ALGORITMICA Y PROGRAMACION II

TRABAJO PRACTICO N° 7

ARBOLES

Nota1: para cada clase hacer un programa que pruebe cada uno de los métodos que provee.

Nota2: documentar cada clase y utilizar Javadoc para generar la misma.

1. Realizar el diagrama de clases del TAD de árbol binario. Cargar un árbol con nombres de métodos que provee el TAD de forma tal que quede balanceado. Recorrer el árbol en preorder, inorder, postorder y por niveles. Mostrar cuales son los hijos de la raíz. La altura desde la raíz. Dar un ejemplo donde se muestre el hermano de un nodo dato. De la profundidad de un nodo que sea hoja.

2. Crear un árbol de expresiones que represente la siguiente expresión aritmética: “(((5+2) ∗ (2−1))/((2+9)+((7−2)−1)) ∗8)”. Recorrer el árbol en postorder para obtener la notación postfija del mismo. Calcular el resultado de la expresión utilizando una pila (apilar los operando, cuando llega un operador desapilar los operando, realizar el cálculo y apilar el mismo)

3. Cambiar la implementación del método *breadthfirst()*, para que utilice una pila en lugar de una cola. Indicar en qué orden se visita cada uno de los nodos.

4. Implementar el método *clone()* y el método *equals()* en un árbol binario enlazado (clase **LinkedBinaryTree**)

5. Implementar los siguientes métodos en la clase **AbstractTree**

```
/**
 * Retorna una colección iterable con las posiciones de los ancestros hasta la
 * raíz
 *
 * @param p posición del primer nodo
 * @return colección iterable con las posiciones de los ancestros hasta la raíz
 */
public Iterable<Position<E>> ancestor(Position<E> p);

/**
 * Busca un elemento dentro del árbol y retorna su posición o null si no se
 * encuentra
 *
 * @param e elemento a buscar
 * @return la posición donde se encuentra el elemento "e" o null si no se
 *         encuentra
 */
public Position<E> search(E e);

/**
 * Retorna una lista con todas las posiciones de los elementos encontrados
 *
 * @param e elemento a buscar
 * @return lista con las posiciones donde se encuentra el elemento "e"
 */
public List<Position<E>> searchAll(E e);

/**
 * Indica si un árbol tienen elementos duplicados
 *
```

TRABAJO PRACTICO N° 7

ARBOLES

```
 * @return "true" si tiene elementos duplicados o "false" caso contrario
 */
public boolean duplicate();

/**
 * Retorna un iterable con todas las posiciones de los nodos externos (hijos)
 *
 * @return iterable de las posiciones de los nodos externos
 */
public Iterable<Position<E>> listChildren();

/**
 * Retorna un iterable con todas las posiciones de la rama más larga de un árbol
 * comenzando desde un nodo externo hasta llegar a la raíz
 *
 * Si hay más de una rama con la misma profundidad, retorna una de las ramas que
 * cumpla con la condición dada
 *
 * @return iterable de las posiciones de rama más larga de un árbol
 */
public Iterable<Position<E>> listGreaterAncestor();

/**
 * Retorna un iterable con los valores de todos los nodos que tienen la
 * profundidad depth
 *
 * @param depth profundidad
 *
 * @return iterable con los valores de los nodos que tienen la profundidad depth
 */
public Iterable<E> listDepth(int depth);
```

6. Implementar los siguientes métodos en la clase **AbstractBinaryTree**

```
/**
 * Un árbol binario es lleno (full) si cada nodo interno tiene dos hijos o
 * ninguno
 *
 * @return retorna "true" si es un árbol binario lleno o "false" en caso
 *         contrario
 */
public boolean full();

/**
 * Un árbol binario de expresiones contiene en sus nodos externos operandos y en
 * sus nodos internos operadores
 *
 * @param op lista de operadores válidos
 * @return retorna "true" si es árbol binario de expresiones o "false" en caso
 *         contrario
 */
public boolean validExpressionTree(List<E> op)
```

7. Implementar los siguientes métodos en la clase **LinkedBinaryTree**

```
/**
 * Reemplaza todas las ocurrencias de un elemento por otro.
 *
 * @param search  elemento a buscar
 * @param replace elemento a remplazar
 * @return retorna un iterable con todas las posiciones de los elementos
 *         reemplazados
 */
public Iterable<Position<E>> replaceAll(E search, E replace)

/**
 * Crea un árbol a partir de un árbol binario pasado en un arreglo
 *
```

ALGORITMICA Y PROGRAMACION II

TRABAJO PRACTICO N° 7

ARBOLES

```
 * Por ejemplo dado el siguiente array:
 *
 * {"/", "*", "+", "+", "4", "-", "2", "3", "1", null, null, "9", "5", null,
 * null }
 *
 * Crea un árbol cuyo recorrido es el siguiente:
 *
 * Por nivel: / * + + 4 - 2 3 1 9 5
 *
 * Inorder: 3 + 1 * 4 / 9 - 5 + 2
 *
 * @param array representación de un árbol binario en un arreglo
 *
 */
public LinkedBinaryTree(E array[])

/**
 * Retorna una lista que contiene la representación de un árbol binario.
 *
 * @return Lista que contiene la representación de un árbol binario.
 */
public List<E> ListBinaryTree()
```