30 de Enero de 2009. Fundamentos de Telemática.
Área de Ingeniería Telemática
Departamento de Ingeniería de Sistemas y Automática
Escuela Superior de Ingenieros
Universidad de Sevilla
Notas provisionales: 6 Febrero
Revisión: 9 Febrero (10:00h)

Ejercicio 1 (4 puntos)

La clase abstracta **`AbstractList<E>`** (`public abstract class` **`AbstractList<E>`** `extends AbstractCollection<E> implements List<E>`) proporciona el esqueleto de la interfaz **`List<E>`** minimizando el esfuerzo requerido para implementar esta interfaz y dar soporte a un almacenamiento de datos de "acceso aleatorio" (tal y como las matrices). El acceso de datos secuencial (tal y como las listas enlazadas), debe utilizar **`AbstractSequentialList<E>`**.

Para implementar una lista que no se va a modificar, el programador necesita heredar de esta clase y proporcionar la implementación de los métodos **`get(int index)`** y **`size()`**.

Para implementar una lista modificable, el programador debería además sobrescribir el método **`set(int index, Object element)`** (sino lanzará una **`UnsupportedOperationException`**). Si la lista tiene un tamaño variable el programador debería adicionalmente sobrescribir los métodos **`add(int index, Object element)`** y **`remove(int index)`**.

A diferencia de otras implementaciones abstractas, el programador no tiene que proporcionar una implementación de **`Iterator<E>`**; ya que **`Iterator<E>`** y **`ListIterator<E>`** son implementados por la clase, con los métodos de 'acceso aleatorio' **`get(int index)`**, **`set(int index, E element)`**, **`add(int index, E element)`** y **`remove(int index)`**.

- Implemente una lista no modificable para tipos genéricos a partir de los elementos (objetos) de una matriz bidimensional con el uso de la clase abstracta **`AbstractList<E>`**. Implemente métodos, constructores y atributos necesarios.
- Configure el método **`set(int index, E element)`** que asigne el elemento al índice proporcionado como parámetro.
- Implemente una clase que contenga el método **`main`** que realice la instanciación de un objeto de la clase implementada con elementos de tipo **`String`**. Realice el recorrido mediante el uso de un *iterador* y mediante la implementación de un bloque **`for-each`**.

Ejercicio 2 (2,5 puntos)

Se quiere diseñar las clases **`Hora`** y **`Fecha`** en el paquete **`misUtilides`**, de tal forma que se le añada una aritmética para incrementar el valor de los atributos de estos objetos.
La clase **`Fecha`** mantiene el día, mes y año. (**Nota**: considerar que los años bisiestos son múltiplos de 400, o bien que sean múltiplos de 4 y no divisibles por 100).
La clase **`Hora`** mantiene la hora, minutos y segundos.

- Una hora está representada mediante un entero de 0 a 23.
- Un minuto está representado mediante un entero de 0 a 59 en la forma habitual.
- Un segundo está representado mediante un entero de 0 a 59.

La clase **`Fecha`** mantiene la hora (según el formato de la clase **`Hora`**), y el día, mes y año.

- Un día está representado por un entero entre 1 y 28, 29, 30 o 31 según el mes y el año (bisiesto o no).
- Un mes está representado por un entero entre 1 y 12.
- Un año está representado por un entero.

Implemente la clase **`Hora`** y **`Fecha`** en el paquete **`misUtilides`** con el constructor para la inicialización de los atributos privados necesarios.

Implemente los métodos `boolean` **`equals`**`(Object obj)` y `String` **`toString`**`()` para ambas clases.

Implemente los siguientes métodos de **`Fecha`**:

- `public void incrementaAño()`
- `public void incrementaMes()`
- `public void incrementaDia()`

Implemente los siguientes métodos de **`Hora`**:

- `public void incrementaSegundo()`
- `public void incrementaMinuto()`
- `public void incrementaHora()`

Ejercicio 3 (3,5 puntos)

Los problemas de ***satisfacibilidad booleana*** utilizan el formato CNF (*Conjunctive Normal Form*) para su formulación utilizando un conjunto de *cláusulas*, donde esta cláusula es una opción de un número de variables o de sus negaciones. Si x_i representa variables que pueden tomar solo los valores *true* o *false*, entonces una fórmula en CNF podría ser de la siguiente forma:

$$(x_1 \vee x_3 \vee \dot{x}_4) \wedge (x_4) \wedge (x_2 \vee \dot{x}_3)$$

donde $\vee$ representa la conectividad booleana ***or***, $\wedge$ representa ***and*** y $\dot{x}_i$ representa la negación de x_i.

Dado un conjunto de cláusulas C_1, C_2,..., C_m, con las variables x_1, x_2, ... , x_n el problema de satisfacibilidad es determinar si satisface la fórmula $C_1 \wedge C_2 \wedge ... \wedge C_m$. Es decir, si hay una asignación de valores para las variables de tal forma que la fórmula sea *true*.

Para representar una instancia de estos problemas, se crean ficheros de entrada que contengan toda la información que necesite para definir el problema. Este fichero será un fichero ASCII que consta de dos partes: el preámbulo y las cláusulas.

- El **preámbulo** contiene información de la instancia. Esta información está contenida en líneas. Cada línea comienza con un carácter seguido por un espacio que determina el tipo de línea. Estos tipos son de la siguiente forma:
    1. **Comentario**: Las líneas de comentario dan información del fichero y son ignoradas por los programas. Las líneas de comentarios comienzan con el carácter **c**.
    2. **Línea de problema**: Hay una línea de problema por fichero de entrada. Las líneas de problemas deberían aparecer antes de cualquier otra línea. Para instancias CNF, la línea de problema tiene el siguiente formato.

        `p FORMATO VARIABLES CLAUSULAS`

        El carácter minúscula **`p`** significa que esta es una línea de problema. El campo **`FORMATO`** permite al programa determinar el formato que se espera encontrar y debería contener la palabra "**`cnf`**". El campo **`VARIABLES`** contiene un valor entero especificando el valor de **n** el número de variables en la instancia. El campo **`CLAUSULAS`** contiene un valor entero especificando **m**, el número de clausulas en la instancia. Esta línea debe ser la última del preámbulo.
- Las **cláusulas** aparecen inmediatamente después de la línea de problema. Las variables se suponen que están numeradas desde 1 a **n**. No es necesario que cada variable aparezca en una instancia. Cada cláusula está representada por una secuencia de número, cada uno de ellos separado por un espacio, tabulador o un carácter nueva línea. La versión no negativa de una variable *i* está representada por ***i***; la versión negada está representada por ***-i***. Las cláusulas están terminadas mediante el valor 0, o bien con el final del fichero.

Ejemplo: Usando el ejemplo $(x_1 \vee x_3 \vee x_4) \wedge (x_4) \wedge (x_2 \vee x_3)$ un fichero de entrada posible podría ser:

```
c Ejemplo de formato de fichero CNF
c
p cnf 4 3
1 3 -4 0
4 0 2
-3
```

Puede utilizar la clase **`RandomAccessFile`**, para la lectura en ficheros de acceso aleatorio, instanciando el objeto **`new RandomAccessFile( "cnf0.txt","r" );`** para permitir lectura del fichero `cnf0.txt`.

Un fichero de acceso aleatorio proporciona como una matriz de bytes muy grande almacenada en el sistema de ficheros. Proporcionando una clase de cursor, o índice en esta matriz, llamado *puntero del fichero*, las operaciones de entrada leen bytes comenzando en el puntero del fichero y avanzando los bytes leídos. Si el fichero de acceso aleatorio se crea en modo lectura/escritura, y las operaciones de salida también están disponibles. Las operaciones de salida escriben bytes comenzando en el puntero del fichero y avanzando los bytes que se escriben. Todas las rutinas de lectura en esta clase si encuentran el final del fichero (*end-of-file*) y se realiza alguna operación de lectura lanza **`EOFException`** (que funciona de forma similar a **`IOException`**). Si cualquier byte no puede leerse por cualquier razón diferente a *end-of-file*, se lanza **`IOException`**.

Se desea mantener la información especificada en el fichero en un formato manejable por una aplicación, para ello:

- Implemente las clases que considere oportunas en el lenguaje de programación Java para que mantengan la información de las instancias de los problemas indicados, tomando la información de los ficheros en la forma descrita.
- Implemente una clase con el método **`main`** que instancie los objetos necesarios implementados en el apartado anterior.

java.util
Class AbstractList<E>

java.lang.Object
  └ java.util.AbstractCollection<E>
    └ java.util.AbstractList<E>

All Implemented Interfaces:
Iterable<E>, Collection<E>, List<E>

Direct Known Subclasses:
AbstractSequentialList, ArrayList, Vector

---

public abstract class **AbstractList<E>**
extends AbstractCollection<E>
implements List<E>

This class provides a skeletal implementation of the List interface to minimize the effort required to implement this interface backed by a "random access" data store (such as an array). For sequential access data (such as a linked list), AbstractSequentialList should be used in preference to this class.

To implement an unmodifiable list, the programmer needs only to extend this class and provide implementations for the get(int index) and size() methods.

To implement a modifiable list, the programmer must additionally override the set(int index, Object element) method (which otherwise throws an UnsupportedOperationException. If the list is variable-size the programmer must additionally override the add(int index, Object element) and remove(int index) methods.

The programmer should generally provide a void (no argument) and collection constructor, as per the recommendation in the Collection interface specification.

Unlike the other abstract collection implementations, the programmer does *not* have to provide an iterator implementation; the iterator and list iterator are implemented by this class, on top the "random access" methods: get(int index), set(int index, Object element), set(int index, Object element), add(int index, Object element) and remove(int index).

The documentation for each non-abstract methods in this class describes its implementation in detail. Each of these methods may be overridden if the collection being implemented admits a more efficient implementation.

This class is a member of the Java Collections Framework.

Since:
1.2

See Also:
Collection, List, AbstractSequentialList, AbstractCollection

Field Summary

protected int	`modCount` The number of times this list has been *structurally modified*.

Constructor Summary

protected	`AbstractList()` Sole constructor.

Method Summary	
boolean	add(E o) Appends the specified element to the end of this List (optional operation).
void	add(int index, E element) Inserts the specified element at the specified position in this list (optional operation).
boolean	addAll(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list at the specified position (optional operation).
void	clear() Removes all of the elements from this collection (optional operation).
boolean	equals(Object o) Compares the specified object with this list for equality.
abstract E	get(int index) Returns the element at the specified position in this list.
int	hashCode() Returns the hash code value for this list.
int	indexOf(Object o) Returns the index in this list of the first occurence of the specified element, or -1 if the list does not contain this element.
Iterator<E>	iterator() Returns an iterator over the elements in this list in proper sequence.
int	lastIndexOf(Object o) Returns the index in this list of the last occurence of the specified element, or -1 if the list does not contain this element.
ListIterator<E>	listIterator() Returns an iterator of the elements in this list (in proper sequence).
ListIterator<E>	listIterator(int index) Returns a list iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
E	remove(int index) Removes the element at the specified position in this list (optional operation).
protected void	removeRange(int fromIndex, int toIndex) Removes from this list all of the elements whose index is between fromIndex, inclusive, and toIndex, exclusive.
E	set(int index, E element) Replaces the element at the specified position in this list with the specified element (optional operation).
List<E>	subList(int fromIndex, int toIndex) Returns a view of the portion of this list between fromIndex, inclusive, and toIndex, exclusive.

Methods inherited from class java.util.AbstractCollection
addAll, contains, containsAll, isEmpty, remove, removeAll, retainAll, size, toArray, toArray, toString

Methods inherited from class java.lang.Object
clone, finalize, getClass, notify, notifyAll, wait, wait, wait

Methods inherited from interface java.util.List
addAll, contains, containsAll, isEmpty, remove, removeAll, retainAll, size, toArray, toArray

java.io
Class RandomAccessFile

java.lang.Object

└ java.io.RandomAccessFile

All Implemented Interfaces:

Closeable, DataInput, DataOutput

public class **RandomAccessFile**

extends Object implements DataOutput, DataInput, Closeable

Instances of this class support both reading and writing to a random access file. A random access file behaves like a large array of bytes stored in the file system. There is a kind of cursor, or index into the implied array, called the *file pointer*; input operations read bytes starting at the file pointer and advance the file pointer past the bytes read. If the random access file is created in read/write mode, then output operations are also available; output operations write bytes starting at the file pointer and advance the file pointer past the bytes written. Output operations that write past the current end of the implied array cause the array to be extended. The file pointer can be read by the getFilePointer method and set by the seek method.

It is generally true of all the reading routines in this class that if end-of-file is reached before the desired number of bytes has been read, an EOFException (which is a kind of IOException) is thrown. If any byte cannot be read for any reason other than end-of-file, an IOException other than EOFException is thrown. In particular, an IOException may be thrown if the stream has been closed.

Since: JDK1.0

Constructor Summary
`RandomAccessFile`(File file, String mode) Creates a random access file stream to read from, and optionally to write to, the file specified by the File argument.
`RandomAccessFile`(String name, String mode) Creates a random access file stream to read from, and optionally to write to, a file with the specified name.

Method Summary	
void	`close`() Closes this random access file stream and releases any system resources associated with the stream.
FileChannel	`getChannel`() Returns the unique FileChannel object associated with this file.
FileDescriptor	`getFD`() Returns the opaque file descriptor object associated with this stream.
long	`getFilePointer`() Returns the current offset in this file.
long	`length`() Returns the length of this file.
int	`read`() Reads a byte of data from this file.
int	`read`(byte[] b) Reads up to b.length bytes of data from this file into an array of bytes.
int	`read`(byte[] b, int off, int len) Reads up to len bytes of data from this file into an array of bytes.
boolean	`readBoolean`() Reads a boolean from this file.
byte	`readByte`() Reads a signed eight-bit value from this file.
char	`readChar`() Reads a Unicode character from this file.

double	readDouble() Reads a double from this file.
float	readFloat() Reads a float from this file.
void	readFully(byte[] b) Reads b.length bytes from this file into the byte array, starting at the current file pointer.
void	readFully(byte[] b, int off, int len) Reads exactly len bytes from this file into the byte array, starting at the current file pointer.
int	readInt() Reads a signed 32-bit integer from this file.
String	readLine() Reads the next line of text from this file.
long	readLong() Reads a signed 64-bit integer from this file.
short	readShort() Reads a signed 16-bit number from this file.
int	readUnsignedByte() Reads an unsigned eight-bit number from this file.
int	readUnsignedShort() Reads an unsigned 16-bit number from this file.
String	readUTF() Reads in a string from this file.
void	seek(long pos) Sets the file-pointer offset, measured from the beginning of this file, at which the next read or write occurs.
void	setLength(long newLength) Sets the length of this file.
int	skipBytes(int n) Attempts to skip over n bytes of input discarding the skipped bytes.
void	write(byte[] b) Writes b.length bytes from the specified byte array to this file, starting at the current file pointer.
void	write(byte[] b, int off, int len) Writes len bytes from the specified byte array starting at offset off to this file.
void	write(int b) Writes the specified byte to this file.
void	writeBoolean(boolean v) Writes a boolean to the file as a one-byte value.
void	writeByte(int v) Writes a byte to the file as a one-byte value.
void	writeBytes(String s) Writes the string to the file as a sequence of bytes.
void	writeChar(int v) Writes a char to the file as a two-byte value, high byte first.
void	writeChars(String s) Writes a string to the file as a sequence of characters.
void	writeDouble(double v) Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the file as an eight-byte quantity, high byte first.
void	writeFloat(float v) Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the file as a four-byte quantity, high byte first.
void	writeInt(int v)

	Writes an int to the file as four bytes, high byte first.
void	writeLong(long v) Writes a long to the file as eight bytes, high byte first.
void	writeShort(int v) Writes a short to the file as two bytes, high byte first.
void	writeUTF(String str) Writes a string to the file using modified UTF-8 encoding in a machine-independent manner.

Solución Problema 1:

```
import java.util.AbstractList;
import java.util.Collection;
import java.util.Iterator;

public class Feb09<E> extends AbstractList<E>{

    private E matriz[][] = null;
    private int  size    = 0;

    public Feb09(E [][] matriz){
        this.matriz = matriz;
        for(int i = 0;i < matriz.length;i++)
            size += matriz[i].length;

    }

    //Dos constructores
    public Feb09(){
    }
    public <E> Feb09(Collection<E> c){
        Object matrix[][] = null;
        matrix = new Object[1][];
        matriz[0] = c.toArray(matriz[0]);
        size = matriz[0].length;
    }

    public E get(int index){
        int desp = 0;

        for(int i = 0; i < matriz.length ; i++){
            if(index < desp + matriz[i].length)
                return matriz[i][index - desp];
            desp += matriz[i].length;
        }
        throw new ArrayIndexOutOfBoundsException(index);
    }

    public E set (int index, E element){
        int desp = 0;
        for(int i = 0; i < matriz.length; i++){
            if(index < desp + matriz[i].length)
                return (matriz[i][index-desp]= element);
            desp += matriz[i].length;
        }
        throw new ArrayIndexOutOfBoundsException(index);
    }

    public int size(){
        return size;
    }
}


class ppal{
    public static void main(String args[]){
        String [][]matriz = {{"uno"},{"dos0","dos1"},{"tres0","tres1","tres2"}};
        Feb09<String> d = new Feb09<String>(matriz);

        System.out.println("Extraccion elemento a elemento con get(indice) "  );
        for(int i = 0; i < d.size(); i++){
            System.out.println(d.get(i));
        }
        System.out.println("............"  );
        System.out.println("Recorrido con un iterador "  );
        Iterator<String> it = d.iterator();
        while(it.hasNext())
            System.out.println(it.next());
        System.out.println("............"  );
        System.out.println("Recorrido con un bucle for-each "  );
        for (Object o : d)
            System.out.println(o);
    }
}
```

Solución Problema 2:

```
package misUtilidades;
import  misUtilidades.Ctes;

public class Fecha {

    private int dia;
    private int mes;
    private int año;

    public Fecha(int dia, int mes, int año){
        this.dia  = dia;
        this.mes  = mes;
        this.año  = año;
    }

    private boolean compruebaFecha(){
    /*Devuelve true si el formato es correcto*/
        boolean local = true;

        if(this.mes < 1 || this.mes > 12){
            local = false;
        }

        if(this.dia < 1){
            local = false;
        }

        switch(this.mes){
            case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                if(this.dia > 31)
                    local = false; //mes de 31 dias
                break;
            case 4: case 6: case 9: case 11:
                if(this.dia > 30)
                    local = false; //mes de 30 dias
                break;
            case 2: ///comprobar bisiesto
                if ((0 == this.año % 400) ||
                    ((0 == this.año % 4) && (0 != this.año % 100))){
                        if(this.dia > 29)
                            local = false;
                }else{         //año no bisiesto (28 dias)
                    if(this.dia > 28)
                        local = false;
                }
                break;
            default:
                local = false;
        }

        return local;
    }

    public void incrementaDia(){
        this.dia++;
        switch(this.mes){
            case 1: case 3:case 5:case 7:case 8:case 10:case 12:
                this.actualizaDiaSegunMes(31); //mes de 31 dias
                break;
            case 4: case 6:case 9:case 11:
                this.actualizaDiaSegunMes(30);//mes de 30 dias
                break;
            case 2: ///comprobar bisiesto
                if ((0 == this.año % 400) ||
                    ((0 == this.año % 4) && (0 != this.año % 100))){
                    this.actualizaDiaSegunMes(29);//bisiesto 29 dias
                }else{         //año no bisiesto (28 dias)
                    this.actualizaDiaSegunMes(28);
                }
                break;
            default:
                throw new IllegalArgumentException();
        }
    }

    private void actualizaDiaSegunMes(int numDiasMes){
        if(this.dia == (numDiasMes + 1)){
            this.dia = 1;
```

```
            this.incrementaMes();
        }
    }

    public void incrementaMes(){
        this.mes++;
        if(this.mes == 13){
            this.mes = 1;
            this.incrementaAño();
        }
    }

    public void incrementaAño(){
        this.año++;
    }
    public String toString(){
        return " " + this.dia + "/"  + this.mes + "/"
                   + this.año;
    }
    public boolean equals(Object obj) {
        if( (obj!= null) && (obj instanceof Fecha ) ) {
            Fecha temp = (Fecha )obj;
            return( this.dia == temp.dia  && this.mes == temp.mes && this.año == temp.año);
        }else {
            return( false );
        }
    }
}
```

```
package misUtilidades;
import misUtilidades.Ctes;

public class Hora {
    private int hora;
    private int minuto;
    private int segundo;
    private Fecha fecha;

    public Hora(int hora, int min, int seg,Fecha fecha){
         this.hora    = hora;
         this.minuto  = min;
         this.segundo = seg;
         this.fecha   = fecha;
    }
    public int getHora(){return this.hora;}
    public int getMinuto(){return this.minuto;}
    public int getSegundo(){return this.segundo;}
    public Fecha getFecha(){return this.fecha;}

    public void incrementaSegundo(){
        this.segundo++;
        if(this.segundo == Ctes.SEGUNDO_RANGO_MAX){
            this.segundo = 0;
            this.incrementaMinuto();
        }
    }

    public void incrementaMinuto(){
        this.minuto++;
        if(this.minuto == Ctes.MINUTO_RANGO_MAX){
            this.minuto = 0;
            this.incrementaHora();
        }
    }

    public void incrementaHora(){
        this.hora++;
        if(this.hora == Ctes.HORA_RANGO_MAX){
            this.hora = 0;
            this.fecha.incrementaDia();
        }
    }
    public String toString(){
        String local = null;
        StringBuffer shora = new StringBuffer(10);
        if(this.hora < 10)
            shora.append('0');
        shora.append(this.hora);
        shora.append(":");
        if(this.minuto < 10)
            shora.append('0');
        shora.append(this.minuto );
        shora.append(":");
        if(this.segundo < 10)
            shora.append('0');
        shora.append(this.segundo );
        local = new String(shora);
        local += this.fecha.toString();
        return local;
    }
    public boolean equals(Object obj) {
        if( (obj!= null) && (obj instanceof Hora ) ) {
            Fecha temp = ((Hora)obj).getFecha();
            return( this.hora         == ((Hora)obj).getHora() &&
                    this.minuto       == ((Hora)obj).getMinuto() &&
                    this.getSegundo() == ((Hora)obj).getSegundo()
                    && this.fecha.equals(temp));
        }else {
            return( false );
        }
    }
}
```

Solución Problema 3:

```
import java.io.RandomAccessFile;
```

```
import java.io.IOException;
import java.io.FileNotFoundException;
import java.util.ArrayList;

class Clausula{
    private int numVar = 0;
    private int numClausulas = 0;
    private String p = null;
    private String []clausulas;

    Clausula(){
    }
    void setP(String p){
        this.p = p;
    }
    void setClausulas(String []clausulas){
        this.clausulas = clausulas;
        numClausulas   = this.clausulas.length;
    }
    void setNumVar(int numVar){
        this.numVar = numVar;
    }
}
class Preambulo{
    private String s = null;
    private int numLineasPreambulo;

    Preambulo(int numLineasPreambulo, String cadena){
        this.numLineasPreambulo = numLineasPreambulo;
        this.s                  = cadena;
    }
    String getCadena(){
        return s;
    }
}

public class CNFInstance{
    private RandomAccessFile miRAFile;
    private int variables;
    private int clausulas;
    private int       pos   =   0;
    private byte[]    bytes = null;
    private Preambulo pre   = null;
    private Clausula  cla   = null;


    CNFInstance (String file){
        try{
            // Abrimos el fichero de acceso aleatorio
            miRAFile = new RandomAccessFile( "cnf0.txt","r" );
            this.variables = 0;
            this.clausulas = 0;

            bytes = new byte[(int)miRAFile.length()];
            miRAFile.read(bytes,0,bytes.length);
            miRAFile.close();

        }catch(FileNotFoundException e){
             System.out.println("Fichero no encontrado: " +e);
        }catch(IOException e){
             System.out.println("Problema i/o: " +e);
        }
    }

    public int getNumVariables(){
        return this.variables;
    }

    public int getNumClausulas(){
        return this.clausulas;
    }
    public void analiza(){
        int pos = 0;
        //Preambulo

        analizaPreambulo();
        analizaClausulas();
    }
```

```
    private void analizaPreambulo(){
        int numLineasPreambulo = 0;
        while (pos < bytes.length && bytes[pos++] == 'c'){
            while(bytes[pos++] != '\n')
                ;
            numLineasPreambulo++;
            System.out.println("linea de preambulo"+numLineasPreambulo);

        }
        pos--;
        this.pre = new Preambulo (numLineasPreambulo ,new  String(bytes, 0, pos) );
        System.out.print(this.pre.getCadena());
    }

    private void analizaClausulas(){
        int numVar1;
        int numVar2;
        System.out.println(new  String(bytes, pos, bytes.length - pos) );
        int inicio = pos;

        while(bytes[pos++]!='\n')
            ;
        System.out.println(new  String(bytes, inicio,  pos - inicio ) );
        this.cla = new Clausula();
        numVar1 = bytes[inicio + 6]-'0';
        numVar2 = bytes[inicio + 8]-'0';

        this.cla.setP(new  String(bytes, inicio,  pos - inicio ));
        this.cla.setNumVar(numVar1);
        String [] string = new String[numVar2];
        for(int i = 0; i <numVar2 ; i++ ){
            inicio = pos;
            while(pos < bytes.length && bytes[pos++]!= '0')
                ;
            string[i] = new  String(bytes, inicio,  pos - inicio );
            System.out.println(string[i]);
        }
        this.cla.setClausulas( string );
    }
}

class Log {
    public static void main( String args[] ) throws IOException {
        CNFInstance cnf = new CNFInstance ( "cnf0.txt" );
        cnf.analiza();

    }
}
```