Nombre y Apellido: ….…… N° Legajo: ……

Primer Parcial de Estructuras de Datos y Algoritmos

Segundo Cuatrimestre de 2011 – 20/09/2011

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*

Condición Mínima de Aprobación: Tener por lo menos dos ejercicios con B-

Consideraciones a tener en cuenta. MUY IMPORTANTE

- **El ejercicio que no respete estrictamente el enunciado será anulado.**
- **Se puede entregar el examen escrito en lápiz.**
- **Se tendrán en cuenta la eficiencia y el estilo de programación.**
- **Los teléfonos celulares deben estar apagados.**

Ejercicio 1

Se tiene la siguiente clase que representa un árbol binario:

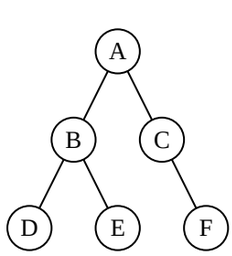
```
public class BinaryTree<T> {
   private T value;
   private BinaryTree<T> left, right;

   public BinaryTree(T value, BinaryTree<T> left, BinaryTree<T> right) {
          this.value = value;
          this.left = left;
          this.right = right;
   }
   public T getValue() {
          return value;
   }
   public BinaryTree<T> getLeft() {
          return left;
   }
   public BinaryTree<T> getRight() {
          return right;
   }
}
```

Implementar un método que dado un árbol binario y una lista de valores verifique si dicha lista se corresponde con el recorrido postorder del árbol:

```
public static <T> boolean checkPostorder(BinaryTree<T> tree, List<T> values)
```

El método no debe crear estructuras auxiliares (es decir, no es válido crear una lista con el recorrido postorder y luego comparar con la lista recibida por parámetro).

Ejemplo: si se tiene el siguiente árbol binario,



La única lista para la cual el algoritmo retornará **true** es {D, E, B, F, C, A}.

Los nodos no deben ser visitados más de una vez

Ejercicio 2

Se cuenta con la siguiente interfaz que representa una lista circular simplemente encadenada:

```
public interface CircularList<T> {

   /**
    * Agrega un elemento al final de la lista.
    */
   public void addLast(T elem);

   /**
    * Permite iterar por los elementos de la lista. La primera vez que se lo
    * invoca retorna el primer elemento, luego el segundo y así sucesivamente.
    * Luego de retornar el último, la próxima invocación vuelve a retornar el primero.
    * Lanza NoSuchElementException únicamente cuando la lista está vacía.
    */
   public T getNext();

   /**
    * Hace que la próxima invocación a getNext retorne el primer elemento de la lista.
    */
   public void reset();

   /**
    * Corta la lista en la posición actual. Esta lista (this) mantiene todos
    * los elementos desde el primero hasta el último retornado por getNext().
    * Los elementos restantes son almacenados en una nueva lista, que es retornada
    * por este método.
    * Lanza IllegalStateException si nunca se invocó a getNext antes de llamar a este
    * método.
    */
   public CircularList<T> split();
}
```

A continuación se muestra un ejemplo de uso:

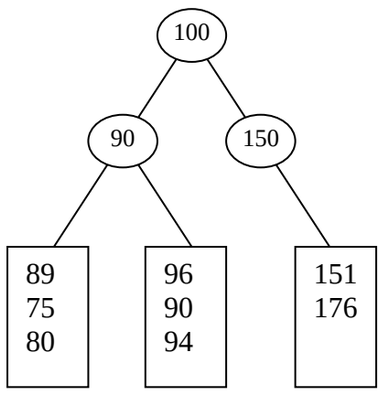
```
public class CircularListImplTest {

   public static void main(String[] args) {
          CircularList<String> list1 = new CircularListImpl<String>();

          list1.addLast("A"); list1.addLast("B"); list1.addLast("C");

          print(list1);       // imprime "A B C A B C A B C A "

          list1.addLast("D"); list1.addLast("E"); list1.addLast("F");

          print(list1);       // imprime "B C D E F A B C D E "

          list1.reset();
          list1.getNext();    // A
          list1.getNext();    // B
          list1.getNext();    // C

          CircularList<String> list2 = list1.split();

          print(list1);       // imprime "A B C A B C A B C A "
          print(list2);       // imprime "D E F D E F D E F D "

          list2.reset(); list2.getNext(); list2.getNext(); list2.getNext();

          CircularList<String> list3 = list2.split();

          print(list2);       // imprime "D E F D E F D E F D "
          print(list3);       // Lanza NoSuchElementException porque la lista está vacía
   }

   private static <T> void print(CircularList<T> list) {
          for (int i=0; i<10; i++) {
                System.out.print(list.getNext() + " ");
          }

          System.out.println();
   }
}
```

Realizar una implementación de CircularList. **Todos los métodos a implementar deben resolverse con complejidad temporal O(1). No utilizar clases de la API de Java.**

Ejercicio 3

Se tiene un árbol binario de búsqueda donde la información (de tipo T) se almacena únicamente en las hojas, que contienen un vector con los datos. En los nodos internos se almacena únicamente una referencia para poder acceder a los datos.

Ejemplo: el siguiente árbol contiene los elementos 75, 80, 89, 90, 94, 96, 151, 176



Las hojas pueden contener N claves, donde N es un entero que se recibe en el constructor del árbol.
Para todo nodo interno, su subárbol izquierdo contiene elementos menores a él, y su subárbol derecho elementos mayores o iguales a él.

Se pide:

a) construir la estructura del árbol

b) escribir el método booleano **belongs** que recibe un tipo T y devuelve true si el mismo pertenece al árbol.