DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Material permitido: NINGUNO. **Duración:** 2 horas

Entregue este folio con sus datos consignados

Alumno:	**Identificación:**
C. Asociado en que realizó la Práctica Obligatoria:	

P1 (1'5 puntos) **Práctica.** La Torre de Control permite añadir una nueva pista a su estructura. Una vez añadida esta nueva pista, se realiza una llamada al método `void redistributeOperations()`, que se encarga de obtener todas las operaciones que ya estaban en una pista, y las vuelve a distribuir teniendo en cuenta a la nueva pista. Se pide programar este método.

1. (1'5 puntos) Prográmese un método `T get (int index)` dentro del tipo `ListIF` que devuelva el elemento index-ésimo de una lista (comenzando desde la cabeza, cuyo índice es el 0). No se permite el uso de iteradores.

2. (2 puntos) Crear un método dentro del tipo cola (`QueueIF`), que devuelva una pila (`StackIF`) cuya cima sea la cabecera de la cola y los elementos siguientes de la pila sean los siguientes de la cola en su mismo orden. Ej: si la cola vista desde la cabeza fuese $1, 2, 3, 4, 5$, la pila devuelta vista desde la cima tendría que ser $1, 2, 3, 4, 5$.

3. El organigrama de una empresa representa gráficamente las relaciones de subordinación entre los empleados de la misma. Por ejemplo, un cierto empleado puede ser el responsable o jefe de un conjunto formado por otros empleados. A su vez, alguno de estos podría ser responsable inmediato de otro grupo de empleados, que serían sus subordinados directos y, transitivamente, subordinados (indirectos) del jefe de éste. Se trata de programar un TAD **Org**anization**Chart** (organigrama), cuyas operaciones serían:

OrgChartIF

```
// Representa el organigrama de una organización
public interface OrgChartIF{
 /* Contratar: añade un empleado (@param e) al conjunto de los
  * subordinados del @param boss (otro empleado)   */
 public void hire (Employee boss, Employee e);
 /* Elimina un empleado de la organización. Atención: puede dejar
  * sin jefe un departamento (justifíquese qué hacer entonces) */
 public void fire (Employee e);
 /* Devuelve cierto si el @param e1 es subordinado (directo o no)
  * del @param e2                                  */
 public boolean isSubordinate (Employee e1, Employee e2);
 /* Devuelve el conjunto de subordinados directos e indirectos del
  * empleado @param                                */
 public ListIF<Employee> subordinates (Employee e);
 // devuelve los subordinados directos del empleado @param
 public ListIF<Employee> department (Employee e);
 /* Devuelve los colegas (subordinados directos del mismo jefe
  * directo) del empleado @param  */
 public ListIF<Employee> colleagues (Employee e); }
```

a) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de este tipo (usando los TAD estudiados en la asignatura). Justifique su elección.

b) (4 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz **`OrgChartIF`**.

c) (0'5) ¿Qué coste asintótico temporal en el caso peor tiene el método de borrado (`fire`) en su implementación?

Consigne sus datos en todas las hojas que entregue. No se puntuarán respuestas sin justificar.

ListIF (Lista)

```
/* Representa una lista de elementos */
public interface ListIF<T>{
   /* Devuelve la cabeza de una lista*/
    *
    public T getFirst ();
    /* Devuelve: la lista excluyendo la cabeza. No
        modifica la estructura */
    public ListIF<T> getTail ();
   /* Inserta una elemento (modifica la estructura)
    * Devuelve: la lista modificada
    * @param elem El elemento que hay que añadir*/
    public ListIF<T> insert (T elem);
    /* Devuelve: cierto si la lista esta vacia */
    public boolean isEmpty ();
    /* Devuelve: cierto si la lista esta llena*/
    public boolean isFull();
    /* Devuelve: el numero de elementos de la lista*/
    public int getLength ();
    /* Devuelve: cierto si la lista contiene el
        elemento.
     * @param elem El elemento buscado */
    public boolean contains (T elem);
    /* Ordena la lista (modifica la lista)
     * @Devuelve: la lista ordenada
     * @param comparator El comparador de elementos*/
    public ListIF<T> sort (ComparatorIF<T>
        comparator);
    /*Devuelve: un iterador para la lista*/
    public IteratorIF<T> getIterator ();
}
```

StackIF (Pila)

```
/* Representa una pila de elementos */
public interface StackIF <T>{
    /* Devuelve:  la cima de la pila */
     public T getTop ();
    /* Incluye un elemento en la cima de la pila
        (modifica la estructura)
     * Devuelve: la pila incluyendo el elemento
     * @param elem Elemento que se quiere añadir */
     public StackIF<T> push (T elem);
    /* Elimina la cima de la pila (modifica la
        estructura)
     * Devuelve: la pila excluyendo la cabeza */
     public StackIF<T> pop ();
    /* Devuelve: cierto si la pila esta vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la pila esta llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos de la pila */
     public int getLength ();
    /* Devuelve: cierto si la pila contiene el
        elemento
     * @param elem Elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la pila*/
    public IteratorIF<T> getIterator ();
}
```

QueueIF (Cola)

```
/* Representa una cola de elementos */
public interface QueueIF <T>{
    /* Devuelve: la cabeza de la cola */
     public T getFirst ();
    /* Incluye un elemento al final de la cola
        (modifica la estructura)
     * Devuelve: la cola incluyendo el elemento
     * @param elem Elemento que se quiere añadir */
     public QueueIF<T> add (T elem);
    /* Elimina el principio de la cola (modifica la
        estructura)
     * Devuelve: la cola excluyendo la cabeza    */
     public QueueIF<T> remove ();
    /* Devuelve: cierto si la cola esta vacia */
     public boolean isEmpty ();
    /* Devuelve: cierto si la cola esta llena */
     public boolean isFull();
    /* Devuelve: el numero de elementos de la cola */
     public int getLength ();
    /* Devuelve: cierto si la cola contiene el
        elemento
     * @param elem elemento buscado */
     public boolean contains (T elem);
    /*Devuelve: un iterador para la cola*/
     public IteratorIF<T> getIterator ();
}
```

TreeIF (Árbol general)

```
/* Representa un arbol general de elementos */
public interface TreeIF <T>{
    public int PREORDER  = 0;
    public int INORDER   = 1;
    public int POSTORDER = 2;
    public int BREADTH   = 3;
    /* Devuelve:  elemento raiz del arbol */
     public T getRoot ();
    /* Devuelve: lista de  hijos de un arbol.*/
     public ListIF <TreeIF <T>> getChildren ();
    /* Establece el elemento raiz.
     * @param elem Elemento que se quiere poner como
        raiz*/
     public void setRoot (T element);
    /* Inserta un subarbol como ultimo hijo
     * @param child el hijo a insertar*/
     public void addChild (TreeIF<T> child);
    /* Elimina el subarbol hijo en la posicion
        index-esima
     * @param index indice del subarbol comenzando
        en 0*/
     public void removeChild (int index);
    /* Devuelve: cierto si el arbol es un nodo hoja*/
     public boolean isLeaf ();
    /* Devuelve: cierto si el arbol es vacio*/
     public boolean isEmpty ();
    /* Devuelve: cierto si la lista contiene el
        elemento
     * @param elem Elemento buscado*/
     public boolean contains (T element);
    /* Devuelve: un iterador para la lista
     * @param traversalType el tipo de recorrido, que
     * sera PREORDER, POSTORDER o BREADTH  */
     public IteratorIF<T> getIterator (int
        traversalType);
}
```

BTreeIF (Árbol Binario)

```
/* Representa un arbol binario de elementos */
public interface BTreeIF <T>{
  public int PREORDER  = 0;
  public int INORDER   = 1;
  public int POSTORDER = 2;
  public int LRBREADTH = 3;
  public int RLBREADTH = 4;
 /* Devuelve: el elemento raiz del arbol */
  public T getRoot ();
 /* Devuelve: el subarbol izquierdo o null si no
     existe */
  public BTreeIF <T> getLeftChild ();
 /* Devuelve: el subarbol derecho o null si no
     existe */
  public BTreeIF <T> getRightChild ();
 /* Establece el elemento raiz
  * @param elem Elemento para poner en la raiz */
  public void setRoot (T elem);
 /* Establece el subarbol izquierdo
  * @param tree el arbol para poner como hijo
     izquierdo */
  public void setLeftChild (BTreeIF <T> tree);
 /* Establece el subarbol derecho
  * @param tree el arbol para poner como hijo
     derecho */
```

No es necesario entregar las hojas de interfaces

```
 public void setRightChild (BTreeIF <T> tree);
/* Borra el subarbol izquierdo */
 public void removeLeftChild ();
/* Borra el subarbol derecho */
 public void removeRightChild ();
/* Devuelve: cierto si el arbol es un nodo hoja*/
 public boolean isLeaf ();
/* Devuelve: cierto si el arbol es vacio */
 public boolean isEmpty ();
/* Devuelve: cierto si el arbol contiene el elemento
 * @param elem Elemento buscado */
 public boolean contains (T elem);
/* Devuelve un iterador para la lista.
 * @param traversalType el tipo de recorrido que
     sera
   PREORDER, POSTORDER, INORDER,  LRBREADTH o
       RLBREADTH */
 public IteratorIF<T> getIterator (int
     traversalType);
}
```

ComparatorIF

```
/* Representa un comparador entre elementos */
public interface ComparatorIF<I>{
  public static int LESS  = -1;
  public static int EQUAL = 0;
  public static int GREATER = 1;
 /* Devuelve: el orden de los elementos
  * Compara dos elementos para indicar si el primero
      es
  * menor, igual o mayor que el segundo elemento
  * @param e1 el primer elemento
  * @param e2 el segundo elemento   */
  public int compare (T e1, T e2);
 /* Devuelve: cierto si un elemento es menor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isLess (T e1, T e2);
 /* Devuelve: cierto si un elemento es igual que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento */
  public boolean isEqual (T e1, T e2);
 /* Devuelve: cierto si un elemento es mayor que otro
  * @param e1 el primer  elemento
  * @param e2 el segundo elemento*/
  public boolean isGreater (T e1, T e2);
}
```

IteratorIF

```
/* Representa un iterador sobre una abstraccion de
    datos */
public interface IteratorIF<T>{
    /* Devuelve: el siguiente elemento de la
        iteracion */
     public T getNext ();
    /* Devuelve: cierto si existen mas elementos en
        el iterador */
     public boolean hasNext ();
    /* Restablece el iterador para volver a recorrer
        la estructura */
     public void reset ();
}
```