Recuperación Segundo Parcial de IIP (ETSInf)
19 de enero de 2018. Duración: 2 horas y 30 minutos

Nota: El examen se evalúa sobre 10 puntos, pero su peso específico en la nota final de IIP es de **3,6 puntos**

NOMBRE:	GRUPO:

1. 6 puntos Se dispone de la clase `Cita` para representar citas en la agenda de una clínica dental. Cada cita corresponde a una intervención a realizar en la clínica. La franja horaria de apertura de la clínica es de 8 de la mañana a 8 de la tarde sin interrupción. Se programan tres tipos de intervenciones: extracción de pieza, limpieza bucal y revisión de ortodoncia. La duración estimada de cada intervención es de 60 minutos para las extracciones, 30 minutos para las limpiezas y 20 minutos para las revisiones de ortodoncia. Para evitar que la clínica cierre más tarde de las 20h, las citas tendrán como hora de inicio cualquiera entre las 08:00h y las 19:00h. Se supone que la clase `Cita` está implementada y lista para ser utilizada. A continuación se muestra un resumen de su documentación, junto con la de la clase `Instante`:

Clase `Cita`:

Field Summary

Fields

Modifier and Type	Field and Description
`static int`	`EXTRACCION` Constante para codificar el tipo de intervención extracción de pieza, su valor es 0.
`static int`	`LIMPIEZA` Constante para codificar el tipo de intervención limpieza bucal, su valor es 1.
`static int`	`ORTODONCIA` Constante para codificar el tipo de intervención revisión de ortodoncia, su valor es 2.

Constructor Summary

Constructors

Constructor and Description
`Cita(int tipo, java.lang.String nombrePaciente, int hora, int minutos)` Crea una cita del tipo de intervención `tipo`, con el nombre del paciente `nombrePaciente`, con hora prevista de inicio indicada por los parámetros `hora` y `minutos`.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`int`	`compareTo(Cita otra)` Devuelve `> 0` si la hora prevista de inicio de `this` es posterior a `otra`, `< 0` si es anterior, y `0` cuando ambas horas de inicio coinciden.
`int`	`getDuracion()` Devuelve la duración prevista en minutos según el tipo de intervención.
`Instante`	`getInicio()` Devuelve el instante de inicio previsto de la intervención.
`java.lang.String`	`getNombre()` Devuelve el nombre del paciente de la cita en curso.
`int`	`getTipo()` Devuelve el tipo de intervención.

Clase `Instante`:

Constructor Summary

Constructors

Constructor and Description
`Instante(int h, int m)` Crea un `Instante` con `h` horas y `m` minutos.

Method Summary

All Methods | **Instance Methods** | **Concrete Methods**

Modifier and Type	Method and Description
`int`	`aMinutos()` Devuelve el número de minutos transcurridos desde las 00:00 hasta el `Instante` en curso.

Se pide: Implementar la clase tipo de datos `Agenda` para representar la programación diaria de una clínica dental en un día determinado. En dicha clase se utilizarán los siguientes atributos y métodos:

a) (0,5 puntos) Atributos:

- `MAX_CITAS`: atributo de clase público constante de tipo entero que indica el máximo de intervenciones que pueden programarse en un día. Este máximo es $34 = (19 - 8) * 3 + 1$, teniendo en cuenta que la duración mínima prevista de una intervención es de 20 minutos, caben tres por hora y, por tanto, caben 33 desde las 8h a las 19h. Falta añadir la última que puede programarse a las 19h por si es una extracción que dura 60 minutos.
- `numCitas`: atributo de instancia privado de tipo entero en el intervalo `[0..MAX_CITAS]` que indica el número de intervenciones planificadas en la agenda en un momento dado.
- `citas`: atributo de instancia privado, un array de tipo base `Cita` para almacenar las citas de un día; las citas se deben disponer en posiciones contiguas del array desde la 0 hasta la `numCitas - 1` inclusive y por orden de hora prevista de inicio, sin que haya solapamiento temporal entre ellas, es decir, que una cita no finalice más tarde de la hora de inicio de la siguiente cita en el array.
- `numOrtodoncias`: atributo de instancia privado de tipo entero que indica el número de revisiones de ortodoncia planificadas.

b) (0,5 puntos) Un constructor por defecto (sin parámetros) que crea el array e inicializa a 0 el número de citas planificadas y el número de ortodoncias.

c) (2,0 puntos) Un método con cabecera o perfil:

```
public boolean insertar(Cita c)
```

que intenta añadir la cita `c` en la agenda, es decir, intenta insertar la cita en el array `citas`. Para simplificar este método, se asumirá como precondición que la nueva cita `c` no se solapa con ninguna de las ya existentes en el array. El método deberá comprobar si hay espacio en el array y que la cita tiene una hora de inicio dentro de la franja permitida. Asumiendo que las citas previamente existentes en el array están ordenadas temporalmente, la nueva cita debe insertarse en la posición que le corresponda, por tanto, si es necesario, deberán desplazarse hacia la derecha las citas ya existentes que tengan una hora de inicio posterior a la nueva cita. El método `insertar` debe devolver `true` cuando ha sido posible insertar la nueva cita, y `false` en caso contrario.

d) (1,0 puntos) Método con cabecera o perfil:

```
public int getNumExtracciones()
```

que devuelva un entero indicando el número de intervenciones del tipo extracción de pieza planificadas para el día.

e) (2,0 puntos) Un método con cabecera o perfil:

```
public String[] getNombresDeOrtodoncia()
```

que devuelva un array de `String` con los nombres de los pacientes para revisión de ortodoncia. La longitud de este array debe ser igual al número de citas planificadas para revisión de ortodoncia, o 0 si no hubiera ninguna.

Solución:

```
/**
 * Clase <code>Agenda</code> para gestionar las citas de una clínica
 * dental. Las citas se mantienen ordenadas por hora estimada de inicio.
 * La franja horaria de atención al público es de 8 de la mañana a 8
 * de la tarde. La última cita que puede asignarse cada día es a las 19h,
 * para asegurar que se cierra a las 20h.
 *
 * Habrá tres tipos de cita, a saber, limpieza bucal, extracción de
 * pieza y ortodoncia. Cada cita tendrá una duración estimada en
 * función del tipo de cita, 20 minutos para revisiones de ortodoncia,
 * 30 minutos para limpiezas y 60 minutos para las extracciones.
 *
 * @author IIP
 * @version Curso 2017-18
 */
public class Agenda {
    /** Constante para indicar el máximo número de citas del día. */
    public static final int MAX_CITAS = (19 - 8) * 3 + 1;

    /** Array para almacenar las citas planificadas en la clínica. */
    private Cita[] citas;
```

```
    /** Contador de citas almacenadas en el array citas. */
    private int numCitas;
    /** Contador de citas almacenadas en el array citas
        que son del tipo Cita.ORTODONCIA. */
    private int numOrtodoncias;

    /**
     * Crea una agenda para gestionar las posibles intervenciones
     * en una clínica dental para un mismo día.
     * Inicialmente la agenda estará vacía, no contendrá ninguna
     * cita. Éstas se irán añadiendo progresivamente.
     */
    public Agenda() {
        this.numCitas = 0;
        this.numOrtodoncias = 0;
        this.citas = new Cita[MAX_CITAS];
    }

    /**
     * Inserta una nueva cita en la agenda ubicándola en la posición
     * que le corresponde dentro del array para asegurar que las citas
     * quedan dispuestas en la agenda en orden cronológico.
     * Precondición: la cita <code>c</code> no solapa con ninguna de
     * las ya existentes en el array <code>citas</code>.
     */
    public boolean insertar(Cita c) {
        // Si no queda espacio no se inserta y se devuelve false.
        if (numCitas == citas.length) { return false; }

        // Si la hora de inicio no está dentro del horario de apertura
        // se devuelve false.
        if (c.getInicio().aMinutos() < 8 * 60
            || c.getInicio().aMinutos() > 19 * 60) { return false; }

        // Se busca la posición que le corresponde a la nueva cita.
        int pos = 0;
        while (pos < numCitas  &&  c.compareTo(citas[pos]) > 0) { pos++; }

        // Se desplazan a la derecha una posición todas las citas que
        // cronológicamente son posteriores a la nueva cita, es decir,
        // las que están en el subarray [pos, numcitas - 1].
        for (int i = numCitas; i > pos; --i) { citas[i] = citas[i - 1]; }

        // Se coloca la nueva cita en la posición que le corresponde.
        citas[pos] = c;

        // Se incrementa el número de citas almacenadas en la agenda.
        ++numCitas;

        // Se incrementa el número de citas almacenadas del tipo
        // Cita.ORTODONCIA si es el caso.
        if (c.getTipo() == Cita.ORTODONCIA) { ++numOrtodoncias; }

        return true;
    }

    /** Devuelve el número de interveciones del tipo extracción. */
    public int getNumExtracciones() {
        int contador = 0;
        for (int i = 0; i < numCitas; i++) {
            if (citas[i].getTipo() == Cita.EXTRACCION) { ++contador; }
        }
        return contador;
    }

    /**
     * Devuelve un array con los nombres de los pacientes que tienen
     * revisión de ortodoncia.
     */
    public String[] getNombresDeOrtodoncia() {
        String[] nombres = new String[numOrtodoncias];
        int k = 0;
        for (int i = 0; i < numCitas && k < numOrtodoncias; i++) {
            if (citas[i].getTipo() == Cita.ORTODONCIA) {
                nombres[k++] = citas[i].getNombre();
            }
        }
        return nombres;
    }
}
```

2. [2 puntos] **Se pide:** implementar un método estático que, dado un entero `n` $\geq$ `3`, escriba en la salida estándar una figura de `n` líneas, en cada una de las cuales se deberán escribir tres caracteres `'N'` con la separación adecuada para que tenga apariencia de letra N mayúscula. Por ejemplo, para `n = 5`, se debe escribir:

```
NN     N
N N    N
N  N   N
N   N  N
N    NN
```

Solución:

```
/** Precondición: n >= 3 */
public static void letra(int n) {
    for (int i = 1; i <= n; i++) {
        System.out.print('N');
        for (int j = 1; j < i; j++) {
            System.out.print(' ');
        }
        System.out.print('N');
        for (int j = 1; j <= n - i; j++) {
            System.out.print(' ');
        }
        System.out.println('N');
    }
}
```

3. [2 puntos] **Se pide:** implementar un método estático que, dado un array de enteros, determine si es **Par Dominante**, esto es, si la suma del valor absoluto de cada uno de los elementos que ocupan posiciones pares, es mayor que todos y cada uno de los elementos, en valor absoluto, que ocupan posiciones impares.

Solución:

```
public static boolean parDominante(int[] v) {
    boolean seguir = true;
    int suma = 0, j = 1;
    for (int i = 0; i < v.length; i = i + 2) {
        suma = suma + Math.abs(v[i]);
    }
    while (j < v.length && seguir) {
        if (Math.abs(v[j]) >= suma) { seguir = false; }
        else { j = j + 2; }
    }
    return seguir;
}
```