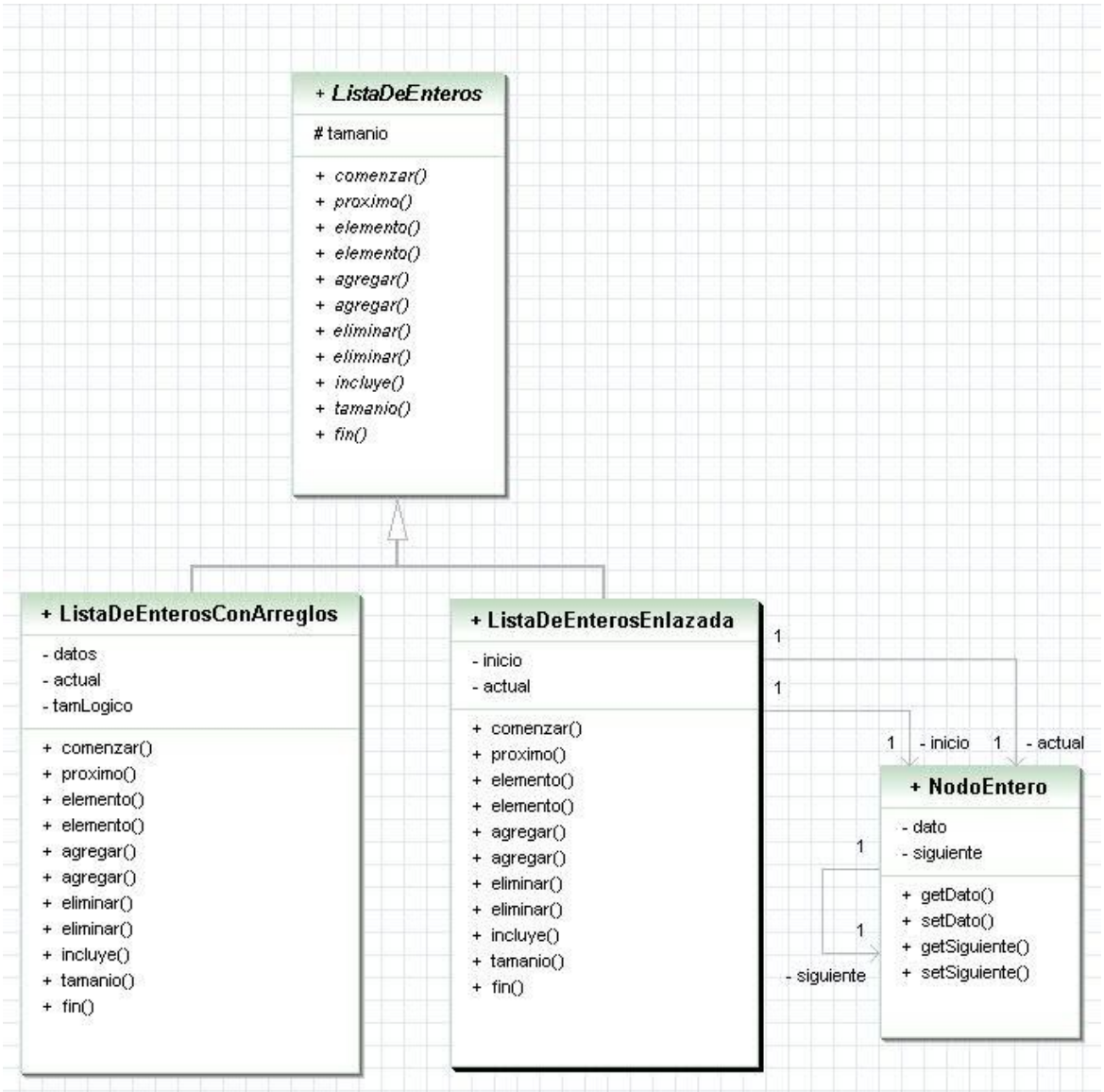
UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2011

Encapsulamiento y Abstracción

Ejercicio 1

Considere la siguiente especificación de operaciones de una lista de enteros:




```
comenzar ()  // Se prepara para iterar los elementos de la lista.
proximo ()  // Avanza al próximo elemento de la lista.
fin()  // Determina si llegó o no al final de la lista.
elemento ()  // Retorna el elemento actual.
```

```
elemento (Integer pos) // Retorna el elemento de la posición indicada (la posición va
desde 0 hasta n-1).
agregar (Integer elem) // Agrega el elemento en la posición actual y retorna true si
pudo agregar y false si no pudo agregar.
agregar (Integer elem, Integer pos) // Agrega el elemento en la posición indicada y
retorna true si pudo agregar y false; si no pudo agregar.
eliminar ()  // Elimina el elemento actual y retorna true si pudo eliminar y false en
caso contrario.
eliminar (Integer pos) // Elimina el elemento de la posición indicada y retorna true si
pudo eliminar y false en caso contrario.
esVacia()// Retorna true si la lista está vacía, false en caso contrario.
incluye (Integer elem)  // Retorna true si elem está contenido en la lista, false en
caso contrario.
tamanio ()  // Retorna la longitud de la lista.
```

a) Implemente en JAVA (pase por máquina) una clase abstracta llamada **ListaDeEnteros** de acuerdo a la especificación dada ubicada dentro del paquete **ejercicio1**.

b) Escriba una clase llamada **ListaDeEnterosConArreglos** como subclase de **ListaDeEnteros** dentro del paquete anterior, de manera que implemente todos los métodos definidos en la superclase haciendo uso de arreglos de longitud fija.

c) Escriba una clase llamada **TestListaDeEnterosConArreglos** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosConArreglos** y luego imprima los elementos de dicha lista.

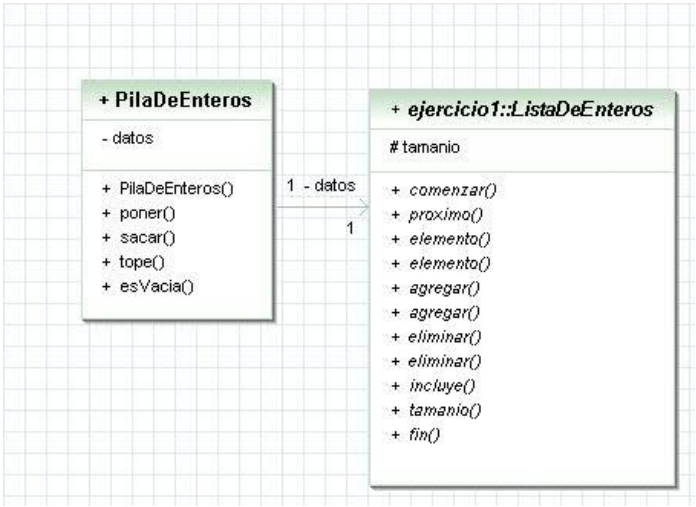
d) Escriba una clase llamada **ListaDeEnterosEnlazada** como otra subclase de **ListaDeEnteros** dentro del paquete ejercicio1, de manera que implemente todos los métodos definidos en la superclase pero haciendo uso de una estructura recursiva. En este caso la lista tendrá una referencia al elemento inicial y al actual. Luego, se definirá otra clase recursiva que contenga el elemento y que referencie a una instancia similar a sí misma.

e) Escriba una clase llamada **TestListaDeEnterosEnlazada** que reciba en su método **main** una secuencia de números, los agregue a un objeto de tipo **ListaDeEnterosEnlazada** y luego imprima los elementos de dicha lista.

f) ¿Es posible implementar el método incluye en la clase **ListaDeEnteros** de manera que solo utilice otras operaciones de lista y no dependa de la estructura interna elegida en las subclases? Si fuera posible, ¿funcionaria el envío de este mensaje a un objeto de tipo **ListaDeEnterosConArreglos**?

Ejercicio 2

Sea la siguiente especificación de una pila de enteros:



```
poner (Integer elem)  // Agrega elem a la pila.
sacar()  //  Elimina y devuelve el elemento en el tope de la pila.
tope()  //  Devuelve el elemento en el tope de la pila sin eliminarlo.
esVacia() // Retorna true si la pila está vacía, false en caso contrario.
```

a. Implemente en JAVA (pase por máquina) la clase **PilaDeEnteros** (utilizando alguna subclase de **ListaDeEnteros**) de acuerdo a la especificación dada ubicada dentro del paquete **ejercicio2**.

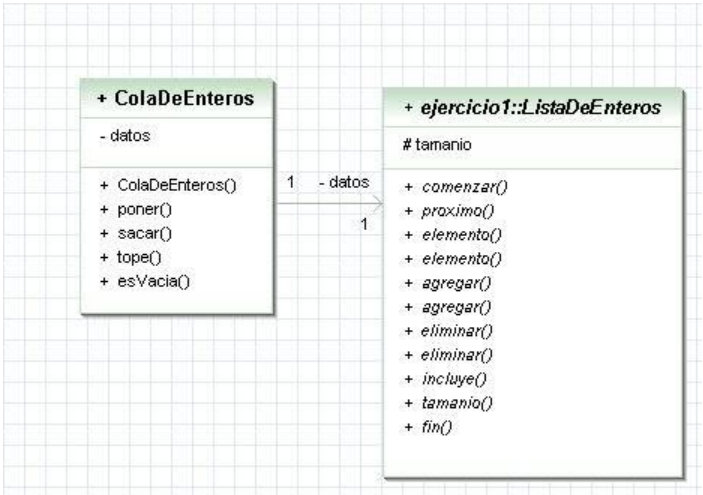
b. Escriba una clase llamada **TestPilaDeEnteros** para ejecutar el siguiente código:

```
PilaDeEnteros p1, p2;
Integer valor2;
p1=new PilaDeEnteros();
p1.poner(1);
p1.poner(2);
p2=p1;
valor2 = p2.sacar();
System.out.println("El valor del tope de la pila p1 es:  " + p1.sacar());
```

c. ¿Qué valor imprime? ¿Qué conclusión saca?

Ejercicio 3

Sea la siguiente especificación de una cola de enteros:




```
poner(Integer elem) // Agrega elem a la cola.
sacar // Elimina y devuelve el  primer elemento de la cola.
tope()  //  Devuelve el elemento en el tope de la cola sin eliminarlo.
esVacia() // Retorna true si la Cola está vacía, false en caso contrario.
```

a. Implemente en JAVA (pase por máquina) la clase **ColaDeEnteros** (utilizando alguna subclase de **ListaDeEnteros**) de acuerdo a la especificación dada. Defina esta clase adentro del paquete **ejercicio3**.

UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2011

Ejercicio 4

El algoritmo conocido como Criba de Eratóstenes permite la obtención de todos los números primos menores que un número dado. Y para conseguir esto se procede del modo siguiente:

Haga una lista de todos los naturales desde 1 hasta un número n dado:

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 … n

Marque el 2 como primer primo y tache de ahí en adelante todos sus múltiplos:

1 [2] 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 … n

De los no tachados que siguen a 2, el primero es el siguiente primo. Marque el 3 y tache todos los múltiplos de éste que no haya tachado en el paso anterior:

1 [2] [3] ~~4~~ 5 ~~6~~ 7 ~~8 9 10~~ 11 ~~12~~ 13 ~~14 15 16~~ 17 ~~18~~ 19 ~~20 21~~ … n

Continúe de esta manera hasta haber marcado un número mayor que la raíz cuadrada de n y entonces marque todos los números mayores que 1 que hasta ese momento hayan permanecido sin ser tachados:

1 [2] [3] ~~4~~ [5] ~~6~~ [7] ~~8 9 10~~ [11] ~~12~~ [13] ~~14 15 16~~ [17] ~~18~~ [19] ~~20 21~~ … n

Los números marcados son todos los primos entre 1 y n.

a. Escriba una clase llamada **ejercicio4.CribaDeEratostenes** como un método llamado **obtenerPrimos()** que tome como parámetro una instancia de **ListaDeEnteros** que contenga los primeros 1000 números naturales y retorne la lista de los primos correspondientes siguiendo el procedimiento antes descrito.

Ejercicio 5

Defina una clase llamada **ejercicio5.ParGenerico** con 2 variables de instancia de tipo genérico. Dicha clase representan una dupla de dos elementos del mismo tipo.

a. Defina un constructor con 2 argumentos que permita crear un par con 2 valores.

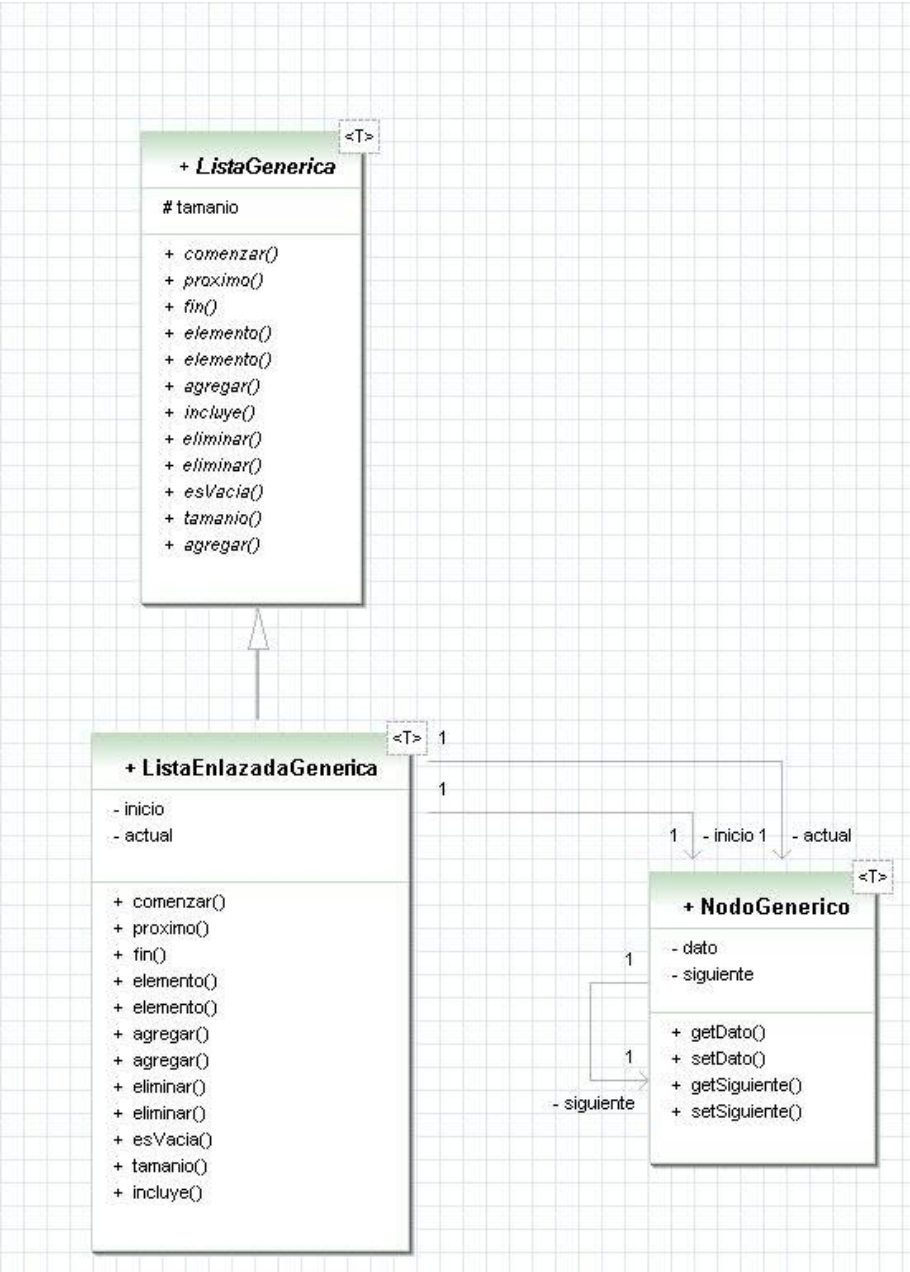
b. Sobrescriba el método **public boolean equals(Object)** de la clase **Object** en **ParGenerico** de manera que la implementación sirva para comparar dos instancias de este tipo.

c. Sobrescriba el **public String toString()** de la clase **Object** en **ParGenerico** de manera que imprima su contenido con el siguiente formato: par[val1,val2].

UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2011

Ejercicio 6

Sea la siguiente especificación de lista genérica:



a. Implemente en JAVA (pase por máquina) una clase abstracta llamada **ListaGenerica** de acuerdo a la especificación dada ubicada dentro del paquete **ejercicio6**.

b. Escriba una clase llamada **ListaEnlazadaGenerica** como subclase de **ListaGenerica** dentro del paquete anterior, de manera que implemente todos los métodos definidos en la superclase pero haciendo uso de una estructura recursiva.

c. Escriba una clase llamada **TestListaEnlazadaGenerica** que reciba en su método **main** una secuencia de strings, los agregue a un objeto de tipo **ListaEnlazadaGenerica** y luego imprima los elementos de dicha lista.

UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2011

Ejercicio 7

Considere un string de caracteres S, el cual comprende únicamente los caracteres: (,),[,],{,}. Decimos que S está balanceado si tiene alguna de las siguientes formas:

```
S = ‘’   S es el string de longitud cero.
S = ‘(T)’
S = ‘[T]’
S = ‘{T}’
S = ‘TU’
```

Donde ambos T y U son strings balanceados. Por ejemplo, ‘{( ) [( )] }’ está balanceado, pero ‘ ([ )]’ no lo está.

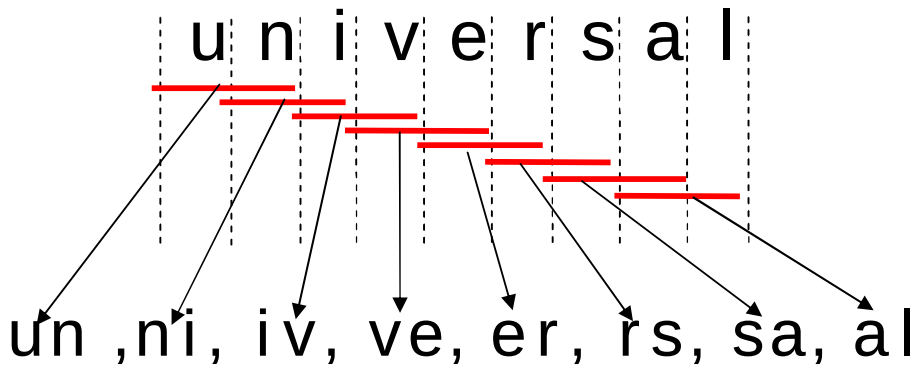
a. Implemente una clase llamada **ejercicio7.TestBalanceo** (pase por máquina), cuyo objetivo es determinar si un string dado está balanceado. El string a verificar es un parámetro de entrada (no es un dato predefinido).

Nota: En caso de ser necesario implemente las estructuras de datos que necesite para resolver el ejercicio.

Ejercicio 8

Las listas invertidas son una herramienta básica utilizada para la búsqueda por contenido en texto. En este caso particular, una lista invertida almacenará el resultado de la factorización de palabras en n-gramas (un n-grama es una subsecuencia de n caracteres de una palabra) guardando las referencias a las palabras donde se encuentran contenidos.

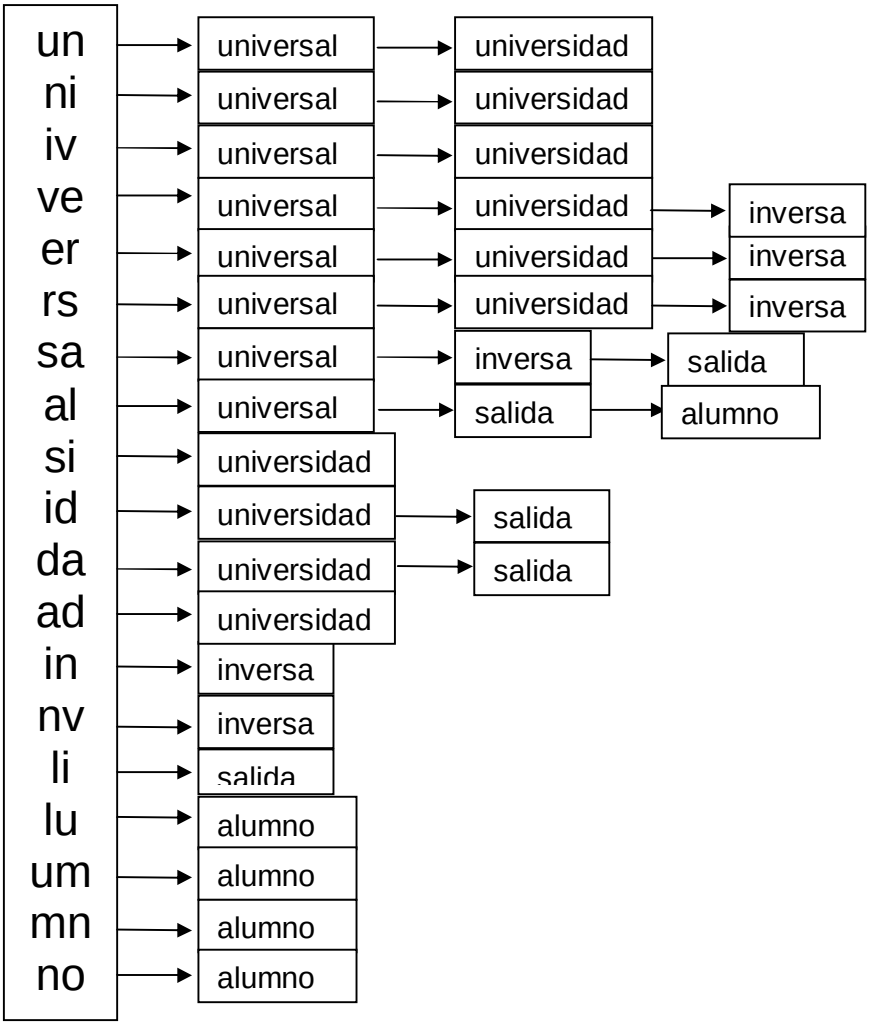
Por Ejemplo, si queremos factorizar la palabra universal en n-gramas de longitud 2 (2-gramas), el resultado será:



Ahora si tenemos la siguiente lista de palabras:

universal, universidad, inversa, salida y alumno

Y deseamos armar una lista invertida con n-gramas de longitud 2, el resultado será:



a. Implemente en JAVA una clase llamada **ListaInvertidaDeGramas** ubicada dentro del paquete **ejercicio8**, cumpliendo la siguiente especificación:

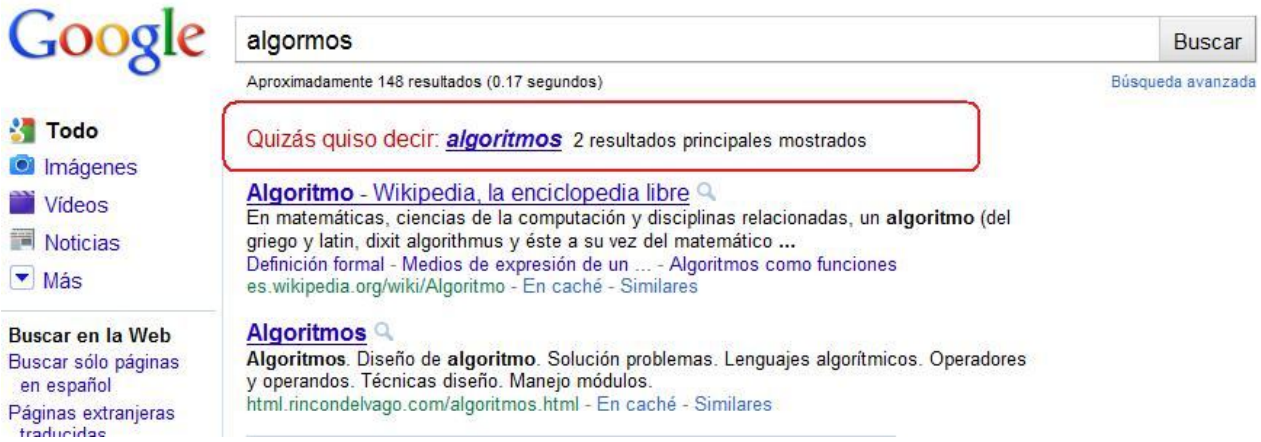
```
ListaInvertidaDeGramas (int longGrama)  // instancia una lista invertida, donde
la longitud de los gramas que almacenará esta dada por el parámetro.

agregar (String palabra) // Agrega los gramas del string enviado como parámetro a
la lista y asocia el parámetro a cada uno de los gramas agregados.

recuperarListaDePalabras(String palabra) // Retorna una lista de todas las
palabras en la lista invertida que poseen al menos uno de los gramas del
parámetro.
```

b. Escriba una clase llamada **TestListaInvertidaDeGramas** que reciba en su método **main** una secuencia de strings, los agregue a un objeto de tipo **ListaInvertidaDeGramas** y luego vuelva a recorrer la secuencia de strings invocando el método **recuperarListaDePalabras** con cada uno de ellos e imprima la lista resultado.

UNLP
Facultad de Informática
Algoritmos y Estructuras de Datos
Curso 2011

Ejercicio 9

Se desea implementar una herramienta del estilo “Quizás quiso decir” de Google:



a. Implemente en JAVA una clase llamada **ejercicio9.Sugeridor** que contenga un método llamado **quisoDecir()** que tome como parámetro una **ListaInvertidaDeGramas** y un string. El resultado de ejecutar el método **quisoDecir()** será una lista con las palabras sugeridas. Dichas palabras sugeridas se encuentran almacenadas en lista invertida enviada como parámetro y cumplen con la condición que comparadas con el parámetro string devuelven un numero menor o igual a 2.

   Nota 1: Para realizar las comparaciones entre las palabras utilice la clase provista por la cátedra llamada **EditDistance** e invoque el método **compararStrings()**, el cual retorna un entero que simboliza el nivel de igualdad entre las palabras comparadas. Si la distancia es 0 -cero-, la palabra buscada es exactamente la encontrada y mientras más grande sea este número más distintas son las palabras.

   Nota 2: Las palabras de la lista invertida con las cuales se debe aplicar la comparación son aquellas que comparten gramas con el string enviado como parámetro.

b. Escriba una clase llamada **TestSugeridor** que en su método **main** reciba una secuencia de strings, los agregue a un objeto de tipo **ListaInvertidaDeGramas** a excepción del último e invoque el método **quisoDecir()** de un objeto de tipo **Sugeridor** enviándole como parámetro el objeto de tipo **ListaInvertidaDeGramas** anterior y el último string de la secuencia enviada al **main**. Además se debe imprimir la siguiente leyenda:

   ENTRADA

   Es como ejecutar el intérprete así: **java TestSugeridor vaca vacaciones playa maya**

   Palabras para armar la lista invertida

   Palabra a buscar

   SALIDA

   ```
   Tal vez quiso decir: str1, str2,…., strn
   ```

   Donde str1, str2,…., strn son los strings de la lista resultante de la ejecución del método **quisoDecir()** .

   En el ejemplo anterior, la salida sería: Tal vez quiso decir: playa