Practico 3 – Árboles Binarios

1. Dadas las siguientes clases, construya una clase Tree, que implemente la interface MyTree. Dicha clase Tree debe comportarse como un árbol binario. Implemente los métodos insert, find y delete de forma recursiva.

```
public class Node<K, T> {
      K key;
      T data;

      Node<K, T> leftChild;
      Node<K, T> rightChild;
}

public interface MyTree<K, T> {

      T find(K key);

      insert (K key, T data, K parentKey);

      void delete (K key);

}
```

2. Incorpore dentro de la interfaz MyTree las siguientes operaciones:
   - size, cuenta la cantidad de elementos dentro del árbol.
   - countLeaf, cuenta la cantidad de hojas de un árbol.
   - countCompleteElements, el número de nodos con subárboles izquierdo y derecho no nulos.

   Realice las 3 operaciones con algoritmos recursivos.

3. Incorpore dentro de la interfaz MyTree las siguientes operaciones que recorren un Arbol
   - List<K> inOrder(), realiza la recorrida del árbol usando el algoritmo inOrder y devuelve una lista con los elementos visitados.
   - List<K> preOrder(), realiza la recorrida del árbol usando el algoritmo preOrder y devuelve una lista con los elementos visitados.
   - List<K> postOrder(), realiza la recorrida del árbol usando el algoritmo postOrder y devuelve una lista con los elementos visitados.

   Realice implementaciones recursivas de las 3 operaciones. Para devolver el resultado en una lista, utilice el TAD lista implementado en prácticos anteriores (puede usar la que considere mas útil para el problema).

4. Siguiendo los algoritmos de recorrida se tiene un cuarto tipo de recorrida que se llama recorrida por nivel. Este algoritmo lo que realiza es recorrer todos los nodos del árbol yendo por nivel de arriba abajo de izquierda a

derecha. Para implementar este algoritmo se requiere utilizar TAD auxiliares. Incorpore esta operación a la operación MyTree.

5. Agregue una operación a la interfaz MyTree que reciba una cadena de caracteres escritas en notación posfija y genere un árbol algebraico.
   - void loadPostFijaExpression (String sPostFija); que lo que realice es tomar la expresión postfija de la cadena de caracteres y genere el árbol aritmético. El algoritmo a realizar es similar a cuando se convierte una expresión de Posfija a Infija usando el TAD Stack.
   - Genere un TestUnitario en JUnit que pruebe esta operación y luego de generar la estructura, realice la recorrida inOrder (del ejercicio 3) y muestre el resultado en pantalla.

6. Genere una nueva interfaz MyBinarySearchTree para representar un árbol binario de búsqueda.

```
public class NodeBST <K extends Comparable<K>, T> {
      K key;
      T data;

      NodeBST <K, T> leftChild;
      NodeBST <K, T> rightChild;
}

public interface  MyBinarySearchTree <K extends Comparable<K>, T>
{

      T find(K key);

      insert (K key, T data);

      void delete (K key);

      …

}
```

Tenga en cuenta que las operaciones find, insert y delete tienen que respetar que es un ABB.

7. Copie las operaciones de recorridas InOrder, PreOrder y PostOrder de la implementación de MyTree a MyBinarySearchTree y verifique que el comportamiento es correcto.