ÁREA DE LENGUAJES Y SISTEMAS INFORMÁTICOS

DEPARTAMENTO DE INFORMÁTICA
E INGENIERÍA DE SISTEMAS

ESCUELA DE INGENIERÍA Y ARQUITECTURA

FUNDAMENTOS DE INFORMÁTICA

1ª CONVOCATORIA (16-junio-2011)

Ejercicio 1 (50% normal, 25% extraordinaria).

Implementar en Java la clase `NaturalGrande` para representar números grandes mayores o iguales a cero. Esta clase tendrá un único campo de tipo ArrayList<Integer>: la lista de los dígitos que componen el número grande. La posición cero de la lista representa las unidades del número; la posición uno, las decenas; la dos, las centenas; etc.

La clase `NaturalGrande` tendrá que tener los siguientes constructores y métodos

```
/**
* Constructor que dado un entero crea un natural grande con las cifras del
* argumento.
* @param n si n >= 0, se construye entero con las cifras de n
*/
public NaturalGrande(int n) {15% problema}

/**
* Si i es mayor que cero, añade todos los dígitos que aparecen en el entero i.
* Por ejemplo, si tenemos el número 478 y añadimos el 235, el número final será
* 235478, es decir, las cifras se añaden por la parte más significativa.
* @param i dígitos a introducir.
*/
public void añadir(int i){30% problema}

/**
* Suma dos naturales grandes
* @param ng numero grande
* @return la suma del objeto que posee el método y ng
*/
public NaturalGrande sumar(NaturalGrande ng){40% problema}

/**
* Transforma a String el natural grande
* @return un String representando al número
*/
public String toString(){15% problema}
```

Observaciones:

- Notar que n % 10 devuelve el dígito menos significativo de n
- Notar que n / 10 elimina el dígito menos significativo de n

```
import java.util.*;
/**
 * Clase para representar números enteros positivos grandes
*/
public class NaturalGrande {
    private ArrayList<Integer> digitos = new ArrayList<Integer>();

    /**
     * Constructor que dado un entero crea un natual grande con las cifras del argumento
     * @param n si n >= 0, se construye entero con las cifras de n
     */
    public NaturalGrande(int n) {
        añadir(n);
    }
    /**
     * Si i es mayor que cero, añade todos los dígitos que aparecen en el entero i
     * @param i dígitos a introducir.
     */
    public void añadir(int i) {
        while(i > 0) {
            digitos.add(i % 10);
            i /= 10;
        }
    }
    /**
     * Suma dos naturales grandes
     * @param ng numero grande
     * @return la suma del objeto que posee el método y ng
     */
    public NaturalGrande sumar(NaturalGrande ng) {
        NaturalGrande res = new NaturalGrande(0);
        int i = 0;
        int llevo = 0;
        for (i = 0; i < digitos.size() && i < ng.digitos.size(); i++) {
            int sumaDigito = digitos.get(i) + ng.digitos.get(i) + llevo;
            res.digitos.add(sumaDigito%10);
            llevo = sumaDigito / 10;
        }
        for (; i < digitos.size(); i++) {
            int sumaDigito = digitos.get(i) + llevo;
            res.digitos.add(sumaDigito%10);
            llevo = sumaDigito / 10;
        }
        for (;i < ng.digitos.size(); i++) {
            int sumaDigito = ng.digitos.get(i) + llevo;
            res.digitos.add(sumaDigito%10);
            llevo = sumaDigito / 10;
        }
        if (llevo != 0) {
            res.digitos.add(llevo);
        }
        return res;
    }
    /**
     * Transforma a String el natural grande
     * @return un String representando al número
     */
    public String toString() {
        String res = "";
        for (int i = digitos.size()-1; i >= 0; i--) {
            res += digitos.get(i);
        }
        if (res.equals("")) {
            return "0";
        } else {
            return res;
        }
    }

}
```

Ejercicio 2 (50% normal, 25% extraordinaria).

Los ficheros con formato bmp de 256 colores tienen las siguientes características:

- los primeros 1078 bytes son diferentes cabeceras, donde entre otros datos, se almacena la anchura y altura de la imagen;
- los siguientes bytes hasta el final del fichero son la información de la imagen, un byte por cada píxel.

Por ejemplo, si disponemos de una imagen con tamaño de 80x40 píxeles, el fichero bmp de 256 colores tendría 1078 bytes de cabeceras más 80x40=3200 bytes de información de la imagen.

Implementar una clase en Java ImagenBMP256 para rotar 180º una imagen bmp a 256 colores de tamaño 80x40 con:

- campos para guardar el nombre del fichero de donde se carga la imagen bmp, un array de 1078 bytes para guardar la cabecera de un fichero bmp y una matriz 40x80 de bytes.
- Un constructor para cargar los datos de la imagen dado un nombre de fichero. (*5% problema*)
- *void setImagen(String nom)*: método para cargar los datos de una imagen bmp del fichero con nombre *nom*. (*40% problema*)
- *void volcarImagen(String nom)*: escribe en el fichero de nombre *nom* los datos de la imagen guardados en los campos, primero los 1078 bytes de la cabecera y después los datos de la matriz de bytes. (*40% problema*)
- *void rotar180()*: rota 180º la imagen guardada en los campos. Notar que para rotar la imagen solo es necesario cambiar los datos de la matriz 40x80 donde se guardan los bytes. Hay que "rotar 180º" la matriz, esto es, el elemento *(i,j)* de la matriz pasará a ser el *(40-i-1, 80-j-1)*. (*15% problema*)

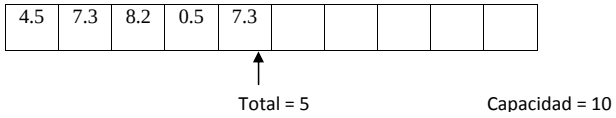
```java
import java.io.*;
/**
 * Clase para rotar imagenes bmp 256 colores 80x40.
 * Los ficheros bmp 256 colores 80x40 se componen: 1078 bytes cabeceras,
 * 80*40 = 3200 bytes de información de la imagen.
*/
public class ImagenBMP256 {
    public static final int ANCHO = 80;
    public static final int ALTO = 40;
    public static final int TAM_CABECERAS = 1078;
    private byte [] cabeceras = new byte[TAM_CABECERAS];
    private byte [][] imagen = new byte[ALTO][ANCHO];
    private String nombreFichero;
    /**
     * Crea una imagen bmp ANCHOxALTO de un fichero de nombre dado.
     * @param nom nombre del fichero para leer bytes de imagen.
     */
    public ImagenBMP256(String nom){
        setImagen(nom);
    }
    /**
     * Carga una imagen bmp ANCHOxALTO del fichero de nombre dado
     * @param nom nombre del fichero a cargar
     */
    public void setImagen(String nom) {
        nombreFichero = nom;
        try {
            FileInputStream in = new FileInputStream(nom);
            in.read(cabeceras);   //leer cabeceras
            //guardar datos en matriz de bytes
            for (int i = 0; i < imagen.length; i++) {
                in.read(imagen[i]);
            }
            in.close();
        } catch (FileNotFoundException e) {
            System.out.println("Fichero no existe");
        } catch (IOException e) {
            System.out.println("Error I/O");
        }
    }
    /**
     * Rota la imagen 180 grados.
     */
    public void rotar180() {
        byte [][] rotada = new byte[ALTO][ANCHO];
        for (int i = 0; i < imagen.length; i++) {
            for (int j = 0; j < imagen[i].length; j++) {
                rotada[i][j] = imagen[ALTO - i - 1][ANCHO - j - 1];
            }
        }
        imagen = rotada;
    }
    /**
     * Guarda en un fichero de nombre dado los datos de la imagen
     * @param nomFichero nombre del fichero donde almacenar los datos de la imagen.
     */
    public void volcarImagen(String nomFichero) {
        try {
            FileOutputStream out = new FileOutputStream(nomFichero);
            out.write(cabeceras);
            for (int i = 0; i < imagen.length; i++) {
                out.write(imagen[i]);
            }
            out.close();
        } catch (IOException e) {
            System.out.println("Error I/O");
        }
    }

}
```

Ejercicio 3 (25% extraordinaria).

1. Crear la clase ListaReales para representar listas de números reales con las siguientes especificaciones:

   - La clase está formado por dos campos: un arrray de reales y un entero que representa el total de elementos significativos de la lista, esto es, los elementos significativos de la lista serán los números que estén entre la posición 0 y el total de elementos menos uno. La lista tendrá una capacidad máxima de elementos que no se podrá sobrepasar.



   - La clase tendrá un constructor con argumento la capacidad máxima de la lista
   - Métodos para observar la lista: la capacidad máxima, el total de elementos de la lista y la componente i-ésima.
   - Métodos para modificar la lista: modificar el elemento i-ésimo de la lista, añadir un elemento (al final de la lista), insertar un nuevo elemento en la posición i-ésima, y borrar el elemento i-ésimo.

(*70% problema*)

2. Implementar la clase ListaRealesOrdenada para representar listas de números reales ordenados crecientemente. Los constructores y métodos serán los mismos que para la clase ListaReales salvo que modificar el elemento i-ésimo e insertar un elemento no deberán hacer nada en caso de no respetar el orden de la lista. El método añadir elemento a la lista deberá introducirlo ordenadamente.

(*30% problema*)

Observaciones:

Notar que al insertar un elemento en la lista, el resto de elementos se tiene que desplazar una posición a la derecha, y al borrar el elemento i-ésimo, los elementos siguientes se tienen que desplazar un posición a la izquierda.

```java
/**
 * Clase para representar listas de reales con un máximo de elementos a guardar.
 *
 */
public class ListaReales {
    private double [] datos;
    private int total;

    /**
     * Crea una lista vacía para guardar a lo sumo una cantidad dada de reales
     * @param max cantidad máxima de reales que se pueden guardar en el lista
     */
    public ListaReales(int max){
        datos = new double[max];
        total = 0;
    }

    /**
     * Devuelve el total de elementos guardados en la lista.
     * @return total de elementos guardados en la lista
     */
    public int getTotal() {
        return total;
    }
    /**
     * Retorna la capacidad máxima de la lista.
     * @return capacidad máxima de la lista
     */
    public int getCapacidad(){
        return datos.length;
    }

    /**
     * Retorna el elemento i-ésimo de la lista.
     * @param i i-ésimo elemento a visitar
     * @return elemento i-ésimo
     */
    public double get(int i) {
        return datos[i];
    }
    /**
     * Modifica la posición i-ésima de la lista con un real dado. Si no se cumple que
     * 0<=i<=total elementos de la lista, no se modifica la lista.
     * @param i posición a modificar
     * @param r elemento nuevo en posición i-ésima
     * @return ciertos, si se ha modificado la lista
     */
    public boolean set(int i, double r) {
        boolean sePuede = (0 <= i) && (i <= getTotal());
        if (sePuede) {
            datos[i] = r;
        }
        return sePuede;
    }
    /**
     * Retorna si la lista está vacía.
     * @return cierto, si la lista está vacía
     */
    public boolean estaVacia() {
        return total == 0;
    }
    /**
     * Añade un real al final de la lista.
     * Retorna si se ha podido introducir o no el real.
     * @param r real a introducir en la lista
     * @return cierto si se puede introducir, falso si no se puede introducir por
     * exceder la capacidad de la lista.
     */
```

```java
    public boolean add(double r) {
        boolean sePuede = getTotal() < getCapacidad();
        if (sePuede) {
            total++;
            datos[getTotal()-1] = r;
        }
        return sePuede;
    }

    /**
     * Inserta en el posición i-ésima un número real.
     * Todos los siguientes elementos se mueven una posición hacia la derecha. Si no se
     * cumple que 0<=i<=total elementos de la lista + 1 o la lista está llena,
     * no se modifica la lista.
     * @param i posición a insertar elemento
     * @param r elemento a insertar
     * @return cierto, si se ha modificado la lista
     */
    public boolean insertar(int i, double r) {
        boolean sePuede = (0 <= i) && (i <= getTotal()) && (getTotal() < getCapacidad());
        if (sePuede) {
            for (int j = getTotal()-1; j >= i; j--) {
                datos[j+1] = datos[j];
            }
            datos[i] = r;
            total++;
        }
        return sePuede;
    }

    /**
     * Elimina el real situado en la posición i-ésima. Todos los siguiente elementos se
     * mueven una posición hacia la izquierda. Si no se cumple que
     * 0<=i<=total elementos de la lista o la lista no está vacía,
     * no se modifica la lista.
     * @param i elemento i-ésimo de la lista a eliminar
     * @return cierto si se ha modificado la lista
     */
    public boolean remove(int i) {
        boolean sePuede = (!estaVacia()) && (0 <= i) && (i < getTotal());
        if (sePuede) {
            //agrupo los elementos al inicio del array
            for (int j = i + 1; j < getTotal(); j++) {
                datos[j-1] = datos[j];
            }
            total--;
        }
        return sePuede;
    }


}
```

```java
/**
 * Clase para guardar una lista ordenada (de menor a mayor) de reales con un máximo de
 * elementos
 */
public class ListaRealesOrdenada extends ListaReales {

    /**
     * Crea una lista ordenada vacía para guardar a lo sumo una cantidad dada de reales
     * @param max cantidad máxima de reales que se pueden guardar en el lista ordenada
     */
    public ListaRealesOrdenada(int max) {
        super(max);
    }

    /**
     * Redefinición del método set. Modifica la lista si respeta la ordenación
     * @param i posición a cambiar
     * @param r nuevo elemento
     * @return cierto, si se puede modificar la lista ordenada.
     */
    public boolean set(int i, double r) {
        if (0 <= i && i < getTotal()) {
            if (i == 0) {
                if (r <= get(i+1) && getTotal() != 1) {
                    return super.set(i, r);
                } else {
                    return super.set(i, r);
                }
            } else if (i == getTotal() - 1 && get(i-1) <= r) {
                return super.set(i, r);
            } else if (get(i-1) <= r && r <= get(i+1)) {
                return super.set(i, r);
            } else {
                return false;
            }
        }
        return false;
    }

    /**
     * Redefinición del método insertar. Modifica la lista si respeta la ordenación.
     * @param i posción donde insertar
     * @param r nuevo elemento
     * @return cierto, si al insertar el elemento, respeta la ordenación de la lista
     */
    public boolean insertar(int i, double r) {
        return false;
    }

    /**
     * Añade un real ordenadamente. Retorna si se ha podido introducir el real.
     * @param r real a introducir en la lista
     * @return cierto si se puede introducir, falso si no se puede introducir por
     * exceder la capacidad de la lista.
     */
    public boolean add(double r) {
        boolean sePuede = getTotal() < getCapacidad();
        if (sePuede) {
            int pos = buscar(r);
            super.insertar(pos, r);
        }
        return sePuede;
    }

    /**
     * Buscar la posición dentro la lista donde debería aparecer un elemento.
     * @param r elemento a buscar
     * @return posición dentro de la lista donde está o debería estar r
     */
    private int buscar(double r) {
        int inf = 0;
```

```
        int sup = getTotal() - 1;
        int med = 0;
        if (!estaVacia()) {
            while (inf != sup) {
                med = (inf + sup) / 2;
                if (get(med) <= r) {
                    sup = med;
                } else {
                    inf = med + 1;
                }
            }
            if ((inf == 0 || inf == getTotal() -1) && get(inf) < r) {
                inf++;
            }
        }
        return inf;
    }
}
```

Ejercicio 4 (25% extraordinaria).

Crear por derivación de la clase `File` del paquete `java.io` la clase `MyFile` para añadirle un método que permita mezclar ordenadamente (orden alfabético) las palabras que aparezcan en dos ficheros de texto ordenados. Supondremos que todos los ficheros que consideremos solo tienen palabras con caracteres en mayúsculas.

Observación:

- Para saber si un String es menor o mayor que otro (orden alfabético) se puede usar el método de la clase `String`, `int compareTo(String s),` que devuelve un número menor que cero si el objeto que posee el método es menor que `s`, cero si es igual, o un número mayor que cero si es mayor.
- No se podrán usar arrays o listas para resolver el problema.

```java
import java.io.*;
import java.util.*;
/**
 * Clase para mezclar ordenadamente dos ficheros de texto
*/
public class MyFile extends File {
    public MyFile(String nom) {
        super(nom);
    }
    /**
     * Mezcla ordenadamente el fichero propietario del método con otro dado en un fichero
     * de nombre nuevo
     * @param f2 fichero a mezclar con propietario del método
     * @param nuevo nombre de fichero con datos mezclados ordenadamente
     */
    public void merge(File f2, String nuevo) {
        try {
            Scanner scan1 = new Scanner(this);
            Scanner scan2 = new Scanner(f2);
            PrintWriter sal = new PrintWriter(nuevo);
            boolean tocaPrimero = trae, tocaSegundo = true;
            boolean fin1 = false, fin2 = false;
            String pal1 = null, pal2 = null;
            if (scan1.hasNext() && scan2.hasNext()) {
                while (!fin1 && !fin2) {
                    if (tocaPrimero) {
                        if (scan1.hasNext()) {
                            pal1 = scan1.next();
                        } else {
                            fin1 = true;
                        }
                    }
                    if (tocaSegundo) {
                        if (scan2.hasNext()) {
                            pal2 = scan2.next();
                        } else {
                            fin2 = true;
                        }
                    }
                    tocaPrimero = pal1.compareTo(pal2) < 0;
                    tocaSegundo = !tocaPrimero;

                    if (tocaPrimero && !fin1) {
                        sal.println(pal1);
                    } else if (tocaSegundo && !fin2) {
                        sal.println(pal2);
                    }
                }
                if (!fin1) {
                    sal.println(pal1);
                } else {
                    sal.println(pal2);
                }
            }
            while (scan1.hasNext()) {
                sal.println(scan1.next());
            }
            while (scan2.hasNext()) {
                sal.println(scan2.next());
            }
            sal.close();
        } catch (FileNotFoundException e) {
            System.out.println("No existe fichero");
        } catch (IOException e) {
            System.out.println("Error I/O");
        }
    }
}
```