Examen escrito de Programación 1

Escuela de Ingeniería y Arquitectura
Departamento de Informática e Ingeniería de Sistemas

12 de septiembre de 2013

- Disponer sobre la mesa en lugar visible un **documento de identificación** provisto de fotografía.
- Escribir **nombre y dos apellidos** en cada una de las hojas de papel que haya sobre la mesa.
- Comenzar a resolver cada una de las partes del examen **en una hoja diferente** para facilitar su corrección por profesores diferentes.
- El tiempo total previsto para realizar el examen es de **dos horas y media**.
- No está permitido consultar libros ni apuntes, excepto los dos documentos facilitados por los profesores de la asignatura: *Un breve resumen del lenguaje Java* y *Documentación de las clases Java utilizadas en la asignatura*.
- En todos los métodos a diseñar en este examen se valorará de forma destacada la adecuada especificación de los mismos.

Problema 1.º (2,5 puntos)

Se debe diseñar una clase Java, de nombre `Producto` cuyos objetos permitan representar productos de consumo que pueden ser perecederos o no. El diagrama UML de dicha clase se muestra a continuación, seguido de explicaciones sobre sus atributos y métodos:

examenSeptiembre.Producto
- nombre: String - calidad: **int** - esPerecedero: **boolean** - diasParaCaducar: **int**
+ Producto(nombre: String; calidad: **int**; esPerecedero: **boolean**; diasParaCaducar: **int**) + nombre(): String + esPerecedero(): **boolean** + diasParaCaducar(): **int** + calidad(): **int** + actualizar(): **void** + haCaducado(): **boolean**

Todos los productos se caracterizan por un nombre determinado, representado a través de un objeto de la clase `String`, y una determinada calidad, representada por un entero no negativo. Los productos perecederos están caracterizados además por un entero positivo que representa el número de días que les falta para caducar. Los productos no perecederos también cuentan con este último atributo, aunque su valor puede ser ignorado. El paso de los días disminuye la calidad de los productos perecederos, que pierden 6 unidades de calidad cada 5 días, hasta que alcanzan su fecha de caducidad, momento a partir del cual su calidad es igual a 0. La calidad de los productos no perecederos no sufre variación aunque pasen los días.

Además de un método constructor y de los métodos de acceso a sus atributos (`nombre()`, `esPerecedero()`, `diasParaCaducar()` y `calidad()`), los objetos de la clase `Producto` dispondrán de un método público denominado `actualizar()` que, cuando es invocado sobre un determinado producto, representa el hecho de que ha pasado un día para el producto y, en consecuencia, en el caso de que se trate de un producto perecedero, actualiza el número de días que le quedan para que caduque y el valor de su calidad. En primer lugar, disminuye en un día el número de días que le falta para caducar y, a continuación, rebaja en dos o una unidades de calidad en función de que el nuevo número de días hasta que caduque sea o no múltiplo de 5, respectivamente. En todo caso, la calidad de un producto tras haber sido actualizado nunca es inferior a cero y si el producto ha caducado (es decir, le quedan 0 días o menos para caducar), su calidad es igual a 0. La calidad de los productos no perecederos no varía nunca, incluso si se invoca su método `actualizar()`.

Los objetos de la clase `Producto` tendrán, además, un método denominado `haCaducado()` que indicará si el objeto sobre el que se invoca ha caducado o no. Es decir, devolverá **`true`** si y solo si se trata de un producto perecedero al que le quedan 0 días para caducar o menos.

Se pide escribir el contenido completo del fichero de código fuente Java «`Producto.java`», ubicado en el paquete `examenSeptiembre`, cuyos métodos públicos tengan un comporta-

miento acorde con la especificación proporcionada en este enunciado. El código debe estar adecuadamente documentado.

Problema 2.º (1 punto)

Utilizando objetos de la clase `Producto` definida en el problema anterior, se pide completar el siguiente método:

```
/**
 * Pre:  «inventario» != null e «inventario».length > 0
 * Post: Ha devuelto la referencia al producto de mayor calidad de entre
 *       todos los productos referenciados en la tabla tabla «inventario».
 *       Las referencias a los productos de la tabla «inventario» no han
 *       sufrido cambios.
 */
public static Producto mejor(Producto[] inventario)
```

Problema 3.º (2,5 puntos)

Utilizando objetos de la clase `Producto` definida en el problema 1.º, se pide completar el siguiente método:

```
/**
 * Pre:  «inventario» != null
 * Post: Ha devuelto la referencia a una nueva tabla de productos en la
 *       que están todas referencias a los productos no caducados de la
 *       tabla «inventario» y solo esas. Las referencias a los
 *       productos de la tabla «inventario» no han sufrido cambios.
 */
public static Producto[] noCaducados(Producto[] inventario) {
```

Problema 4.º (4 puntos)

La información de productos representables como objetos de la clase `Producto` definida en el problema 1.º se almacena, por motivos históricos y de mantenimiento, en dos tipos de ficheros: ficheros de texto y ficheros binarios.

Por una parte, en los ficheros de texto, cada línea representa un producto en el que, separados por comas, se encuentran los atributos de cada producto en este orden: nombre, calidad, si es perecedero y el número de días que faltan para que caduque. En los nombres de los productos no hay comas.

La estructura de los ficheros de texto de productos es la siguiente:

```
<fichero_productos_txt> ::= {<producto_txt>}
<producto_txt> ::= <nombre_txt> "," <calidad_txt> ","
                    <perecedero_txt> "," <caducidad_txt>
<nombre_txt> ::= literal_string (sin comas)
<calidad_txt> ::= literal_entero
<perecedero_txt> ::= literal_booleano
<caducidad_txt> ::= literal_entero
```

Por otra parte, en los ficheros binarios, por cada producto la información almacenada es, en el orden que se indica a continuación, un dato **int** que representa la longitud del nombre nombre del producto, seguido de tantos datos de tipo **char** como la longitud del nombre indique, que representan cada uno de los caracteres que forman el nombre; un dato de tipo **int** que representa la calidad del producto; un dato de tipo **boolean** que representa si el producto si es perecedero o no y un dato de tipo **int** que representa el número de días que faltan para que el producto caduque.

La estructura de los ficheros binarios de productos es la siguiente:

```
<fichero_productos_bin> ::= {<producto_bin>}
<producto_bin> ::= <nombre_bin><calidad_bin><perecedero_bin>
                   <caducidad_bin>
<nombre_bin> ::= int {char}
<calidad_bin> ::= int
<perecedero_bin> ::= boolean
<caducidad_bin> ::= int
```

Se pide completar el siguiente método que, partiendo de la información de un lote de productos almacenada en un fichero de texto, actualiza sus calidades y días para caducar y almacena la información de los productos actualizados como un fichero binario:

```
/**
 * Pre:  «nombreFicheroTexto» es distinto de null y hace referencia a un
 *       fichero de texto existente con la estructura establecida en el
 *       enunciado; «nombreFicheroBinario» es distinto de null y hace
 *       referencia a un fichero que no existe o que, si existe, puede ser
 *       reescrito.
 * Post: Ha leído el contenido del fichero «nombreFicheroTexto»,
 *       interpretando cada línea del mismo como un producto que, tras
 *       haber sido actualizado de acuerdo con las especificaciones de la
 *       clase producto establecidas en el enunciado del problema 1.º, ha
 *       sido escrito como dato binario en el fichero
 *       «nombreFicheroBinario».
 */
public static void convertirDeTextoABinario(File nombreFicheroTexto,
                                            File nombreFicheroBinario)
```

Al diseño de este método se le debe aplicar la metodología de diseño descendente utilizada en el curso, plasmada en el desarrollo de, al menos, un par de métodos privados auxiliares.

Solución al problema 1.º

```
package examen.curso2012_13.septiembre;

/**
 * Los objetos de esta clase representan productos que pueden ser
 * perecederos o no. Todos los productos se caracterizan por un nombre
 * determinado y una cierta calidad, representada por un valor entero no
 * negativo. Los productos perecederos están caracterizados además por el
 * número de días que les falta para caducar. El paso de los días disminuye
 * la calidad de los productos perecederos, que pierden 6 unidades de
 * calidad cada 5 días, hasta que alcanzan su fecha de caducidad, momento a
 * partir del cual su calidad es igual a 0. La calidad de los productos no
 * perecederos no sufre variación aunque pasen los días.
 */
public class Producto {

    /**
     * Nombre de este producto
     */
    private String nombre;

    /**
     * Entero positivo que representa la calidad de este producto
     */
    private int calidad;

    /**
     * Booleano que indica si este producto es perecedero o no
     */
    private boolean esPerecedero;

    /**
     * Entero positivo que representa el número de días que falta para que
     * caduque este producto perecedero. Si este producto no es perecedero,
     * el valor de este atributo puede ser ignorado.
     */
    private int diasParaCaducar;

    /**
     * Pre: «nombre» != null; «calidad» >= 0; si «esPerecedero», entonces
     * «diasParaCaducar» >= 0
     *
     * Post: Ha inicializado los atributos de un nuevo objeto de esta clase
     * a partir de los valores de los parámetros
     */
    public Producto(String nombre, int calidad, boolean esPerecedero,
            int diasParaCaducar) {
        this.nombre = nombre;
        this.esPerecedero = esPerecedero;
        this.diasParaCaducar = diasParaCaducar;
        this.calidad = Math.max(calidad, 0);
```

```
}

/**
 * Post: Ha devuelto el nombre de este producto.
 */
public String nombre() {
    return nombre;
}

/**
 * Post: Ha devuelto true si y solo si este producto es perecedero.
 */
public boolean esPerecedero() {
    return esPerecedero;
}

/**
 * Pre: Este producto es perecedero
 *
 * Post: Ha devuelto el número de días que quedan para que caduque este
 * producto perecedero. Si este producto no es perecedero, ha devuelto
 * un valor indeterminado.
 */
public int diasParaCaducar() {
    return diasParaCaducar;
}

/**
 * Post: Ha devuelto un entero positivo que representa la calidad de
 * este producto.
 */
public int calidad() {
    return calidad;
}

/**
 * Post: Una invocación a este método representa el hecho de que ha
 * pasado un día para el producto y, en consecuencia, en el caso de que
 * se trate de un producto perecedero, ha actualizado el número de días
 * que quedan para que caduque este producto y su valor de calidad.
 * Primero habrá disminuido en un día el número de días que le faltan
 * para caducar y, a continuación, habrá rebajado en dos o una unidades
 * de calidad en función de que el nuevo número de días hasta que
 * caduque sea o no múltiplo de 5, respectivamente. En todo caso, la
 * calidad de un producto tras haber sido actualizado nunca es inferior
 * a cero y si el producto ha caducado (es decir, le quedan 0 días o
 * menos para caducar), su calidad es igual a 0. Si este producto no es
 * perecedero, su valor de calidad no ha variado.
 */
public void actualizar() {
    if (esPerecedero) {
        diasParaCaducar = diasParaCaducar - 1;
```

```
            if (diasParaCaducar <= 0) {
                calidad = 0;
            }
            else if (diasParaCaducar % 5 == 0) {
                calidad = Math.max(calidad - 2, 0);
            }
            else {
                calidad = Math.max(calidad - 1, 0);
            }
        }
    }

    /**
     * Post: Ha devuelto true si y solo si este producto ha caducado, es
     * decir, si se trata de un producto perecedero al que le quedan 0 días
     * para caducar o menos.
     */
    public boolean haCaducado() {
        return esPerecedero && diasParaCaducar <= 0;
    }
}
```

Solución a los problemas 2.º, 3.º y 4.º

```
package examen.curso2012_13.septiembre;

import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Scanner;

public class Problemas {

    /**
     * Pre: «inventario» != null e «inventario».length > 0
     *
     * Post: Ha devuelto la referencia al producto de mayor calidad de entre
     * todos los productos referenciados en la tabla tabla «inventario». Las
     * referencias a los productos de la tabla «inventario» no han sufrido
     * cambios.
     */
    public static Producto mejor(Producto[] inventario) {
        // Referencia al producto de mayor calidad de la tabla «inventario»,
        // inicialmente, el referenciado por la primera componente
        Producto mejor = inventario[0];

        // Recorrido del resto de las componentes de la tabla «inventario»,
        // actualizando la referencia «mejor» cada vez que se encuentra un
        // elemento de calidad mayor
        for (int i = 1; i < inventario.length; i++) {
            if (inventario[i].calidad() > mejor.calidad()) {
                mejor = inventario[i];
            }
        }

        // Devolución del resultado
        return mejor;
    }

    /**
     * Pre: «inventario» != null
     *
     * Post: Ha devuelto la referencia a una nueva tabla de productos en la
     * que están todas referencias a los productos no caducados de la tabla
     * «inventario» y solo esas. Las referencias a los productos de la tabla
     * «inventario» no han sufrido cambios.
     */
    public static Producto[] noCaducados(Producto[] inventario) {
        // Contador del número de elementos no caducados de «inventario»
        int numeroNoCaducados = 0;

        // Recorrido completo de la tabla para averiguar el número de
        // productos no caducados
```

```
        for (int i = 0; i < inventario.length; i++) {
            if (!inventario[i].haCaducado()) {
                numeroNoCaducados++;
            }
        }

        // Creación de la tabla que contendrá las referencias a los
        // productos no caducados
        Producto[] noCaducados = new Producto[numeroNoCaducados];

        // Recorrido de las componentes de la tabla «inventario», copiando a
        // la componente adecuada de la tabla «noCaducados» las referencias
        // los productos no caducados
        int indiceNoCaducados = 0;
        for (int i = 0; i < inventario.length; i++) {
            if (!inventario[i].haCaducado()) {
                noCaducados[indiceNoCaducados] = inventario[i];
                indiceNoCaducados++;
            }
        }

        // Devolución del resultado
        return noCaducados;
    }

    /**
     * Pre: «nombreFicheroTexto» es distinto de null y hace referencia a un
     * fichero de texto existente con la estructura establecida en el
     * enunciado; «nombreFicheroBinario» es distinto de null y hace
     * referencia a un fichero que no existe o que, si existe, puede ser
     * reescrito.
     *
     * Post: Ha leído el contenido del fichero «nombreFicheroTexto»,
     * interpretando cada línea del mismo como un producto que, tras haber
     * sido actualizado de acuerdo con las especificaciones de la clase
     * producto establecidas en el enunciado del problema 1.º, ha sido
     * escrito como dato binario en el fichero «nombreFicheroBinario».
     */
    public static void convertirDeTextoABinario(File nombreFicheroTexto,
            File nombreFicheroBinario) {
        try {
            // Creación de los objetos para la lectura y escritura de los
            // ficheros
            Scanner texto = new Scanner(nombreFicheroTexto);
            DataOutputStream binario =
                    new DataOutputStream(new FileOutputStream(
                            nombreFicheroBinario));

            // Lectura del fichero «nombreFicheroTexto», leyendo una línea
            // cada vez, creando un producto a partir de lo leído,
            // actualizándolo y escribiéndolo en el fichero
            // «nombreFicheroBinario»
```

```
        while (texto.hasNextLine()) {
            Producto p = leerProducto(texto.nextLine());
            p.actualizar();
            escribir(binario, p);
        }

        // Liberación de los recursos utilizados por «texto» y «binario»
        texto.close();
        binario.close();
    }
    catch (IOException ex) {
        // En caso de producirse un error, se informa de ello por
        // pantalla
        System.out.println("Error␣al␣leer␣o␣escribir␣en␣los␣ficheros:␣"
                + ex.getMessage());
    }
}

/**
 * Pre: «linea» es distinto de null y tiene el formato de una línea de
 * fichero de texto que representa un producto, según el enunciado.
 *
 * Post: Ha devuelto la referencia a un nuevo objeto de la clase
 * Producto de acuerdo con el contenido de «linea».
 */
private static Producto leerProducto(String linea) {
    // Creación de un objeto de la clase Scanner para procesar «linea»
    Scanner s = new Scanner(linea);

    // Establecimiento de la coma como separador de elementos
    s.useDelimiter(",");

    // Lectura de los cuatro atributos del producto representado por
    // «linea»
    String nombre = s.next();
    int calidad = s.nextInt();
    boolean esPerecedero = s.nextBoolean();
    int diasParaCaducar = s.nextInt();

    // Creación y devolución del producto representado por «linea»
    return new Producto(nombre, calidad, esPerecedero, diasParaCaducar);
}

/**
 * Pre: «f» != null y «f» no ha sido cerrado; «p» != null
 *
 * Post: Ha escrito en «f» los atributos de «p»: la longitud de su
 * nombre, cada uno de los caracteres que forman el nombre, el entero
 * que indica su calidad, un booleano que indica si es perecedero o no y
 * el número de días que le faltan para caducar, en ese orden. El
 * fichero «f» sigue sin haberse cerrado.
 */
```

```
    private static void escribir(DataOutputStream f, Producto p) {
        try {
            // Escritura de los cuatro atributos de «p»
            escribir(f, p.nombre());
            f.writeInt(p.calidad());
            f.writeBoolean(p.esPerecedero());
            f.writeInt(p.diasParaCaducar());
        }
        catch (IOException ex) {
            // En caso de producirse un error, se informa de ello por
            // pantalla
            System.out.println("Error␣al␣escribir␣este␣producto:␣"
                    + ex.getMessage());
        }
    }

    /**
     * Pre: «f» != null y «f» no ha sido cerrado; «cadena» != null
     *
     * Post: Ha escrito en «f» la longitud de «cadena», seguido de cada uno
     * de los caracteres que la forman. El fichero «f» sigue sin haberse
     * cerrado.
     */
    private static void escribir(DataOutputStream f, String cadena) {
        try {
            // Escritura de la longitud de «cadena»
            f.writeInt(cadena.length());

            // Escritura de cada uno de los caracteres que forman «cadena»
            for (int i = 0; i < cadena.length(); i++) {
                f.writeChar(cadena.charAt(i));
            }
        }
        catch (IOException ex) {
            // En caso de producirse un error, se informa de ello por
            // pantalla
            System.out.println("Error␣al␣escribir␣el␣nombre␣este␣producto:␣"
                    + ex.getMessage());
        }
    }
}
```