ESTRUCTURA DE DATOS Y ALGORITMOS **8-Febrero-2007**
Titulación: Ingeniero Técnico en Informática de Gestión **Curso: 2º**

1. Ejercicio (1 puntos)

Dado el array A={8, 3, 7, 1, 4, 9, 5, 2, 6}

(a) Escribir la secuencia de ordenación del array A por medio del algoritmo de InsertionSort.

{8, 3, 7, 1, 4, 9, 5, 2, 6}→{3, 8, 7, 1, 4, 9, 5, 2, 6}→{3, 7, 8, 1, 4, 9, 5, 2, 6}→{1, 3, 7, 8, 4, 9, 5, 2, 6}
→{1, 3, 4, 7, 8, 9, 5, 2, 6}→{1, 3, 4, 7, 8, 9, 5, 2, 6}→{1, 3, 4, 5, 7, 8, 9, 2, 6}→{1, 2, 3, 4, 5, 7, 8, 9, 6}
→{1, 2, 3, 4, 5, 6, 7, 8, 9}

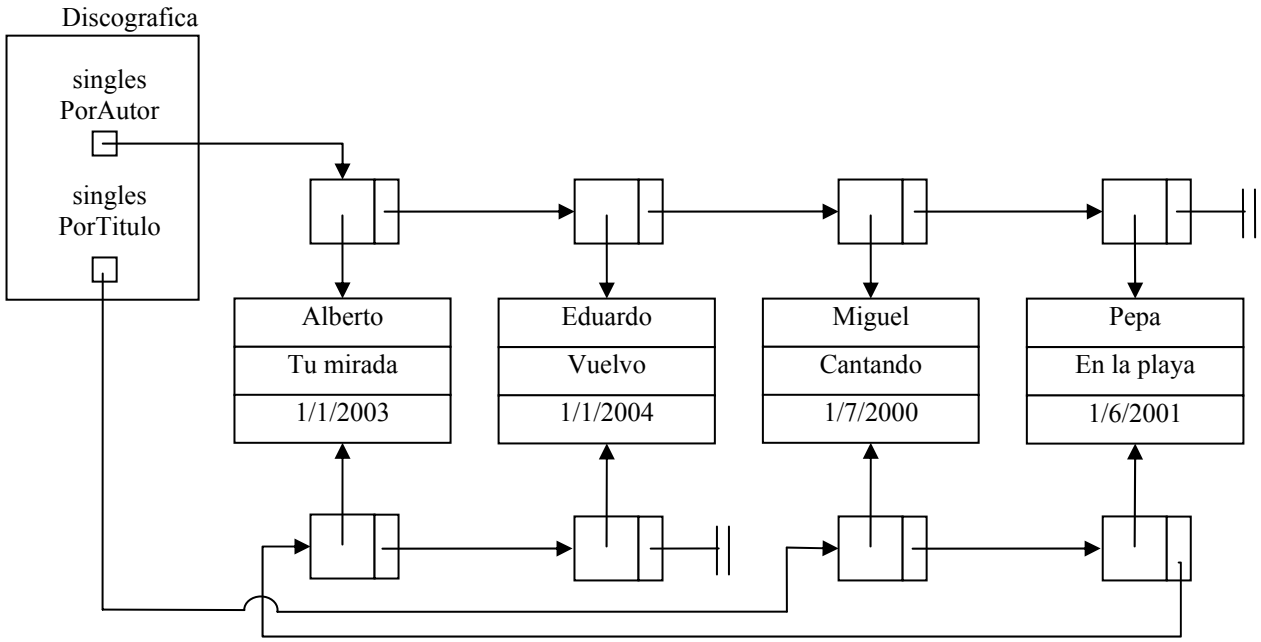
2. Ejercicio. (3 puntos)

a) Complentar la especificación

```
public class Discografica {
      LinkedList singlesPorTitulo;
      LinkedList singlesPorAutor;
      LinkedListItr itrSinglesPorTitulo;
      LinkedListItr itrSinglesPorAutor;
}
class Node {
      Object element;
      Node next;
}
```

```
public class LinkedList{
      Node top;
}
public class LinkedListItr {
      LinkedList theList;
      Node current;
      Node preceding;
}
```

b) Representación de la información en listas:



c) Especificación de los métodos que se van a utilizar

```
public void advance();
```
// avanza en la lista con la variable *current*, pasando al siguiente nodo del nodo *current*
```
public boolean isOnList();
```
// verifica si la variable *current* referencia un nodo de la lista, en caso afirmativo devuelve *true* y en
// caso contrario *false*
```
public void goFirst();
```
//actualiza la variable *current*, el cuál referencia el primer nodo de la lista
```
public Object getCurrent();
```
// devuelve el objeto que esta referenciado en el nodo, el cuál es accesible mediante la variable *current*
```
public void insert(Object obj);
```
// inserta la información obj en la lista, creando un nuevo nodo después del nodo referenciado por la
// variable *current*. Finalmente, *current* referencia el nuevo nodo creado.
```
public void insertFirst(Object obj);
```
// inserta la información en la primera posición de la lista, creando un nuevo nodo con la información
// obj. Finalmente, *current* referencia el nuevo nodo creado.

Implementación:

```
public void añadir(SingleMusical single){
  SingleMusical sAntT = null;
  SingleMusical sAntA = null;
  boolean enc = false;
  SingleMusical sm   = null;

  //buscar posición en la lista por titulo, NO puede haber repetidos
  itrSinglesPorTitulo.goFirst();

  while ( (itrSinglesPorTitulo.isOnList()) && !enc ) {
    sm = (SingleMusical) itrSinglesPorTitulo.getCurrent();
    if (sm.getTitulo().compareTo(single.getTitulo())<0) {
      sAntT = sm;
      itrSinglesPorTitulo.advance();
    }
    else enc = true;
  }

  if ((enc) && (sm.getTitulo().compareTo(single.getTitulo())==0))
      System.out.println("Error, el single con el titulo " +
      single.getTitulo() + " ya esta");
  else {
      enc = false;

      //buscar posición en la lista por autores, puede haber repetidos y
      //están ordenados por titulo
      itrCuadrosPorAutor.goFirst();
      while ( (itrCuadrosPorAutor.isOnList()) && !enc ) {
        sm = (SingleMusical)itrCuadrosPorAutor.getCurrent();
        if ((sm.getAutor().compareTo(single.getAutor())<0) ||
            ((sm.getAutor().compareTo(single.getAutor())==0) &&
             (sm.getTitulo().compareTo(single.getTitulo())<0))){
              sAntA = c;
              itrCuadrosPorAutor.advance();
        }
        else enc = true;
```

```
      }

      // insertar el single en la lista por titulo
      if (sAntT == null)
        itrSinglesPorTitulo.insertFirst(single);
      else {
        itrSinglesPorTitulo.find(sAntT);
        itrSinglesPorTitulo.insert(single);
      }
      // insertar el single en la lista por autor
      if (sAntA == null)
        itrCuadrosPorAutor.insertFirst(single);
      else {
        itrCuadrosPorAutor.find(sAntA);
        itrCuadrosPorAutor.insert(single);
      }
    }
}
```

3. Ejercicio (3 puntos)

(a) Especificar, diseñar e implementar un subprograma tal que dado un árbol binario de elementos devuelva cierto si el árbol **es completo** y falso en caso contrario. Un árbol **es completo** si todos los nodos, excepto las hojas, tienen 2 hijos.

1. Especificar / Parametrizar

Entrada: A, un arbol binario.
Salida: cierto si A es **completo**, falso en caso contrario
Llamada inicial: COMPLETO(nodo)

2. Diseño

Casos triviales

Si Vacio (A) → true
Si Hoja (A) → true
Si No Hoja (A) y ((Izqdo(A)=null) o (Dcho(A)=null)) → false

Caso General

Si No Hoja (A) y (Izqdo(A)!=null) y (Dcho(A)!=null) →
COMPLETO(Izqdo(A)) && COMPLETO(Dcho (A))

3. Implementación

```
public boolean completo(BTNode pNode) {

   if (pNode ==null)
     return true;
   else {
      if((pNode.left==null) && (pNode.right==null))
           return true;
      else {
           if((pNode.left==null) || (pNode.right==null))
                return false;
           else
                return completo(pNode.left) && completo(pNode. right);
      }
   }
}
```

(b) Especificar, diseñar e implementar un subprograma tal que dado un árbol binario de elementos devuelva cierto si el árbol **es lleno** y falso en caso contrario. Un árbol **es lleno** si, además de ser completo, todas las hojas están en el mismo nivel.

1. Especificar / Parametrizar

Entrada: A, un arbol binario.
Salida: cierto si A es **lleno**, falso en caso contrario

public boolean lleno();

A la hora de tratar un nodo debemos conocer la altura de sus hijos y si el árbol que forma dicho nodo es lleno o no → INMERSION

2. Diseño

Entrada: A, un arbol binario.
Salida: La altura y el valor booleano indicando si el árbol es lleno.

Se necesita una clase Contexto para poder devolver dos valores

```
public class Contexto {
        int altura;
        boolean es_lleno;
}
```

public Contexto lleno1(BTNodeA);

Casos triviales

Si Vacio (A) → devolver new Contexto(-1,true)
Si Hoja (A) → devolver new Contexto(0,true)
Si No Hoja (A) y ((Izqdo(A)=null) o (Dcho(A)=null)) →
devolver new Contexto(**a**, false)
// el valor de **a**, sería la altura del hijo que no es vacío más uno

Caso General

```
Si No Hoja (A) y (Izqdo(A)!=null) y (Dcho(A)!=null) →
   contextoIzq = LLENO1 (Rama_izq(A));
   contextoDch = LLENO1 (Rama_dch(A));
   Si (contextoIzq. altura > contextoDch.altura) entonces
      altura = contextoIzq.altura+1;
   si no
      altura = contextoDch.altura+1;
   fin si
      es_lleno = (contextoIzq.es_lleno && contextoDch.es_lleno
                                       && (contextoIzq.altura ==  contextoDch.altura));

contexto = new Contexto(altura, es_lleno);
devolver contexto;
```

3-Implementación

```
public boolean lleno() {
      Contexto contexto = lleno1(bTree.root);
      return contexto.es_lleno;
}

private Contexto lleno1 (BTNode nodo) {
      int altura;
      boolean es_lleno;
      if (nodo==null)
            return new Contexto(0, true);
      else if ((nodo.left==null) && (nodo.right==null) ) {
            return new Contexto(1, true);
      }
      else if ((nodo.left==null) || (nodo.right==null) ) {
            return new Contexto(0, false);
      }
      else {
            Contexto contextoIzq = lleno(nodo.left);
            Contexto contextoDch = lleno(nodo.right);
            if (contextoIzq. altura > contextoDch. altura) {
                  altura = contextoIzq. altura + 1;
            }
            else if (contextoDch.profundidad >= contextoIzq.profundidad) {
                  altura = contextoDch. altura + 1;
            }

            es_lleno = (contextoIzq. prof.es_lleno &&
                        contextoDch. prof.es_lleno &&
                        (contextoIzq. altura == contextoDch. altura));

            Contexto contexto = new Contexto(altura, es_lleno);
            return contexto;
      }
}

public class Contexto {
      int altura;
      boolean es_lleno;
      public Contexto(int altura, boolean es_lleno) {
        this. altura = altura;
        this.es_lleno = es_lleno;
      }
      public void print() {
        System.out.println("profundidad: "+profundidad+" es_lleno: "+es_lleno);
      }
}
```

4. Ejercicio (3 puntos)

Para realizar la funcionalidad que se pide en el enunciado, se requiere cambiar el orden cronológico de los datos del fichero antes de ser imprimidos. Para ello, utilizaremos la estructura de datos Pila con la siguiente estrategia:

1. La información que se almacenará, serán objetos de tipo Desastre.
2. La cima de la pila indicará el desastre más reciente, en el cual el número de victimas es el mayor de la historia comparándolo con años anteriores.
3. Los datos de un desastre serán apilados, si el número de victimas del desastre es mayor al número de victimas del desastre histórico anterior, es decir, el desastre que está en la cima de la pila.

```
Public void imprimirDesastres(String nombreFichero){
      Token fichero = new Token(nombreFichero) ;
      Queue historico  =new Queue() ;
      Desastre des;
      //Leer datos del fichero y calcular el histórico
      While(fichero.hasMoreTokens){
            Datos d = fichero.getToken() ;
            Desastre des = ;
            If (historico.isEmpty())
                  historico.enqueue(des);
            else{
                  Desastre des = historico.top():
                  If (d.getNumVictimas() > des.getNumVictimas())
                        historico.enqueue(new Desastre(d.getFecha(),
                                                      d.getNumVictimas()));
            }
      }
      //imprimir el histórico
      While (!historico.isEmpty()){
            des = historico.pop();
            System.out.println(des.getFecha()+" "+des.getNumVictimas());
      }
}
```