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**. Impleméntese el siguiente comando de usuario para añadirlo a las funcionalidades del sistema de ficheros de la práctica.

head head - lista la primera parte de un fichero

SINTAXIS

`head path -n num`

Lista por salida estándar los primeros `num` bytes del fichero `path`

OPCIONES

-n indica que el siguiente parámetro es el número de bytes que hay que listar del principio del fichero

---

1. (1'5 puntos) Prográmese el método `int numLeafs()` dentro del tipo `TreeIF` que calcula el número de hojas que tiene un árbol. **Nota:** no se permite el uso de iteradores.

2. (1'5 puntos) Prográmese el método `void removeRepeated()` dentro del tipo `ListIF` que elimina los elementos repetidos de una lista de modo que cada elemento aparezca sólo una vez.

3. Una cola de dos terminaciones (`DequeIF`) es un tipo de datos que permite inserciones y consultas tanto por el principio como por el final. La interfaz del tipo es la siguiente:

DequeIF

```
// Representa una cola de dos terminaciones (deque)
public interface DequeIF<T> {

  /* añade un elemento por el principio
   * @param e el elemento que se desea añadir [0'75 puntos] */
   public void addFirst(T e);

  /*añade un elemento por el final
   * @param e el elemento que se desea añadir [0'75 puntos] */
  public void addLast(T e);

  /* Elimina el primer elemento de la deque (modifica la estructura)
   * @return la deque excluyendo el primer elemento [0'75 puntos] */
  public DequeIF<T> removeFirst();

  /* Elimina el último elemento de la deque (modifica la estructura)
   * @return la deque excluyendo el último elemento [0'75 puntos] */
  public DequeIF<T> removeLast();

 /* Devuelve el primer elemento de la deque
  * @return el primer elemento de la deque [0'5 puntos] */
  public T getFirst();
```

```
    /* Devuelve el último elemento de la deque
     * @return el último elemento de la deque [0'5 puntos] */
     public T getLast();

    /* Devuelve el número de elementos de la deque
     * @return el número de elementos de la deque [0'5 puntos] */
     public int getLength();
}
```

a) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de este tipo y detalle el constructor de una clase que implemente esta interfaz.

b) (4'5 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz `DequeIF<T>`. (**Nota:** las puntuaciones asignadas a cada método se indican en la especificación de dicho método en la interfaz del tipo)

c) (0'5 puntos) ¿Cuál es el coste asintótico temporal en el caso peor del método `removeLast` 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 */
   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<T>{
  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 ();
}
```