UNIVERSITAT JAUME·I

1º Ingeniería Informática
1º Matemática Computacional

EI1008/MT1008 Programación II

Examen final. Segunda Convocatoria (28 de junio de 2021)

PREGUNTA 1 (5 PUNTOS)

La evaluación de un trabajo de una asignatura se organiza mediante dos exámenes orales. Los estudiantes pueden decidir si se presentan solo a uno de ellos, en cuyo caso la nota del trabajo coincide con la de ese examen, o a ambos, en cuyo caso la nota del trabajo es la máxima de las notas obtenidas en los dos exámenes. Disponemos de una clase, `Puntuación`, que proporciona los siguientes constructores y métodos públicos:

- `Puntuación(String dni, double nota)`: establece la nota obtenida por un estudiante en una actividad evaluable.
- `String getDni()`: devuelve el DNI del estudiante.
- `double getNota()`: devuelve la nota obtenida en esa actividad.

Necesitamos definir una nueva clase, `EvaluaciónTrabajo`, que gestione las calificaciones del trabajo. Para almacenar las puntuaciones obtenidas por los estudiantes que presentan el trabajo, decidimos utilizar un vector de objetos de la clase `Puntuación`, *ordenado de menor a mayor* por DNI. Considera la siguiente definición incompleta de la clase `EvaluaciónTrabajo`:

```
public class EvaluaciónTrabajo {

    private Puntuación[] puntuaciones;  // Vector de puntuaciones (ordenado por DNI)
    private int presentados;            // Estudiantes presentados

    public EvaluaciónTrabajo(int matriculados) {
        this.puntuaciones = new Puntuación[matriculados]; // Capacidad para todos los estudiantes
                                                          // matriculados en la asignatura
        this.presentados = 0;
    }
}
```

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

a) `void añadirNota(String DNI, double nota)` **(1,5 puntos)**

Añade una nueva nota al vector de puntuaciones de modo que siga estando ordenado de menor a mayor por DNI. Supón que el vector de puntuaciones no contiene el DNI recibido. Dado que el vector de puntuaciones tiene capacidad para todos los estudiantes matriculados en la asignatura, nunca será necesario aumentar su tamaño. El coste temporal de este método debe ser $\mathcal{O}(n)$, donde n representa el número de estudiantes presentados.

b) `double consultarNota(String DNI)` **(1,75 puntos)**

Recibe el DNI de un estudiante y devuelve la nota que le corresponde. Si el DNI dado no aparece en el vector de puntuaciones, el método debe lanzar la excepción `NoSuchElementException`. El coste temporal de este método debe ser $\mathcal{O}(log\ n)$, donde n representa el número de estudiantes presentados.

c) `EvaluaciónTrabajo calificarTrabajo(EvaluaciónTrabajo segundoExamen)` **(1,75 puntos)**

Calcula la nota final de cada trabajo a partir de las notas obtenidas en los dos exámenes orales, `this` y `segundoExamen`, los cuales no deben ser modificados. El coste temporal de este método debe ser $\mathcal{O}(n)$, donde n representa el número de estudiantes presentados.

Pregunta 2 (5 puntos)

Estamos desarrollando una aplicación para la gestión de tarjetas de transporte público recargables. Para ello, ya disponemos de la clase `Fecha` implementada en prácticas y de una clase, `Recarga`, que proporciona los siguientes constructores y métodos públicos:

- `Recarga(String código, Fecha fecha, int viajes)`: crea una nueva recarga y la marca como pendiente de confirmación. Los viajes asociados a la recarga no serán definitivos hasta que no se confirme la recarga.
- `String getCódigo()`: devuelve el código identificador de la recarga.
- `Fecha getFecha()`: devuelve la fecha en la que se realizó la recarga.
- `int getViajes()`: devuelve el número de viajes correspondiente a la recarga.
- `boolean pendiente()`: devuelve `true` cuando la recarga siga pendiente de confirmación y `false` cuando haya sido confirmada.
- `void confirmar()`: establece la recarga como confirmada.

Necesitamos definir una nueva clase, `Tarjeta`, que gestione las recargas de una tarjeta. Para almacenar el historial de recargas de una tarjeta utilizaremos una lista de nodos con enlace simple. Los datos de las nuevas recargas se añadirán siempre al comienzo de la lista, con lo que esta quedará *ordenada por fecha de recarga de más reciente a más antigua*. Considera la siguiente definición incompleta de la clase `Tarjeta`:

```
public class Tarjeta {

    private static class Nodo {
        Recarga recarga;
        Nodo siguiente;

        Nodo(Recarga recarga, Nodo siguiente) {
            this.recarga = recarga;
            this.siguiente = siguiente;
        }
    }

    // Atributos
    private long id;            // Número que identifica a la tarjeta
    private int viajes;         // Total de viajes disponibles en la tarjeta
    private Nodo primero;       // Referencia al primer nodo de la lista de recargas
    private int totalRecargas;  // Total de recargas realizadas (incluidas las canceladas)

    // Constructor
    public Tarjeta(long id) {
        this.id = id;
        this.viajes = 0;
        this.primero = null;
        this.totalRecargas = 0;
    }

    ... // Métodos
}
```

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

a) `void recargar(int viajes)` **(0,5 puntos)**

Crea una nueva recarga correspondiente a la fecha actual[1] y a la cantidad de viajes indicada y la añade al comienzo de la lista de recargas. El número de viajes de la tarjeta no debe variar y el contador de recargas debe incrementarse en 1.

Como código de la nueva recarga se debe utilizar una cadena con el formato Tid/Rn, donde id represente el número de la tarjeta y n el total de recargas de la tarjeta, incluida la recarga actual. Por ejemplo, el código para la cuarta recarga de una tarjeta con identificador 58755 sería `T58755/R4`.

b) `int confirmarPendientes()` **(1,25 puntos)**

Para cada una de las recargas pendientes de confirmar, establece la recarga como confirmada y actualiza el total de viajes disponibles en la tarjeta. Además, devuelve como resultado la cantidad de recargas que acaban de ser confirmadas.

Nota: Ten en cuenta que las recargas pendientes de confirmar ocuparán los primeros nodos de la lista de recargas.

c) `boolean cancelarRecarga(String código)` **(1,75 puntos)**

Si el código dado no corresponde a ninguna de las recargas de la lista, el método debe lanzar la excepción `NoSuchElementException`. Cuando el código dado corresponda a una recarga pendiente de confirmar, el método debe eliminarla de la lista de recargas y devolver `true`. En otro caso, no se modifica la lista de recargas y el método devuelve `false`.

d) `void borrarAnteriores(Fecha fecha)` **(1,5 puntos)**

Borra del historial todas las recargas realizadas antes de la fecha dada[2].

[1] Recuerda que la clase `Fecha` dispone del método estático `hoy()`, que devuelve la fecha actual.

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