Examen de Programación 1. Viernes 31/enero/2014

- Disponer un documento de identificación con fotografía sobre la mesa.
- Comenzar a resolver cada problema del examen en una hoja de papel diferente. Escribir en cada hoja de papel nombre y apellidos y utilizar, en su caso, ambas caras de la hoja.
- Tiempo máximo para realizar este examen: 3 horas

Problema 1º (2.5 puntos)

Se pide diseñar el código Java del método **cerificar** que se especifica a continuación. Debe entregarse el método pedido completo, es decir, incluyendo su especificación y, en su caso, acompañado de los métodos auxiliares que se hubieran desarrollado para completar su diseño.

```
/**
 * Pre: n>0
 * Post: Devuelve un entero que al ser escrito en base 10 presenta las cifras no nulas
 *       de [n] y en el mismo orden y, entre cada par de cifras no nulas consecutivas,
 *       el número devuelto presenta un cero. Ejemplos:
 *          cerificar (7) = 7
 *          cerificar (17) = 107
 *          cerificar (113) = 10103
 *          cerificar (170) = 107
 *          cerificar (1203) = 10203
 *          cerificar (1000203) = 10203
 *          cerificar (912000) = 90102
 *          cerificar (91002000) = 90102
 */
public static int cerificar (int n)
```

Problema 2º (4.0 puntos)

Un objeto de la clase *Alumno*, ubicada en el *package cap3*, gestiona la información de un estudiante universitario: el número NIP de identificación universitaria del estudiante y su nombre completo (nombre y apellidos). El constructor *Alumno(nip,nombre)* permite crear un objeto que gestiona el NIP (*nip*) y nombre completo (*nombre*) del alumno. El método *nip()* devuelve el número NIP del alumno y el método *nombre()* devuelve la referencia a un objeto de la clase *String* que gestiona su nombre completo.

cap3.Alumno
+ Alumno (nip: **int**; nombre: **String**) + nip(): **int** + nombre(): **String**

Se desea almacenar de forma permanente la información de todos los alumnos de un centro universitario, de una titulación o de una asignatura en un ***fichero binario de alumnos*** con la siguiente estructura:

```
<fichero_de_alumnos> ::=  <alumno> { <alumno> }
<alumno> ::= <nip> <nombre>
<nip> ::= int
<nombre> ::= <numCaracteres> <secuenciaCaracteres>
<numCaracteres> ::= int
<secuenciaCaracteres> ::=   { char }
```

Observación: La longitud de la secuencia <***secuenciaCaracteres***> de datos de tipo **char** de un nombre coincide con el valor de <***numCaracteres***>.

Se pide diseñar el código Java de los métodos *guardarAlumnos(nombre, T)* y *leerAlumnos(nombre)* que se especifican a continuación. Deben entregarse los métodos pedidos completos, es decir, incluyendo sus especificaciones y, en su caso, acompañados de los métodos auxiliares que se hubieran desarrollado para completar sus diseños. En cambio, el código de la clase **cap3.Alumno** no ha de ser ni desarrollado ni entregado.

```
/**
 * Pre: [nombre] ha de definir un nombre válido para un fichero que va a ser creado.
 *      T!=null y la tabla [T] no almacena referencias null
 * Post: Crea un fichero denominado [nombre] con la estructura de un fichero binario
 *      de alumnos y almacena en él los datos de todos los alumnos referenciados
 *      desde la tabla [T]. Devuelve true si la operación se ha completado con éxito
 *      y false en caso contrario
 */
public static boolean guardarAlumnos (String nombre, Alumno[] T)

/**
 * Pre: [nombre] es la referencia al nombre de un fichero binario de alumnos
 * Post: Crea una tabla de referencias a objetos Alumno cada uno de los cuales gestiona
 *      la información de uno de los alumnos cuyos datos constan en el fichero binario
 *      de alumnos [nombre] y devuelve la referencia a la tabla creada. En el caso de
 *      que no pueda ser completada con éxito la lectura del fichero binario de alumnos
 *      [nombre] entonces devuelve una referencia null.
 */
public static Alumno[] leerAlumnos (String nombre)
```

Problema 3º (3.5 puntos)

Un objeto de la clase ***Reloj*** gestiona la hora que marca un reloj. Ofrece dos métodos constructores que permiten definir la hora inicial del reloj gestionado por el objeto creado. El método **adelantar(segs)** permite modificar la hora de un reloj y el método **toString()** permite consultar la hora de un reloj en forma textual. La clase ***Reloj*** incluye también el método estático **estanOrdenados(TR)** que permite comprobar si una colección de relojes está ordenada cronológicamente o no lo está.

Reloj
+ Reloj (hh: **int**; mm: **int**; ss: **int**) + Reloj (hh: **int**) + adelantar(segs: **int**) + estanOrdenados(TR: **Reloj[]**): **boolean** + toString(): **String**

Una invocación ***new Reloj(12,40,30)*** crea un objeto de la clase **Reloj** que gestiona un reloj cuya hora inicial son las 12:40:30.

Una invocación ***new Reloj(22)*** al segundo de los constructores crea un objeto de la clase **Reloj** que gestiona un reloj cuya hora inicial son las 22:00:00.

La hora de un reloj ha de estar comprendida entre las 00:00:00 y las 23:59:59.

Asumamos que **miReloj** es una referencia a un objeto de la clase **Reloj** cuya hora actual es 23:59:08. Una invocación **miReloj.adelantar(20)** modifica la hora de dicho reloj adelantándola 20 segundos, por lo que la nueva hora del reloj pasa a ser las 23:59:28. Una segunda invocación **miReloj.adelantar(120)** modifica la hora de dicho reloj adelantándola 120 segundos, por lo que la nueva hora del reloj pasa a ser las 00:01:28. Una invocación **miReloj.toString()** devuelve la referencia a un String de 8 caracteres que corresponde a la hora marcada por el reloj, es decir, devuelve la referencia a un String que almacena la secuencia `"00:01:28"`.

El método estático **estanOrdenados(TR)** se aplica a una tabla de referencias no nulas a objetos de la clase **Reloj**. Devuelve true si y solo si todos los relojes referenciados desde la tabla están ordenados cronológicamente según se explica a continuación:

- están ordenados cronológicamente si cada uno de ellos, el reloj T[i], marca una hora igual o más temprana que el siguiente reloj, T[i+1]
- o también están ordenados cronológicamente si cada uno de ellos, el reloj T[i], marca una hora igual o más tardía que el siguiente reloj, T[i+1]

Se pide diseñar el código Java de la clase **Reloj**, incluyendo la definición de datos de cada objeto y el código de los cinco métodos públicos citados en el diagrama UML de la clase, incluyendo sus especificaciones y, en su caso, acompañados de los métodos auxiliares que se hubieran desarrollado para completar sus diseños.

Una solución del problema 1º

```
/**
 * Pre: n>0
 * Post: Devuelve un entero que al ser escrito en base 10 presenta las cifras no nulas
 *       de [n] y en el mismo orden y, entre cada par de cifras no nulas consecutivas,
 *       el número devuelto presenta un cero. Ejemplos:
 *          cerificar (7) = 7
 *          cerificar (17) = 107
 *          cerificar (113) = 10103
 *          cerificar (170) = 107
 *          cerificar (1203) = 10203
 *          cerificar (1000203) = 10203
 *          cerificar (912000) = 90102
 *          cerificar (91002000) = 90102
 */
public static int cerificar (int n) {
    /*
     * Asigna a [resultado] un valor nulo y dispone [pot] señalando la cifra de las unidades
     */
    int resultado = 0;
    int pot = 1;
    /*
     * Completa la construcción del número pedido sobre la varable [resultado]
     */
    while (n!=0) {
        if (n%10>0) {
            /*
             * Incorpora a [resultado] la siguiente cifra no nula de [n] en la posición
             * determinada por el valor de [pot] y dispone a [pot] señalando una cifra
             * dos posiciones a la izquierda
             */
            resultado = (n%10)*pot + resultado;
            pot = 100*pot;
        }
        /*
         * Elimina la cifra menos significativa de [n]
         */
        n = n/10;
    }
    /*
     * Devuelve el número pedido
     */
    return resultado;
}
```

Una solución del problema 2º

```
/**
 * Pre: [nombre] ha de definir un nombre válido para un fichero que va a ser creado.
 *      T!=null y la tabla [T] no almacena referencias null
 * Post: Crea un fichero denominado [nombre] con la estructura de un fichero binario
 *      de alumnos y almacena en él los datos de todos los alumnos referenciados
 *      desde la tabla [T]. Devuelve true si la operación se ha completado con éxito
 *      y false en caso contrario
 */
public static boolean guardarAlumnos (String nombre, Alumno[] T) {
    try {
        FileOutputStream f = new FileOutputStream(nombre);
        DataOutputStream fa = new DataOutputStream(f);
        for (int i = 0; i<T.length; ++i) {
            fa.writeInt (T[i].nip ());
            int longitud = T[i].nombre().length ();
            fa.writeInt ( longitud );
            for (int j = 0; j<longitud; ++j) {
                fa.writeChar(T[i].nombre().charAt(j ));
            }
        }
        fa.close ();
        return true;
    }
    catch (IOException e) {
        return false;
    }
}

/**
 * Pre: [nombre] es la referencia al nombre de un fichero binario de alumnos
 * Post: Devuelve el número de alumnos almacenados en el fichero de alumnos
 *      [nombre]. En el caso de que no pueda ser completada con éxito la
 *      lectura del fichero devuelve un valor negativo.
 */
private static int contarAlumnos (String nombre) {
    final int LECTURA_ERRONEA = -10;
    DataInputStream fa = null;
    int cuenta = 0;
    try {
        FileInputStream f = new FileInputStream(nombre);
        fa = new DataInputStream(f);
        while (true) {
            /*
             * Lee los datos correspondientes al siguiente alumno almacenado
             * en el fichero e incrementa en una unidad el valor de [cuenta]
             */
            fa.readInt ();
            int longitud = fa.readInt ();
            for (int j = 0; j<longitud; ++j) {
                fa.readChar ();
            }
            ++cuenta;
        }
    }
    catch (EOFException e) {
        try {
            fa.close ();
        }
        catch (IOException e1) {
```

```
                return LECTURA_ERRONEA;
            }
            return cuenta;
        }
        catch (IOException e) {
            return LECTURA_ERRONEA;
        }
    }

    /**
     * Pre: [nombre] es la referencia al nombre de un fichero binario de alumnos
     * Post: Crea una tabla de referencias a objetos Alumno cada uno de los cuales gestiona
     *       la información de uno de los alumnos cuyos datos constan en el fichero binario
     *       de alumnos [nombre] y devuelve la referencia a la tabla creada. En el caso de
     *       que no pueda ser completada con éxito la lectura del fichero binario de alumnos
     *       [nombre] entonces devuelve una referencia null.
     */
    public static Alumno[] leerAlumnos (String nombre) {
        DataInputStream fa = null;
        Alumno[] T = new Alumno[contarAlumnos(nombre)];
        try {
            FileInputStream f = new FileInputStream(nombre);
            fa = new DataInputStream(f);
            int cuenta = 0;
            while (true) {
                int nip = fa.readInt ();
                int longitud = fa.readInt ();
                String alumno = "";
                for (int j = 0; j<longitud; ++j) {
                    alumno = alumno + fa.readChar ();
                }
                T[cuenta] = new Alumno(nip,alumno);
                ++cuenta;
            }
        }
        catch (EOFException e) {
            try {
                fa.close ();
            }
            catch (IOException e1) {
                return null;
            }
            return T;
        }
        catch (IOException e) {
            return null;
        }
    }
```

Una solución del problema 3º

```
public class Reloj {
    /**
     * Un objeto de la clase Reloj gestiona la hora señalada por un reloj
     */

    /*
     * Hora, minuto y segundo que marca el reloj que satisfacen :
     *     hora>=0, hora<24
     *     minuto>=0, minuto<60
     *     segundo>=0, segundo<60
     */
    private int hora, minuto, segundo;

    /**
     * Pre: hh>=0, hh<24, mm>=0, mm<60, ss>=0, ss<60
     * Post: El reloj creado marca la hora hh:mm:ss
     */
    public Reloj (int hh, int mm, int ss) {
        hora = hh;
        minuto = mm;
        segundo = ss;
    }

    /**
     * Pre: hh>=0, hh<24
     * Post: El reloj creado marca la hora hh:00:00
     */
    public Reloj (int hh) {
        hora = hh;
        minuto = 0;
        segundo = 0;
    }

    /**
     * Pre: ---
     * Post: Devuelve un entero positivo que al ser escrito en base 10 tiene la
     *       forma hhmmss que representan la hora hh:mm:ss de este reloj
     */
    private int laHora () {
        return 10000*hora + 100*minuto + segundo;
    }

    /**
     * Pre: segs>=0
     * Post: Modifica la hora actual de este reloj adelantándola un número
     *       de segundos igual a [segs]
     */
    public void adelantar (int segs) {
        segundo = segundo + segs;
        if (segundo>=60) {
            minuto = minuto + segundo/60;
            segundo = segundo % 60;
            if (minuto>=60) {
                hora = hora + minuto/60;
                minuto = minuto % 60;
                if (hora>=24) {
                    hora = hora % 24;
                }
            }
```

```
    }
}

/**
 * Pre: ---
 * Post: Devuelve la referencia a un String que representan la hora de este reloj
 *       con ocho caracteres de la forma "hh:mm:ss"
 */
public String toString () {
    String resultado = "";
    if (hora<10) {
        resultado = resultado + "0" + hora + ":";
    }
    else {
        resultado = resultado + hora + ":";
    }
    if (minuto<10) {
        resultado = resultado + "0" + minuto + ":";
    }
    else {
        resultado = resultado+ minuto + ":";
    }
    if (segundo<10) {
        resultado = resultado + "0" + segundo;
    }
    else {
        resultado = resultado+ segundo;
    }
    return resultado ;
}

/**
 * Pre: T!=null y T no almacena ninguna referencia null
 * Post: Devuelve true si y sólo si las horas que marcan los relojes referenciados
 *       desde T está ordenados cronológicamente de forma que o bien la hora del
 *       primero es igual o más temprana que la del segundo, la de segundo igual o
 *       más temprana que la del tercero y así sucesivamente, o bien la hora del
 *       primero es igual o más tardía que la del segundo, la de segundo igual o más
 *       tardía que la del tercero y así sucesivamente.
 */
public static boolean estanOrdenados(Reloj[] T) {
    boolean loEstan = true;
    /*
     * Comprueba si los relojes están ordenados empezando por la hora más temprana
     * avazando hacia horas igual o más tardías
     */
    int indice = 0;
    while (loEstan && indice<T.length-2) {
        if (T[indice].laHora()>T[indice+1].laHora()) {
            loEstan = false;
        }
        else {
            ++indice;
        }
    }
    if (!loEstan) {
        /*
         * Comprueba si los relojes están ordenados empezando por la hora más tardía
         * avazando hacia horas igual o más tempranas
         */
        loEstan = true;
```

```
            indice = 0;
            while (loEstan && indice<T.length-2) {
                if (T[indice].laHora()<T[indice+1].laHora()) {
                    loEstan = false;
                }
                else {
                    ++indice;
                }
            }
        }
        return loEstan;
    }

}
```