UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Segunda Convocatoria (01 de julio de 2014)

PREGUNTA 1 (5 PUNTOS)

Como parte de un sistema de control de trabajos de impresión, necesitamos definir una nueva clase, `ColaTrabajos`, que permita gestionar los trabajos pendientes de imprimir. De cada trabajo nos interesa almacenar su identificador (valor entero que lo identifica), el documento que se debe imprimir y el usuario que ha solicitado su impresión. Para ello, disponemos ya de una clase, `Trabajo`, que proporciona los siguientes métodos de consulta:

- `public int getIdentificador()`.
- `public String getDocumento()`.
- `public String getUsuario()`.

Para decidir qué trabajo será imprimido, seguiremos una estrategia *FIFO* (*First in, first out*), es decir, seleccionaremos el trabajo que ocupe la primera posición de la cola de trabajos pendientes de imprimir. Considera la siguiente definición de la clase `ColaTrabajos`:

```
public class ColaTrabajos {

    private static class Nodo {
        Trabajo trabajo;
        Nodo siguiente;

        Nodo(Trabajo trabajo, Nodo siguiente) {
            this.trabajo = trabajo;
            this.siguiente = siguiente;
        }
    }

    // Atributos
    private Nodo primero;  // Referencia al primer trabajo pendiente de imprimir
    private Nodo último;   // Referencia al último trabajo pendiente de imprimir

    // Constructor (por defecto)
}
```

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

a) (1 PUNTO) Un método, `insertarTrabajo`, que reciba un trabajo y lo añada al final de la cola de trabajos pendientes de imprimir. Puedes considerar que no habrá otro trabajo con el mismo identificador. El coste temporal de este método debe ser O(1).

b) (1 PUNTO) Un método, `extraerTrabajo`, que elimine el primero de los trabajos que quedan en la cola y devuelva el nombre del documento asociado. Si la cola está vacía, el método debe devolver `null`. El coste temporal de este método debe ser O(1).

c) (1,5 PUNTOS) Un método, `cancelarTrabajos`, que reciba el nombre de un usuario y elimine de la cola todos los trabajos de ese usuario. El coste temporal de este método debe ser O(n), siendo n el número de trabajos que hay en la cola.

d) (1,5 PUNTOS) Un método, `adelantarTrabajo`, que reciba el identificador de un trabajo y desplace ese trabajo en la cola para que pase a ocupar la primera posición. El coste temporal de este método debe ser O(n), siendo n el número de trabajos que hay en la cola.

Pregunta 2 (5 puntos)

Para organizar una boda, hemos creado la clase `Invitado`, cuya declaración es la siguiente:

```
public class Invitado {
    private String nombre;
    private String[] debeEstarCon;
    private String[] noDebeEstarCon;

    public Invitado(String nombre, String[] debeEstarCon, String[] noDebeEstarCon) {
        this.nombre = nombre;
        this.debeEstarCon = debeEstarCon;
        this.noDebeEstarCon = noDebeEstarCon;
    }

    public String getNombre() {
        return nombre;
    }
    public String[] getDebeEstarCon() {
        return debeEstarCon;
    }
    public String[] getNoDebeEstarCon() {
        return noDebeEstarCon;
    }
}
```

Como ves, para cada invitado nos guardamos su nombre y dos vectores que contienen los nombres de las personas que obligatoriamente deben estar en su mesa (por ejemplo, su pareja) y de aquellas que no pueden estar en su mesa por alguna razón.

Guardamos la información de la boda en la clase `Boda`, que tiene esta forma:

```
public class Boda {
    private Invitado[] invitados;

    public Boda(Invitado[] invitados) {
        this.invitados = invitados;
    }
    ...
 }
```

El vector `invitados` tiene todos los invitados *ordenados lexicográficamente por nombre*[1].

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

a) (1,5 puntos) Un método, `void añadirInvitado(Invitado nuevo)`, que reciba un invitado y lo añada al vector `invitados`. Al añadirlo, debe redimensionar el vector y hacer que se mantenga el orden lexicográfico. Puedes suponer que `nuevo` no estará en el vector.

b) (2 puntos) Un método, `Invitado buscarInvitado(String nombre)`, que devuelva el invitado cuyo nombre sea `nombre`. Si el `nombre` no está en el vector, el método debe devolver `null`. Debes utilizar una estrategia de búsqueda que te garantice que el coste temporal de este método sea O($log\ n$), siendo n la talla del vector.

c) (1,5 puntos) Un método, `boolean mesaCompatible(String[] nombresMesa)`, que reciba un vector con los nombres de los invitados que se sientan en una mesa y compruebe que todos cumplen las restricciones establecidas. Para determinar si una mesa es compatible, para cada nombre de `nombresMesa` debes recuperar el objeto `Invitado` correspondiente (usando el método `buscarInvitado`) y comprobar que **todos** los nombres de su vector `debeEstarCon` aparecen en `nombresMesa` y que **ningún** nombre de su vector `noDebeEstarCon` aparece en `nombresMesa`.

[1] Te recordamos que, si `c1` y `c2` son de tipo `String`, `c1.compareTo(c2)` devuelve un número negativo, cero o un número positivo según `c1` sea menor que, igual o mayor que `c2`, respectivamente, de acuerdo con el orden lexicográfico de las cadenas.