Resolución del Segundo Parcial de EDA (5 de Mayo de 2014)

1.- En la clase `MonticuloBinario` (ver Anexo), **se pide** implementar un método `igualesA` que devuelva el número de elementos de un Montículo Minimal iguales a uno dado `e`. El coste temporal del método diseñado deberá ser lineal con el número de elementos del Montículo menores o iguales que `e`; así, por ejemplo, su coste temporal será Θ(`talla`) si todos los elementos del Montículo sobre el que se aplica son menores o iguales que `e` y Θ(1) si todos son mayores. (**2.5 puntos)**

```
public int igualesA(E e) {
   return igualesA(e, 1);
}
private int igualesA(E e, int i) {
   int res = 0;
   if (i <= talla) {
     int resC = elArray[i].compareTo(e);
     if (resC <= 0) {
       if (resC == 0) res++;
       res += igualesA(e, 2 * i) + igualesA(e, 2 * i + 1);
     }
   }
   return res;
}
```

2.- Se pide diseñar un método genérico, estático e iterativo `fusion` que dadas `qP1` y `qP2`, dos Colas de Prioridad implementadas mediante Min-Heaps, devuelva una Lista Con PI que contiene los datos de `qP1` y `qP2` ordenados ascendentemente; dicho método no puede hacer uso de ninguna estructura de datos auxiliar para calcular su resultado y, además, `qP1` y `qP2` deben estar vacías al concluir su ejecución. El coste temporal del método diseñado deberá ser $\Theta(x \cdot \log x)$, siendo `x` la suma de las tallas de `qP1` y `qP2` o, equivalentemente, la talla de la Lista resultado. **(2.5 puntos)**

NOTA: por si resulta de utilidad, en el Anexo figuran las interfaces `ListaConPI` y `ColaPrioridad`.

```
public static <E extends Comparable<E>> ListaConPI<E> fusion(ColaPrioridad<E> qP1, ColaPrioridad<E> qP2) {
   ListaConPI<E> res = new LEGListaConPI<E>();
   while (!qP1.esVacia() && !qP2.esVacia()) {
     if ((qP1.recuperarMin()).compareTo(qP2.recuperarMin()) < 0)
            res.insertar(qP1.eliminarMin());
     else   res.insertar(qP2.eliminarMin());
   }
   while (!qP1.esVacia()) res.insertar(qP1.eliminarMin());
   while (!qP2.esVacia()) res.insertar(qP2.eliminarMin());
   return res;
}
```

3- Se pide diseñar un método `modaDe` que, con el menor coste temporal posible, devuelva la moda de una Lista Con Punto de Interés `l` dada, es decir que devuelva el elemento de `l` que se repite más veces. Para ello se deberá usar un `Map` (ver Anexo) implementado mediante una Tabla Hash. **(2.5 puntos)**

```
public static <E> E modaDe(ListaConPI<E> l) {
    E moda = null; int frecModa = 0;
    Map<E, Integer> d = new TablaHash<E, Integer>(l.talla());
    for (l.inicio(); !l.esFin(); l.siguiente()) {
       E e = l.recuperar();
       Integer freqE = d.recuperar(e);
       if (frecE != null) frecE++; else frecE = 1;
       d.insertar(e, freqE);
       if (frecE > frecModa) {
         frecModa = frecE;
         moda = e;
       }
    }
    return moda;
}
```

4- En la clase `TablaHash` (ver Anexo), **se pide** diseñar un método `masDeLaMedia` que, con el menor coste posible, devuelva el número de cubetas de una Tabla Hash que tienen una longitud mayor que la media. **(2.5 puntos)**

- SI las cubetas se implementan mediante Listas Con PI.

```
public int masDeLaMedia() {
    int res = 0; double longMedia = this.factorCarga();
    for (int i = 0; i < elArray.length; i++) if (elArray[i].talla() > longMedia) res++;
    return res;
}
```

- SI las cubetas se implementan mediante Listas Directas.

```
public int masDeLaMedia() {
    int res = 0; double longMedia = this.factorCarga();
    for (int i = 0; i < elArray.length; i++) {
       int longCubeta = 0;
       for (EntradaHash<C, V> e = elArray[i]; e != null; e = e.siguiente) longCubeta++;
       if (longCubeta > longMedia) res++;
    }
    return res;
}
```

ANEXO

Las interfaces `ListaConPI`**,** `Map` **y** `ColaPrioridad` **del paquete** `modelos`**.**

```
public interface ListaConPI<E> {
  void insertar(E e);
  /** SII !esFin() */ void eliminar();
  void inicio();
  /** SII !esFin() */ void siguiente();
  void fin();
  /** SII !esFin() */ E recuperar();
  boolean esFin();
  boolean esVacia();
  int talla();
}
```

```
public interface Map<C, V> {
      V insertar(C c, V v);
      V eliminar(C c);
      V recuperar(C c);
      boolean esVacio();
      int talla();
      ListaConPI<C> claves();
}
```

```
public interface ColaPrioridad<E extends Comparable<E>> {
   void insertar(E e);
  /** SII !esVacia() */ E  eliminarMin();
  /** SII !esVacia() */ E  recuperarMin();
  boolean esVacia();
}
```

Las clases `TablaHash` **y** `EntradaHash` del paquete `deDispersion`**.**

- Implementación de **cubetas** con **Listas Con Punto de Interés:**

```
class EntradaHash<C, V> {
    C clave; V valor;
    public EntradaHash(C clave, V valor) { this.clave = clave; this.valor = valor; }
}
public class TablaHash<C, V> implements Map<C, V> {
    protected ListaConPI<EntradaHash<C,V>>[] elArray; protected int talla;
    protected int indiceHash(C c) {…}
    public TablaHash(int tallaMaximaEstimada) {…}
    public V recuperar(C c) {…}
    public V eliminar(C c) {…}
    public V insertar(C c, V v) {…}
    public final double factorCarga() {…}
    public final boolean esVacio() {…}
    public final int talla() {…}
    public final ListaConPI<C> claves() {…}
    … //otros métodos
}
```

- Implementación de **cubetas** con **Listas Directas:**

```
class EntradaHash<C, V> {
    C clave; V valor;
    EntradaHash<C, V> siguiente;
    public EntradaHash(C clave, V valor, EntradaHash<C, V> siguiente){
          this.clave = clave; this.valor = valor; this.siguiente = siguiente;
    }
}
public class TablaHash<C, V> implements Map<C, V> {
    protected EntradaHash<C,V>>[] elArray; protected int talla;
    … // Los nombres de los métodos coinciden con los de la anterior clase TablaHash
}
```

La clase `MonticuloBinario` del paquete `jerarquicos`**.**

```
public class MonticuloBinario<E extends Comparable<E>> implements ColaPrioridad<E> {
    protected E elArray[]; protected static final int CAPACIDAD_POR_DEFECTO = 11;
    protected int talla;
    @SuppressWarnings("unchecked") public MonticuloBinario() {…}
    public void insertar(E e) {…}
    /** SII !esVacia() */ public E eliminarMin {…}
    /** SII !esVacia() */ public E recuperarMin(){…}
    // Otros métodos
}
```