UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional

EI1008/MT1008 Programación II

Examen final. Primera Convocatoria (26 de mayo de 2022)

PROBLEMA 1 (4,5 PUNTOS)

Estamos realizando una aplicación para gestionar un almacén de medicamentos. Ya hemos implementado una clase, `Medicamento`, que proporciona, entre otros, los siguientes constructores y métodos públicos:

- `Medicamento(String código, int cantidad)`: Inicializa un objeto con el código que identifica al medicamento y la cantidad de unidades indicados. Establece la fecha actual como fecha de última operación realizada sobre el medicamento.
- `String getCódigo()`: Devuelve el código identificativo del medicamento.
- `int getStock()`: Devuelve la cantidad de unidades que tenemos de ese medicamento.
- `Fecha getFecha()`: Devuelve la fecha de la última operación realizada sobre ese medicamento.

También tenemos la siguiente implementación parcial de la clase `AlmacenMedicamentos`:

```
public class AlmacenMedicamentos {
    private static final int TAM_INICIAL = 10; // Tamaño inicial del vector de medicamentos

    private Medicamento[] vector; // Vector de medicamentos
                                  // (ordenado de menor a mayor por código de medicamento)
    private int ocupados;         // Cantidad real de medicamentos almacenados en el vector

    public AlmacenMedicamentos() {
        vector = new Medicamento[TAM_INICIAL];
        ocupados = 0;
    }
}
```

El atributo `ocupados` indica la cantidad real de objetos de la clase `Medicamento` que hay en el vector, agrupados de forma contigua al principio del mismo y ordenados lexicográficamente de menor a mayor por su código identificativo. El resto de componentes del vector contendrá siempre `null`.

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

a) **(1,5 puntos)** `int getStock(String código)`

Busca el código indicado en el vector de medicamentos y devuelve la cantidad de unidades disponibles de ese medicamento. Si dicho código no existe, debe devolver un valor negativo.

Teniendo en cuenta que el vector de medicamentos está ordenado por código de medicamento, se exige que la búsqueda del código aproveche esta circunstancia y el coste en el peor de los casos de tu solución sea $\mathcal{O}(log\ n)$, donde `n` es la cantidad de medicamentos almacenados en el almacén.

b) **(1,5 puntos)** `void eliminar(Fecha f)`

Elimina del almacén todos los medicamentos cuya fecha de última operación sea anterior[1] a `f`. El coste temporal de este método debe ser $\mathcal{O}(n)$.

[1] Recuerda que la clase `Fecha` proporciona un método `compareTo`.

c) **(1,5 puntos)** `AlmacenMedicamentos unir(AlmacenMedicamentos otro)`

Devuelve un nuevo almacén de medicamentos que será la unión de los almacenes `this` y `otro`. La fecha de última operación de todos los medicamentos en el nuevo almacén debe ser la fecha actual, de lo que ya se encarga el constructor de la clase `Medicamento`. Cuando un mismo medicamento esté disponible en los dos almacenes, `this` y `otro`, en el nuevo almacén solo deberá aparecer una vez con un stock igual a la suma de las cantidades existentes en ambos almacenes. La capacidad del vector en el nuevo almacén de medicamentos puede ser cualquiera, siempre que sea suficiente para almacenar todos los medicamentos necesarios.

Teniendo en cuenta que los dos almacenes existentes están ordenados y que el almacén resultante también debe estarlo, se exige que el coste temporal de este método sea $\mathcal{O}(n)$, donde `n` es la cantidad de medicamentos en ambos almacenes.

Problema 2 (4,5 puntos)

Una conocida ONG acepta donaciones de particulares para ayudar a personas necesitadas. En su sistema informático se guardan ficheros de texto en los que, en cada línea, aparece la información de una donación: el DNI del donante y el importe en euros de su donación. Un mismo donante puede hacer varias donaciones. A continuación tienes un ejemplo de uno de estos ficheros (su contenido se muestra en dos columnas para reducir espacio en el enunciado):

```
19197846E  50  |  41585456F 300
64764868L  60  |  19197846E  60
41585456F 130  |  64764868L 200
49578315K   5  |  19197846E  70
```

Queremos desarrollar un programa que permita llevar un registro de todas las donaciones realizadas. Para ello, utilizaremos una lista de nodos con enlace simple. Tenemos ya una clase, `Donación`, que proporciona los siguientes constructores y métodos públicos:

- `Donación(String dni, double importe)`: Inicializa un objeto de la clase `Donación` con el DNI y el importe dados.
- `String getDni()`: Devuelva el DNI del donante.
- `double getImporte()`: Devuelve el importe de la donación.

Disponemos, además, de la implementación parcial de la clase `RegistroDonaciones`:

```
public class RegistroDonaciones {
    private static class Nodo {
        Donación donación;
        Nodo sig;

        Nodo(Donación laDonación, Nodo elSiguiente) {
            this.donación = laDonación;
            this.sig = elSiguiente;
        }
    }
    // Atributos:
    private Nodo primero;
    private Nodo último;
    private double importeTotal;

    public RegistroDonaciones() {}  // Constructor sin parámetros
}
```

Añade a la clase `RegistroDonaciones` los siguientes constructores y métodos públicos:

a) **(1 punto)** `void añadir(String dni, double importe)`

Añade un nuevo nodo al final de la lista con el DNI y el importe indicados y actualiza el atributo `importeTotal`. Su coste temporal debe ser $\mathcal{O}(1)$.

b) **(1,5 puntos)** `RegistroDonaciones(String nombreFichero) throws FileNotFoundException`

Construye la lista de nodos a partir de los datos leídos desde un fichero (como el del ejemplo) cuyo nombre se recibe como argumento. La información de cada línea del fichero se guardará en un nodo de la lista. Este constructor debe llamar al método `añadir` del apartado anterior.

c) **(2 puntos)** `RegistroDonaciones extraerDonacionesPremium(int importePremium)`

Elimina de la lista aquellos nodos correspondientes a donaciones *premium* y devuelve un nuevo objeto de la clase `RegistroDonaciones` con las donaciones eliminadas. Se considera que una donación es *premium* cuando su importe es mayor o igual que el importe del parámetro `importePremium`. El coste temporal del método debe ser $\mathcal{O}(n)$, donde n es la cantidad de elementos en la lista.