igual a otra. La cola tiene los mismos elementos y en el
 * mismo orden.
 *
 * @param obj cola a verificar
 * @return true si obj es igual o false si no lo es
 */
	public boolean equals(Object obj)
```

Correr el siguiente Test de JUnit y probar su funcionamiento.

```
package test;

import static org.junit.jupiter.api.Assertions.assertFalse;
import static org.junit.jupiter.api.Assertions.assertTrue;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import net.datastructures.ArrayQueue;
import net.datastructures.LinkedQueue;
import net.datastructures.Queue;

class TestArrayQueue {

	private Queue<String> q1;

	@BeforeEach
	void setUp() throws Exception {
		q1 = new ArrayQueue<String>(5);
		q1.enqueue("Juan");
		q1.enqueue("Pedro");
		q1.enqueue("Ana");
	}

	@Test
	void testThis() {
		assertTrue(q1.equals(q1));
	}

	@Test
	void testNull() {
		assertFalse(q1.equals(null));
	}

	@Test
	void testClass() {
		Queue<String> q2 = new LinkedQueue<String>();
		q2.enqueue("Juan");
		q2.enqueue("Pedro");
		q2.enqueue("Ana");
		assertFalse(q1.equals(q2));
	}

	@Test
	void testIgual1() {
		Queue<String> q2 = new ArrayQueue<String>();
		q2.enqueue("Juan");
		q2.enqueue("Pedro");
		q2.enqueue("Ana");
		assertTrue(q1.equals(q2));
	}

	@Test
	void testIgual2() {
		Queue<String> q2 = new ArrayQueue<String>(3);
		q2.enqueue("Juan");
		q2.enqueue("Juan");
		q2.enqueue("Juan");
		q2.dequeue();
		q2.dequeue();
		q2.enqueue("Pedro");
		q2.enqueue("Ana");
		assertTrue(q1.equals(q2));
	}
}
```

2) Agregar a la clase **AbstractBinaryTree.java** el siguiente método:

```
/**
 * Verifica si el árbol t es un árbol completo
 *
 * @return true si es un árbol completo o false si no lo es
 */
	public boolean complete()
```



Árbol binario completo: Se dice que un árbol binario de altura k está completo si está lleno hasta la altura k-1 y el último nivel está ocupado de izquierda a derecha.

Árbol binario lleno: se dice que un árbol binario está lleno si es un árbol binario de altura k que tiene 2^{k+1} - 1 nodos. (Para esta definición consideramos que la altura de la raíz es cero)

Para su desarrollo considere la posibilidad de utilizar una lista de nodos (*List<Node<E>>*) del tamaño de un árbol binario lleno inicializada en *null*.

Luego complete la lista con los nodos del árbol. Dado el padre es fácil determinar la posición que ocupará el hijo dentro de la lista:

Hijo izquierdo: 2 * posición del padre + 1
Hijo derecho: 2 * posición del padre + 2

Por último recorra la lista y si hay un *null* intercalado en la misma el árbol binario no es completo.

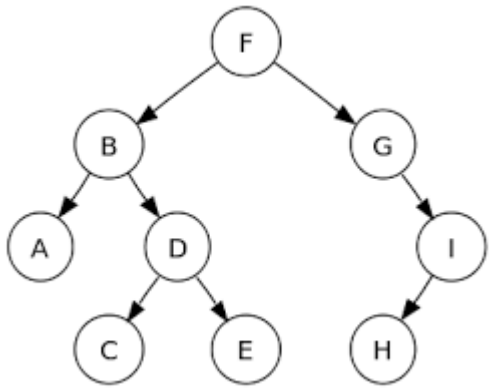
Por ejemplo:

Completo	1	2	3	4	5	6	7
Completo	1	2	3	4	5	6	null
No completo	1	2	3	4	5	null	7

3) Agregar a la clase **AbstractTree.java** los siguientes métodos:

```
/**
 * Retorna una lista con todos los nodos externos
 */
public List<Position<E>> externals()

/**
 * Retorna un map de todas las ramas de un árbol. La clave es el nodo externo y
 * la rama una lista de todos los nodos internos hasta llegar a la raíz
 */
public Map<Position<E>, List<Position<E>>> branchMap() {
```

Por ejemplo, dado el árbol:



El resultado mostrado por consola sería similar al siguiente:

Externos: H E C A

H: I G F
E: D B F
A: B F
C: D B F

IMPORTANTE:

Subir solamente los archivos con extensión .java que contiene el código elaborado.