Nombre y Apellido: ....…… N° Legajo: ……

Primer Parcial de Estructuras de Datos y Algoritmos

Segundo Cuatrimestre de 2010 – 17/09/2010

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.**

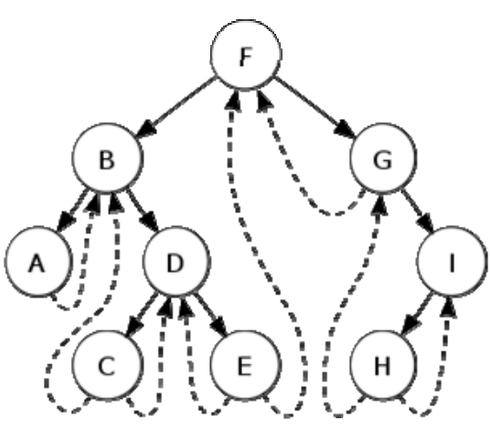
Ejercicio 1

```
public class BST<T> implements BinarySearchTree<T> {

   private Node root;
   private Comparator<? super T> cmp;

   public BST(Comparator<? super T> cmp) {
      this.root = null;
      this.cmp = cmp;
   }

   public void add(T value) {
      root = add(root, value);
   }


   private Node add(Node node, T value) {
          ...
   }



   private class Node {
      T value;
      Node left;
      Node right;

      Node(T value) {
         this.value = value;
      }
   }
}
```

Un "Threaded Binary Tree" es un árbol binario tal que

a. si el hijo derecho de un nodo debe ser null, apunta al sucesor inorder del nodo
b. si el hijo izquierdo debe ser null, apunta al predecesor inorder del nodo
c. las inserciones y borrados se realizan de la misma forma que en un BST, pero actualizando -de ser necesario- las referencias al sucesor y predecesor inorder.

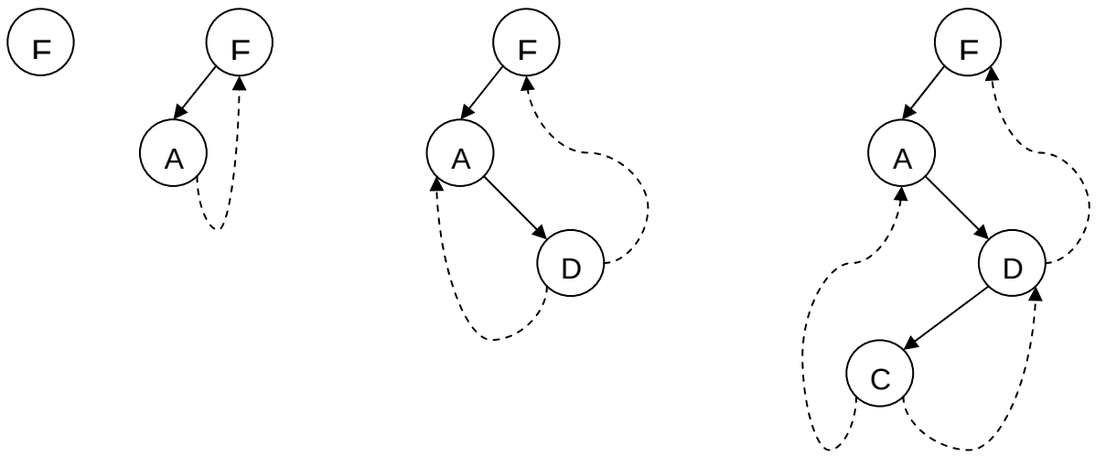
Ejemplo



Basándose en la clase BST crear la clase ThreadedBT, que contenga los siguientes métodos

a. inOrder que no reciba parámetros e imprima el valor de los nodos en forma inorder.
b. add, que recibe un valor y lo inserta en forma ordenada. Si el valor ya estaba, no hace nada.

Ejemplo: si se insertan las claves F, A, D y C el árbol, en cada paso, sería



Ejercicio 2

Se cuenta con la siguiente interfaz que representa una lista ordenada (se admiten elementos repetidos), en donde las operaciones ofrecidas son: insertar un elemento, imprimir la lista, y "deshacer" la última inserción realizada. Para el criterio de ordenación se debe proporcionar un comparador como parámetro en el constructor de la clase que implemente la interfaz.

```
public interface SortedList<T> {

   /**
    * Agrega un elemento a la lista (en la posición correspondiente según el comparador).
    * No se debe permitir almacenar valores null, si se invoca con null
    * lanza IllegalArgumentException.
    */
   public void add(T elem);

   /**
    * Elimina el elemento agregado más recientemente. Se lo puede invocar sucesivas veces,
    * eliminando así los elementos en el orden inverso al que fueron agregados a la lista.
    * Si la lista está vacía no hace nada.
    */
   public void undo();

   /**
    * Imprime los elementos de la lista.
    */
   public void print();
}
```

A continuación se muestra un ejemplo de uso:

```
public static void main(String[] args) {

   SortedList<String> list = new SortedListImpl<String>(new Comparator<String>() {
          public int compare(String o1, String o2) {
                 return o1.compareTo(o2);
          }
   });

   list.add("C");
   list.add("A");
   list.add("D");
   list.add("B");
   list.print();       // imprime "A B C D"
   list.undo();
   list.print();       // imprime "A C D"
   list.undo();
   list.undo();
   list.print();       // imprime "C"
}
```

Realizar una implementación de dicha interfaz, sin utilizar colecciones ya implementadas en la API de Java, y cumpliendo con los siguientes requerimientos de eficiencia:

- Los métodos **add** y **print** deben tener complejidad temporal O(N), siendo N la cantidad de datos almacenados.
- El método **undo** debe tener complejidad temporal O(1).
- El espacio utilizado por la estructura para almacenar los datos debe ser O(N), siendo N la cantidad de datos almacenados.

Ejercicio 3

Agregar a la clase BST un método getInOrder, que reciba dos números enteros positivos **inf** y **sup** y retorne una lista ordenada que contenga los elementos cuyo orden dentro del recorrido inorder del árbol esté entre **inf** y **sup**, donde el primer elemento del recorrido inorder recibe el orden 1. El método no debe crear estructuras auxiliares (no es aceptable armar una lista con el recorrido inorder completo y retornar una sublista de la misma).
Si el rango es inválido retorna la lista vacía

Ejemplo:

inf	**sup**	respuesta
1	1	{10}
1	3	{10, 30, 40}
4	6	{50, 70, 100}
6	20	{100, 250, 260, 270}
10	5	{}
-1	2	{}
50	60	{}
0	1	{}