Ejercicios de clase
TEMA 3 – MAPS y Tablas de Dispersión (Hash)

Ejercicio 1

Implementa el siguiente método:

```
public static String traducir(String textoCastellano, Map<String,String> map);
```

, teniendo en cuenta que:

- La clave en el diccionario es la palabra en castellano y el valor es su traducción al inglés.
- El método traducir devuelve una cadena con la traducción al inglés, palabra a palabra, de la cadena *textoCastellano*.

Si una palabra no se encuentra en el diccionario, deberá sustituirla por "<error>" en la cadena resultante.

SOLUCIÓN:

```
public static String traducir(String textoCastellano, Map<String,String> map) {
  String palabras[] = textoCastellano.split(" +");
  String res = "";
  for (int i = 0; i < palabras.length; i++) {
    String ingles = map.recuperar(palabras[i]);
    if (ingles != null) res += ingles;
    else res += "<error>";
    if (i < palabras.length - 1) res += " ";
  }
  return res;
}
```

Ejercicio 2

Usando la interfaz *Map*, y disponiendo de la clase *TablaHash* que la implementa, diseña un programa que lea un texto por teclado y devuelva el número de palabras distintas que hay en dicho texto junto con la frecuencia de aparición de cada una de ellas.

Nota: la frecuencia se calcula como el número de veces que aparece la palabra en el texto dividido entre el número total de palabras del texto

SOLUCIÓN:

```
public class ContarApariciones {

  public static void main(String[] args) {
    // Lectura del texto
    Scanner teclado = new Scanner(System.in);
    System.out.println("Escriba un texto:");
    String texto = teclado.nextLine();
    String[] palabras = texto.split(" +");
    // Cuenta del número de palabras distintas
    Map<String,Integer> dic = new TablaHash<String,Integer>(palabras.length);
    for (int i = 0; i < palabras.length; i++) {
      Integer numApariciones = dic.recuperar(palabras[i]);
      if (numApariciones == null)
          numApariciones = 0;
      dic.insertar(palabras[i], numApariciones + 1);
    }
    // Mostrar el resultado
    System.out.println("Hay " + dic.talla() + " palabras distintas");
    ListaConPI<String> pDistintas = dic.claves();
    pDistintas.inicio();
    while (!pDistintas.esFin()) {
      String palabra = pDistintas.recuperar();
      int numApariciones = dic.recuperar(palabra).intValue();
      double frec = numApariciones / (double) palabras.length;
      System.out.println("La frecuencia de la palabra ‘" +
                         palabra + "’ es " + frec);
      pDistintas.siguiente();
    }
  }

}
```

Ejercicio 3

La Policía quiere utilizar una *TablaHash* para poder consultar eficientemente el dueño de un coche determinado.

- Los coches se identifican por su matrícula, compuesta por un número (entero) y una secuencia de letras (String). Es importante mantener estos dos atributos por separado para acelerar otras posibles operaciones.
- Del dueño del coche sólo nos interesa saber su nombre.

Indica de qué tipo sería la *TablaHash* y diseña las clases que sean necesarias.

SOLUCIÓN:

La *TablaHash* sería de tipo <Matricula, String>, donde la clase Matricula se definiría como:

```
public class Matricula {
  private String letras;
  private int numero;

  // Constructor
  public Matricula(String letras, int numero) {
    this.letras = letras;
    this.numero = numero;
  }

  // Método requerido: comparación de dos objetos Matricula
  public boolean equals(Object x) {
    if (x instanceof Matricula) {
      Matricula m = (Matricula) x;
      return m.numero == numero && m.letras.equals(letras);
    } else return false;
  }

  // Método requerido: hashCode
  public int hashCode() {
    return (numero + letras).hashCode();
  }

  // Otros métodos recomendados
  public String getLetras() { return letras; }
  public int getNumero()    { return numero; }
  public String toString()  { return numero + letras; }
}
```

Ejercicio 4

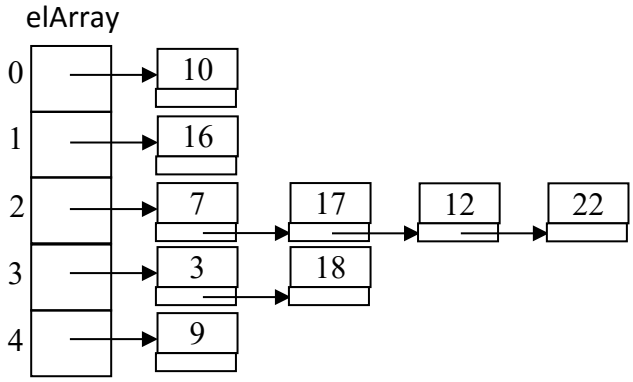
Inserta los enteros 9, 7, 3, 17, 18, 16, 12, 10, 22 en una tabla hash enlazada de tamaño 5 y con la función de dispersión H(x) = x.

a) Sin hacer rehashing, dibuja la tabla resultante y calcula su FC.

b) ¿Y si hiciéramos rehashing cuando el FC > 1.5, aumentando la capacidad de la tabla a 11?

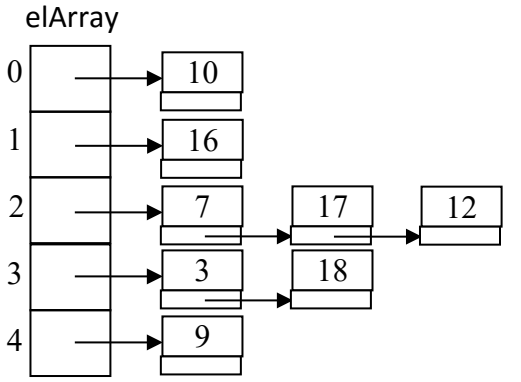
SOLUCIÓN:

a)

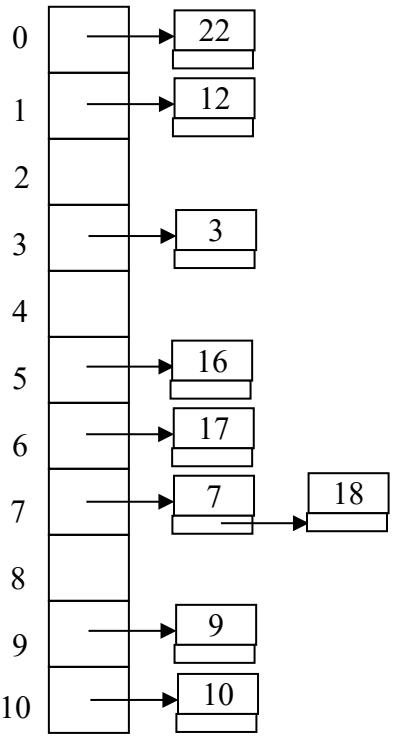


FC = 9/5 = 1.8

b)



FC = 8/5 = 1.6 > 1.5 → **rehashing**



FC = 9/11 = 0.82

Ejercicio 5

Se quiere saber las entradas de un *Map*, implementado mediante una *TablaHash*, cuyo valor es igual a uno dado.

Se pide: implementar en la clase *TablaHash* un nuevo método que devuelva una *ListaConPI* con las claves de las entradas cuyo valor sea igual a uno dado.

SOLUCIÓN:

```
public ListaConPI<C> clavesConValor(V valor) {
  ListaConPI<C> res = new LEGListaConPI<C>();
  for (int i = 0; i < elArray.length; i++) {
       ListaConPI<EntradaHash<C, V>> lista = elArray[i];
       for (lista.inicio(); !lista.esFin(); lista.siguiente()) {
            EntradaHash<C, V> e = lista.recuperar();
            if (e.valor.equals(valor)) res.insertar(e.clave);
       }
  }
  return res;
}
```

Ejercicio 6

Diseña un método en la clase *TablaHash* que reciba otro *Map* como parámetro y devuelva uno nuevo que sea la intersección de ambas, es decir, que contenga los elementos (misma clave y valor) que están en ambos *Maps*.

SOLUCIÓN:

```
public Map<C,V> interseccion(Map<C,V> otro) {
  Map<C,V> res = new TablaHash<C,V>(Math.min(talla, otro.talla()));
  for (int i = 0; i < elArray.length; i++) {
    ListaConPI<EntradaHash<C,V>> lista = elArray[i];
    while (lista.inicio(); !lista.esFin(); lista.siguiente()) {
      EntradaHash<C, V> e = lista.recuperar();
      V valor = otro.recuperar(e.clave);
      if (valor != null && valor.equals(e.valor))
          res.insertar(e.clave, e.valor);
    }
  }
  return res;
}
```