



Guía de Estudio

Control 3

Profesor: Andrés Muñoz

Profesores Auxiliares: Agustín Almonte

Marcelo Muñoz

Patricio Salles



CC10A - 2004

Tabla de Contenidos

TABLA DE CONTENIDOS	2
1. MARATÓN (CONTROL 3 - 1998)	4
2. ARCHIVOS Y ESTRUCTURAS DE DATOS (PROPUESTO)	8
3. CONVERTIR UN ABB A LISTA ENLAZADA	10
4. COLA - PILA	12
5. OTRA IMPLEMENTACIÓN DE UNA PILA	14
6. EL ASCENSOR	17
7. EL SUPERMERCADO (PROPUESTO)	20
8. ARREGLO QUE SIMULA UNA LISTA ENLAZADA	22
9. MERGESORT EN LISTAS ENLAZADAS	26
10. LA RULETA	28
11. LISTA DE ÍNDICES DE UN ARREGLO	30
12. AJUSTE DE TEXTO	32
13. EL TRADUCTOR	37
14. DICCIONARIO	42
15. LIBROS INTERACTIVOS	45
16. PRINTTREE	48
17. CODIFICADOR MORSE (PROPUESTO)	49
18. DIVISIBLES Y NO DIVISIBLES	50
19. TRANSFORMADA DE BURROWS-WHEELER.	52
20. EMULADOR HP.	54
21. IN THE NAVY.(PROPUESTO)	56
22. AGENDA. (PROPUESTO)	57
23. CLASE PARRAFO.(PROPUESTO)	59
24. INSERCIÓN ORDENADA.	61
25. CLASE LISTA.(PROPUESTO)	63
26. CASINO (PROPUESTO)	64
27. LA VUELTA A FRANCIA (PROPUESTO)	65
28. ENVIANDO MENSAJES (PROPUESTO)	66
29. NOTAS	67

30. LISTA DE FRECUENCIAS	70
31. ORDENAMIENTO RESTRINGIDO (PROPUESTO)	73
32. OLIMPIADAS	74
33. VECTORES LIGEROS.	77

1. Maratón (Control 3 - 1998)

Escribir un programa que se utilice en la meta de una maratón, de modo de registrar el orden de llegada de los atletas. Los corredores se identifican con números consecutivos a partir del 2001. El programa debe establecer un diálogo como el que se muestra a continuación:

```
Ingrese los números que identifican a los atletas
Lugar 1 : 2027
Lugar 2 : 2042
Lugar 3 : 2001
...
Lugar 84: 0
Fin de la Maratón
```

Una vez ingresado el último corredor, el programa debe emitir las siguientes listas de resultados:

Resultados por Identificador

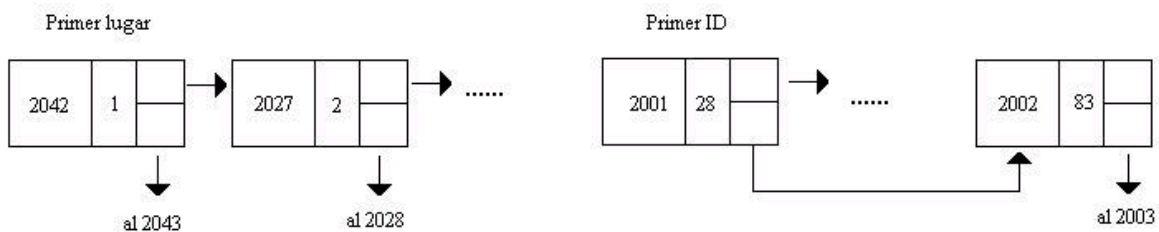
Nº	Lugar
2001	28
2002	83
.....	
2042	1

Resultados por Lugar

Lugar	Nº
1	2042
2	2027
...	
28	2001
...	
83	2002

Importante: El programa debe resolverse utilizando una lista con dos enlaces:

- uno con la referencia del atleta que llega en el lugar inmediatamente siguiente
- uno con la referencia del atleta con el N° de identificador siguiente
- debe declarar la clase (y definirla completamente) que implementa el objeto y usarlo en un programa que vaya construyendo la lista a medida que van llegando los corredores a la meta.



Solución

```
import java.io.*;

class Nodo{

    //referencia a usar, una al lugar siguiente y otra al id siguiente
    private Nodo lug_sig;
    private Nodo id_sig;

    //con estas guardamos el id y el lugar de llegada
    private int numero;
    private int lugar;

    //constructor, recibe id y lugar de llegada como parámetros
    public Nodo(int n, int l){
        numero = n;
        lugar = l;
        lug_sig = null;
        id_sig = null;
    }

    //métodos para obtener las variables de instancia
    public Nodo lugarSiguiente(){
        return lug_sig;
    }

    public Nodo idSiguiente(){
        return id_sig;
    }

    public int numeroCorredor(){
        return numero;
    }

    public int lugarCorredor(){
        return lugar;
    }

    //métodos para realizar los enlaces. Distinguimos si enlazamos por
    //id's o si es por lugar
    public void enlazarLugar(Nodo n){
        lug_sig = n;
    }

    public void enlazarId(Nodo n){
        id_sig = n;
    }
}
```

```
class maraton{

    //recibe un nodo y la cabeza de la lista, y devuelve el nodo con el
    //id inmediatamente siguiente

    private static Nodo buscaIdSiguiente(Nodo n, Nodo cabeza){

        //casos especiales: lista vacía o lista de un elemento

        if(cabeza == null)return null;
        if(cabeza.numeroCorredor()==n.numeroCorredor()+1)
            return cabeza;

        //caso normal

        Nodo aux = cabeza;
        while(aux.lugarSiguiente() != null){

            //si encontramos el siguiente, retornamos de inmediato
            if(aux.lugarSiguiente().numeroCorredor()==n.numeroCorredor()+1)
                return aux.lugarSiguiente();
            else
                aux = aux.lugarSiguiente();
        }
        return null;
    }

    //recibe la cabeza de la lista y devuelve el nodo que tiene menor ID
    //(PrimeroID en la figura)

    private static Nodo buscarMenor(Nodo cabeza){
        Nodo candidato = cabeza;
        Nodo aux;
        for(aux = cabeza; aux!=null ; aux = aux.lugarSiguiente()){
            if(aux.numeroCorredor()<candidato.numeroCorredor())
                candidato = aux;
        }
        return candidato;
    }

    //usando buscaMenor, recorremos la lista y vamos enlazando los
    //elementos con su sucesor

    private static void actualizar(Nodo cabeza){
        Nodo aux;
        for(aux = cabeza ; aux != null ; aux = aux.lugarSiguiente())
            aux.enlazarId(buscaIdSiguiente(aux,cabeza));
    }

    //el main, en un ciclo lo corredores van llegando. Luego de que
    //llega el último, enlazamos por ID (ya que a mediada que van
    //llegando enlazamos por lugar) y así obtenemos las dos listas
    //ordenadas en una sola

    public static void main(String[] args)throws IOException{

        Nodo primero = null;
        Nodo recorrer = primero;
        BufferedReader b = new BufferedReader(new InputStreamReader(
                                                                    System.in));
```

```
String s = null;
int n = 1;

while(true){

    System.out.print("Lugar "+n+" ?");

    //si es el ultimo, salimos del ciclo
    if((s=b.readLine()).equals("0")){
        System.out.println("Fin de la Maraton");
        break;
    }

    //si es el primero en llegar, creamos la lista
    if(recorrer == null){
        recorrer = new Nodo(Integer.parseInt(s),n++);
        primero = recorrer;
    }

    //el resto de los casos
    else{
        recorrer.enlazarLugar(
            new Nodo(Integer.parseInt(s),n++));
        recorrer = recorrer.lugarSiguiente();
    }
} //fin del while

actualizar(primero);
System.out.println("Resultados por Identificador");
System.out.println("No\tLugar");

for(recorrer = buscarMenor(primero); recorrer != null ; recorrer =
    recorrer.idSiguiente())
    System.out.println(recorrer.numeroCorredor()+"\t"+
        recorrer.lugarCorredor());

System.out.println("\nResultados por Lugar");
System.out.println("Lugar\tNo");

for(recorrer = primero; recorrer != null; recorrer =
    recorrer.lugarSiguiente())
    System.out.println(recorrer.lugarCorredor()+"\t"+
        recorrer.numeroCorredor());
}
```

2. Archivos y Estructuras de Datos (Propuesto)

El registro civil posee una base de datos compuesta por 2 archivos: un archivo de personas que almacena los datos de cada habitante de Chile, un archivo de matrimonios que indica las parejas que han contraído matrimonio y un archivo de hijos que señala quien es hijo de quien.

Para cada archivo Ud. Dispone de una clase para representar el contenido de cada línea del archivo. La clase posee los métodos usuales vistos en su sección para leer y escribir líneas en el archivo. La siguiente tabla es una descripción detallada de cada archivo:

Nombre de archivo:	Nombre de la clase que representa una línea del archivo:	Datos almacenados en la clase (variables de instancia o campos):	El archivo se encuentra ordenado ascendentemente por:
personas.dat	Persona	String cedula String nombre String fechaNac String sexo ("m", "f")	cedula
matrimonios.dat	Matrimonio	String cedulaEsposo String cedulaEsposa	cedulaEsposo
hijos.dat	Hijo	String cedulaHijo String cedula (del padre o la madre)	cedula

Por ejemplo, supongamos que una persona con cedula x tiene padre y madre vivos y es a su vez padre de otra persona. Entonces, su cedula aparecerá en tres líneas en el archivo de hijos:

cedulaHijo	cedula
x	Cedula del padre de x
x	Cedula de la madre de x
...	...
cedula del hjo de x	x

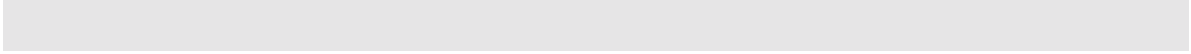
Como en Chile una persona puede estar casada con una sola persona, solo puede aparecer en una sola línea del archivo de matrimonios. Si la persona es soltera, no aparece en el archivo de matrimonios.

Escriba un programa que produzca un archivo de nombre padres-solteros.dat con todos los padres solteros (varones solteros con hijos). El archivo debe contener cedula, nombre, fecha de nacimiento y sexo (al igual que el archivo de personas).

Observaciones: El tamaño de los archivos hace imposible colocar su contenido en un arreglo en la memoria del computador. Su programa debe tomar tiempo razonable de tiempo (no mas de un día).

El formato de cada línea en los archivos no es relevante porque Ud. Debe suponer que las clases Persona, Matrimonio e Hijo ya están programadas y poseen los métodos y/o constructores adecuados para leer y escribir en el archivo-

Indicación: Utilice un archivo auxiliar para almacenar resultados intermedios.



3. Convertir un ABB a Lista Enlazada¹

Escribir un método que reciba un árbol de búsqueda binaria y que lo convierta en un lista enlazada (ordenada). La firma del método debe ser:

public NodoLista abb2Lista (NodoArbol raiz)

Las definiciones de los NodoLista y NodoArbol son:

```
class NodoLista {
    public int info;
    public NodoLista sgte;

    public NodoLista (int x) {
        this.info = x;
        this.sgte = null;
    }
}

class NodoArbol {
    public int info;
    public NodoArbol izq, der;

    public NodoArbol (int x) {
        this.info = x;
        this.izq = null;
        this.der = null;
    }
}
```

Nota: Debe hacerlo directamente trabajando con los nodos. Recuerde los patrones de programación utilizados tanto en árboles como en listas.

Solución

```
public NodoLista abb2Lista (NodoArbol raiz) {
    // Un árbol nulo es una lista
    if (raiz == null)
        return null; // de NodoLista

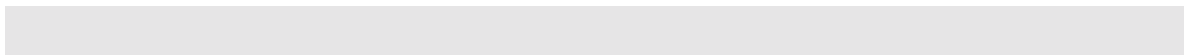
    // La lista se construye como la lista de la izquierda
    // más el nodo raiz y más la lista de la derecha del ABB
    NodoLista ant = abb2Lista (raiz.izq);
    NodoLista x = new NodoLista (raiz.info);
    NodoLista pos = abb2Lista (raiz.der);

    // Enlazamos la sublista siguiente con el nodo x
    x.sgte = pos;

    // Buscamos el último elemento de la sublista anterior
    NodoLista p = ant;
    while (ant != null && ant.sgte != null)
        ant = ant.sgte;

    // Enlazamos dependiendo donde va
    if (ant != null)
        ant.sgte = x;
    else
        p = x;
    return p;
}
```

¹ Solo es para que vean la solución del ejercicio en forma rápida y recursiva. Es más importante la parte de la Lista que hacer un recorrido a través del árbol.



4. Cola – Pila

Implementar la estructura de datos **Cola** utilizando la clase **Pila** vista en clases. Debe escribir:

```
public class Cola {
```

(a) Variables de Instancia de la clase **Cola** (**Hint**: Se pueden utilizar 2 pilas).

Solución

```
private Pila primaria;  
private Pila secundaria;
```

(b) Escribir un constructor que inicialice la **Cola** vacía.

Solución

```
public Cola () {  
    this.primaria = new Pila();  
    this.secundaria = new Pila();  
}
```

(c) Implementar los métodos **poner(x)** y **sacar()** utilizando la representación interna de la cola (ie, métodos de los objetos de las pilas declaradas **push(x)** y **pop()**).

Solución

```
public void poner (Object x) {  
    this.primaria.push(x);  
}  
  
public Object sacar () {  
    // Volcar pila primaria en secundaria  
    Object y;  
    while ((y = this.primaria.pop()) != null)  
        this.secundaria.push(y);  
  
    // Sacamos de la secundaria el primer elemento  
    Object x = this.secundaria.pop();  
  
    // Volvemos a volcar secundaria en primaria  
    while ((y = this.secundaria.pop()) != null)  
        this.primaria.push(y);  
  
    return x;  
}
```

(d) Implemente el método **boolean estaVacia()**.

Solución

```
public boolean estaVacia() {  
    Object x = this.primaria.pop();  
    If (x != null)  
        this.primaria.push(y);  
    return x == null;  
}
```

}	}
---	---

5. Otra Implementación de una PILA

Suponga a siguiente representación de **Nodo**:

```
class Nodo {  
    public double info;  
    public Nodo sgte;  
  
    public Nodo (double x) {  
        this.info = x;  
        this.sgte = null;  
    }  
}
```

(a) Escriba una clase abstracta **Lista** que modele una lista enlazada con los siguientes métodos:

- **Lista ()**: Constructor que crea una lista enlazada vacía.
- **void agregar (double x)**: [abstracta] Agrega el valor x a la lista.
- **double eliminar (int n)**: Elimina el elemento n-ésimo de la lista y retorna su valor.
- **int contar ()**: Retorna la cantidad de elementos que posee la lista.

Solución

```
public abstract class Lista {  
    private Nodo cabeza;  
  
    public Lista () {  
        this.cabeza = null;  
    }  
  
    public abstract void agregar (double x);  
  
    public double eliminar (int n) {  
        int cont = this.contar();  
        if (cont < n)  
            DataError e =  
                new DataError("No hay tantos elementos");2  
        Nodo q = this.cabeza;  
        for (int m=0; m<n; m++)  
            q = q.sgte;  
        return q.info;  
    }  
  
    public int contar () {  
        Nodo q = this.raiz;  
        int c = 0;  
        while (q != null) {  
            q = q.sgte;  
            c++;  
        }  
        return c;  
    }  
}
```

² Supondremos que esta clase permite generar un ERROR en la pantalla de la muerte con el texto del constructor. Así podremos controlar este caso.

- (b) Implemente una clase **ListaOrdenada** derivada de **Lista** que permita insertar los elementos en forma ordeanada.

Solución

```
public class ListaOrdenada extends Lista {
    public void agregar (double x) {
        Nodo q = super.raiz;
        Nodo p = null;
        while (q != null && q.info < x) {
            p = q;
            q = q.sgte;
        }
        Nodo t = new Nodo(x)
        if (p != null) {
            t.sgte = q;
            p.sgte = t;
        }
        else {
            t.sgte = this.cabeza;
            this.cabeza = t;
        }
    }
}
```

- (c) Programe una clase **ListaPila** derivada de **Lista** que permita insertar elementos al principio de la lista.

Solución

```
public class ListaOrdenada extends Lista {
    public void agregar (double x) {
        Nodo t = new Nodo(x)
        t.sgte = this.cabeza;
        this.cabeza = t;
    }
}
```

- (d) Enmascare la clase **ListaPila** creando una nueva clase que la utilice para implementar una **Pila** con sus métodos:

- **void push (double x):** Pone el elemento x en el tope de la pila.
- **double pop ():** Saca y elimina el elemento del tope de la pila.
- **Boolean estaVacia():** Retorna si está vacía la pila.

Solución

```
public class Pila {
    private Lista p;

    public Pila () {
        this.p = new ListaPila();
    }

    public void push(double x) {
```

```
        this.p.agregar(x);3
    }

    public double pop() {
        return this.p.eliminar(1);
    }

    public boolean estaVacia() {
        return this.p.contar() == 0;
    }
}
```

³ En este caso también está correcto poner ((ListaPila) this.p).agregar(x) ya que el método se define en la clase heredada, sin embargo, por estar definido abstracto en la superclase, no es necesario ponerlo (el enlace dinámico es implícito).

6. El Ascensor

Existe la clase **ArbolBinario** con los siguientes métodos:

- **ArbolBinario()**: Constructor que crea un ABB vacío.
- **void agregarValor(int x)**: Agrega x al ABB.
- **int eliminarIzquierda()**: Elimina el nodo más a la izquierda del ABB (menor).
- **int eliminarDerecha()**: Elimina el nodo más a la derecha del ABB (mayor).
- **boolean estaVacio()**: Retorna TRUE si el ABB está vacío.

(a) Invente una clase **Ascensor** que utilice la clase **ArbolBinario** para que simule el funcionamiento de un ascensor en la vida real. Los métodos que permite el ascensor son:

- **Ascensor (int n)**: Crea un ascensor de un edificio de con n pisos.
- **boolean presionaPiso (int x)**: Presiona el botón del piso x. Retorna verdadero si el ascensor sube y el piso que ingresa es mayor que el actual o si el ascensor baja y el piso es menor al actual. Si el ascensor está detenido, este método solo verifica que el piso sea distinto al actual y pone la dirección de desplazamiento.
- **int siguientePiso ()**: El ascensor viaja al piso y retorna cuál es su parada. Si retorna -1, no hay pisos por visitar.

El funcionamiento del ascensor es:

- El ascensor siempre que no queda nadie espera a que suba alguien.
- Visita los pisos de manera ordenada (ascendente o descendente) dependiendo de la dirección de viaje.
- Debe guardar la dirección de viaje para saber qué piso es el siguiente. Por ejemplo si está en el 7mo piso, va bajando y se presiona el piso 9, no debería aceptar esto.

Nota: Las variables de instancia deben ser: # de pisos, dirección de desplazamiento (1=arriba, -1=abajo, 0=detenido), piso actual y el ABB correspondiente con los pisos para tenerlos ordenados.

Solución

```
public class Ascensor {
    private ArbolBinario abb;
    private int nPisos;
    private int piso;
    private int dir;

    public Ascensor (int n) {
        this.nPisos = n;
        this.abb = new ArbolBinario();
        this.piso = 1;           // El ascensor parte en el 1
        this.dir = 0;           // No tiene dirección inicial
    }

    public boolean presionaPiso (int x) {
        // No se puede si pasamos el tope
    }
}
```

```
// o si el piso no está en el camino del ascensor
if (x > this.nPisos ||
    (this.dir != 0 && (x * this.dir) <= this.piso))
    return false;

// Agrega el piso
this.abb.agregarValor (x);

// Toma la dirección del desplazamiento
this.dir = Math.sign(this.piso - x);

return true;
}

public int siguientePiso () {
    // Si no hay pisos
    if (this.abb.estaVacio ())
        return -1;

    // Depende de la direccion
    if (this.dir > 0)
        this.piso = this.abb.eliminarIzquierda ();
    else
        this.piso = this.abb.eliminarDerecha ();

    return this.piso;
}
}
```

(b) Simule el siguiente diálogo:

Ascensor en piso 1	Opción: PISO 5
Opción: PISO 3	Piso 5 marcado
Piso 3 marcado	Opción: IR
Opción: PISO 7	Ascensor en piso 5
Piso 7 marcado	Opción: IR
Opción: IR	Ascensor en piso 7
Ascensor en piso 3	Opción: IR
Opción: PISO 2	Ascensor detenido
No puede ir a ese piso	...

Nota: Simplifique el problema pensando en que el edificio tiene 10 pisos y sin subterráneos.

Solución

```
public class Simulacion {
    public static void main (String args[]) {
        BufferedReader in = new BufferedReader (
            new InputStreamReader (System.in));
        Ascensor a = new Ascensor (10);
        System.out.println("Ascensor en piso 1");
        while (true) {
            System.out.print ("Opción: ");
            String op = in.readLine();

            if (op.indexOf("PISO") == 0) {
                int piso = Integer.parseInt(
                    op.substring(4));
                if (ascensor.presionaPiso(piso))
                    System.out.println("Piso " + piso +
                        " marcado");
            }
        }
    }
}
```

```
        else
            System.out.println("No puede ir" +
                               " a ese piso");
    }
    else if (op.indexOf("IR") == 0) {
        int piso = ascensor.siguientePiso();
        if (piso > 0)
            System.out.println("Ascensor en " +
                               "piso " + piso);
        else
            System.out.println("Ascensor " +
                               "detenido");
    }
    else
        System.out.println("Opción Inválida");
    }
    in.close();
}
```

7. El Supermercado (Propuesto)

En un supermercado desean hacer un inventario de cuántos productos por categoría tienen. Para ello se ha decidido que las categorías de productos son las siguientes:

- Alimentación (Fideos, Arroz, Puré en caja, Sopa, Salsa de Tomates, etc)
- Lacteos (Leche, Yogurt, Mantequilla, Manjar, Queso, Quesillo)
- Carnes (Vacuno, Pollo, Pavo, Cerdo)
- Frutas y Verduras (Congeladas y Frescas)
- Bebidas (Jugo, Gaseosa, Licor)
- Limpieza (Cera, Limpiador, Detergente, Lavaloz, Desinfectante)
- Cosmética (Shampoo, Jabón, Crema)
- General (productos que no caen en ninguna categoría)

¡Como podemos ver es una lista bastante larga!

Se le pide modelar el inventario a través de una "tabla de acceso rápido" utilizando un arreglo para las categorías (así se puede organizar más fácil la información). Cada posición del arreglo posee un puntero a un Nodo de la forma:

```
public class Nodo {
    public String producto, categoria;
    public int precio, cantidad;
    public Nodo siguiente;

    public Nodo (String p, String c, int v) {
        this.producto = p;
        this.categoria = c;
        this.precio = v;
        this.cantidad = 0;
    }
}
```

El cual es la cabeza de una lista con los productos de esa categoría.

(a) Enuncia la clase abstracta Inventario indicando:

- Variables de Instancia para simular lo indicado.
- Constructor que permita iniciar una tabla vacía (solo con la categoría General y vacía).

Nota: Suponga un máximo de 100 categorías.

(b) Construya un método público:

public void crearCategoria(String nombre)

que inserta una nueva categoría a la tabla de acceso rápido del inventario.

(c) Construya un método público:

```
public void insertarProducto(String categoria, String nombre,  
                             int cantidad, int precio)
```

que permita insertar un producto en la tabla. Si el producto existe, el método debe incrementar la cantidad que posee la lista y cambiar el precio, pero sin perder el producto anterior.

Para resolver esto se le sugiere que construya primero un método privado:

```
private Nodo buscarCategoría(String nombre)
```

que busca una categoría dentro de la tabla y que retorna el Nodo a la lista de productos de esa categoría.

(d) Escriba el método:

```
public void eliminarProducto(String categoria, String nombre)
```

que elimina un producto del inventario.

8. Arreglo que Simula una Lista Enlazada

(Algo de comprensión de conceptos, y mucha práctica en transformar requerimientos en programas hechos en Java, es decir, para soltar la mano)

Conocemos cómo funcionan las listas enlazadas. Se desea simular una lista enlazada a través de un arreglo. ¿Cómo?. Aquí construiremos la idea:

- (a) Construya una estructura de datos que permita almacenar un valor String y un entero. Además provea un constructor para esos datos. Este elemento debe ser *comparable*, es decir, implementar la interface **Comparable** y construir el método **compareTo(x)** utilizando el compareTo del String.

Solución

```
public class Elemento extends Comparable {
    public String info;
    public int sgte;

    public Elemento (String x, int n) {
        this.info = x;
        this.sgte = n;
    }

    public int compareTo (Object x) {
        return this.info.compareTo(x);
    }
}
```

- (b) Escriba la interface **Lista** que posea métodos de mirada, inserción y eliminación de un arreglo que simulen sus pares en los arreglos reales, es decir:

- **Obtener Elemento** es como obtener el valor del i-ésimo elemento del arreglo.
- **Insertar Elemento** permite agregar un nuevo elemento (**Object**) al arreglo en la posición i-ésima.
- **Eliminar Elemento** permite sacar definitivamente del arreglo uno de los elementos ubicado en la posición i-esima.

Solución

```
public interface Lista {
    // Para obtener el n-ésimo elemento
    public Object obtener (int n);

    // Para insertar un elemento en la posición n
    public void inserar (Object x, int n);

    // Para eliminar un elemento de la posición n
    // Retorna el elemento eliminado
    public Object eliminar (int n);
}
```

- (c) Implemente una clase **ListaElemento** que implemente **Lista** y que permita almacenar objetos de tipo **Elemento** en sus posiciones. Solo escriba la definición (variables de instancia y constructor que crea vacío un arreglo de n elementos).

Solución

```
public class ArregloElemento {
    private Elemento[] arreglo;
    private int largo;
    public int inicio, final;

    public ArregloElemento (int n) {
        this.largo = n;
        this.arreglo = new Elemento[n];
        this.inicio = 0;
        this.final = 0;
    }
}
```

- (d) Programe el método **obtener**.

Solución

```
public Object obtener(int n) {
    return this.arreglo[n];
}
```

- (e) Programe el método **insertar** pensando en:

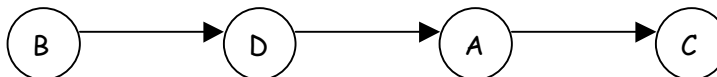
- Al insertar en el *enésimo* elemento, se inserta al final del arreglo real.
- El campo **inicio** de **ArregloElemento** guarda la cabeza de la lista.
- El campo **sgte** de **Elemento** es como el puntero de las listas enlazadas: Indica el índice del siguiente elemento.
- Si el campo **sgte** es -1, significa que la lista llega hasta ahí.

Por ejemplo:

```
Lista al = new ListaArreglo(100);
```

```
al.insertar("A", 0)           // Inserta al principio de la lista
al.insertar("B", 0)           // Inserta al principio de la lista
al.insertar("C", al.final)    // Inserta al final de la lista
al.insertar("D", 1)           // Inserta en la posición 1
```

La lista quedaría así:



Pero físicamente dentro del arreglo está así:

A	B	C	D								
2	3	-1	0								
0											

final

largo

Como se puede ver, casualmente "final" concuerda exactamente con la cantidad de elementos.

Solución

```
public void insertar (Object x, int n) {
    Elemento elto = (Elemento) x;

    // Se verifica que exista espacio.
    if (this.final < this.largo) {
        this.arreglo[this.final] = elto;

        // Para enganchar, buscamos el n-1 elemento
        int c = 0;
        int i = this.inicio;
        while (c < n-1)
            i = this.arreglo[i].sgte;

        if (n == this.final)
            this.arreglo[this.final].sgte = -1;
        else
            this.arreglo[this.final].sgte =
                this.arreglo[i].sgte;

        this.arreglo[i].sgte = this.final;

        // Corremos el final
        this.final++;
    }
}
```

(f) Programe el método **eliminar** pensando en:

- El índice a eliminar no es el mismo que en la lista.
- Al eliminar un elemento de en medio, se debe correr el arreglo a la izquierda (es decir, ocupar el espacio vacío con los siguientes)

Solución

```
public Object eliminar (int n) {
    // Buscamos el que se borra
    int c = 0;
    int i = this.inicio;
    if (n > 0) {
        while (c < n-1)
            i = this.arreglo[i].sgte;

        // Se borra lógicamente
        c = this.arreglo[i].sgte;
        this.arreglo[i].sgte = this.arreglo[c].sgte;
    }
    else {
        // Se borra lógicamente
        c = i;
        this.inicio = this.arreglo[i].sgte;
    }
}
```



```
// Para retornarlo
Elemento x = this.arreglo[c];

// Se corren los elementos que están de más
for (int j=c; j<this.final-1; j++)
    this.arreglo[j] = this.arreglo[j+1];

// Se eliminó físicamente
this.final--;

return x;
}
```

9. MergeSort en Listas Enlazadas

Implementa el algoritmo de MergeSort para que funcione con listas enlazadas, El encabezado del método es:

```
public static Nodo mergeSort (Nodo p, int n)
```

Este método recibe la cabeza de una lista formada por Nodos (ver clase adjunta), y retorna la cabeza de la lista ya ordenada.

El algoritmo de mergesort divide en dos los datos que tiene que ordenar, los ordena recursivamente (llamándose a si mismo con cada mitad) posteriormente realiza el merge, es decir, la mezcla de las sublistas ordenadas.

Para listas pequeñas (≤ 2 Nodos), mergesort no utiliza recursión, si no que las entrega en el orden correcto comparando los campos de información de los Nodos.

```
class Nodo {  
    public Comparable info ;  
    public Nodo sgte ;  
  
    public Nodo (Comparable x) {  
        this.info = x ;  
    }  
}
```

Solución

```
// Mergesort para listas de Nodos  
  
public static Nodo mergeSort (Nodo p, int n) {  
    if (n > 2) {  
        // buscamos el punto de corte  
        Nodo r = p ;  
        for (int i = 0; i < n/2 ; i++)  
            r = r.sgte ;  
  
        // cortamos y dejamos la segunda mitad en q  
        Nodo q = r.sgte ;  
        r.sgte = null ;  
  
        // ordenamos las mitades recursivamente  
        p = mergeSort(p,n/2) ;  
        q = mergeSort(q,n-n/2) ;  
  
        // retornamos la cabeza de la lista resultado  
        return merge(p,q) ;  
    }  
    else { // son a lo mas 2 nodos los que hay que ordenar  
        if (n <= 1 || p.info.compareTo(p.sgte.info) < 0)  
            return p ;  
  
        // aca sabemos que son 2 y que estan al revés  
        Nodo q = p.sgte ;  
        p.sgte = null ;  
        q.sgte = p ;  
        return q ;  
    }  
}
```

```
    }  
}  
  
// mezcla dos listas ordenadas, retornando la cabeza del resultado  
private static Nodo merge (Nodo p, Nodo q) {  
    Nodo n = null, pi = p, qi = q, aux ;  
  
    // nos quedan nodos en ambas listas  
    while (pi != null && qi != null) {  
        if ( pi.info.compareTo(qi.info) > 0) {  
            aux = pi ;  
            pi = pi.sgte ;  
        }  
        else {  
            aux = qi ;  
            qi = qi.sgte ;  
        }  
  
        // se agrega el nodo seleccionado (aux) a la lista  
        // resultado que comienza en n  
        aux.sgte = n ;  
        n = aux ;  
    }  
  
    // salimos pq se nos acabo qi antes que pi  
    while (pi != null) {  
        aux = pi ;  
        pi = pi.sgte ;  
        aux.sgte = n ;  
        n = aux ;  
    }  
  
    // salimos pq se nos acabo pi antes que qi  
    while (qi != null) {  
        aux = qi ;  
        qi = qi.sgte ;  
        aux.sgte = n ;  
        n = aux ;  
    }  
    return n;  
}
```

10. La Ruleta

La clase **Casillero** corresponde a un número de una Ruleta; tiene enlaces el siguiente Casillero y al anterior, y guarda un contador que se puede incrementar utilizando el método **inc()**.

```
class Casillero {
    public int num, frec ;
    public Casillero sgte, ant ;
    public Casillero (int n) {
        this.num = n ;
        frec = 0 ;
    }
    public void inc() {
        frec++ ;
    }
}
```

Implementa la clase Ruleta, con los métodos y variables de instancia que se enuncian a continuación:

Variables de Instancia	
Casillero cero ;	referencia al primer casillero
final int N = 37 ;	número total de casilleros
int jugadas ;	lleva la cuenta de cuántas veces se ha jugado
Métodos	
Ruleta()	constructor, crea los N casilleros y establece que se han jugado cero veces.
int lanzar()	avanza o retrocede (con la misma probabilidad) un numero aleatorio de casilleros, partiendo desde el apuntado por cero . Realiza esto N veces, y finalmente incrementa el contador del casillero que
int porcentaje(int k)	retorna el porcentaje de veces que salio

Propuesto: crear los casilleros en orden aleatorio.

Solución

```
class Ruleta {
    Casillero cero ;
    final int N = 37 ;
    int jugadas ;

    public Ruleta () {
        jugadas = 0 ;
        cero = new Casillero(0) ;
        Casillero p = cero ;
        for (int i = 0 ; i < N ; ++i) {
            p.sgte = new Casillero (i+1) ;
            p.sgte.ant = p ;
        }
    }
}
```

```
        p = p.sgte ;
    }
    p.sgte = cero ;
    cero.ant = p ;
}

public int lanzar () {
    jugadas++ ;
    Casillero p = cero ;
    for (int i = 0 ; i < N ; i++) {
        if (Math.random() < 0.5)
            p = avanzar(p,(int)(Math.random()*N)) ;
        else
            p = retroceder(p,(int)(Math.random()*N)) ;
    }
    p.inc() ;
    return p.num ;
}

private Casillero avanzar (Casillero p, int n) {
    while (n-- > 0)
        p = p.sgte ;
    return p ;
}

private Casillero retroceder (Casillero p, int n) {
    while (n-- > 0)
        p = p.ant ;
    return p ;
}

public double porcentaje (int k) {
    Casillero p ;
    for (int i = 0 ; i < k ; k++ )
        p = p.sgte ;
    return 1.0*p.frec/jugadas ;
}
}
```

11. Lista de Índices de un Arreglo

Cuando se tiene un arreglo grande, en el cual hay algunas posiciones ocupadas y otras que no, es muy lento el proceso de encontrar una posición desocupada para insertar un nuevo elemento.

Para eso se quiere crear la clase **ListaIndice**, la cual debe mantener una lista de las posiciones libres del arreglo, y además permitir hacer de forma fácil y eficiente:

- La inserción secuencial de un elemento nuevo
- La inserción aleatoria de un elemento nuevo
- La inserción aleatoria de un conjunto de elementos nuevos

Para cumplir tales requerimientos, se ha diseñado la clase **ListaIndice** de la siguiente forma:

Variables de Instancia	
Object A[]	Referencia al arreglo de Object's del cual se encarga
NodoIndice primero	Referencia al primer nodo de la lista
int libres	Lleva la cuenta de las posiciones libres
Métodos	
ListaIndice(int n, Object X[])	Constructor que recibe un arreglo de objetos y su tamaño, e inicializa la lista con todas las posiciones como libres
boolean ubicarSecuencialmente(Object x)	Coloca al objeto x en la primera posición libre que hay, retornado true si pudo insertarse o false si no quedaba espacio
boolean ubicarAleatoriamente (Object x)	Coloca al objeto x en una posición aleatoria dentro de todas las posiciones libres que hay, retornado true si pudo insertarse o false si no quedaba espacio
boolean llenarAleatoriamente (Object X[], int n)	Ubica aleatoriamente los n elementos del arreglo de objetos X, solo si hay espacio para todos.

Nota: este tipo de listas se utiliza en muchos computadores y programas para poder asignar eficientemente la memoria (si no, sería un parto cargar un programa nuevo en un computador con el 80% de su memoria ocupada).

Solución

```
class NodoIndice {
    public int indice ;
    public NodoIndice sgte ;
    public NodoIndice (int k) {
        this.indice = k ;
        this.sgte = null ;
    }
}

class ListaIndice {
    Object A[] ;
    NodoIndice primero ;
    int libres ;
}
```

```
public ListaIndice (int n, Object X[]) {
    primero = null ;
    NodoIndice p ;
    for ( int i = 0 ; i < n ; i++) {
        p = new NodoIndice(n-i) ;
        p.sgte = primero ;
        primero = p ;
    }
    A = X ;
    libres = n ;
}

public boolean ubicarSecuencialmente (Object x) {
    if (libres < 1)
        return false ;
    int pos = primero.indice ;
    primero = primero.sgte ;
    A[pos] = x ;
    libres-- ;
    return true ;
}

public boolean ubicarAleatoriamente (Object x) {
    if (libres < 1)
        return false ;
    int n = (Math.random()*libres+1) ;
    int pos = primero.indice ;
    for (int i = 1 ; i < n ; ++i)
        p = p.sgte ;
    A[pos] = x ;
    libres-- ;
    return true ;
}

public boolean llenarAleatoriamente (Object x[], int n) {
    if (libres < n)
        return false ;
    for (int i = 0 ; i < n ; i++)
        ubicarAleatoriamente(x[i]) ;
    libres -= n ;
    return true ;
}
}
```

12. Ajuste de Texto

Una de las cosas que mas se aborrecen en la red (y en general, en cualquier medio de publicacion) es el tener que leer lineas muy largas. Para resolver el problema, te proponemos la implementacion de un programita que lee un archivo y ajusta el texto para encajar en un ancho de caracteres dado.

Dado que un archivo no es otra cosa que una lista de listas (una lista de lineas, cada una de las cuales es una lista de palabras), utilizamos las siguientes clases para modelar los elementos relevantes:

NodoPal Almacena una palabra (que en el texto esta separada de otras por espacios) y una referencia al siguiente **NodoPalabra**
ListaPal Simula una linea de texto (compuesta por palabras)
NodoListaPal Almacena una referencia a una **ListaPal** (linea de texto) y una al siguiente **NodoListaPal**
SuperListaPal Simula un conjunto de lineas de texto (una lista de ellas)

El siguiente programa utiliza una **SuperListaPal** para ajustar el texto del archivo que se le entrega como parametro.

```
public static void main (String args[]) throws IOException {
    BufferedReader file = new BufferedReader(
        new FileReader(args[0])) ;
    String linea ;
    SuperListaPal T = new SuperListaPal() ;
    while ((linea = file.readLine()) != null)
        T.agregar(new ListaPal(linea)) ;
    T.ajustar(Integer.parseInt(args[1])) ;
    for (NodoListaPal A = T.primeros ; A != null ; A = A.sgte)
        System.out.println(A.L.toString()) ;
}
```

¿Cómo funciona?

Supongamos que tenemos el siguiente archivo **test.txt**:

```
Este es un archivo de prueba para el programa que acabo de hacer.
Este programa pesca un archivo de texto (como este, claro) y un
numero entero >1, y con eso ajusta el texto para que utilice solo
esa cantidad de columnas y no mas. Obviamente, no corta las
palabras,
y si hay alguna mas larga que el nmero dado, la deja pasar.
```

y ejecutamos:

```
java FijarAncho test.txt 30
```

entonces obtenemos:

Este es un archivo de prueba para el programa que acabo de hacer. Este programa pesca un archivo de texto (como este, claro) y un numero entero >1, y con eso ajusta el texto para que utilice solo esa cantidad de columnas y no mas. Obviamente, no corta las palabras, y si hay alguna mas larga que el nmero dado, la deja pasar.

Los nodos utilizados son:

```
class NodoPal {
    public String pal ;
    public NodoPal sgte ;
    public NodoPal (String x) {
        this.pal = x ;
    }
}
```

```
class NodoListaPal {
    public ListaPal L ;
    public int largo ;
    public NodoListaPal sgte ;

    public NodoListaPal (ListaPal x) {
        this.L = x ;
        this.largo = x.largo() ;
        this.sgte = null ;
    }
}
```

Los métodos de las clases ListaPal y SuperListaPal son:

```
class ListaPal {
    NodoPal primero ;
    public ListaPal () {...}
    public ListaPal (String linea) {...}
    public void agregarAlFinal (String s) {...}
    public void agregarAlPrincipio (String s) {...}
    public NodoPal sacarUltimo() {...}
    public int largo() {...}
    public String toString() {...}
}
```

```
class SuperListaPal {
    NodoListaPal primero ;
    public SuperListaPal() {...}
    public void agregar (ListaPal L) {...}
    public int ajustar (int c) {...}
}
```

Se te pide implementar:

(a) La función de la clase **SuperListaPal**: **public int ajustar (int c)** suponiendo que todo el resto ya está listo. Esta función recibe un número c (máximo de caracteres) y se encarga de que cada ListaPal no tenga un largo() mayor a c, colocando las palabras que sobren al final en la ListaPal siguiente (creándola si no queda más).

(b) De la clase ListaPal, las funciones:

- **public void agregarAlFinal(String s)**: agrega un String en un nuevo NodoPal al final de la lista
- **public void agregarAlPrincipio(String s)**: agrega un String en un nuevo NodoPal al principio de la lista
- **public NodoPal sacarUltimo()**: saca el ultimo NodoPal de la lista y lo retorna

Solución

Esta solución está completa pues si lo copian a un archivo .java podrán ejecutarlo en Jolie o JDK.

```
import java.io.* ;
import java.util.* ;

class NodoPal {
    public String pal ;
    public NodoPal sgte ;

    public NodoPal (String x) {
        this.pal = x ;
    }
}

class ListaPal {
    NodoPal primero ;

    public ListaPal () {
        primero = null ;
    }

    public ListaPal (String linea) {
        primero = null ;
        int c ;
        while ((c=linea.indexOf(" ")) != -1) {
            if (c != 0)
                agregarAlFinal(linea.substring(0,c)) ;
            linea = linea.substring(c+1) ;
        }
        if (linea.length() != 0)
            agregarAlFinal(linea) ;
    }

    public void agregarAlFinal (String s) {
        NodoPal n = new NodoPal(s) ;
        if (primero == null) {
            primero = n ;
            return ;
        }
        else {
            NodoPal aux = primero ;
            while (aux.sgte != null)
```

```
        aux = aux.sgte ;
        aux.sgte = n ;
    }

    public void agregarAlPrincipio (String s) {
        NodoPal n = new NodoPal(s) ;
        n.sgte = primero ;
        primero = n ;
    }

    public NodoPal sacarUltimo() {
        if (primero == null)
            return null ;
        else if (primero.sgte == null) {
            NodoPal aux = primero ;
            primero = null ;
            return aux ;
        }
        else {
            NodoPal p = primero ;
            while (p.sgte.sgte != null)
                p = p.sgte ;
            NodoPal aux = p.sgte ;
            p.sgte = null ;
            return aux ;
        }
    }

    public int largo() {
        int l = 0 ;
        for (NodoPal p = primero ; p != null ; p = p.sgte )
            l += p.pal.length()+1 ;
        return l-1 ;
    }

    public String toString() {
        String s = "" ;
        for (NodoPal n = primero ; n != null ; n = n.sgte)
            s += " " + n.pal ;
        return (primero==null?s:s.substring(1)) ;
    }
}

class NodoListaPal {
    public ListaPal L ;
    public int largo ;
    public NodoListaPal sgte ;

    public NodoListaPal (ListaPal x) {
        this.L = x ;
        this.largo = x.largo() ;
        this.sgte = null ;
    }
}

class SuperListaPal {
    NodoListaPal primero ;

    public SuperListaPal() {
        primero = null ;
    }
}
```

```
public void agregar (ListaPal L) {
    if (primero == null)
        primero = new NodoListaPal(L) ;
    else {
        NodoListaPal N = primero ;
        for ( ; N.sgte != null ; N = N.sgte) ;
        N.sgte = new NodoListaPal(L) ;
    }
}

public int ajustar (int c) {
    NodoListaPal p = primero ;
    int n = 0 ;
    while (p != null) {
        while (p.largo > c &&
            !(p.L.primero.sgte == null)) {
            String temp = p.L.sacarUltimo().pal ;
            if (p.sgte == null)
                p.sgte = new NodoListaPal(
                    new ListaPal(temp)) ;
            else
                p.sgte.L.agregarAlPrincipio(temp) ;
            p.largo = p.L.largo() ;
            p.sgte.largo = p.sgte.L.largo() ;
        }
        p = p.sgte ;
        n++ ;
    }
    return n ;
}

}

class FijarAncho {
    public static void main (String args[]) throws IOException {
        BufferedReader file = new BufferedReader(
            new FileReader(args[0])) ;
        String linea ;
        SuperListaPal T = new SuperListaPal() ;
        while ((linea = file.readLine()) != null)
            T.agregar(new ListaPal(linea)) ;
        T.ajustar(Integer.parseInt(args[1])) ;
        for (NodoListaPal A = T.primero ; A != null ;
            A = A.sgte)
            System.out.println(A.L.toString()) ;
    }
}
```

13. El Traductor

Una de las grandes gracias de los computadores es su inmensa capacidad de leer y procesar información. De hecho algo muy utilizado en la actualidad son los traductores.

Existe un archivo que se llama "diccionario.txt" que posee todas las palabras del español que tienen traducciones a otros idiomas. El formato de este archivo es:

- Palabra en Español (ej: "crear")
- @
- Traducciones en Inglés separadas por comas (ej: "to made, to make")

Nos damos cuenta que en general podemos tener más de una palabra en español que poseen la misma traducción en inglés y viceversa.

Se desea construir la clase **Traductor** que almacene esta información a través de un árbol de texto con las siguientes funcionalidades:

- **Traductor(String dic):** Constructor que lee el diccionario con el nombre del archivo indicado en el parámetro.
- **String traducir(String palabra):** Devuelve todas las traducciones al inglés de la palabra (separadas por comas).
- **String significado(String palabra):** Devuelve todos los significados (separados por comas) que la palabra (en inglés) tiene en español.

Entonces, por ejemplo, si utilizamos el diccionario, éste sería:

```
Traductor t = new Traductor("diccionario.txt");

// esto despliega "up, over"
System.out.println(t.traducir("arriba"));

// eso despliega "término, final, conclusión"
System.out.println(t.significado("end"));
```

Para hacer esto:

- (a) Defina la clase abstracta **Nodo** que posee el campo de información como si fuera un String y un método set y get que manipulan este campo.

Solución:

```
public abstract class Nodo {
    protected String info;

    public String get() { return this.info; }
    public void set(String x) { this.info = x; }
}
```

- (b) Defina las clases **NodoSignificado** y **NodoPalabra** que son heredadas de tipo **Nodo** y que permiten manipular las palabras en inglés y español respectivamente. Para el caso de las palabras inglesas, suponga que las traducciones se guardan en una lista enlazada y las palabras en español son parte de un árbol de búsqueda binaria (con Strings), que posee una cabeza de lista a una lista enlazada para las traducciones de ella al inglés.

Solución:

```
public class NodoPalabra extends Nodo {
    public NodoPalabra izq, der;
    public NodoSignificado trad;
    public NodoPalabra(String x) {
        super.set(x);
        this.izq = this.der = null;
        this.trad = null;
    }
}

public class NodoSignificado extends Nodo {
    public NodoSignificado sgte;
    public NodoSignificado(String x) {
        super.set(x);
        this.sgte = null;
    }
}
```

Suponiendo la siguiente definición de la clase **Traductor**:

```
public class Traductor {
    private NodoPalabra diccionario;
```

- (c) Programe el método **private void insertar(String linea)** que inserta en el árbol una palabra y sus significados. La información viene dentro de la línea según lo indicado en la definición del archivo⁴.

Solución:

```
private void insertar(String linea) {
    String palabra = linea.substring(0, linea.indexOf("@"));
    String traducc = linea.substring(palabra.length()+1);

    // Creamos el nodo para la palabra
    NodoPalabra pal = new NodoPalabra(palabra);

    // Le ponemos todos los significados a esa palabra
    while (traducc.indexOf(",") >= 0) {
        // Obtenemos la palabra
        String t = traducc.substring(0, traducc.indexOf(","));
        traducc = traducc.substring(t.length()+1);

        // Le creamos un Nodo y lo insertamos al comienzo
        // de la lista de significados
    }
}
```

⁴ Es muy importante saber que las comparaciones no consideran el "case" de la palabra.

```
        NodoSignificado sig = new NodoSignificado(t);
        sig.sgte = pal.trad;
        pal.trad = sig;
    }
    // Falta la última palabra:
    // Le creamos un Nodo y la insertamos al comienzo de la
    // lista de significados
    NodoSignificado sig = new NodoSignificado(traducc);
    sig.sgte = pal.trad;
    pal.trad = sig;

    // Insertamos el nodo con la palabra en el árbol, recorriendo
    // hasta la hoja donde debemos insertar
    NodoPalabra q = this.diccionario;
    NodoPalabra a = null;                                // anterior a q

    while (q != null) {
        a = q;
        if (q.get().compareTo(pal.get()) > 0) {
            q = q.izq;
        }
        else if (q.get().compareTo(pal.get()) < 0) {
            q = q.der;
        }
        else {                                           // palabra repetida
            break;
        }
    }

    // Una vez que llegamos a Null debemos insertarla.
    if (q == null) {
        if (a == null) {                                // está vacío el árbol
            this.diccionario = pal;
        }
        else {
            if (a.get().compareTo(pal.get()) > 0) {
                a.izq = pal;
            }
            else if (a.get().compareTo(pal.get()) < 0) {
                a.der = pal;
            }
        }
    }
}
```

(d) Implemente el **constructor** de la clase Traductor que, utilizando insertar, vaya poniendo todas las palabras del archivo en el diccionario.

Solución:

```
public Traductor(String archivo) {
    try {
        BufferedReader file = new BufferedReader(
            new FileReader(archivo));

        while (true) {
            String linea = file.readLine();
            if (linea == null) { break; }
            this.insertar(linea);
        }
        file.close();
    }
    catch (Exception e) {
```

```
        System.out.println("ERROR AL USAR EL ARCHIVO");  
    }  
}
```

(e) Escriba los métodos **traducir** y **significado** que se indican en la definición de traductor.

Solución:

```
// Este método lo único que hace es recorrer el árbol hasta  
// encontrar la palabra  
public String traducir(String palabra) {  
    NodoPalabra q = this.diccionario;  
  
    while (q != null && q.get().compareTo(palabra) != 0) {  
        if (q.get().compareTo(palabra) > 0) {  
            q = q.izq;  
        }  
        else if (q.get().compareTo(palabra) < 0) {  
            q = q.der;  
        }  
    }  
  
    if (q != null) {  
        String res = ""; // se encontró  
  
        // Recorremos la lista de significados  
        NodoSignificado ns = q.trad;  
        while (ns != null) {  
            res = res + ns.get() + ", ";  
            ns = ns.sgte;  
        }  
  
        return res;  
    }  
    else { // no se encontró  
        return null;  
    }  
}
```

```
// Este otro es un poquito más complicado, porque debemos revisar  
// TODAS las traducciones de cada palabra. Lo mejor es hacerlo  
// recursiva  
public String significado(String palabra) {  
    return this.significado(this.diccionario, palabra);  
}  
  
private String significado(NodoPalabra p, String x) {  
    // Caso base, que no encontremos más palabras  
    if (p == null) {  
        return "";  
    }  
  
    // En cada nodo revisamos si existe la palabra x  
    NodoSignificado ns = p.trad;  
    while (ns != null && ns.get().compareTo(x) != 0) {  
        ns = ns.sgte;  
    }  
  
    // Si lo encontramos, no salimos con ns = null  
    String res = "";  
    if (ns != null) {
```



```
        res = p.get() + ", ";  
    }  
  
    // Buscamos por ambos lados  
    res = res + this.significado(p.izq, x);  
    res = res + this.significado(p.der, x);  
    return res;  
}  
} // Acá cerramos la clase Traductor
```

- (f) Realice un programa que utilice esta clase para hacer un pequeño traductor, en donde se solicite una palabra (en español) y se devuelvan sus traducciones al inglés y viceversa.

Solución:

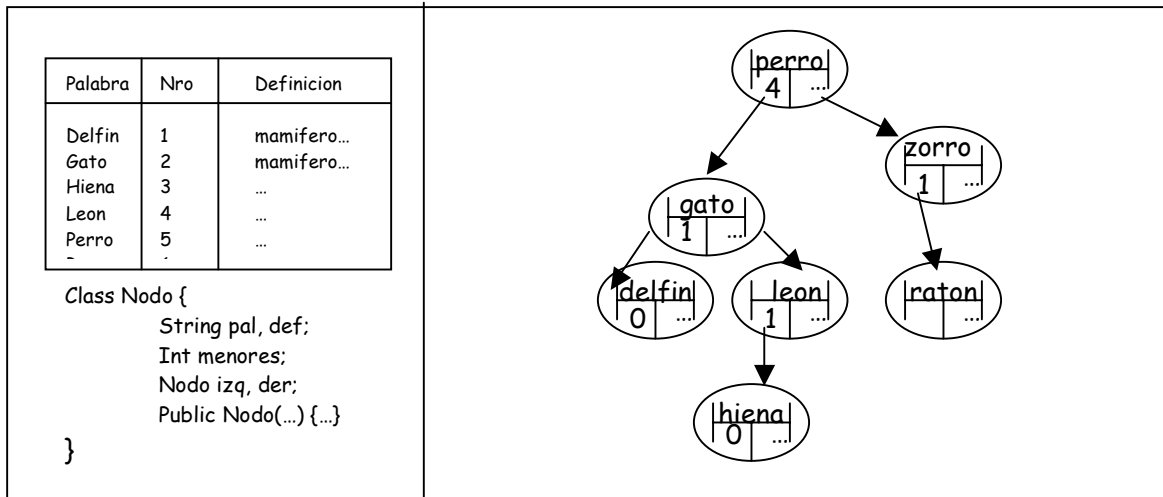
```
public class Programa {  
    public static void main(String[] args) {  
        Console c = new Console();  
        Traductor dic = new Traductor("diccionario.txt");  
        while(true) {  
            c.print("Lenguaje Origen(ESP/ING)?");  
            String lang = c.readLine();  
            c.print("Palabra Origen?");  
            String pal = c.readLine();  
            if (lang.equals("ESP")) {  
                c.println("Traducciones (ESP -> ING) = " +  
                    dic.traducir(pal));  
            }  
            else if (lang.equals("ING")) {  
                c.println("Traducciones (ING -> ESP) = " +  
                    dic.significado(pal));  
            }  
        }  
    }  
}
```

14. Diccionario

Se desea implementar el tipo de datos abstracto diccionario ordenado. Un diccionario ordenado es un diccionario enriquecido con una operación para obtener la I-ésima palabra. Esta última corresponde a la I-ésima palabra que se leer al ordenar alfabéticamente las palabras del diccionario. La siguiente tabla especifica sus operaciones:

Ejemplos	Significado	Encabezado
DiccOrd d=new DiccOrd();	Crea un diccionario ordenado inicialmente vacío	public DiccOrd()
d.definir("correcaminos", "aves velosus"); d.definir("coyote", "coyotis hambrientus");	Establece la definición de correcaminos. Establece la definición de coyote.	void definir (String pal, String def)
String definicion=d.consultar("gcoyote")	Entrega la definición de "coyote": "coyotis hambrientus"	String consultar(String pal)
String palabra=d.palabra(2)	Entrega la segunda palabra del diccionario ordenado alfabéticamente.	String palabra (int i)
int n=d.size();		int size()

Se ha decidido implementar el diccionario ordenado mediante un árbol de búsqueda binaria (ABB) modificado para hacer eficiente la obtención de la I-ésima palabra. Esta modificación consiste en que se incluye en cada nodo el campo menores que indica cuantos nodos contiene el subárbol izquierdo. La siguiente figura muestra arriba a la izquierda un ejemplo de diccionario ordenado y arriba a la derecha su representación mediante un ABB modificado:



Observe que en la raíz del árbol aparece un 4, porque el subárbol izquierdo del nodo raíz posee 4 nodos. Abajo a la izquierda se encuentra la estructura de cada nodo del ABB y a la derecha el bosquejo de la definición de la clase DiccOrd. La llave para el ordenamiento del ABB es el campo pal, es decir la palabra que se define.

Programa los métodos **palabra** y **definir** eficientemente (el tiempo de ejecución de estos métodos debe ser proporcional a la altura de árbol en el peor caso). Al programar el método **definir** usted debe actualizar el número de nodos del subárbol izquierdo.

Propuesto: Redefinir la declaración de los métodos **palabra** y **definir**, y programarlos usando recursividad.

Solución:

```
class Nodo{

    public String palabra;
    public String def;
    public int menores;
    public Nodo izq;
    public Nodo der;

    public Nodo(String p, String d){
        palabra = p;
        def = d;
        izq = der = null;
        menores = 0;
    }
}

class DiccOrd{

    public Nodo raiz;

    public DiccOrd(){
        raiz = null;
    }

    public void definir(String p, String d){

        if(raiz == null){
            raiz = new Nodo(p,d);
            return;
        }

        Nodo aux = raiz;

        while(true){

            if(aux.palabra.compareTo(p)>0){
                if(aux.izq!=null){
                    aux.menores++;
                    aux = aux.izq;
                }
                else{
                    aux.menores++;
                    aux.izq = new Nodo(p,d);
                    break;
                }
            }

            else{
                if(aux.der!=null)
```

```
                aux = aux.der;
            else{
                aux.der = new Nodo(p,d);
                break;
            }
        }
    }
}

public String palabra(int n){
    if(raiz == null)
        return null;

    Nodo aux = raiz;
    int cont = 0;
    while(true){
        if(aux == null)break;
        if(aux.menores + cont == n-1)
            return aux.palabra;
        else if(aux.menores + cont > n-1)
            aux = aux.izq;
        else {
            cont+= (aux.menores+1);
            aux = aux.der;
        }
    }
    return null;
}
```

15. Libros Interactivos

Los libros interactivos son antiguas metodologías que permitían al lector decidir sobre la historia, en cierto sentido. Por ejemplo, si en el Capítulo III el personaje principal debía o no beber la poción negra que encontraba en el cofre era decisión del lector, pero ambos casos podían llevar a desenlaces completamente distintos.

Existe un archivo de texto con todas las escenas del libro con el siguiente formato:

`<nº>@<título>@<texto>@<op1>@<des1>@<op2>@<des2>`

en donde

<code><nº></code>	es el número de escena correlativo
<code><título></code>	es el título de la escena
<code><texto></code>	es el detalle de la escena
<code><op1></code> y <code><des1></code>	indican la opción 1 a decidir y a qué escena lo conduce
<code><op2></code> y <code><des2></code>	indican la opción 2 a decidir y a qué escena lo conduce

Se desea representar este tipo de libros como un árbol con los posibles caminos y así poder interactuar con el lector directamente sin que éste pueda "usmear" en las otras escenas. Cuando ambos destinos son 0, indican que es el final de la aventura.

(a) Escriba una clase que represente un nodo de un árbol que permita guardar esta información.

Solución

```
public class Nodo {
    // No es necesario guardar el número de la
    // escena ya que es el correlativo que se
    // usa solo para los enlaces que se
    // reemplazarán con punteros
    public String titulo;
    public String texto;
    // Guardaremos también las opciones como parte
    // del texto, porque no agrega más o menos
    // valor
    public Nodo op1, op2;
    // Es facil generalizar las opciones usando un
    // arreglo o una lista enlazada
}
```

(b) Escriba la clase LibroInteractivo con

- (i) Variable(s) de Instancia
- (ii) Constructor que reciba el nombre del archivo a leer y construya el libro

Nota: Para construir el libro deberá recorrer el archivo como si éste fuera un árbol. Puede duplicar tantas escenas como nodos apuntados a ellas tenga, es decir, si desde la escena 27 y de la escena 48 puede llegar a la escena 63, puede crear dos escenas 63, una hija de la escena 27 y la otra como hija de la escena 48.

Solución

```
import java.io.*;
public class LibroInteractivo {
    // Solo es necesario conocer el inicio del
    // libro
    public Nodo inicio;

    // El constructor será quien arme el libro
    public LibroInteractivo(String nombre) {
        try {
            this.inicio = this.siguiente(nombre,
                1);
        }
        catch (Exception e) {
            e.printStackTrace();
            this.inicio = null;
        }
    }

    // Como es recursivo, requiere del siguiente
    // paso
    public Nodo siguiente(String nombre,
        int escena) throws Exception {
        // Como determinamos si es final
        if (escena == 0) return null;

        // Se busca la escena
        BufferedReader b = new BufferedReader(
            new FileReader(nombre));
        String linea = null;
        for (int i=0; i<escena; i++)
            linea = b.readLine();
        b.close();

        // Se obtiene, y se buscan los hijos
        Nodo p = new Nodo();
        String[] comp = linea.split("@");
        p.titulo = comp[1];
        p.texto = comp[2];
        if (comp[3].length() > 0)
            p.texto += "\n(1) " + comp[3];
        if (comp[5].length() > 0)
            p.texto += "\n(2) " + comp[5];
        p.op1 = this.siguiente(nombre,
            Integer.parseInt(comp[4]));
        p.op2 = this.siguiente(nombre,
            Integer.parseInt(comp[6]));
        return p;
    }
}
```

- (c) Implemente el método `public void leer(Console c)` de la clase `LibroInteractivo` que recibe como parámetro una consola y nos permite realizar el ciclo de lectura con interacción del usuario.

Nota: El libro se puede recorrer mientras tenga hijos el nodo. Suponga que el libro está bien construido.

Solución

```
public void leer(Console c) {
    // Se declara un contador para indicar en
    // que escena va
    int n = 1;
    // Se declara el nodo que recorrerá el
    // camino
    Nodo p = this.inicio;
    while (true) {
        // Se imprime la escena actual
        c.println("Escena " + n + ": " +
            p.titulo);
        c.println(p.texto);
        // Se verifica de que no sea el final
        if (p.op1 == null && p.op2 == null)
            break;
        // Se espera la opcion
        int op = c.readInt();
        // Se cambia de escena
        if (op == 1 && p.op1 != null)
            p = p.op1;
        else if (op == 2 && p.op2 != null)
            p = p.op2;
        else {
            c.println("Opción no Permitida");
            n--;
        }
        n++;
    }
    c.println("FIN");
}
```

Y para probar:

```
public class Programa {
    public static void main(String[] args) {
        Console c = new Console();
        c.print("Libro?");
        String libro = c.readLine();
        LibroInteractivo l =
            new LibroInteractivo(libro);
        l.leer(c);
    }
}
```

Pruébelo con el siguiente archivo "aventura.txt":

```
1@Inicio@...@Izquierda@2@Derecha@3
2@Intriga@...@Izquierda@4@@@0
3@Intriga@...@@@0@Derecha@2
4@Fin@...@@@0@@@0
```

16. PrintTree

Escribir un método que reciba como parámetro un ABB, y que despliegue en pantalla en orden de menor a mayor los valores contenidos en este árbol (los valores son Strings). El encabezado del método debe ser de la forma.

```
public String imprime(Nodo x){...}
```

Considere la siguiente implementación de la clase Nodo

```
Public class Nodo {  
    String info;  
    Nodo izq, der;  
    public Nodo() {  
        info="";  
        izq=der=null;  
    }  
    public Nodo( String x) {  
        info=x;  
        izq=der=null;  
    }  
}
```

Solución:

```
public String in_orden(Nodo subraiz){  
    if (subraiz==null){  
        return "";  
    }  
    else  
        return in_orden(subraiz.izq)+subraiz.info+  
               in_orden(subraiz.der);  
}
```


17. Codificador Morse (Propuesto)

Se desea implementar un codificador de clave morse. Para lo cual se propone utilizar un ABB, con la definición de este código (se encuentra al final del enunciado). Considere que sólo se pide codificar letras (sin acentos ni caracteres especiales), y que los espacios en el texto encriptado son también espacios en el texto. Además se utiliza el separador "/" para cada carácter encriptado. Utilice la estructura de un ABB para codificar *eficientemente* el texto ingresado. (Suponga este árbol ya creado para simplificar)

Ejemplo de Uso:

```
% java Morse.java
% Bienvenido(a) a El Codificador Morse 1.0, Ingrese Texto
% hola mundo
% .../---/.-/.- --/..-/..-/---
%
```

A ·-	B -··	C -·-	D -··	E ·	F ···	G --·	H	I ..
J ·---	K -·-	L ···	M --	N ·-		O ---	P ···	Q ---·
R ·-	S ...	T -	U --	V ...-	Ñ ---·-	X ---	Y ---·	Z ---·
					W ·--			

18. Divisibles y no divisibles

a) Escriba un método de encabezamiento `int reordenar(int n, int[]x, int y)` que reordene los primeros `n` elementos del arreglo `x` en dos grupos:

a la izquierda los divisibles por `y`
a la derecha los que no son divisibles por `y`.

El método debe entregar adicionalmente como resultado la cantidad de elementos que son divisibles por `y`.

Por ejemplo:

```
int[] a = { 5, 8, 6, 9, 3, 2, 4 }; //arreglo de 7 elementos
int i = reordenar(6, a, 2); //reordenar los primeros 6 elementos del arreglo
//deja el arreglo a con los valores {8,6,2,5,9,3,4} y la variable i con el valor 3
```

Solución:

```
static public int reordenar(int n, int[] x, int y){
    //inicializar contador de divisibles
    int k=0;

    //recorrer elementos de arreglo
    for(int i=0; i < n; ++i) {
        //detectar divisibles por y
        if( x[i] % y == 0 ) {
            //poner divisibles al comienzo
            int aux = x[k];
            x[k] = x[i];
            x[i] = aux;
            //contar divisibles
            ++k;
        }
    }
    //devolver contador de divisibles
    return k;
}
```

b) Escribir un programa que use el método anterior para escribir todos los números pares entre 1 y 1000 que son divisibles por 3 y por 5.

Solución:

```
public static void main(String[] args) {
    //declarar e inicializar arreglo
    final int N=1000;
    int[] a = new int[N];
    for(int i=0; i < N; ++i)
        a[i] = i+1;

    //reordenar arreglo
    int n = reordenar(N,a,2);
}
```

```
n = reordenar(n,a,3);  
n = reordenar(n,a,5);  
  
//escribir pares divisibles por 3 y 5  
for(int i=0; i < n; ++i)  
    System.out.println(a[i]);  
}
```

19. Transformada de Burrows-Wheeler.

Esta transformada es muy utilizada en el área de la compresión. Lo que hace es convertir un String x en un resultado compuesto por un String y un número:



En este caso, convirtió "BANANA" en el par ordenado ("NNBAAA", 4).

B	A	N	A	N	A
A	N	A	N	A	B
N	A	N	A	B	A
A	N	A	B	A	N
N	A	B	A	N	A
A	B	A	N	A	N

Se calcula de la siguiente manera. Primero toma el String, y lo "rota circularmente" hacia la izquierda, escribiendo así todas las palabras resultantes dentro de una matriz cuadrada:

A	B	A	N	A	N
A	N	A	B	A	N
A	N	A	N	A	B
B	A	N	A	N	A
N	A	B	A	N	A
N	A	N	A	B	A

Luego ordenamos las palabras (las filas) en orden lexicográfico. La transformada se obtiene observando, dentro de la matriz resultante, viendo la última columna y la posición en que quedó la palabra original.

Se pide escribir:

a) el método `public static ordenarArregloChars(char[][] c, int n, int m)`, que ordene las filas de la matriz c , dejándolas en orden lexicográfico en el mismo arreglo.

Solución:

```
// solucion con metodo de la burbuja
public void ordenarArregloChars(char[][] c, int n, int m)
{
    // vamos agrandando el conjunto de los "listos"
    for (int i = n-1; i>=0; i--)
    {
        // subimos la burbuja hasta que queda "lista"
        for(int j = 0 ; j < i ; j++)
        {
            // sacamos palabras de las fila j y (j+1)
            String pal_j = "", pal_j1 = "" ;
            for(int k = 0 ; k < m ; k++)
            {
                pal_j += c[j][k] ;
                pal_j1 += c[j+1][k] ;
            }
            // si la palabra en j va despues de
            // la de (j+1), las intercambiamos
            if(pal_j.compareTo(pal_j1) > 0)
            {
                // intercambiamos las palabras
                String temp = pal_j;
                pal_j = pal_j1;
                pal_j1 = temp;
            }
        }
    }
}
```

```
        for (int k=0; k<m; k++)
        {
            char aux = c[j][k] ;
            c[j][k] = c[j+1][k] ;
            c[j+1][k] = aux ;
        }
    }
}
```

b) el método public static String BW(String x), que retorne la transformada de x (un String como "NNBAAA 4", colocando el número tras un espacio).

Solución:

```
public String BW(String x)
{
    int Tam = x.length() ;
    char[][] arr = new char[Tam][Tam] ;
    // llenamos primera fila del arreglo
    for (int j = 0 ; j < Tam ; j++)
        arr[0][j] = x.charAt(j) ;
    // llenamos las siguientes con los "shifteos"
    for (int i=0; i<Tam; i++)
        for (int j=1; j<Tam; j++)
            arr[j][(i+Tam-j)%Tam] = arr[0][i];
    // ordenamos
    ordenarArregloChars(arr,Tam,Tam) ;
    // obtenemos ultima columna
    String pal = "" ;
    for (int i=0; i<Tam; i++)
        pal += arr[i][Tam-1] ;
    // buscamos donde quedo la palabra original
    int i = 0 ;
    for (; i < Tam ; i++)
    {
        String aux = "" ;
        for (int j = 0 ; j < Tam ; j++)
            aux += arr[i][j] ;
        if (aux.equals(x))
            break ;
        else
            aux = "" ;
    }
    return pal + " " + (i+1);
}
```

20. Emulador HP.

La notación polaca es una manera de definir expresiones en las que el orden en la que aparecerán los operadores trae implícita la prioridad.

Ejemplo de expresiones en notación polaca y su correspondencia en notación normal.

Normal	Notación Polaca
$a+b-c$	$ab+c-$
$a+b*c$	$abc*+$
$a+b*c+d$	$abc*+d+$
$(a+b)*c$	$ab+c*$
$a+(b-c)$	$abc-+$
$a*b+c$	$ab*c+$
$a+b/c$	$abc/+$
$(a+b)/c$	$ab+c/$

Se pide escribir un programa que dada una expresión en notación polaca, la evalúe e imprima el resultado en pantalla. *Indicación:* use la clase Stack cuyos métodos son:

<code>Stack s = new Stack();</code>	Crea el Stack s vacío.
<code>s.push(x)</code>	Inserta el elemento x en el tope del stack.
<code>s.pop()</code>	Retira el elemento del tope del Stack y lo devuelve
<code>s.empty()</code>	Retorna true si el stack está vacío, false si no.
<code>s.full()</code>	Retorna true si el stack está lleno, false si no.
<code>s.reset()</code>	Vacía el stack

Solución:

```
public class Polaca {
    public static void main(String[] args) throws Exception{
        BufferedReader I = new BufferedReader(new
                                InputStreamReader(System.in));
        Stack stack = new Stack();

        System.out.println("Ingrese la expresion en notacion
                                polaca");
        String expression = I.readLine();

        while(expression.length() != 0){
```

```
        char c = expresion.charAt(0);

        if(esNumero(c)){
            stack.push(Integer.parseInt(c+""));
        }
        else if(esOperador(c)){
            double op2 = stack.pop();
            double op1 = stack.pop();
            stack.push(evaluar(op1, c, op2));
        }
        else{
            System.out.println("caracter invalido");
            System.exit(0);
        }
        expresion = expresion.substring(1);
    }
    System.out.println("El resultado de la expresion es: "+
        stack.pop());
    I.close();
}

private static double evaluar(double op1, char c, double op2) {
    switch(c){
        case '*': return op1*op2;
        case '/': return op1/op2;
        case '+': return op1+op2;
        case '-': return op1-op2;
    }
    return 0;
}

private static boolean esOperador(char c) {
    String aux = ""+c;
    if(aux.equals("*") || aux.equals("/") || aux.equals("+")
        || aux.equals("-") ){
        return true;
    }
    return false;
}

private static boolean esNumero(char c) {
    switch(c){
        case '0': case '1': case '2': case '3': case '4':
        case '5': case '6': case '7': case '8': case '9':
            return true;
        default:
            return false;
    }
}
}
```

21. In the Navy.(Propuesto)

La operación OLITAS consiste en maniobras navales en las que participan naves pertenecientes a 9 países. La superficie marítima se cuadricula para representarla en una matriz de 100 por 80 caracteres. Consecuentemente, cada barco se representa por una secuencia ininterrumpida horizontal o vertical del dígito (1 a 9) que representa al país. Los lugares no ocupados por un barco se representan con un dígito cero.

Escriba un programa que escriba el número del país y el tamaño (cantidad de espacios que ocupa) del barco más grande que participa en la operación. La información se encuentra grabada en las 100 líneas del archivo "barcos.txt". Cada línea contiene 80 caracteres (del '0' al '9').

22. Agenda. (Propuesto)

R.B. es una persona que viaja mucho y tiene amigos repartidos por todo el mundo. R.B. ha ideado una estructura muy eficiente para guardar los números de teléfono de sus amigos por medio de un árbol cuyos nodos responden a la siguiente declaración:

```
public class Nodo {  
    public class int numero;  
    public String nombre;  
    //para la raíz esta información es irrelevante  
    public Nodo[] hijo=new Nodo[10];  
    //cantidad máxima de hijos  
    public int hijos=0;  
    //cantidad efectiva de hijos  
}
```

De acuerdo a lo anterior:

- * La raíz del árbol (nivel 1) contiene como información el número que debe marcar para iniciar una llamada internacional (ejemplo 00)
- * Los hijos de la raíz (nivel 2) tienen los números correspondientes a los códigos de cada país donde R.B. tiene registrado el teléfono de al menos un amigo. Estos nodos también contienen el nombre del país. Por ejemplo, 56, "Chile".
- * Los hijos de cada nodo del nivel anterior (nivel 3) contienen los códigos de las ciudades del país representado por el padre, donde R.B. tiene por lo menos un número registrado. Estos nodos contienen además el nombre de la ciudad. Por ejemplo, 2, "Santiago".
- * Finalmente los nodos del nivel 4 contienen la parte final del número telefónico para personas. Estos nodos contienen además el nombre de la persona que tiene ese número. Por ejemplo, 1234567, "Pato".

Por lo anterior, se puede ver que para obtener el número telefónico internacional de una persona dada, basta concatenar los números desde la raíz hasta una de las hojas. Suponga que la clase que implementa esta agenda en forma de árbol como:

```
public class Agenda {  
    private Nodo raiz=new Nodo();  
  
    public String nombre(int numero_pais, int numero_ciudad,  
                        int numero_persona)  
    //devuelve nombre de la persona con ese número de teléfono, null si  
    //no está  
  
    public void agregar (int numero_pais, String nombre_pais,  
                        int numero_ciudad, String nombre_ciudad, int numero_persona,  
                        String nombre_persona)  
  
    //agrega a la persona de nombre dado con el telefono dado a la  
    //estructura sin duplicar nodos de país ni ciudad  
  
    public String fono (String x);  
  
    //busca el nombre x y entrega el fono en la forma pais-ciudad-numero  
    //ejemplo : 56-2-1234567
```

- a) Escriba un método `public static Nodo Hijo (Nodo x, int y)` que retorna el nodo `x` que contiene en su componente `numero` un valor igual a `y`.
- b) Usando a) escriba el método `nombre`.
- c) Escriba el método `fono`.



23. Clase Parrafo.(Propuesto)

La clase Parrafo sirve para administrar la información contenida en un párrafo de un texto. Sus operaciones son :

Método	Descripción
Parrafo(String S, int cols)	Crea un párrafo cuyo contenido es el String S, coles es el máximo de columnas por línea
String palabra(int n)	Devuelve la enésima palabra del párrafo
void escribe(Console c)	Escribe el contenido del párrafo en la consola C. Las líneas tienen un máximo de cols caracteres. Suponga que no hay palabras más largas que el máximo de caracteres por línea (cols). Cada palabra debe estar separada por un blanco de la siguiente.

Ejemplo:

```
Console c=new Console();
Parrafo p=new Parrafo ("Un hipopótamo se balanceaba sobre la tela de una araña", 17);
c.println(p.palabra(4));      //escribe "balanceaba"
p.escribe(c);                //escribe lo siguiente en la consola
                             //Un hipopotamo se
                             //balanceaba sobre
                             //la tela de una
                             //araña
```

La representación interna de la clase es una lista simplemente enlazada de palabras:

```
class Palabra
{
    public String info; //una sola palabra
    public Palabra sgte; //puntero a la siguiente palabra
}
class Parrafo
{
    Palabra primera=null; //inicialmente null
    int cols;
    ...
}
```

- Usando esta representación escriba el método void escribe (Console C)
- La clase Editor es una extensión de la clase Párrafo permite modificar la información contenida en un párrafo de texto. Esta clase equivale a un párrafo que además tiene un cursor que permite insertar letras. Sus métodos son:

Método	Descripción
Editor (String s, int cols)	Igual al constructor de Parrafo, pero además coloca el cursor en la primera columna y primera fila
void avanza()	Corre el cursor una posición a la derecha. Si la columna es mayor

	que el número máximo de columnas se pasa a la primera columna de la siguiente fila.
void borra()	Borra el caracter bajo el cursor
char caracter()	Devuelve el caracter apuntado por el cursor
void inserta(char c)	Inserta el caracter en la posición del cursor

La declaración de la clase es

```
class Editor extends Parrafo {  
    int XCursor=0, YCursor=0; //posición del cursor  
    .... métodos ....  
}
```

Ejemplo de uso :

```
Editor e=new Editor("Una hipopótamo se balanceaba sobre la tela", 13)  
char x=e.caracter(); // en x se guarda U  
e.avanza();  
e.borra(); // borra la a de Una  
x=e.caracter(); // en x queda un espacio  
e.avanza(); e.avanza(); e.avanza(); e.inserta('p');  
Console c=new Console();  
e.escribe(c); //escribe en la consola:
```

```
Un hipopótamo  
se balanceaba  
sobre la tela
```

Escriba el método `public char caracter();`

c) Escriba el método `static void reemplazar(Editor e, char x, char y)` que recibe un objeto de la clase Editor y cambia todas las ocurrencias de la letra x por la letra y usando sólo los métodos de la clase Editor.

24. Insercion Ordenada.

a)Escriba un método de encabezamiento void insertar(String x,String[]y,int max,int n) que inserte x en el lugar que le corresponde en el arreglo y que está ordenado ascendentemente. Los parámetros max y n representan la cantidad máxima y actual de elementos del arreglo y.

Ejemplo: String[]a = new String[4]; a[0]="A"; a[1]="C"; a[2]="E";
insertar("B",a,4,3); //deja a={"A","B","C","E"}
insertar("D",a,4,4); //deja a={"A","B","C","D"} y se pierde el último valor ("E")

b)Utilice el método anterior en un programa que lea el archivo "palabras.txt" que contiene una cantidad indeterminada de palabras (cada una en una línea) y escriba las primeras 100 según el orden alfabético. Suponga que el archivo tiene al menos 100 líneas.

Solución:

```
public class Ordenar {  
    public static void insertar(String x, String[] y, int max,  
                                int n) {  
        //max: cantidad máxima de elementos (tamaño del  
        //arreglo)  
        //n: numero actual de elementos del arreglo y  
  
        //arreglo auxiliar del mismo tamaño que y  
        String[] aux = new String[max];  
  
        int i=0;  
  
        //recorro todos los elementos del arreglo inicial que  
        //sean menores que x  
        for (i = 0; i < n; i++) {  
            if (x.compareTo(y[i]) <= 0)  
                //x < y --> debo insertar  
                break; //debo insertar x  
            else  
                aux[i] = y[i];  
        }  
  
        // si i<max significa que "cabe" x  
        if (i < max) {  
            //inserto x y muevo i en 1  
            aux[i++] = x;  
            //recorro el resto del arreglo  
            for (; i < Math.min(n+1,max); i++) {  
                aux[i] = y[i - 1];  
            }  
        }  
        //copio el arreglo de vuelta a y  
        for (i = 0; i < Math.min(n+1,max); i++)  
            y[i] = aux[i];  
    }  
}
```

```
public static void main(String args[]) throws IOException {
    BufferedReader in=new BufferedReader(new
                                   FileReader("palabras.txt"));

    String linea;
    String arreglo[]=new String[100];
    int contador=0;
    while((linea=in.readLine())!=null){
        insertar(linea,arreglo,100,contador);
        if (contador<100) contador++;
    }
    for (int i=0; i<100; i++)
        System.out.println(arreglo[i]);

}

}
```

25. Clase Lista.(Propuesto)

La clase Lista permite mantener una lista de enteros positivos a través de los siguientes métodos:

ejemplo	resultado	significado
L = new Lista()	-	constructor que inicializa L con cero elementos
L.agregar(x)	void	agregar el N° entero x a la lista L
L.borrar(x)	void	borra todas las apariciones del N° x de la lista L
L.copiar()	Lista	entrega una copia de la lista L

Suponiendo que una Lista se representa con una lista enlazada:

- Escriba las declaraciones que permitan implementar la representación
- Escriba el método borrar de manera que las apariciones del número sólo se marquen como borradas (sin eliminarlas físicamente de la lista enlazada).
- Escriba el método copiar de modo que la nueva Lista contenga sólo los números que no están marcados como borrados.

26. Casino (Propuesto)

La clase Cola tiene definidas las siguientes operaciones:

ejemplo	resultado	significado
<code>C = new Cola()</code>	-	constructor que inicializa cola como vacía (sin elementos)
<code>C.poner(x)</code>	void	agregar String x al final de la cola C
<code>C.sacar()</code>	String	extraer (y devolver) primer String de la cola C
<code>C.vacia()</code>	boolean	devolver true si cola C está vacía (o false si no)

Utilice (sin escribirla) la clase Cola en un programa que simule la operación del casino de alumnos de la Escuela. Al respecto, el archivo "casino.txt" contiene líneas que registraron la operación del casino en un día normal en la forma indicada en el siguiente ejemplo:

L1200 llegó un alumno a las 12:00 y se puso en la cola
L1201 llegó un alumno a las 12:01 y se puso en la cola
A1202 se atendió al primer alumno de la cola a las 12:02
L1204 llegó un alumno y se puso en la cola
A1205 se atendió al primer alumno de la cola

...

El programa debe leer el archivo y producir los siguientes resultados:

- N° total de alumnos atendidos
- Largo máximo de la cola (expresado en N° de alumnos)
- Tiempo de espera promedio, es decir, el promedio de minutos que esperaron los alumnos
- Tiempo máximo de espera, es decir, la cantidad de minutos que más tuvo que esperar un alumno

27. La vuelta a Francia (Propuesto)

El Tour de Francia es una competencia ciclística que se disputa en tres semanas. Cada día se corre una etapa y el tiempo que toma cada ciclista se acumula de modo que resultará ganador del Tour el que totalice el menor tiempo. Al respecto, escriba un programa que se ejecute al final de una etapa y que actualice la clasificación acumulada regrabando el archivo "tour.txt". Los resultados de la etapa se ingresan de acuerdo al diálogo indicado en el siguiente ejemplo:

Nº del ciclista ganador ? 13

Nº del ciclista que llegó en el lugar 2 ? 21

Diferencia en segundos respecto del primero ? 10

Nº del ciclista que llegó en el lugar 3 ? 146

Diferencia en segundos respecto del primero ? 12

...

Nº del ciclista que llegó en el lugar x ? 0 (fin de los datos)

Notas

- Inicialmente compiten 200 ciclistas identificados con números del 1 al 200
- El archivo "tour.txt" está ordenado por lugar y cada línea contiene el lugar (Nº entre 1 y 200), el Nº del ciclista (Nº entre 1 y 200) y la diferencia de tiempo (en segundos) respecto del que marcha primero en la clasificación acumulada. Por supuesto, el primero tendrá una diferencia con valor cero.
- El archivo actualizado debe quedar ordenado por lugar.
- Los ciclistas que no completen la etapa deben ser eliminados de la carrera y por lo tanto no deben aparecer en el archivo actualizado.

28. Enviando Mensajes (Propuesto)

Para transmitir un mensaje grande de un computador a otro se acostumbra enviarlo en partes ("paquetes") de tamaño reducido. El computador receptor debe reconstruir el mensaje original aunque reciba las partes desordenadas. Para apoyar la solución de este problema se dispone de las sgtes clases:

```
class Paquete{
    public int numero;
    //numero que identifica la parte del mensaje (1, 2, ...)
    public String informacion;
    //informacion de la parte del mensaje
}
class Mensaje{
    //representacion: lista enlazada de objetos de clase Paquete
    ...
    public void recibir(Paquete p){...}
    //recibe paquete y lo agrega a lista enlazada(si está repetido
    //lo ignora)
    public String mensaje(int x){
        //devuelve mensaje completo (de x partes) o null si no está
        //completo
        ...
    }
}
```

Escriba la clase Mensaje.

29. Notas

Suponga que existe la clase `Nota` con la siguiente representación:

```
class Nota{  
  
    public double nota;  
    public String apellido;  
    public String nombre;  
  
    ...//constructor y métodos  
}
```

Los profesores de Computación desean llevar un ranking de los alumnos de su sección, según la nota promedio que llevan hasta el momento en el ramo. Los profesores tienen un arreglo con las notas de sus alumnos, sin embargo éste no tiene ningún orden en particular.

Implemente el método `void ordenar(Nota x[])` que ordena el arreglo que recibe como parámetro en orden decreciente de notas. Los alumnos que tengan la misma nota deben quedar ordenados alfabéticamente, primero por apellido y luego por nombre.

Ejemplo: En el arreglo de notas se tienen los objetos definidos por los valores:

```
3,5#Pérez#Juan  
4,5#Aceval#Domingo  
4,0#Carrasco#Marcela  
4,5#Toledo#Rodrigo  
6,0#Gómez#Jorge  
4,0#Carrasco#Armando
```

Luego de aplicar el método `ordenar`, el arreglo queda:

```
6,0#Gómez#Jorge  
4,5#Aceval#Domingo  
4,5#Toledo#Rodrigo  
4,0#Carrasco#Armando  
4,0#Carrasco#Marcela  
3,5#Pérez#Juan
```

Solución:

```
import java.io.*;  
  
class Nota {  
  
    double nota;  
    String apellido;  
    String nombre;  
  
    public Nota(double n, String a, String nom){
```

```
        nota=Math.round(n);
        apellido=a;
        nombre=nom;

    }

    public void imprimir(){

        System.out.println(nota+"-"+apellido+", "+nombre);

    }
}

class UsoNotas{

    public static void ordenarNotasBurbuja(Nota[] a){

        Nota aux;
        for(int i=a.length-1; i>0; i--){

            for(int j=0; j<i; j++){

                if(a[j].nota<a[j+1].nota ||
                    (a[j].nota==a[j+1].nota &&
                     a[j].apellido.compareTo(a[j+1].apellido)>0
                    ||(a[j].apellido.compareTo(a[j+1].apellido)==0 &&
                     a[j].nombre.compareTo(a[j+1].nombre)>0))){

                    aux=a[j];
                    a[j]=a[j+1];
                    a[j+1]=aux;

                }

            }

        }

    }

    public static void main(String args[]) throws IOException{

        BufferedReader b=new BufferedReader(new
                                                FileReader("nombres.txt"));

        int n=0;
        while(b.readLine()!=null) n++;

        b.close();

        Nota notas[]=new Nota[n];
        String nom[]=new String[notas.length];
        String ap[]=new String[notas.length];

        b=new BufferedReader(new FileReader("nombres.txt"));
        String linea;
        int i=0;
        while(i<n){
            linea=b.readLine();
            nom[i]=linea.substring(0,linea.indexOf(" "));
            ap[i++]=linea.substring(linea.indexOf(" ")+1);
        }

        b.close();

    }

}
```

```
        for(i=0; i<notas.length; i++){
            notas[i]=new Nota(Math.random()*7,ap[i],nom[i]);
        }

        for(i=0; i<notas.length; i++)
            notas[i].imprimir();

        System.out.println("ordenado");
        ordenarNotasBurbuja(notas);

        for(i=0; i<notas.length; i++)
            notas[i].imprimir();

    }
}
```

30. Lista de Frecuencias

Implemente una clase Lista que corresponda a lista enlazada simple para almacenar números enteros y su frecuencia. La lista debe estar siempre ordenada de mayor a menor con respecto a los números.

- a) Definir el tipo Nodo y sus métodos.

Solución:

```
class Nodo{
    // variables de instancia
    int numero;
    int frecuencia;
    Nodo sgte;

    // constructor
    public Nodo(int n, Nodo s){
        this.numero = n;
        this.frecuencia = 1;
        this.sgte = s;
    }

    public int obtenerNumero(){
        return this.numero;
    }

    public int obtenerFrecuencia(){
        return this.frecuencia;
    }

    public void fijarFrecuencia(int n){
        this.frecuencia = n;
    }
}
```

- b) Implementar la clase Lista considerando la siguiente representación.

```
class Lista{
    public Nodo primero;

    public Lista() // crea la lista vacia

    public void agregar(int x) // inserta x en el lugar que le corresponde para mantener
    // la lista ordenada. Si x ya esta, incrementa su frecuencia en uno.

    public void eliminar(int x) // decrementa la frecuencia de x, si su frecuencia es 1
    // lo elimina.

    public int frecuencia(int x) // retorna la frecuencia de x. Si no está, retorna 0.
}
```

Solución:

```
class Lista {
    Nodo primero;

    public Lista() {
        primero = null;
    }

    // metodo que ve si un numero esta en la lista
    public boolean esta(int i) {
        for (Nodo aux = primero; aux != null; aux = aux.sgte) {
            if (aux.obtenerNumero() == i)
                return true;
        }
        return false;
    }

    // metodo agregar
    public void agregar(int a) {
        if (primero == null) {
            primero = new Nodo(a, null);
            return;
        }

        if (primero.obtenerNumero() == a) {
            primero.fijarFrecuencia(primero.obtenerFrecuencia() + 1);
            return;
        }

        if (primero.obtenerNumero() < a) {
            primero = new Nodo(a, primero);
            return;
        }

        Nodo aux;
        for (aux = primero; aux.sgte != null; aux = aux.sgte) {
            if (aux.obtenerNumero() == a) {
                aux.fijarFrecuencia(aux.obtenerFrecuencia() + 1);
                return;
            }
            else if (aux.sgte.obtenerNumero() < a) {
                //Nodo nuevo = new Nodo(a, aux.sgte);
                aux.sgte = new Nodo(a, aux.sgte); //nuevo;
                return;
            }
        }

        if(aux.obtenerNumero()==a){
            aux.fijarFrecuencia(aux.obtenerFrecuencia()+ 1);
            return;
        }
        aux.sgte = new Nodo(a, null);
    }

    // metodo eliminar
    public void eliminar(int a) {
        if (esta(a)) {
```

```
        if (primero.obtenerNumero()==a &&
            primero.obtenerFrecuencia() == 1){
            primero = primero.sgte;
            return;
        }

        if (primero.obtenerNumero() == a){
            primero.fijarFrecuencia(primero.obtenerFrecuencia()-
                                   1);
            return;
        }
        Nodo aux;
        for (aux = primero; aux.sgte != null; aux = aux.sgte) {
            if (aux.sgte.obtenerNumero() == a) {
                if(aux.sgte.obtenerFrecuencia()== 1) {
                    // eliminar nodo
                    aux.sgte = aux.sgte.sgte;
                    return;
                }
                else {
                    aux.sgte.fijarFrecuencia(
                        aux.sgte.obtenerFrecuencia() - 1);
                    return;
                }
            }
        }
    }

    public int frecuencia(int a) {
        if (esta(a)) {
            for (Nodo aux=primero; aux!=null; aux=aux.sgte)
                if (aux.obtenerNumero() == a)
                    return aux.obtenerFrecuencia();
        }
        return -1;
    }

    public void imprimir() {
        for (Nodo aux = primero; aux != null; aux = aux.sgte)
            System.out.println(
                "nodo: numero="
                + aux.obtenerNumero()
                + " frecuencia: "
                + aux.obtenerFrecuencia());
    }
}
```


31. Ordenamiento restringido (Propuesto)

a) Programe el método (o procedimiento) **ordenarXcols** que ordena parcialmente un arreglo de strings. El encabezado del método debe ser:

Void ordenarXcols(String[] a, int posini, int posfin, int colini, int colfin)

Este método debe ordenar el arreglo de strings *a* entre los índices *posini* y *posfin*. El ordenamiento se debe hacer ascendentemente y lexicográficamente considerando sólo una parte de cada string. Esta parte está delimitada entre las posiciones *colini* y *colfin* de cada string. Por ejemplo, la siguiente figura muestra, a la izquierda, un arreglo desordenado y, a la derecha, el resultado de invocar 2 veces **ordenarXcols** con distintos argumentos.

b) Con el objeto de promocionar el lanzamiento del sitio www.camisetas.cl, su propietario a donado a la Facultad 20 camisetas alusivas al último eclipse del siglo. La Facultad ha dispuesto que esas camisetas se regalen a los alumnos que obtuvieron las 20 mejores notas en el último control de Astronomía I.

Los resultados del control se encuentran en el archivo *notas.txt*. Cada línea contiene en los 30 primeros caracteres los dos apellidos y el nombre de un alumno, y en dos caracteres su nota en el control en el formato *nn*. Por ejemplo el archivo podría ser:

Fernandez Concha Patricia	51
Perez Urrutia Juan	62
Hernandez Jimenez Gloria	40

...

Escriba un programa que seleccione a los 20 alumnos que obtendrán una camiseta y luego despliegue el resultado en pantalla indicando apellidos y nombres de los 20 alumnos en orden lexicográfico (sin desplegar la nota). Ud debe usar la parte a.- para resolver este problema, sin programar nuevamente otro algoritmo de ordenamiento.

Suponga que el curso tiene a lo mas 200 alumnos. Observe que en el orden lexicográfico, "40" es menor que "41", "49" es menor que "50", "Fernandez Concha Patricia" es menor que "Perez Urrutia Juan", etc. Observe también que el resultado se pide ordenado por apellidos y nombres, no por nota.

32. Olimpiadas

Con motivo de las proximas olimpiadas, el comité olimpico le solicita que haga un estudio sobre la incidencia de la cantidad de habitantes de los paises en su rendimiento deportivo. Para tener un estimador de esta situacion el comité requiere una lista de los paises competidores en que el orden este dado por la razon medallas/habitantes para cada pais.

Ud debe crear un programa que genere el ranking solicitado y lo guarde en un archivo RelacionMH.txt, los datos necesarios para su programa los obtendra de dos archivos, paises.txt y medallas.txt que contienen por linea en los primero 19 caracteres el nombre del pais y en los caracteres restantes la cantidad de habitantes y la cantidad de medallas obtenidas por el pais respectivamente. Los archivos no tienen ningun orden aparente y el numero de paises registrados es de 100.

Indicacion: cree una estructura para representar a cada pais para luego ordenarlos de acuerdo al criterio mencionado.

Solución:

Comenzamos creando la clase rankeado que contendra la informacion referente a un pais. Luego podremos utilizar esta clase para formar arreglos de paises.

```
public class Rankeado {
    public String pais;
    public String medallas;
    public String habitantes;

    public Rankeado(String x, String y, String z){
        pais=x;
        medallas=y;
        habitantes=z;
    }
}
```

```
import java.io.*;

public class Preguntal {

    //se supondra los paises sin medallas aparecen con cero en el
    //ranking
    //definiremos un objeto con el nombre del pais, cantidad de
    //medallas y hab,
    //un arreglo de estos.
    //Luego ordenaremos estos objetos de acuerdo a la relacion
    //señalada
    //finalmente mostraremos en pantalla el resultado del ranking.

    static public void main(String args[]){

        Preguntal gato=new Preguntal();
    }
}
```

```
public Preguntal(){

    try{
        BufferedReader medallas=new BufferedReader(new
            FileReader("Medallas.txt"));
        BufferedReader habitantes=new BufferedReader(new
            FileReader("Habitantes.txt"));

        Rankeado[] lista=new Rankeado[100];
        int i=0;
        String aux=habitantes.readLine();
        while(aux!=null){
            lista[i]=new Rankeado(aux.substring(0,19),
                "",(aux.substring(19)).trim());
            ++i;
            aux=habitantes.readLine();
        }

        aux=medallas.readLine();

        while(aux!=null){
            for(int k=0;k<i;++k){
                if((aux.substring(0,19)).equals(lista[k].pais)){
                    lista[k].medallas=aux.substring(19,aux.length());
                    break;
                }
            }
            aux=medallas.readLine();
        }

        //este metodo compara segun el criterio pedido
        //esta implementado mas abajo
        burbuja(lista,i);

        PrintWriter relacion = new PrintWriter(new
            FileWriter("RelacionMH.txt"));

        System.out.println("\nPregunta 1, Paises Ordenados por
            relacion medallas/habitantes");
        System.out.println("-----");
        for(int k=0;k<i;++k){
            relacion.println(lista[k].pais+
                (Double.parseDouble(lista[k].medallas.trim())/
                Double.parseDouble(lista[k].habitantes.trim())));
        }
        System.out.println();
        relacion.close();

    }catch(Exception e){
        System.out.println("Error al acceder a archivos o algo
similar "+e);
    }
}

public void burbuja(Rankeado[] x, int size){

    Rankeado aux;
    int p=0;
    while(size-p-1!=0){
```

```
        for(int i=0;i<size-1;++i){
            if(compareTo(x[i],x[i+1])<0){
                aux=x[i+1];
                x[i+1]=x[i];
                x[i]=aux;
            }
        }
        ++p;
    }

}

public int compareTo(Rankeado x, Rankeado y){

    double relacionx=Double.parseDouble((x.medallas).trim())/
        Integer.parseInt((x.habitantes).trim());
    double relaciony=Double.parseDouble((y.medallas).trim())/
        Integer.parseInt((y.habitantes).trim());
    if(relacionx>relaciony) return 1;
    if(relacionx<relaciony) return -1;

    return 0;

}

}
```

33. Vectores ligeros.

Con el fin de reducir el espacio de almacenamiento de coordenadas el ejercito diseña la clase VectDiet que tiene la particularidad de utilizar menos espacio de almacenamiento eliminando coordenadas con valor 0. Para mantener la funcionalidad la clase VectDiet mantiene una lista con el indice y valor de la coordenada. Por ejemplo, la coordenada (0,2,5,0,3) en la clase VectDiet sera representada por la lista compuesta de los nodos {1,2}->{2,5}->{4,3} que indicara que el vector esta compuesto por un 2 en la posicion 1, un 5 en la 2, un 3 en la 4 y ceros en las posiciones restantes.

Tomando en cuenta esta clase se le encarga terminar la definicion de la misma agregando operaciones entre vectores de tipo VectDiet, puntualmente se le pide implementar las operaciones producto punto y proyeccion. Recuerde que el producto punto realiza la multiplicación paralela de las componentes del vector y luego retorna la suma (ppunto([1,3,4,2],[1,3,3,1])=1+9+12+2=24) y la proyeccion anula la componente sobre la que se proyecta (proyectar [12,3,5,2] sobre la componente 2 resultara [12,0,5,2]).

Considere la clase VectDiet de la siguiente forma.

```
Class VectDiet{
    Nodo inicio;//primer nodo de la lista enlazada.
    public VectDiet(double [] x){...}
}
```

y los encabezados de las funciones requeridas como:

```
public double ppunto(VectDiet v){...}
public proyectar(int x){...}
```

Donde ppunto aplica el producto punto del vector parametro con el vector que llama a la funcion y proyectar aplica la proyeccion sobre la componente x del vector que llama.

Solución:

```
public double pPunto(VectorDiet q){

    //declaracion de variables auxiliares
    double resultado=0.0;
    Nodo y=inicio;
    Nodo x=q.inicio;

    //un nesting, solo para mostrar como usarlo
    //al operar nunca retrocedo en las listas el algoritmo
    //es en el peor caso de Orden(n+m) siendo estas las
    //dimensiones de cada vector
    nido:
    while(true){
        if(x==null || y==null ||
            ((x.indice==--1 || y.indice==--1) &&
             (x.sig==null || y.sig==null)))
            break nido;
    }
}
```

```
        if(x.indice==y.indice){
            resultado+=x.valor*y.valor;
            y=y.sig;
            x=x.sig;
        }else{
            if(x.indice>y.indice)    y=y.sig;
            else                    x=x.sig;
        }

    }

    return resultado;
}

//suponemos el metodo recibe solo el buen parametro, x>0
//para poder crear un nuevo objeto debo usar el constructor
//osea volver a la representacion de arreglo
public VectorDiet proyectar(int x){
    int size=0;

    //se podra hacer todo en una pasada como pPunto?
    Nodo k=inicio;
    while(k!=null){
        size=k.indice+1;
        k=k.sig;
    }

    if(size==0)
        return new VectorDiet(null);

    System.out.println("Porte del vector= "+size);
    double[] valores=new double[size];

    for(Nodo p=inicio;p!=null;p=p.sig){
        if(p.indice!=-1)
            valores[p.indice]=p.valor;
    }
    valores[x]=0.0;

    return new VectorDiet(valores);
}

}

class Nodo{

    public int indice;
    public double valor;
    public Nodo sig;

    public Nodo(int x,double y,Nodo z){
        indice=x; valor=y; sig=z;
    }

}
```

