EJERCICIO 1. Método *masFrecuente*

Implementa en la clase **BuscadorDeLaBibl** un método llamado **masFrecuente**, que reciba una **ListaConPI<Termino>** y devuelva, de entre los términos de la lista, el **Termino** con mayor frecuencia de aparición en la Biblioteca Digital (BD).

El método debe imprimir (en salida estándar) tanto el término más frecuente como su frecuencia antes de devolver el resultado.

Los términos de la lista pueden no aparecer en el índice de la BD. Procesar un término inexistente en el índice **NO** debe interrumpir el método; este siempre debe acabar. Al detectar un término que no aparece, el programa debe imprimir un aviso, siguiendo el formato del ejemplo.

Puedes asumir que la lista que se pasa como parámetro no estará vacía y que al menos uno de los términos de la lista aparecerá en el índice de la BD.

Se recuerda que, en la clase **BuscadorDeLaBibl,** la información de la biblioteca digital (BD) se almacena en un **Map** llamado **index** cuyas claves son objetos **Termino** y cuyos valores son objetos **ListaConPI<BuscadorDeLaBibl.Posting>**.

Test. Añade la clase **TestMasFrecuente** (que está en el fichero adjunto) al paquete **biblioteca** de tu proyecto BlueJ. Ejecuta el método **main** en la clase **TestMasFrecuente**. El resultado debe ser el siguiente:

Atención: El Termino tocadiscos no está en el índice.
El Termino más frecuente es tabla, que aparece 614 veces

Estas dos líneas se mostrarán después de las 12 líneas que informan de la indexación de los 10 libros a partir de los cuales se crea la BD. Esta salida se obtiene porque la lista que, en el **main**, se le pasa al método **masFrecuente** es: **["ficheros", "tabla", "tocadiscos", "programa", "estrellas", "ladrillos"]**, y es **"tabla"** el término más frecuente, con 614 apariciones, y **"tocadiscos"** el único término de esa lista que no está en el índice.

Solución

```
public Termino masFrecuente(ListaConPI<Termino> lt) {
    int maxFreq = -1;
    Termino res = null;
    ListaConPI<BuscadorDeLaBibl.Posting> postings;
    for (lt.inicio(); !lt.esFin(); lt.siguiente()) {
        Termino term = lt.recuperar();
        postings = this.index.recuperar(term);
        if (postings != null) {
            int termFreq = postings.talla();
            if (termFreq > maxFreq) {
                maxFreq = termFreq;
                res = term;
            }
        } else {
            System.out.println("Atención: El Termino " + term.termino
                               + " no está en el índice. ");
        }
    }
    System.out.println("El Termino más frecuente es " + res.termino
                       + ", que aparece " + maxFreq + " veces.");
    return res;
}
```

EJERCICIO 2. Método *comienzanPor*

Se pide añadir un nuevo método público a la clase **BuscadorDeLaBibl** con el siguiente perfil:

public ListaConPI<Termino> comienzanPor(String prefijo, String libro)

El método recibe dos **String *prefijo*** y ***libro***, y devuelve una **ListaConPI** de términos de la BD que comiencen por la **String *prefijo*** dentro del libro cuyo título se indica en la **String *libro***.

A tener en cuenta:

- Si el parámetro **prefijo** no es un término válido conforme a los requisitos de la BD, el método devolverá un valor **null**. Puedes utilizar el método **esTermino(String)** de la clase **BuscadorDeLaBibl** para comprobar si una cadena es un término válido.
- Puedes usar el método **boolean startsWith(String str)** de Java **String** que devuelve **true** si la cadena **this** comienza por la cadena **str**, y **false** en caso contrario.

Se recuerda que, en la clase **BuscadorDeLaBibl,** la información de la biblioteca digital (BD) se almacena en un **Map** llamado **index** cuyas claves son objetos **Termino** y cuyos valores son objetos **ListaConPI<BuscadorDeLaBibl.Posting>**.

Test. Añade la clase **TestComienzanPor** (que está en el fichero adjunto) al paquete **biblioteca** de tu proyecto BlueJ. Ejecuta el método **main** en la clase **TestComienzanPor**. El resultado debe ser el siguiente:

relacionalmente (646029615)
relaciona (-23349676)
relacionada (-964198959)
relacionado (-964198945)
relacionadas (174603458)
relación (-554938455)
relacionales (174611270)
relacional (-723839848)
relacionan (-723839846)
relacionar (-723839842)
relaciones (-723839717)
relacionarse (174617456)
objectinputstream (-2107549909)
objetivo (90992444)
object (-1023368385)
objeto (-1023367863)
objetos (-1659632566)
obj (109815)
objectoutputstream (-1858559872)
objetoutputstream (-418540889)

Estas líneas se mostrarán después de las 12 líneas que informan de la indexación de los 10 libros a partir de los cuales se crea la BD. Estas líneas se mostrarán porque, en el **main**, se invoca dos veces el método **comienzanPor**, la primera vez buscando términos que empiezan por **"relac"** en el libro **"Bases-de-Datos"**, y la segunda vez buscando términos que empiezan por **"obj"** en el libro **"AprendiendoJava-y-POO"**.

Solución

```
public ListaConPI<Termino> comienzanPor(String prefijo, String libro) {
    if (!esTermino(prefijo)) return null;
    ListaConPI<Termino> res = new LEGListaConPI<Termino>();
    ListaConPI<Termino> c = index.claves();
    ListaConPI<BuscadorDeLaBibl.Posting> p;
    for (c.inicio(); !c.esFin(); c.siguiente()) {
        Termino clave = c.recuperar();
        if (clave.termino.startsWith(prefijo)) {
            p = index.recuperar(clave);
            for (p.inicio(); !p.esFin(); p.siguiente()) {
                BuscadorDeLaBibl.Posting posting = p.recuperar();
                if (posting.tituloLibro.equals(libro)) {
                    res.insertar(clave);
                    break;
                }
            }
        }
    }
    return res;
}
```

EJERCICIO 3. Método *enTitulo*

Se pide añadir un nuevo método público a la clase **BuscadorDeLaBibl** con el siguiente perfil:

public ListaConPI<Termino> enTitulo()

Dado el contenido de la Biblioteca Digital (la BD), el método devuelve una lista con los términos que aparecen en la primera línea de los libros a partir de los cuales se ha generado la BD.

El método debe ser eficiente, y para ello, la operación a realizar sobre una lista de **Posting** debe ser una búsqueda, y no un recorrido.

Se recuerda que, en la clase **BuscadorDeLaBibl,** la información de la biblioteca digital (BD) se almacena en un **Map** llamado **index** cuyas claves son objetos **Termino** y cuyos valores son objetos **ListaConPI<BuscadorDeLaBibl.Posting>**.

Test. Añade la clase **TestEnTitulo** (que está en el fichero adjunto) al paquete **biblioteca** de tu proyecto BlueJ. Ejecuta el método **main** en la clase **TestEnTitulo**. El resultado debe ser el siguiente:

a (97)
ataques (-692411040)
revisión (-260782121)
bb (3136)
de (3201)
abierto (-1200707504)
aprendizaje (-1196686925)
absoluto (1728122241)
java (3254818)
acceso (-1423461024)
mixto (103910871)
traficantes (853805944)
comité (-1354818563)
cero (3050015)
aprendiendo (-1197300219)
sueños (-891318926)
bases (93508546)
autores (-646297455)
datos (95356859)

Estas líneas se mostrarán después de las 12 líneas que informan de la indexación de los 10 libros a partir de los cuales se crea la BD. Se debe obtener este resultado porque esos son los términos que aparecen en la primera línea de los libros indexados en nuestra BD. (Es decir, el método **main** contiene una llamada al método **enTitulo** y, a continuación, lista los términos devueltos por **enTitulo**).

Solución

```
public ListaConPI<Termino> enTitulo() {
    ListaConPI<Termino> res = new LEGListaConPI<Termino>();
    ListaConPI<Termino> c = index.claves();
    ListaConPI<BuscadorDeLaBibl.Posting> valor;
    for (c.inicio(); !c.esFin(); c.siguiente()) {
        Termino clave = c.recuperar();
        valor = index.recuperar(clave);
        for (valor.inicio(); !valor.esFin(); valor.siguiente()) {
            Posting p = valor.recuperar();
            if (valor.recuperar().lineaLibro == 1) {
                res.insertar(clave);
                break;
            }
        }
    }
    return res;
}
```

EJERCICIO 4. Método *listaValores*

Implementa en la clase **TablaHash** un método llamado **listaValores**, con un argumento entero **n**, que devuelva una **ListaConPI<V>** tal que contenga los valores de la tabla hash cuya frecuencia de aparición sea mayor o igual a **n**.

El método tiene como precondición: **n > 1**

El método debe tener la siguiente cabecera:

public ListaConPI<V> listaValores(int n)

IMPORTANTE: La lista resultado no debe contener ningún valor repetido.

PISTA: Se puede usar un **Map<V, Integer>** como estructura auxiliar.

Ejemplo: dada una **TablaHash t** con claves y valores **String**, y que contiene las siguientes entradas (clave -> valor):

"hola" -> "hello"
"mundo" -> "world"
"golpe" -> "hit"
"éxito" -> "hit"
"nota" -> "note"
"billete" -> "note"
"apunte" -> "note"
"ola" -> "wave"
"saludar" -> "wave"

La llamada **t.listaValores(2)** devolverá la lista **["hit", "wave", "note"]** (cualquier orden de los elementos de las listas sería válido). La llamada **t.listaValores(3)** devolverá la lista **["note"]**.

Test. Añade la clase **TestListaValores** (que está en el fichero adjunto) al paquete **deDispersion** de tu proyecto BlueJ. Ejecuta el método **main** en la clase **TestListaValores**. El resultado debe ser el siguiente:

[note wave hit]
[note]

Dado que en el método **main** se crea la **TablaHash** del ejemplo anterior y se hacen las 2 llamadas a **listaValores** del mismo ejemplo, mostrando en la salida los elementos de las listas.

Solución

```
public ListaConPI<V> listaValores(int n) {
    ListaConPI<V> res = new LEGListaConPI<>();
    Map<V, Integer> m = new TablaHash<>(talla);
    for (int i = 0; i < elArray.length; i++) {
        ListaConPI<EntradaHash<C, V>> l = elArray[i];
        for (l.inicio(); !l.esFin(); l.siguiente()) {
            V v = l.recuperar().valor;
            Integer j = m.recuperar(v);
            if (j == null) m.insertar(v, 1);
            else {
               m.insertar(v, j + 1);
               if (j + 1 == n) res.insertar(v);
            }
        }
    }
    return res;
}
```