UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Primera Convocatoria (02 de junio de 2017)

PREGUNTA 1 (5 PUNTOS)

Necesitamos desarrollar una aplicación para gestionar las actas de calificación de asignaturas de una titulación. Ya hemos definido una clase, `NotaEstudiante`, que proporciona los siguientes constructores y métodos públicos:

- `NotaEstudiante(String DNI, int convocatoria)`: constructor de la clase. Almacena el DNI y el número de convocatoria de un estudiante e inicialmente lo marca como no presentado.
- `String getDNI()`: devuelve el DNI del estudiante.
- `int getConvocatoria()`: devuelve el número de convocatoria del estudiante.
- `double getNota()`: devuelve la nota del estudiante.
- `void setNota(double nota)`: establece la nota del estudiante y lo marca como presentado.
- `boolean getPresentado()`: devuelve `true` si se ha establecido una nota para el estudiante y `false` en caso contrario.

Necesitamos definir una nueva clase, `Acta`, que gestione el acta de una asignatura. Considera que ya tenemos la siguiente definición parcial:

```
public class Acta {

    // Atributos
    private String código;           // Código de la asignatura
    private int curso;               // Curso académico
    private NotaEstudiante[] notas;  // Datos de los estudiantes que hay en el acta
                                     // (ordenados de menor a mayor por DNI)

    // Constructor
    public Acta(String código, int curso, NotaEstudiante[] notas) {
        this.código = código;
        this.curso = curso;
        this.notas = notas;
    }

    // Métodos
    ...
}
```

Añade a la clase `Acta` los siguientes métodos públicos:

a) `void calificar(String DNI, double nota)` **(1,5 puntos)**

Cuando el DNI dado aparezca en el acta, se establecerá la nota indicada para ese estudiante. En caso contrario, se lanzará la excepción `NoSuchElementException`.

El coste temporal de este método debe ser $\mathcal{O}(log\ n)$, siendo n el número de estudiantes en el acta.

b) `void enviarSMS(String[] vectorDNI)` **(1,75 puntos)**

El parámetro `vectorDNI` contiene, ordenados lexicográficamente de menor a mayor, los DNI de todos los estudiantes de la titulación que están suscritos al servicio de comunicación de calificaciones a través de SMS. Para todo estudiante que figure en el acta como presentado y que esté suscrito a este servicio

de mensajería, se le debe enviar un mensaje con la calificación obtenida. Para el envío del mensaje, habrá que usar el método estático `enviar` de la clase `SMS`:

```
static void enviar(String DNI, double nota) // Envía la nota dada al estudiante indicado
```

Dado que el atributo `notas` y el parámetro `vectorDNI` están ordenados por DNI, el coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n la suma de elementos de los dos vectores.

c) `Acta siguienteConvocatoria()` **(1,75 puntos)**

El método debe crear y devolver una nueva `Acta`, con el mismo código de asignatura y curso académico, pero en la que no aparezcan los estudiantes que ya han aprobado (es decir, que solo incluya los estudiantes no presentados o suspensos). El número de convocatoria se debe mantener para los no presentados y se debe incrementar en uno para los suspensos. En cualquier caso, el vector del acta resultado debe contener nuevos objetos de la clase `NotaEstudiante` inicialmente no calificados.

Si el coste temporal de este método es mayor que $\mathcal{O}(n)$, siendo n el número de estudiantes en el acta, la nota máxima de este apartado será de 1 punto.

Pregunta 2 (5 puntos)

Estamos desarrollando una aplicación para gestionar almacenes de productos. Para ello, ya disponemos de la clase `Fecha`, implementada en prácticas, y de una clase, `Producto`, que proporciona los siguientes constructores y métodos públicos:

- `Producto(String código, int cantidad, Fecha fecha)`: constructor de la clase.
- `String getCódigo()`: devuelve el código del producto.
- `int getCantidad()`: devuelve la cantidad de unidades de ese producto.
- `void setCantidad(int cantidad)`: establece la cantidad de unidades de ese producto.
- `Fecha getÚltimaFecha()`: devuelve la fecha de la última operación sobre ese producto.
- `void setÚltimaFecha(Fecha fecha)`: establece la fecha de la última operación sobre ese producto.

Necesitamos definir una nueva clase, `Almacén`, que gestione el almacén de productos. Para mantener la información necesaria de los productos vamos a utilizar una *lista simplemente enlazada en la que los nodos no guardan ningún orden concreto*. Considera que ya tenemos la siguiente definición parcial de la clase:

```
public class Almacén {

    private static class Nodo {
        Producto producto;
        Nodo     siguiente;

        Nodo(Producto producto, Nodo siguiente) {
            this.producto = producto;
            this.siguiente = siguiente;
        }
    }

    // Atributos
    private Nodo primero;

    // Constructor (por defecto)
}
```

Añade a la clase `Almacén` los siguientes métodos públicos:

a) `boolean añadirProducto(String código, int cantidad)` **(1,5 puntos)**

Cuando el almacén contenga un producto cuyo código coincida con el código dado, habrá que aumentar la cantidad de unidades de ese producto en la cantidad dada (supón que es un entero positivo) y devolver `false`. En caso contrario, habrá que insertar el nuevo producto en una posición cualquiera de la lista (recuerda que los productos no guardan ningún orden concreto) y devolver `true`. En ambos casos, la fecha de la última operación del producto debe quedar actualizada a la fecha de hoy[1].

b) `int venderProducto(String código, int cantidad)` **(1,75 puntos)**

Si el almacén no contiene ningún producto cuyo código coincida con el código dado, se deberá lanzar la excepción `NoSuchElementException`. En caso contrario, cuando haya suficientes unidades del producto para poder vender la cantidad solicitada (supón que es un entero positivo), se decrementará esa cantidad en el almacén y, cuando no haya suficientes unidades, solo se venderán las unidades disponibles. Si la cantidad de unidades del producto tras la venta queda a cero, el producto deberá eliminarse de la lista.

El método debe devolver como resultado la cantidad real de unidades que se han vendido. Además, la fecha de la última operación de ese producto debe quedar actualizada a la fecha de hoy[1].

c) `int eliminarObsoletos(Fecha fecha)` **(1,75 puntos)**

Debe eliminar del almacén todos los productos cuya fecha de última operación sea anterior a la fecha dada[2] y devolver como resultado el total de unidades eliminadas.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n el número de productos en el almacén.

[1] Recuerda que la clase `Fecha` dispone del método estático `hoy()` que devuelve la fecha actual.

[2] Recuerda que la clase `Fecha` dispone del método `compareTo`. Si `f1` y `f2` son de tipo `Fecha`, `f1.compareTo(f2)` devuelve un número negativo, cero o un número positivo según `f1` sea menor que, igual o mayor que `f2`, respectivamente.