UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional
EI1008/MT1008 Programación II
Examen final. Segunda Convocatoria (07 de julio de 2015)

Pregunta 1 (5 puntos)

Queremos realizar una aplicación para la gestión de las inscripciones a una serie de MOOC (Massive Online Open Course). Para ello, disponemos de una clase, `Inscripción`, que proporciona los siguientes métodos de consulta:

- `public String getMOOC()`: devuelve el código del curso.
- `public String getDNI()`: devuelve el DNI de la persona.

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

```
public class ReservasMOOC {

    private Inscripción[] vector;  // Datos de las inscripciones ordenados de menor a mayor
                                   // por código del curso. Para un mismo curso,
                                   // los DNI aparecen en cualquier orden.
    public ReservasMOOC() {
        vector = new Inscripción[0];
    }
}
```

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

a) `void agregarInscripción(Inscripción nueva)` **(1,5 puntos)**

El método recibe una nueva inscripción y la añade al vector. Al añadirla, debes redimensionar el vector y hacer que se mantenga el orden de los cursos. Puedes suponer que `nueva` no estará en el vector.

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

Ejemplo de uso:
```
ReservasMOOC reservas = new ReservasMOOC();
Inscripción inscripción = new Inscripción("MOOC-123", "12345678X");
reservas.agregarInscripción(inscripción);
```

b) `String cursoMásInscripciones()` **(1,5 puntos)**

Devuelve el código del curso para el que hay un mayor número de inscripciones. En caso de empate entre varios cursos, puedes devolver cualquiera de ellos.

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

Ejemplo de uso:
```
String cursoMásSolicitado = reservas.cursoMásInscripciones();
```

c) `String[] inscritosMOOC(String códigoCurso)` **(2 puntos)**

Devuelve un vector con los DNI de las personas inscritas al curso `códigoCurso`. La capacidad del vector devuelto como resultado debe coincidir exactamente con el número de personas inscritas al curso. En el caso concreto de que no se haya inscrito ninguna persona al curso dado, se devolverá un vector de cero elementos.

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

Ejemplo de uso:
```
String[] inscritosCurso = reservas.inscritosMOOC("MOOC-123");
```

Pregunta 2 (5 puntos)

Queremos desarrollar una aplicación que nos permita gestionar un sistema de préstamos de bicicletas públicas en una ciudad. Para cada préstamo de un usuario concreto necesitamos saber en qué fecha se ha producido, el lugar de donde el usuario toma prestada una bicicleta, el lugar donde la devuelve y la duración en minutos del préstamo. Considera que ya disponemos de la clase `Fecha` (la misma que has desarrollado en las prácticas de la asignatura y que, entre otros, proporciona el método `compareTo`[1]) y de la clase `Préstamo`, que proporciona los siguientes métodos:

- `Fecha getFecha()`.
- `String getOrigen()`.
- `String getDestino()`.
- `int getDuración()`.

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

```
public class ListaPréstamos {
    class Nodo {
        Préstamo dato;
        Nodo siguiente;

        Nodo(Préstamo unPréstamo, Nodo unNodo) {
            dato = unPréstamo;
            siguiente = unNodo;
        }
    }

    private Nodo primerNodo;

    // Constructor (por defecto)
    ...
}
```

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

a) `void añadir(Préstamo unPréstamo)` **(1,5 puntos)**

El método recibe un préstamo y lo añade a la lista de préstamos. Recuerda que la lista debe estar ordenada por fechas de menor a mayor[1]. En el caso de que varios préstamos tengan la misma fecha, el orden en el que se almacenen los nodos es indiferente.

Ejemplo de uso:
```
ListaPréstamos miLista = new ListaPréstamos();
Fecha unaFecha = new Fecha(18, 5, 2015);
Préstamo unPréstamo = new Préstamo(unaFecha, "UJI", "Gran Vía", 15);
miLista.añadir(unPréstamo);
```

b) `ListaPréstamos extraerActividadSancionable(int límite)` **(1,5 puntos)**

El método recibe un valor `límite` en minutos y devuelve un nuevo objeto de la clase `ListaPréstamos` que contiene aquellos préstamos sancionables por tener una duración mayor que `límite`. La lista original no debe modificarse.

Ejemplo de uso:
```
ListaPréstamos sancionable = miLista.extraerActividadSancionable(120);
```

c) `void eliminarPréstamos(Fecha laFecha)` **(2 puntos)**

El método recibe una fecha y elimina de la lista todos los préstamos cuya fecha coincida con esa. Los préstamos restantes deben quedar ordenados por fecha.

Ejemplo de uso:
```
miLista.eliminarPréstamos(unaFecha);
```

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