UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Primera Convocatoria (05 de junio de 2015)

PREGUNTA 1 (5 PUNTOS)

Queremos realizar una aplicación para la gestión de una agenda telefónica. Para ello, disponemos de la clase `Fecha` implementada en prácticas y de una clase, `Contacto`, que proporciona los siguientes métodos de consulta:

- `public String getNombre()`: devuelve el nombre del contacto.
- `public String getTeléfono()`: devuelve el teléfono asociado al contacto.
- `public Fecha getAniversario()`: devuelve la fecha de nacimiento del contacto.

Necesitamos definir una nueva clase, `Agenda`, que gestione una colección de contactos. Considera que ya tenemos la siguiente definición parcial:

```
public class Agenda {
    private static final int TAMAÑO_INICIAL = 10;  // Capacidad inicial del vector de contactos.

    private Contacto[] vectorContactos;  // Datos de los contactos que hay en la agenda
                                         // (ordenados de menor a mayor por nombre).
    private int numContactos;            // Número de contactos en la agenda.

    public Agenda() {
        vectorContactos = new Contacto[TAMAÑO_INICIAL];
        numContactos = 0;  // Inicialmente la agenda está vacía.
    }
}
```

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

a) `boolean agregarContacto(Contacto nuevo)` **(1,5 puntos)**

Cuando la agenda contenga un contacto cuyo nombre coincida con el del contacto dado, habrá que reemplazar el contacto antiguo por el nuevo y devolver `false`. En caso contrario, habrá que insertar el nuevo contacto en la posición apropiada (recuerda que los contactos están ordenados por nombre) y devolver `true`. Siempre que sea preciso aumentar la capacidad del vector de contactos, deberás duplicar su capacidad.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n el número de contactos en la agenda.

b) `boolean borrarContacto(String nombre)` **(1,25 puntos)**

Cuando la agenda contenga un contacto con el nombre dado, habrá que borrar ese contacto y devolver `true`. En caso contrario, no se deberá modificar la agenda y el método devolverá `false`. La capacidad del vector no se debe modificar en ningún caso.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n el número de contactos en la agenda.

c) `String consultarTeléfono(String nombre)` **(1,25 puntos)**

Devuelve el teléfono correspondiente al nombre dado, o el valor `null` si la agenda no contiene ningún contacto con ese nombre.

El coste temporal de este método debe ser $\mathcal{O}(log\ n)$, siendo n el número de contactos en la agenda.

d) `Contacto[] cumplenHoy()` **(1 punto)**

Devuelve un vector con todos los contactos a los que hay que felicitar por ser su cumpleaños. Recuerda que la clase `Fecha` proporciona el método estático `hoy()`, que permite averiguar la fecha actual del sistema, así como los métodos de consulta `getDía()` y `getMes()`.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n el número de contactos en la agenda.

Pregunta 2 (5 puntos)

Como encargado del procesamiento de los datos electorales para una emisora local, debes recoger los datos de las distintas mesas electorales para, posteriormente, preparar un informe. Para hacer más fácil tu tarea, decides usar dos estructuras de datos. Por un lado, utilizarás `ListaResultados` para almacenar los resultados de cada partido político. Por otro lado, `Partido` será la clase que guarde las características de los partidos.

La clase `ListaResultados` tiene la siguiente declaración:

```
public class ListaResultados {

    private static class Nodo {
        String nombrePartido;
        int votos;
        Nodo sig;

        public Nodo(String nombrePartido, int votos, Nodo sig) {
            this.nombrePartido = nombrePartido;
            this.votos = votos;
            this.sig = sig;
        }
    }

    // Atributo
    private Nodo primero;

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

Como ves, es una lista simplemente enlazada en la que tenemos una referencia al primer elemento. En cada nodo de la lista se almacena el nombre del partido y la cantidad total de votos que ha recibido en todas las mesas electorales.

La clase `Partido` proporciona los siguientes métodos:

- `public String getNombre()`: devuelve el nombre del partido.
- `public boolean puedePactarCon(String otroPartido)`: devuelve `true` para aquellos partidos con los que estaría dispuesto a pactar.

Los resultados de las elecciones se guardan en un fichero de texto donde cada línea tiene tres campos: el número de mesa electoral, el nombre del partido y la cantidad de votos. El comienzo de uno de esos ficheros podría ser así:

```
1 Partido_A 250
2 Partido_B 125
1 Partido_B 225
3 Partido_A 150
2 Partido_C 250
...
```

Date cuenta de que las mesas no están ordenadas y de que un mismo partido recibe votos en varias mesas.

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

a) `ListaResultados(String nombreFichero) throws FileNotFoundException` **(2 puntos)**

Un constructor que recibe el nombre de un fichero con el formato descrito anteriormente y construye la lista con los resultados electorales[1]. La lista debe tener un nodo por cada partido que recoja la cantidad total de votos que ha obtenido en todas las mesas electorales. Solo se permite leer los datos del fichero una vez.

b) `ListaResultados filtraUmbral(double umbral)` **(1,5 puntos)**

Un método que recibe el porcentaje mínimo de votos que debe tener un partido para tener representación. Como resultado, devuelve una nueva lista con solo aquellos partidos que superan ese porcentaje de los votos. Ten presente que deberás recorrer la lista una primera vez para averiguar el total de votos y una segunda para ir construyendo la lista resultado. El método no debe modificar la lista original y su coste debe ser $\mathcal{O}(n)$, donde n es el número de elementos en la lista.

c) `Partido[] mayoría(Partido[] partidos)` **(1,5 puntos)**

Un método que recibe un vector con todos los partidos que se han presentado a las elecciones y cuyo resultado es uno de los siguientes:

- Si un partido tiene más votos que todos los demás partidos juntos, un vector con ese partido como único elemento.
- Si hay dos partidos que pueden pactar (utiliza el método `puedePactarCon`) y entre los dos tienen más votos que el resto de partidos, un vector con esos dos partidos, en cualquier orden. Si hubiese más de una pareja posible, se puede devolver cualquiera de ellas.
- En cualquier otro caso, `null`.

El método no debe modificar la lista original.

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