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.** Para la realización de este ejercicio se asume que se incorpora el método `void cancelOperation (OperationIF op)` a `ControlTower`. Este método se utiliza para cancelar una determinada operación (la que se le pasa por parámetro) que ya ha sido colocada en una pista de entre las gestionadas por la torre de control. El cancelar una operación supone eliminarla de la pista en la que se encuentra. Se pide implementar el método `cancelOperation` utilizando los métodos públicos de `ControlTowerIF` y `RunwayIF`.

---

1. (1'5 puntos) Prográmese un método `ListIF<T> leaves()` que devuelva el contenido de todas las **hojas** de un árbol binario (según la interfaz `BTreeIF<T>`) *sin utilizar iteradores.*

2. (2 puntos) Sean una pila (según `StackIF`) y una cola (según `QueueIF`), ambas de caracteres. Ambas estructuras contienen, por precondición, los mismos elementos (entre los que no hay repetidos) pero en, posiblemente, distinto orden. Se trata de implementar un método del TAD StackIF `QueueIF<Integer> posDiff (QueueIF<Char> pq)` que, para cada elemento, en la posición que éste aparezca en la pila (contando desde la cima, que será la cero) se devuelva en dicha posición de la cola de salida la diferencia con la posición en la que dicho elemento aparece en la cola parámetro.

   Por ejemplo, si a la pila cliente vista desde la cima $[A, B, C, D]$ se le aplica el método `posDiff` con la cola $[D, C, A, B]$, el resultado debería ser la cola $[-2, -2, 1, 3]$, ya que A está en la posición 0 de la pila (cima) y en la 2 en la cola, etc. Se valorará la eficiencia de la solución.

3. Una máquina de refrescos dispone de un cierto número de depósitos, identificados por su número de orden, cada uno de los cuales puede contener una cantidad máxima de latas de refresco. Los refrescos están mezclados, es decir, pueden estar en cualquier depósito y cada uno de estos puede contener cualquier colección de refrescos (iguales o distintos). Cada refresco tiene asignado un precio (cada tipo de refresco tiene un precio único) y la máquina es capaz de realizar cierta contabilidad (llevar el importe total de los refrescos vendidos). Se trata de programar un TAD **V**ending**Mac**hine, cuyas operaciones serían:

VMacIF

```
// Representa una máquina de venta de refrescos
public interface VMacIF{
 /* Añade un refresco de precio price al depósito container, encima
  *  de todos los anteriores (sería el último accesible) */
public void addCan (int container, float price);
/* Elimina el primer refresco del depósito container. Esta
 * operación debe incrementar la venta acumulada en el
 * precio del producto eliminado                    */
public void releaseCan (int container);
// Devuelve cierto si el depósito container está vacío
public boolean isEmpty (int container);
// Devuelve cierto si el depósito container está lleno
public boolean isFull (int container);
// Devuelve el precio del primer producto del depósito container
public float getPrice (int container);
/* Devuelve la venta acumulada desde la última vez que se puso a
 * cero la contabilidad de la máquina */
```

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

```
    public float getSales ();
    // Pone a cero las ventas acumuladas
    public void resetSales();
    /* Devuelve el mínimo gasto necesario para obtener un cierto
    producto de precio price */
    public float getMinCost (float price)
}
```

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) (0'5 puntos) Implemente el constructor (o constructores) del tipo. Justifique sus decisiones.

c) (3'5 puntos) Basándose en las respuestas anteriores, implemente todos los métodos de la interfaz **`VMacIF<float>`**

d) (0'5) ¿Qué coste asintótico temporal en el caso peor tiene el método de consulta del mínimo gasto que garantiza un producto (`getMinCost`) en su implementación?

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 ();
}
```