UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Primera Convocatoria (31 de mayo de 2018)

PREGUNTA 1 (5 PUNTOS)

Para tratar de evitar la reventa de entradas de ciertos espectáculos, el número de entradas que una persona puede comprar se limita a una pequeña cantidad. Así, por ejemplo, para el partido amistoso que la selección española de fútbol disputa este domingo en Villarreal, cada persona puede adquirir un máximo de cuatro entradas.

Necesitamos desarrollar una aplicación para gestionar la venta limitada de entradas de espectáculos. Ya hemos definido una clase, `Compra`, que proporciona los siguientes constructores y métodos públicos:

- `Compra(String dni, int cantidad)`: constructor de la clase. Almacena el DNI de la persona que realiza una compra y la cantidad de entradas adquiridas.
- `String getDni()`: devuelve el DNI de la persona que ha realizado la compra.
- `int getCantidad()`: devuelve la cantidad de entradas compradas.
- `void setCantidad(int cantidad)`: establece la cantidad de entradas compradas.

Necesitamos definir una nueva clase, `ComprasEspectáculo`, que gestione la venta limitada de entradas de un espectáculo. Considera que ya tenemos la siguiente definición parcial:

```
public class ComprasEspectáculo {

   // Atributos
   private final int TAMAÑO_INICIAL = 10;  // Capacidad inicial del vector de compras.

   private String descripción;  // Descripción del espectáculo.
   private int máximo;          // Número máximo de entradas que una misma persona
                                // puede adquirir para ese espectáculo.
   private Compra[] compras;    // Datos de las compras realizadas para ese espectáculo
                                // (ordenados de menor a mayor por DNI del comprador).
   private int ocupados;        // Indica las posiciones realmente ocupadas en el vector.

   // Constructor
   public ComprasEspectáculo(String descripción, int máximo) {
      this.descripción = descripción;
      this.máximo = máximo;
      this.compras = new Compra[TAMAÑO_INICIAL];
      this.ocupados = 0; // Inicialmente el vector no contiene datos.
   }

   // Métodos
}
```

Añade a la clase `ComprasEspectáculo` los siguientes métodos:

a) `public int consultarEntradas(String dni)` **(1,5 puntos)**

Devuelve la cantidad de entradas que ha adquirido la persona con el DNI indicado o un valor cero cuando dicho DNI no aparezca en el vector de compras.

Dado que el vector de compras está ordenado por DNI, el coste temporal de este método debe ser $\mathcal{O}(log\ n)$, siendo n el número de elementos del vector.

b) `private void redimensionar(int nuevoTamaño)` **(0,5 puntos)**

Redimensiona el tamaño del vector de compras al nuevo tamaño indicado, que puede ser mayor o menor que el actual. En caso de que el nuevo tamaño sea inferior al valor del atributo `ocupados`, el método debe lanzar la excepción `InvalidSizeException`[1].

c) `public boolean añadirCompra(String dni, int cantidad)` **(1,5 puntos)**

El método debe comprobar inicialmente si se acepta esa compra o no. Una compra se aceptará siempre que la suma de la cantidad de entradas previamente adquiridas por ese comprador y la cantidad solicitada no supere el máximo permitido por persona para el espectáculo considerado. Si la compra se acepta, el método debe registrar la compra y devolver `true`; si no se acepta, debe devolver `false`.

Para registrar la nueva compra, primero hay que comprobar si ya existía una compra previa para el DNI dado, en cuyo caso se actualizará la cantidad de entradas asociada al mismo. En caso contrario, se guardará la nueva compra en el vector de modo que este siga *ordenado de menor a mayor por DNI*. Siempre que sea necesario aumentar la capacidad del vector, esta se duplicará llamando al método `redimensionar` del apartado anterior.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n el número de elementos del vector de compras.

d) `public int contarComunes(ComprasEspectáculo otro)` **(1,5 puntos)**

Devuelve la cantidad de personas que han adquirido entradas para ambos espectáculos.

Dado que los dos vectores de compras 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.

Pregunta 2 (5 puntos)

Un grupo de amigos se ha reunido para organizar una fiesta. Cada vez que compran un artículo anotan en un fichero la cantidad gastada y el nombre del comprador. Un ejemplo de fichero sería:

```
40 José_Antonio
100 Marta
50 Ana
60 José_Antonio
60 Alberto
50 Marta
```

Te han encargado un programa para facilitar el trabajo de hacer las cuentas. El programa pedirá el nombre del fichero y mostrará el saldo de cada persona, es decir, la diferencia entre lo que ha pagado y el gasto medio. Cada vez que una persona pague a otra, se ajustarán sus saldos. Así, hasta que todos estén en paz. Un ejemplo de ejecución para el fichero anterior, en el que el gasto medio es de 90, sería:

```
Introduce el nombre del fichero: fiesta.txt

+ Alberto debe 30.0
+ Ana debe 40.0
+ A Marta le deben 60.0
+ A José_Antonio le deben 10.0

Nombre del pagador: Ana
Nombre del receptor: Marta
Cantidad: 40

+ Alberto debe 30.0
+ A Marta le deben 20.0
+ A José_Antonio le deben 10.0

Nombre del pagador: Alberto
Nombre del receptor: Marta
Cantidad: 20

+ Alberto debe 10.0
+ A José_Antonio le deben 10.0

Nombre del pagador: Alberto
Nombre del receptor: José_Antonio
Cantidad: 10
Todos en paz
```

[1] Considera que se trata de una excepción ya definida del tipo `RuntimeException`.

Para implementar el programa decides crear una clase, `Cuentas`, que asociará a cada persona su saldo. La definición inicial de `Cuentas` es:

```
public class Cuentas {
    private static class Nodo {
        String nombre;
        double cantidad;
        Nodo sig;

        Nodo(String nombre, double cantidad, Nodo sig) {
            this.nombre = nombre;
            this.cantidad = cantidad;
            this.sig = sig;
        }
    }

    private Nodo primero;
}
```

Añade a la clase `Cuentas` los siguientes métodos y constructores:

a) `private void insertar(String nombre, double cantidad)` **(1,5 puntos)**

Si el nombre ya aparece en la lista, añade `cantidad` a su cantidad asociada. Si no, crea un nuevo nodo y lo inserta en una posición cualquiera de la lista.

b) `private void convertirASaldos()` **(1 punto)**

Calcula el gasto medio por persona (la suma de las cantidades almacenadas en la lista dividido por el número de personas) y lo resta a la cantidad almacenada en cada nodo de la lista. Se admite que el saldo resultante pueda ser cero. Puedes asumir que la lista no estará vacía.

c) `public Cuentas(String nombreFichero) throws FileNotFoundException` **(0,75 puntos)**

Este constructor recibe el nombre de un fichero con el formato descrito anteriormente y construye un objeto de tipo `Cuentas` para almacenar el saldo de cada persona. Para ello, leerá los datos de cada línea del fichero indicado[2] y los irá insertando en la lista mediante llamadas al método `insertar`, con lo que cada nodo contendrá el gasto total de una persona. A continuación, llamará al método `convertirASaldos` para transformar el gasto total de cada persona en su saldo.

d) `public void asentarPago(String pagador, String receptor, double cantidad)` **(1,75 puntos)**

Añade `cantidad` al saldo de `pagador` y lo resta del saldo de `receptor`. Si la cantidad resultante de cualquiera de ellos es cero, elimina de la lista el nodo correspondiente. Ten en cuenta que podría ser necesario eliminar los dos nodos correspondientes al pagador y al receptor. Por cuestiones de eficiencia, *no puedes recorrer más de una vez la lista*. Puedes asumir que tanto `pagador` como `receptor` están en la lista, pero no puedes asumir en qué orden.

[2]Recuerda que para leer el fichero puedes utilizar la clase `Scanner` y sus métodos `hasNext`, `next` y `nextDouble`.