UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional

EI1008/MT1008 Programación II

Examen final. Primera Convocatoria (31 de mayo de 2019)

PREGUNTA 1 (5 PUNTOS)

Disponemos de la clase `Libro` que proporciona varios constructores y métodos públicos. De entre todos ellos, los que utilizaremos en este ejercicio son los siguientes:

- `Libro(Libro otro)`: inicializa un libro con los mismos datos que el libro dado como parámetro.
- `String getIsbn()`: devuelve el ISBN del libro.
- `String getAutor()`: devuelve el autor del libro.
- `int getEjemplares()`: devuelve la cantidad de ejemplares disponibles del libro.
- `void setEjemplares(int cantidad)`: establece la cantidad de ejemplares del libro.

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

```
public class Biblioteca {

   // Atributo
   private Libro[] v;    // Vector de libros ordenado de menor a mayor por ISBN.
                         // El tamaño del vector coincidirá siempre con la cantidad de libros
                         // en la biblioteca (es decir, nunca habrá componentes a null).
   // Constructor
   public Biblioteca() {
      this.v = new Libro[0];  // Vector con 0 libros
   }

   // Métodos

   private int buscar(String isbn) {
     // Este método busca en el vector v el ISBN dado y
     // - si lo encuentra: devuelve la posición correspondiente
     // - si no lo encuentra: devuelve la posición en la que debería insertarse para mantener el orden.
     //   Existe la posibilidad de que devuelva v.length en el caso de que el ISBN dado deba colocarse
     //   al final del vector.

     ...
   }

}
```

Añade a la clase `Biblioteca` los siguientes métodos:

a) `public boolean añadir(Libro libro)` **(1,75 puntos)**

Si el `libro` dado ya estaba en la biblioteca, incrementa la cantidad de ejemplares y devuelve `false`. Si el `libro` no estaba en la biblioteca, lo inserta en la posición adecuada y devuelve `true`.

En este apartado, debes hacer uso del método privado `buscar`.

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

b) `public String[] autores()` **(1,75 puntos)**

Devuelve un vector de cadenas con los autores de todos los libros de la biblioteca pero sin repeticiones, es decir, si varios libros han sido escritos por un mismo autor, este solo debe aparecer una vez en el vector resultado. El tamaño del vector resultado debe ajustarse a la cantidad de elementos que contenga, sin incluir ninguna referencia a `null`.

c) `public Biblioteca mezclar(Biblioteca otra)` **(1,5 puntos)**

Devuelve una nueva biblioteca que contendrá los ejemplares de las bibliotecas `this` y `otra`. Cuando un mismo libro aparezca en las dos bibliotecas, en la nueva aparecerá solamente una vez con el número de ejemplares dado por la suma de ambas.

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

En la nueva biblioteca debes almacenar nuevos objetos de la clase `Libro` con los mismos datos que los originales, pero no referencias a los objetos ya existentes.

Pregunta 2 (5 puntos)

Para hacer un videojuego disponemos de una clase, `Personaje`, que almacena y gestiona toda la información necesaria de los distintos personajes como su fuerza, la cantidad de vida, la posición en la que se encuentran, la velocidad a la que se mueven, etc. La clase `Personaje` proporciona varios constructores y métodos públicos. De entre todos ellos, los métodos que utilizaremos en este ejercicio son los siguientes:

- `void mover()`: actualiza la posición del personaje según su velocidad.
- `boolean estáMuerto()`: devuelve `true` si la cantidad de vida del personaje está a cero.
- `boolean estáCerca(Personaje otro)`: devuelve `true` si la distancia entre los personajes `this` y `otro` está por debajo de un umbral.
- `void luchar(Personaje otro)`: aleatoriamente simula un combate entre los personajes `this` y `otro` y reduce la cantidad de vida de `this`, `otro` o ambos.

Para controlar todos los personajes del juego utilizaremos una lista simplemente enlazada en la clase `ListaPersonajes` de la que tenemos la siguiente implementación parcial:

```
public class ListaPersonajes {
    private static class Nodo {
        Personaje personaje;
        Nodo sig;

        public Nodo(Personaje personaje, Nodo sig) {
            this.personaje = personaje;
            this.sig = sig;
        }
    }

    private Nodo primero;

    // Constructor por defecto

}
```

Añade a la clase `ListaPersonajes` los siguientes métodos:

a) `public void mover()` **(1 punto)**

Actualiza la posición de todos los personajes no muertos de la lista. Para ello, llamará al método `mover` de la clase `Personaje`.

b) `public void luchar()` **(2 puntos)**

Este método debe formar parejas de personajes tales que ambos estén vivos y cerca el uno del otro. Para cada una de esas parejas debes llamar al método `luchar`, teniendo que cuenta que debes hacerlo sólo una vez por pareja, es decir, si llamas al método `luchar` del personaje `A` con el personaje `B` como parámetro, no debes llamar al método `luchar` de `B` con `A` como parámetro. Lógicamente, si como consecuencia de la lucha un personaje muere, no participará en más luchas.

Aunque el resultado final de quién sobrevive y quién muere dependerá del orden en el que se realicen las luchas, eso no debe preocuparte en tu implementación. Puedes elegir un orden cualquiera para establecer las parejas que se enfrentarán.

c) `public void quitarMuertos()` **(2 puntos)**

Elimina de la lista todos los personajes para los que `estáMuerto` devuelve `true`.