UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Segunda Convocatoria (21 de junio de 2023)

PROBLEMA 1 (4,5 PUNTOS)

Estamos realizando una aplicación para gestionar un parking de vehículos. Ya hemos implementado una clase, `Instante`, que almacena una fecha y una hora y se utilizará para registrar los instantes en los que se realizan las entradas y las salidas del parking. También tenemos otra clase, `Registro`, que proporciona, entre otros, los siguientes constructores y métodos públicos:

- `Registro(String matrícula)`: Inicializa un objeto con la matrícula dada. Establece la fecha y hora actuales como el instante de entrada al parking.
- `void setSalida()`: Establece la fecha y hora actuales como el instante de salida del parking.
- `String getMatrícula()`: Devuelve la matrícula del vehículo.
- `Instante getEntrada()`: Devuelve el instante en el que el vehículo entró al parking.
- `Instante getSalida()`: Devuelve el instante en el que el vehículo salió del parking. Si el vehículo continúa en el parking, devuelve `null`.
- `int tiempoUso()`: Devuelve la cantidad de minutos transcurridos desde la entrada hasta la salida. Si el vehículo no ha salido, se produce una excepción.

Hemos empezado a implementar una clase, `Parking`, que internamente mantiene una lista de nodos con enlace simple:

```
public class Parking {
    private static class Nodo {
       Registro reg;
       Nodo sig;
       Nodo(Registro r, Nodo s) {
          this.reg = r;
          this.sig = s;
       }
    }
    // Atributo
    private Nodo primero;
    // Constructor (por defecto)
}
```

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

a) **(0,5 puntos)** `public void entraCoche(String matrícula)`

Crea un nuevo registro para la matrícula dada que automáticamente incluirá el instante de entrada. Este registro se añade en un nuevo nodo al principio de la lista.

b) **(1,25 punto)** `public boolean saleCoche(String matrícula)`

Busca el primer nodo de la lista en cuyo registro aparezca la matrícula dada. Si dicho registro no tiene almacenada una salida (es decir, contiene `null`), almacena el instante actual como instante de salida y devuelve `true`. Si el registro ya tiene almacenada una salida, el método debe devolver `false`. Si la matrícula no aparece en la lista, se debe lanzar la excepción `NoSuchElementException`.

c) **(1,25 punto)** `public int[] pagosPendientes(String[] matrículas)`

El método tiene como parámetro un vector de matrículas y devuelve como resultado un vector de enteros con los minutos pendientes de facturar para cada una de las matrículas dadas. Los minutos a facturar para una matrícula se calculan como la suma de los tiempos de uso de todos los registros que se correspondan con dicha matrícula para los que conste su salida.

d) **(1,5 puntos)** `public int facturar(String matrícula)`

El método devuelve como resultado el total de minutos que se debe facturar a la matrícula dada (la suma de los tiempos de uso de esa matrícula para aquellos registros en los que consta una la salida). Además, debe borrar de la lista todos los nodos cuyos importes se han tenido en cuenta en la facturación.

Problema 2 (4,5 puntos)

Queremos desarrollar una aplicación que nos permita gestionar un catálogo de componentes electrónicos. Se dispone de la clase `Componente` que proporciona, entre otros, los siguientes métodos públicos:

- `String getCódigo()`: Devuelve una cadena con el código identificativo del componente.
- `void marcarObsoleto()`: Marca el componente como obsoleto.
- `boolean esObsoleto()`: Devuelve `true` si el componente ha sido marcado como obsoleto y `false` en caso contrario.

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

```
public class Catálogo {
    private static final int TAM_INICIAL = 10;
    private Componente[] vector; // Vector de componentes
                                 // (ordenados lexicográficamente por código)
    private int cantidad; // Cantidad real de componentes almacenados en el vector

    public Catálogo() {
        vector = new Componente[TAM_INICIAL];
        cantidad = 0;
    }
    ...
}
```

El atributo `cantidad` indica la cantidad de componentes del vector que realmente se están utilizando. Todos los elementos del vector que contengan un componente están agrupados de forma contigua al principio del vector. Los componentes están **ordenados lexicográficamente por código**.

Añade a la clase `Catálogo`:

a) **(1,5 puntos)** El método `public boolean añadir(Componente componente)`.

El método debe comprobar si en el catálogo ya existe un componente con el mismo código. De ser así, devuelve `false` y el catálogo no se modifica. Si no, añade el componente al catálogo, de modo que este mantenga el orden requerido, y devuelve `true`. Cuando el vector esté lleno y sea necesario aumentar su tamaño, este se duplicará.

El coste temporal del método debe ser $\mathcal{O}(n)$, siendo n la cantidad de componentes en el catálogo.

b) **(1,5 puntos)** El método `public int marcarObsoletos(String[] v)`, que tiene como parámetro un vector no ordenado de códigos de componentes. El método debe buscar en el catálogo cada código que aparece en el vector `v`. Si lo encuentra, lo marca como obsoleto. Si no, simplemente lo ignora. El método debe devolver como resultado la cantidad de componentes que han sido marcados como obsoletos.

Dado que el vector `v` no está ordenado, pero los componentes en el catálogo sí lo están, el coste temporal del método debe ser $\mathcal{O}(m \log n)$, siendo m la cantidad de elementos en el vector `v` y n la cantidad de componentes en el catálogo.

c) **(1,5 puntos)** El método `public int eliminarObsoletos()` que elimina del catálogo todos los componentes que estén marcados como obsoletos y devuelve como resultado la cantidad de componentes que se han eliminado.

El coste temporal del método debe ser $\mathcal{O}(n)$, siendo n la cantidad de componentes en el catálogo.