Programación II - 2012/2013 - Universitat Jaume I
Examen final - Primera convocatoria

30 de mayo de 2013

La duración máxima de este examen es de 3 horas. No puedes consultar libros ni apuntes. Para aprobar debes obtener al menos 4 puntos en el examen final y al menos 5 puntos en la nota media del curso.

Ejercicio 1 — 0,5 puntos

Considera la siguiente implementación de la clase `Par`:

```
public class Par {

    private int primero, segundo;

    public Par(int elPrimero, int elSegundo) {
        primero = elPrimero;
        segundo = elSegundo;
    }

    public Par intercambia() {
        return new Par(segundo, primero);
    }

    public String toString() {
        return "(" + primero + "," + segundo + ")";
    }

}
```

Indica qué se escribe en la salida estándar al ejecutar el siguiente programa:

```
public class PruebaPar {

    public static void métodoA(Par par) {
        par = par.intercambia();
    }

    public static Par métodoB(Par par) {
        return par.intercambia();
    }

    public static Par métodoC(Par par) {
        par.intercambia();
        return par;
    }

    public static void main(String[] args) {

        Par a = new Par(2, 3);
        métodoA(a);
        System.out.println("A: " + a);

        Par b1 = new Par(4, 5),
            b2 = métodoB(b1);
        System.out.println("B1: " + b1);
        System.out.println("B2: " + b2);

        Par c1 = new Par(6, 7),
            c2 = métodoC(c1);
        System.out.println("C1: " + c1);
        System.out.println("C2: " + c2);

    }

}
```

Ejercicio 2 — 0,5 puntos

Indica qué se escribe en la salida estándar al ejecutar el siguiente programa y describe brevemente para qué sirve el método `enigma`.

```
public class Enigma {

    public static void main(String[] args) {
        int[] fibonacci = {1, 1, 2, 3, 5, 8, 13, 21, 34};
        int[] tresdoses = {2, 2, 2};
        int[] aleatorios = {5, 3, 9};
        System.out.println("A: " + enigma(fibonacci));
        System.out.println("B: " + enigma(tresdoses));
        System.out.println("C: " + enigma(aleatorios));
    }

    public static boolean enigma(int[] v) {
        return misterio(v, 1);
    }

    public static boolean misterio(int[] v, int i) {
        if (i >= v.length)
            return true;
        if (v[i] < v[i - 1])
            return false;
        return misterio(v, i + 1);
    }

}
```

Ejercicio 3 — 4,5 puntos

Queremos desarrollar una aplicación que nos permita gestionar un catálogo de libros. De cada libro solamente nos interesa su ISBN (que es un código alfanumérico que lo identifica) y la cantidad de ejemplares. Considera que ya disponemos de la siguiente clase `Libro`:

```
public class Libro {
    private String ISBN;
    private int cantidad;
    public Libro(String elISBN, int cantidadInicial) {
        ISBN = elISBN;
        cantidad = cantidadInicial;
    }
    public String getISBN() { return ISBN; }
    public int getCantidad() { return cantidad; }
    public void setCantidad(int nuevaCantidad) { cantidad = nuevaCantidad; }
}
```

Haciendo uso de esa clase `Libro`, sin modificarla, queremos completar la siguiente clase `Catálogo`:

```
public class Catálogo {
    private Libro[] libros;
    public Catálogo() { libros = new Libro[0]; }
    // Aquí irían los métodos que se piden
}
```

Internamente, los libros se deben guardar *ordenados de menor a mayor por ISBN* en el vector[1]. Para no consumir memoria innecesaria, la longitud del vector debe coincidir en todo momento con la cantidad de elementos almacenados (en particular, la longitud será 0 si no hay libros). Teniendo en cuenta eso, te pedimos implementar los siguientes métodos, que se deben poder utilizar como ilustran los ejemplos. Si quieres hacer uso de cualquier otro método de la clase `Catálogo`, debes implementarlo también.

En cada uno de los siguientes apartados **solo se aceptan soluciones eficientes**, que tengan en cuenta que el vector de libros está ordenado por ISBN. Además, debes expresar, empleando la notación O, el tiempo de ejecución de tu solución en cada apartado.

a) **[1 punto]** Un método privado `consultarPosición` que tenga como parámetro un ISBN. Si el vector contiene un libro con ese ISBN, devolverá su posición en el vector. En caso contrario, devolverá la posición en la que debería ir ese libro si se insertase ordenadamente en el vector. Como caso particular, la posición devuelta podría coincidir con la talla actual del vector si ese ISBN fuera mayor que todos los del vector.

Ejemplo de uso: `int posición = consultarPosición(ISBN);`

b) **[0,5 puntos]** Un método público `consultarEjemplares` que tenga como parámetro un ISBN y devuelva la cantidad de ejemplares del libro cuyo ISBN coincida con ese. Si no existe en el catálogo, devolverá cero.

Ejemplo de uso:
```
Catálogo miCatálogo = new Catálogo();
// Aquí podría ir código que inserta datos en miCatálogo
int ejemplares = miCatálogo.consultarEjemplares("0-937383-18-X");
```

c) **[1,5 puntos]** Un método público `añadir` que tenga como parámetros un ISBN y una cantidad de ejemplares. El método debe comprobar si ya existe un libro con ese ISBN. Si es así, se incrementará su cantidad de ejemplares sumando la cantidad dada. Si no es así, se insertará el nuevo libro en el catálogo, manteniendo el orden por ISBN.

Ejemplo de uso: `miCatálogo.añadir("0-937383-18-X", 5);`

d) **[1,5 puntos]** Un método público `añadir` que tenga como parámetro otro catálogo. El método debe añadir todos los ejemplares de ese catálogo, incrementando su cantidad si ya existen o insertándolos en caso contrario. Como antes, los libros deben quedar ordenados por ISBN en un vector cuya longitud coincida con la cantidad de libros diferentes. El catálogo dado como parámetro no se debe modificar y los cambios posteriores en uno de los catálogos no deben afectar al otro. Recuerda que solo se aceptan soluciones eficientes y que los dos vectores están ordenados.

Ejemplo de uso: `miCatálogo.añadir(catálogoRegalado);`

[1] Recuerda que, si `c1` y `c2` son de tipo `String`, `c1.compareTo(c2)` devuelve un número negativo, cero o un número positivo según `c1` sea menor que, igual o mayor que `c2`, respectivamente, de acuerdo con el orden lexicográfico de las cadenas.

Ejercicio 4 — 4,5 puntos

Queremos desarrollar una aplicación que nos permita gestionar una agenda de tareas pendientes. Cada tarea tiene una fecha y una descripción. Considera que ya disponemos de la clase `Fecha` (que, entre otros, proporciona un método público `compareTo`[2]) y de la siguiente clase `Tarea`:

```
public class Tarea {
    private Fecha fecha;
    private String descripción;
    public Tarea(Fecha laFecha, String laDescripción) {
        fecha = laFecha;
        descripción = laDescripción;
    }
    public Fecha getFecha() { return fecha; }
    public String getDescripción() { return descripción; }
}
```

Haciendo uso de esa clase `Tarea`, sin modificarla, queremos completar la siguiente clase `Agenda`:

```
public class Agenda {
    private Nodo primerNodo;
    // Aquí irían los métodos que se piden
}
```

Internamente, las tareas se guardan *ordenadas de menor a mayor por fecha* en una lista simplemente enlazada de nodos en la que cada nodo tiene un dato de tipo `Tarea`. Considera además que la definición de la clase `Nodo` incluye únicamente lo siguiente:

```
Tarea tarea;
Nodo sig; // Referencia al Nodo siguiente
Nodo(Tarea tarea, Nodo sig) { this.tarea = tarea; this.sig = sig; }
```

Te pedimos implementar los siguientes métodos públicos, que se deben poder utilizar como ilustran los ejemplos. Si quieres hacer uso de cualquier otro método de las clases `Nodo` o `Agenda`, debes implementarlo también.

En cada uno de los siguientes apartados **solo se aceptan soluciones eficientes**, que tengan en cuenta que la lista de tareas está ordenada por fecha. Además, debes expresar, empleando la notación O, el tiempo de ejecución de tu solución en cada apartado.

a) **[1,5 puntos]** Un método `consultarPeorFecha`, sin parámetros, que devuelva la fecha con más tareas pendientes. En caso de que dos o más fechas tengan el máximo número de tareas pendientes, el método puede devolver cualquiera de esas fechas. Si la agenda está vacía debe devolver `null`.

Ejemplo de uso:
```
Agenda agendaTrabajo = new Agenda();
// Aquí podría ir código que inserta datos en agendaTrabajo
Fecha díaNegro = agendaTrabajo.consultarPeorFecha();
```

b) **[1,5 puntos]** Un método `consultarTareas` que tenga como parámetro una fecha y devuelva como resultado un vector de cadenas que contenga las descripciones de las tareas cuya fecha coincida con esa. Como caso particular, la longitud del vector devuelto será cero si no se encuentra ninguna tarea con esa fecha.

Ejemplo de uso:
```
String[] cosasDelDíaNegro = agendaTrabajo.consultarTareas(díaNegro);
```

c) **[1,5 puntos]** Un método `eliminarTareas` que tenga como parámetro una fecha y elimine de la lista todas las tareas cuya fecha coincida con esa. Por supuesto, las tareas restantes deben quedar ordenadas por fecha.

Ejemplo de uso:
```
agendaTrabajo.eliminarTareas(díaNegro);
```

Antes de entregar no olvides indicar el tiempo de ejecución en cada apartado de los ejercicios 3 y 4.

[2] 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.