UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Primera convocatoria (06 de junio de 2016)

Pregunta 1 (5 puntos)

Queremos desarrollar una aplicación para tiendas de alquiler de videojuegos y necesitamos implementar una clase que nos permita gestionar la información de un catálogo de videojuegos. Para ello, disponemos de una clase, `Juego`, que proporciona los siguientes métodos públicos:

- `String getTítulo()`: devuelve el título del juego.
- `int getCantidad()`: devuelve la cantidad de ejemplares del juego.
- `void setCantidad(int cantidad)`: establece la cantidad de ejemplares del juego.

Necesitamos definir una nueva clase, `CatálogoDeJuegos`, que gestione el catálogo de videojuegos. Considera que ya tenemos la siguiente definición parcial:

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

    private Juego[] vector;  // Datos de los juegos que hay en el catálogo
                             // (ordenados de menor a mayor por título).
    private int numJuegos;   // Número de juegos en el catálogo.

    public CatálogoDeJuegos() {
        vector = new Juego[TAMAÑO_INICIAL];
        numJuegos = 0;  // Inicialmente el catálogo está vacío.
    }
}
```

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

a) `int consultarEjemplares(String título)` **(1,25 puntos)**

Cuando el catálogo contenga un juego cuyo título coincida con el dado, se devolverá la cantidad de ejemplares del mismo. En caso contrario, se devolverá cero.

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

b) `boolean agregarJuego(Juego nuevo)` **(1,25 puntos)**

Cuando el catálogo contenga un juego cuyo título coincida con el del juego dado, habrá que aumentar la cantidad de ejemplares del mismo en la cantidad correspondiente al nuevo juego y devolver `false`. En caso contrario, habrá que insertar el nuevo juego en la posición apropiada (recuerda que los juegos están ordenados por título) y devolver `true`. Siempre que sea preciso aumentar la capacidad del vector de juegos, deberás duplicar su capacidad.

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

c) `boolean borrarJuego(String título)` **(1,25 puntos)**

Cuando el catálogo contenga un juego con el título dado, habrá que disminuir en uno la cantidad de ejemplares de ese juego y devolver `true`. En caso contrario, no se deberá modificar el catálogo y el método devolverá `false`. Cuando al reducir la cantidad de ejemplares de un juego el resultado sea 0, se debe eliminar el juego del vector, que debe continuar ordenado. 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 juegos en el catálogo.

d) `CatálogoDeJuegos mezclar(CatálogoDeJuegos otroCatálogo)` **(1,25 puntos)**

Devuelve un nuevo catálogo que contiene todos los juegos incluidos en los catálogos `this` y `otroCatálogo`. Cuando un juego aparezca en ambos catálogos, en el catálogo resultado aparecerá solamente una vez, pero el número de ejemplares será la suma de los ejemplares de ambos catálogos. Ten en cuenta que los catálogos que ya existen están ordenados y que el catálogo que debes devolver también debe quedar ordenado.

El coste temporal de este método debe ser $\mathcal{O}(n)$, siendo n la suma de los números de juegos en los dos catálogos.

Pregunta 2 (5 puntos)

Queremos desarrollar una aplicación que permita el seguimiento de la dispersión del mosquito tigre en una región. Para ello se cuenta con la colaboración de voluntarios que proporcionan información sobre avistamientos de estos insectos. Para almacenar la información de un avistamiento suministrada por un voluntario disponemos de la clase `Avistamiento` que tiene los siguientes métodos públicos (considera que ya existen las clases `Fecha`[1] y `Punto`[2], las mismas que has desarrollado en las prácticas de la asignatura):

- `String getNombreDelVoluntario()`: devuelve el nombre del voluntario que ha hecho el avistamiento.
- `Fecha getFecha()`: devuelve la fecha del avistamiento.
- `Punto getCoordenadas()`: devuelve un punto que representa las coordenadas del avistamiento en el plano de la región.

Para gestionar los avistamientos vamos a utilizar una **lista simplemente enlazada en la que los nodos están ordenados por fecha**. Considera la siguiente definición (incompleta) de la clase `ListaAvistamientos`:

```
public class ListaAvistamientos {

    private static class Nodo {
        Avistamiento avistamiento;
        Nodo         siguiente;

        Nodo(Avistamiento avistamiento, Nodo siguiente) {
            this.avistamiento = avistamiento;
            this.siguiente = siguiente;
        }
    }

    // Atributos
    private Nodo primerNodo;
    ...
}
```

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

a) **(1,5 puntos)** Un método, `void añadir(Avistamiento avistamiento)`, que reciba un avistamiento y lo añada a la lista de avistamientos. Recuerda que la lista debe estar ordenada por fecha de menor a mayor. En el caso de que varios avistamientos tengan la misma fecha el orden en el que se almacenen esos nodos es indiferente.

b) **(1 punto)** Un método, `Avistamiento másCercano(Punto p)`, que devuelva el avistamiento más cercano al punto `p`. En caso de empate entre varios avistamientos, el método puede devolver cualquiera de ellos.

c) **(1 punto)** Un método, `void eliminarAvistamientosAntesDe(Fecha fecha)`, que reciba una fecha y elimine de la lista todos los avistamientos anteriores a la fecha dada. Recuerda que los nodos están ordenados por fecha.

d) **(1,5 puntos)** Un método, `void eliminarAvistamientosDe(String voluntario)`, que reciba el nombre de un voluntario y elimine de la lista todos los avistamientos proporcionados por ese voluntario.

[1] La clase `Fecha` dispone del método `compareTo`. Si `f1` y `f2` son de tipo `Fecha`, la llamada `f1.compareTo(f2)` devuelve un número negativo, cero o un número positivo según `f1` sea menor que, igual o mayor que `f2`, respectivamente.

[2] La clase `Punto` dispone del método `distancia`. Si `p1` y `p2` son del tipo `Punto`, la llamada `p1.distancia(p2)` devuelve la distancia entre los puntos `p1` y `p2`.