Examen de Programación 1. Jueves 31/ENE/2013

- 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.
- Tiempo para realizar el examen: 3 horas

Problema 1º (2.0 puntos)

Un objeto de la clase *Calificacion*, ubicada en el *package examenFebrero*, gestiona la información de una calificación: el número NIP de identificación universitaria del estudiante y su calificación en la asignatura expresada como un entero entre 0 y 100. El método *nip()* devuelve el número NIP del estudiante calificado y el método *nota()* devuelve el valor de su calificación.

examenFebrero.Calificacion
+ Calificacion (estudiante: **int**; nota: **int**) + nip(): **int** + nota(): **int**

Se pide diseñar el código Java del método **mejores** que se especifica a continuación. En su diseño se puede contar con los recursos públicos que ofrece la clase *Calificacion*, que no debe ser programada.

```
/**
 * Pre: n>0 y clase!=null y clase.length>=n y no hay ninguna referencia con valor null en la
 *      tabla [clase]
 * Post: Devuelve la referencia a una nueva tabla que almacena las referencias a las [n]
 *       calificaciones más altas entre las almacenadas en la tabla referenciada por [clase].
 *       No altera el contenido de la tabla referenciada por [clase]
 */
public static Calificacion [] mejores ( Calificacion [] clase, int n)
```

Problema 2º (2.0 puntos)

Se pide diseñar también el código Java del método **agrupar(*Calificación[]*)** que se especifica a continuación. Se valorará especialmente la eficiencia del método diseñado. Se desaconseja programarlo como un algoritmo de ordenación por resultar una solución poco eficiente.

```
/**
 * Pre: notas!=null, notas.length>0 y no hay ninguna referencia con valor null en la
 *      tabla [notas]
 * Post: Reordena permutando los datos de [notas] de forma que todas las calificaciones
 *       correspondientes a aprobados (notas iguales o superiores a 50) precedan en ella
 *       a las correspondientes a suspensos (notas inferiores a 50)
 */
public static void agrupar ( Calificacion [] notas)
```

Problema 3º (3.0 puntos)

Diseñar una clase de nombre *Fecha*, ubicada en el *package examenFebrero*, que gestione la información de una fecha del calendario y cuyo diagrama UML es el siguiente:

examenFebrero.Fecha
+ Fecha (dia: **int**; mes: **int**; año: **int**) + Fecha (unaFecha: **int**) + dia(): **int** + mes(): **int** + año(): **int** + esValida(): **boolean** + esPrevia (Fecha otra): **boolean** + toString (): **String**

La especificación de todos los métodos públicos de la clase **examenFebrero.Fecha** se detalla a continuación.

```
/**
 * Pre: dia/mes/año definen una fecha que puede ser una fecha válida del calendario o puede
 *      no serlo
 * Post: Los parámetros [dia], [mes] y [año] definen los valores del día, mes y año,
 *       respectivamente, de esta fecha
 */
public Fecha (int dia, int mes, int año)

/**
 * Pre: El parámetro [unaFecha] es un entero que, escrito en base 10, consta de ocho
 *      cifras: aaaammdd.
 * Post: Las cuatro más significativas del parámetro, [aaaa], definen el año de esta fecha,
 *       las dos cifras siguientes, [mm] definen el mes y las dos cifras menos significativas,
 *       [dd], definen el día de esta fecha. Por ejemplo, un valor de [unaFecha] igual a
 *       20130131 define esta fecha como el 31 de Enero de 2013
 */
public Fecha (int unaFecha)

   /**
 * Post: Devuelve el valor del día correspondiente a esta fecha
 */
public int dia()

/**
 * Post: Devuelve el valor del mes correspondiente a esta fecha
 */
public int mes()

/**
 * Post: Devuelve el valor del año correspondiente a esta fecha
 */
public int año()

/**
 * Pre: ---
 * Post: Devuelve true si y solo si esta fecha corresponde a una fecha existente en
 *       el calendario igual o posterior al 1 de Enero de 1600
 */
public boolean esValida()
```

```
/**
 * Pre:  this.esValida() y otra.esValida()
 * Post: Devuelve true si y solo si esta fecha es anterior a la fecha representada
 *       por [otra]
 */
public boolean esPrevia (Fecha otra)

/**
 * Pre:  this.esValida()
 * Post: Devuelve la referencia a un String que describe esta fecha de la forma
 *       [dia] de [mes] de [año]. Por ejemplo: 31 de Enero de 2013
 */
public String toString ()
```

Se pide diseñar la clase **examenFebrero.Fecha** incluyendo sus **atributos** y el **código de cinco de sus ocho métodos**: los dos métodos constructores y los métodos públicos ***esValida()***, ***esPrevia(Fecha)*** y ***toString()***. En cambio, no se pide el código de los métodos públicos ***dia()***, ***mes()*** y ***año()***.

En el diseño de esta clase no está permitido importar ninguna otra clase.

Información complementaria: Son bisietos exclusivamente los años divisibles por 4 a excepción de los años múltiplo de 100 que no sean también múltiplo de 400.

Problema 4º (3.0 puntos)

Las notas de cada uno de los examenes se almacenan en un fichero binario con la siguiente estructura. Este tipo de ficheros binarios los denominaremos ficheros de notas.

```
<fichero_de_notas> ::=  <calificación> { <calificación> }
<calificación> ::= <nip>  <nota>
<nip> ::= int
<nota> ::= int
```

Se pide diseñar el código Java del método **separar(*String*, *String*, *String*)** que se especifica a continuación.

```
/**
 * Pre: [nombreFNotas] es el nombre de un fichero de notas
 * Post: Devuelve la nota media de las calificaciones de los alumnos cuyas notas se almacenan
 *       en el fichero de notas [nombreFNotas] y crea dos nuevos ficheros de notas de nombres
 *       [nombreFAprobados] y [nombreFSuspensos]. El primero de ellos almacena una copia de
 *       las calificaciones de todos los alumnos aprobados (su nota es igual o mayor que 50) y
 *       en el segundo de ellos se almacena una copia de las calificaciones de todos los alumnos
 *       suspendidos (su nota es menor que 50). En caso de producirse alguna excepción en el
 *       trabajo con cualquiera de los tres ficheros el método se limitará a devolver un valor
 *       negativo, sin emitir ningún mensaje.
 */
public static double separar (String nombreFNotas, String nombreFAprobados,
                              String nombreFSuspensos)
```

Una solución del problema 1º

```
/**
 * Pre: n>0 y clase!=null y clase.length>=n y no hay ninguna referencia con valor null en la
 *      tabla [clase]
 * Post: Devuelve la referencia a una nueva tabla que almacena las referencias a las [n]
 *       calificaciones más altas entre las almacenadas en la tabla referenciada por [clase].
 *       No altera el contenido de la tabla referenciada por [clase]
 */
public static Calificacion [] mejores ( Calificacion [] clase, int n) {
    /*
     * Duplica en [copia] la tabla [clase]
     */
    Calificacion [] copia = new Calificacion [clase.length];
    for (int i=0; i<clase.length; ++i) { copia[i] = clase[i]; }
    /*
     * Crea una tabla capaz de almacenar [n] referencias a objetos
     * [Calificación]
     */
    Calificacion [] T = new Calificacion [n];
    for (int i=0; i<n; ++i) {
        /*
         * Asigna al elemento i-ésimo de la tabla [T] la referencia a la
         * calificación más alta de la subtabla copia[i,copia.length-1]
         */
        int iMax = i;
        for (int j=i+1; j<copia.length; ++j) {
            if (copia[j].nota()>copia[iMax].nota()) {
                iMax = j;
            }
        }
        T[i] = copia[iMax];
        copia[iMax] = copia[i];
    }
    /*
     * Devuelve la referencia a la tabla [T]
     */
    return T;
}
```

Una solución del problema 2º

```
/**
 * Pre: notas!=null, notas.length>0 y no hay ninguna referencia con valor null en la
 *      tabla [notas]
 * Post: Reordena permutando los datos de [notas] de forma que todas las calificaciones
 *       correspondientes a aprobados (notas iguales o superiores a 50) precedan en ella
 *       a las correspondientes a suspensos (notas inferiores a 50)
 */
public static void agrupar ( Calificacion [] notas ) {
    final int NOTA_MINIMA = 50;
    int inf = 0, sup = notas.length-1;
    while ( inf<sup) {
        if ( notas[ inf ].nota()>=NOTA_MINIMA) {
            ++inf;
        }
        else if ( notas[sup].nota()<NOTA_MINIMA) {
            --sup;
        }
        else {
            Calificacion aux = notas[ inf ];
            notas[ inf ] = notas[sup]; notas[sup] = aux;
            ++inf; --sup;
        }
    }
}
```

Una solución del problema 3º

```
package examenes.febrero2013;

/**
 * Cada objeto de esta clase permite gestionar la información que describe una
 * fecha del calendario
 */
public class Fecha {

    private int dia;                /* Día de esta fecha */
    private int mes;                /* Mes de esta fecha */
    private int año;                /* Año de esta fecha */

    /**
     * Pre: dia/mes/año definen una fecha que puede ser una fecha válida del calendario o puede
     *      no serlo
     * Post: Los parámetros [dia], [mes] y [año] definen los valores del día, mes y año,
     *       respectivamente, de esta fecha
     */
    public Fecha(int dia, int mes, int año) {
        this.dia = dia;
        this.mes = mes;
        this.año = año;
    }

    /**
     * Pre: El parámetro [unaFecha] es un entero que, escrito en base 10, consta de ocho
     *      cifras: aaaammdd.
     * Post: Las cuatro más significativas del parámetro, [aaaa], definen el año de esta fecha,
     *       las dos cifras siguientes, [mm] definen el mes y las dos cifras menos significativas,
     *       [dd], definen el día de esta fecha. Por ejemplo, un valor de [unaFecha] igual a
     *       20130131 define esta fecha como el 31 de Enero de 2013
     */
    public Fecha(int unaFecha) {
        dia = unaFecha%100;
        mes = (unaFecha/100)%100;
        año = unaFecha/10000;
    }

    /**
     * Post: Devuelve el valor del día correspondiente a esta fecha
     */
    public int dia() {
        /* No se pedía el código de este método */
    }

    /**
     * Post: Devuelve el valor del mes correspondiente a esta fecha
     */
    public int mes() {
        /* No se pedía el código de este método */
    }


    /**
     * Post: Devuelve el valor del año correspondiente a esta fecha
     */
    public int año() {
        /* No se pedía el código de este método */
    }
```

```
    /**
     * Pre: ---
     * Post: Devuelve true si y sólo si el año de esta fecha es bisiesto
     */
    private boolean esBisiesto (int año) {
        return (año%4==0) && ((año%100!=0) || (año%400==0));
    }

    /**
     * Pre: mes>=1, mes<=12, año>=1600
     * Post: Devuelve el número de días del mes [mes] del año [año]
     */
    private int diasMes (int mes, int año) {
        final int[] DIAS = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

        if ((mes==2) && esBisiesto(año)) {
            return 29;
        }
        else {
            return DIAS[mes-1];
        }
    }

    /**
     * Pre: ---
     * Post: Devuelve true si y sólo si esta fecha corresponde a una fecha del
     *       calendario igual o posterior al 1 de Enero de 1600
     */
    public boolean esValida () {
        return (dia>=1) && (dia<=diasMes(mes,año))
                && (mes>=1) && (mes<=12) && (año>=1600);
    }

    /**
     * Pre: this.esValida() y otra.esValida()
     * Post: Devuelve true si y sólo si esta fecha es anterior a
     *       la fecha representada por [otra]
     */
    public boolean esPrevia (Fecha otra) {
        int f_otra = otra.dia() + 100* otra.mes() + 100000*otra.año();
        int f_esta = dia + 100*mes + 100000*año;
        return f_esta < f_otra;
    }

    /**
     * Pre: mes>=1, mes<=12
     * Post: Devuelve la referencia a un String que describe esta fecha de la forma
     *       [dia] de [Mes] de [año]. Por ejemplo: 6 de Septiembre de 2009
     */
    public String toString () {
        final String[] NOMBRE_MES = { "Enero", "Febrero", "Marzo", "Abril", "Mayo", "Junio",
            "Julio", "Agosto", "Septiembre", "Octubre", "Noviembre", "Diciembre"};
       return dia + " de " + NOMBRE_MES[mes-1] + " de " + año;
    }

}
```

Una solución del problema 4º

```
/**
 * Pre: [nombreFNotas] es el nombre de un fichero de notas
 * Post: Devuelve la nota media de las calificaciones de los alumnos cuyas notas
 *       se almacenan en el fichero de notas [nombreFNotas] y crea dos nuevos
 *       ficheros de notas de nombres [nombreFAprobados] y [nombreFSuspensos].
 *       En el primero de ellos almacena una copia de las notas de todos los
 *       alumnos aprobados (su calificación es igual o mayor que 50) y en el
 *       segundo de ellos se almacena una copia de las notas de todos los alumnos
 *       suspendidos (su calificación es menor que 50). En caso de producirse
 *       alguna excepción en el trabajo con cualquiera de los tres ficheros el
 *       método se limitará a devolver un valor negativo, sin emitir ningún mensaje.
 */
public static double separar (String nombreFNotas, String NombreFAprobados,
                              String nombreFSuspensos) {
    final int MINIMO_PARA_APROBAR = 50;
    DataInputStream fNotas = null;
    DataOutputStream fAprobados = null;
    DataOutputStream fSuspensos = null;;
    int sumaNotas = 0, numAlumnos = 0;
    try {
        fNotas = new DataInputStream (new FileInputStream(nombreFNotas));
        fAprobados = new DataOutputStream (new FileOutputStream(NombreFAprobados));
        fSuspensos = new DataOutputStream (new FileOutputStream(nombreFSuspensos));
        while (true) {
            int nip = fNotas.readInt ();
            int nota = fNotas.readInt ();
            sumaNotas = sumaNotas + nota;  ++numAlumnos;
            if (nota>=MINIMO_PARA_APROBAR) {
                fAprobados.writeInt (nip); fAprobados.writeInt (nota);
            }
            else {
                fSuspensos.writeInt (nip); fSuspensos.writeInt (nota);
            }
        }
    }
    catch (EOFException e) {
        try {
            fNotas.close ();  fAprobados.close ();  fSuspensos.close ();
        }
        catch (Exception e1) {
            return -1.0;
        }
        return (double) sumaNotas / numAlumnos;
    }
    catch (Exception e) {
        return -1.0;
    }
}
```