P1 (1'5 puntos) **Práctica**. Implementar el comando `ls path -s num`, que asume que la ruta `path` se corresponde con un directorio y que muestra por pantalla el nombre de los ficheros del directorio `path` cuyo número de bytes es mayor a `num`, además de los subdirectorios de `path` que contienen algún fichero cuyo número de bytes es mayor a `num` ?

---

1. (1'5 puntos) Prográmese el método `void insertAtTheEnd(T e)` dentro del tipo `ListIF<T>` que inserte un dato como último elemento de una lista.

2. (1'5 puntos) Prográmese el método `StackIF<T> reverse()` dentro del tipo `StackIF<T>` que devuelva el reverso de una pila. Ej: Dada la pila (vista desde la cima) `[1,2,3,4]`, su reverso (vista desde la cima) sería `[4,3,2,1]`.

3. Una máquina automática de venta de productos alimenticios dispone de un cierto número de *coronas* horizontales dispuestas en un eje vertical e identificadas por un ordinal comenzando por la situada más arriba, que identificaremos como número 0. Cada una de éstas puede almacenar una cantidad - la misma en todos los casos - de productos. A su vez, cada producto tiene unas características diferentes, entre las que destacaremos su nombre y su precio de venta al público (pvp) y su fecha de caducidad. Una corona puede contener cualquier mezcla de productos, de manera que la máquina dispone de un botón por cada corona que la hace avanzar para que el cliente pueda ver cada producto. Además, existe otro botón para cada corona que permite visualizar el precio del producto en exposición - el que está seleccionado, que será el único accesible -, su nombre y su fecha de caducidad.

   Se trata de programar un TAD **dispenser**, cuyas operaciones serían:

   DispenserIF

```
// Representa una máquina dispensadora de productos alimenticios
public interface DispenserIF{
/* Obtiene el producto seleccionado, que será el primero,
 * de la corona crown [0'5 puntos] */
 public ProductIF getProduct (int crown);
/* Añade un producto a la corona crown de
 * manera que sea el  último accesible) [0'5 puntos] */
 public void addProduct (int crown, ProductIF product);
 /* Elimina el primer producto de la corona crown [0'5 puntos] */
 public void removeProduct (int crown);
 /* Mueve la selección de la corona crown una posición de forma
  * circular (el primer elemento pasa a ser el último)[0'5 puntos]*/
 public void advanceCrown (int crown);
 /* Devuelve cierto si la corona crown esta vacía [0'5 puntos]*/
 public boolean isEmptyCrown (int crown);
 /* Devuelve cierto si la corona crown esta llena [0'5 puntos]*/
 public boolean isFullCrown (int crown);
 /* Devuelve cuántos productos de la máquina tienen una caducidad
  * que no supera el parámetro expiry (en días, con hoy==0)[1 punto]*/
 public int getPerishables (int expiry);
 /* Devuelve la colección de productos de la máquin cuya caducidad
  * no supera el parámetro expiry (en días, con hoy==0)[1 punto]  */
 public ListIF<Product> getAllPerishables (int expiry);
}
```

   Considérese la siguiente interfaz para productos (**ProductIF**). No se pide programar ninguna clase que implemente esta interfaz sino utilizarla.

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

ProductIF

```
// Representa un producto
public interface ProductIF{
 // Devuelve el precio de un producto
 public float getPrice ();
 // Devuelve la etiqueta de un producto
 public String getLabel ();
 // Devuelve la caducidad del producto en días (hoy==0)
 public int getExpiry ();
 // Pone el precio a un producto
 public void setPrice (float price);
 // Pone la etiqueta a un producto
 public void setLabel (String label);
 // Pone la caducidad del producto en días (hoy==0)
 public void setExpiry (int expiry);
}
```

a) (0'5 puntos) Describa detalladamente cómo realizaría la representación interna de este tipo y programe el constructor del mismo usando, en ambos casos, los TAD estudiados en la asignatura. Justifique sus decisiones.

b) (4'5 puntos) Basándose en las respuestas anteriores, implemente todos los métodos de la interfaz **DispenserIF**

c) (0'5 puntos) ¿Qué coste asintótico temporal en el caso peor tiene el método `getAllPerishables` que obtine la colección de productos que caducan en un cierto número de días?

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 ();
}
```