UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Primera Convocatoria (25 de mayo de 2023)

PROBLEMA 1 (4,5 PUNTOS)

Para gestionar el acceso de los mortales al cielo o al infierno, *San Pedro* desea implementar en Java una cola con nodos con enlace simple. Para ello, ya dispone de una clase, `Mortal`, que proporciona, entre otros:

- Un método `public String getNombre()`, que devuelve el nombre de la persona.
- Un método `public void setAlCielo(boolean alCielo)`, que establece el destino de la persona: `true` cuando se dirige al cielo y `false` cuando va al infierno.
- Un método `public boolean getAlCielo()`, que devuelve el destino establecido para la persona.

Además, ya dispone de la siguiente implementación parcial de la clase `ColaCielo`:

```
public class ColaCielo {
    private static class Nodo {
        Mortal mortal;
        Nodo sig;

        Nodo(Mortal mortal, Nodo sig) {
            this.mortal = mortal;
            this.sig = sig;
        }

    }

    // Atributos
    private Nodo primero;
    private Nodo último;

    // Constructor por defecto
    ...
```

Añade a la clase `ColaCielo` los siguientes métodos:

a) **(1 punto)** `public void añadir(Mortal mortal)`

Añade un nuevo mortal al final de la cola. El coste temporal de este método debe ser $\mathcal{O}(1)$.

b) **(1,75 puntos)** `public boolean cambiarDestino(String nombre)`

De vez en cuando, llegan órdenes *de arriba* para cambiar el destino preestablecido de un mortal. Si el mortal cuyo nombre se indica aparece en la cola, el método debe cambiar su destino (si era cielo, cambia a infierno y viceversa), situarlo en la última posición de la cola y devolver `true`. Si no aparece en la cola, el método debe devolver `false`. Supón que en la cola nunca habrá dos personas con el mismo nombre.

c) **(1,75 puntos)** `public void intercambiar()`

Como en ocasiones hay que poner en práctica la idea de que, en el reino de los cielos, *los últimos serán los primeros*, este método cambia las posiciones del primer y el último elemento de la cola: el primero pasa a ser el último y el último pasa a ser el primero (los demás elementos mantienen sus posiciones). Si en la cola hay menos de dos elementos, se debe lanzar la excepción `NoSuchElementException`.

IMPORTANTE: En este apartado se exige que tu solución modifique las *posiciones* de los nodos indicados. Es decir, no se permite crear nuevos nodos ni modificar el atributo `mortal` de los nodos existentes.

Problema 2 (4,5 puntos)

La clase `Lanzamiento` permite almacenar la información del lanzamiento de un cohete espacial. La clase proporciona, entre otros:

- Un método `public Fecha getFecha()`, que devuelve la fecha en la que se produjo el lanzamiento.
- Un método `public boolean esCargaCivil()`, que devuelve `true` cuando la carga útil fue civil y `false` cuando la carga útil fue militar.

Para almacenar la fecha de lanzamiento tenemos la clase `Fecha`, tal como fue implementada en las prácticas de la asignatura. Recuerda que la clase `Fecha` proporciona un método `compareTo`.

Necesitamos implementar una nueva clase, `SitioLanzamiento`, que nos permita guardar todos los lanzamientos realizados desde un lugar determinado. Los lanzamientos se guardan en un vector ordenado por fecha de lanzamiento de más antigua (menor) a más reciente (mayor). El tamaño del vector coincidirá siempre con la cantidad de lanzamientos, es decir, nunca contendrá componentes con referencia `null`. Considera que ya tenemos la siguiente definición parcial de la clase:

```
public class SitioLanzamiento {

    // Atributos
    private String nombre;
    private Lanzamiento[] lanzamientos; // ordenados de menor a mayor por fecha

    // Constructor
    public SitioLanzamiento(String unNombre) {
        this.nombre = unNombre;
        this.lanzamientos = new Lanzamiento[0];
    }
    ...
```

Añade a la clase `SitioLanzamiento` los siguiente métodos:

a) **(2 puntos)** `public boolean tuvoCargaCivil(Fecha f)`

Devuelve `true` si el lanzamiento de la fecha dada tenía una carga civil y `false` en caso contrario. Supón que no se realizan dos lanzamientos el mismo día. Si no existe ningún lanzamiento en la fecha dada, el método debe lanzar la excepción `NoSuchElementException`. El coste temporal de este método debe ser $\mathcal{O}(log\ n)$.

b) **(2,5 puntos)**

`public SitioLanzamiento unir(SitioLanzamiento otroSitio, String nuevoNombre, boolean sonCiviles)`

El método une la información de dos sitios de lanzamiento de manera que si el parámetro `sonCiviles` es `true` solo se tienen en cuenta los lanzamientos civiles y, si es `false`, solo se tienen en cuenta los militares. Una llamada como `capeCanaveral.unir(kennedySpaceCenter, "Florida", true)` devolverá como resultado un nuevo objeto de la clase `SitioLanzamiento` cuyo nombre será `"Florida"` y cuyo vector de lanzamientos contendrá los lanzamientos civiles de `capeCanaveral` y los lanzamientos civiles de `kennedySpaceCenter`. Supón que no habrá lanzamientos con la misma fecha en los dos sitios de lanzamiento.

Teniendo en cuenta que los vectores de lanzamientos siempre deben estar ordenados por fecha, se exige que el coste temporal de este método sea $\mathcal{O}(n)$, donde `n` es el total de lanzamientos en los dos vectores.