P1 (1'5 puntos) **Práctica.** El comando `mkdir` del sistema de ficheros de la práctica consideraba un error si la ruta completa hasta el directorio padre del que se iba a crear no existía previamente a su invocación. Impleméntese dicho comando pero modificando ese comportamiento para que, en lugar de dar error, cree la parte de la ruta que no existiese con antelación.

mkdir make directory - crea directorios

SINTAXIS

`mkdir path`

Crea un directorio. Si alguno de los directorios antecesores en la ruta `path` no existe, lo crea.

---

1. (1'5 puntos) Prográmese el método `StackIF<T> insertBottom(T e)` dentro del tipo `StackIF` que devuelve la pila resultado de insertar un elemento en la base de una pila dada. Por ejemplo, si se ejecuta el método `insertBottom(5)` sobre la pila (vista desde la cima) `[1,2,3,4]`, el resultado será `[1,2,3,4,5]`.

2. (1'5 puntos) Prográmese el método `ListIF<BTreeIF<T>> parentsLeaves()` dentro del tipo `BTreeIF <T>` que devuelve una lista con todos los subarboles que tienen *al menos* un hijo que sea una hoja.

3. Flickr es un repositorio de imagenes generado por los internautas y etiquetado por ellos mismos. En este sistema, cada imagen contiene una colección de etiquetas de texto libre que - a juicio de los usuarios- caracterizan la misma y sirve para poder localizarla mediante una búsqueda por palabras. Supongamos que internamente el sistema tiene definido un tipo de datos PhotoIF para representar las fotos y utiliza el interfaz GalleryIF para gestionar las imagenes.

GalleryIF

```
// Representa una Galería o colección de fotos
public interface GalleryIF {
   /* Devuelve la primera foto de las que esten etiquetadas con tag
    * @param tag la etiqueta con la que se compara
    * @return la primera foto de las etiquetadas con tag
    * [0'5 puntos] */
   public PhotoIF findPhoto (String tag);

   /* Obtiene la lista de todas las fotos que tienen el parametro como
    * etiqueta.  No es necesario eliminar repetidos del resultado
    * @param tag la etiqueta
    * @return la lista de todas las fotos etiquetadas con tag
    * [0'75 puntos] */
   public ListIF<PhotoIF> findAllPhotos (String tag);

   /* Obtiene la lista de todas las fotos que tienen todas las etiquetas
    * de la lista parametro. No es necesario eliminar repetidos
    * del resultado
    * @param tags la lista de etiquetas
    * @return la lista de todas las fotos que tienen todas las
    * etiquetas del parámetro [1 punto]*/
   public ListIF<PhotoIF> findAllPhotos (ListIF<String> tags);

   /* Obtiene la lista de todas las fotos de la coleccion
    * No es necesario eliminar repetidos del resultado
    * @return la lista de fotos de la colección [0'25 puntos] */
   public ListIF<PhotoIF> findAllPhotos ();

   /* Añade una foto a la coleccion
```

```
     * @param photo la foto que se desa añadir [0'5 puntos] */
    public void addPhoto (PhotoIF photo);

    /* Borra una foto de la coleccion
     * @param photo la foto que se desea borrar [0'5 puntos] */
     public void removePhoto (PhotoIF photo);

    /* Asocia la etiqueta parametro (tag) a la foto parametro
     * @param photo la foto
     * @param tag la etiqueta [0'5 puntos] */
    public void addTag (PhotoIF photo, String tag);

    /* Elimina la etiqueta parametro (tag) de la lista de etiquetas
     * de la foto
     * @param photo la foto de la que se quiere eliminar la etiqueta
     * @param tag la etiqueta que se desea eliminar de la foto
     * [0'5 puntos] */
    public void removeTag (PhotoIF photo, String tag);
}
```

a) (0'5 puntos) Describa con precisión cómo realizaría la representación interna de este tipo y detalle el constructor de una clase que implemente esta interfaz usando, en ambos casos, los TAD estudiados en la asignatura. Justifique sus decisiones.

b) (4'5 puntos) Basándose en la respuesta anterior, implemente todos los métodos de la interfaz `GalleryIF` (**Nota:** las puntuaciones asignadas a cada método se indican en la especificación de dicho método en la interfaz del tipo)

c) (0'5) ¿Cuál es el coste del método `findAllPhotos (ListIF<String> tags)` 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 ();
}
```