Segundo Parcial de IIP (ETSInf). Recuperación
21 de enero de 2020. 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 Recientemente se ha celebrado en el Palacio de Congresos de Madrid la COP25 sobre el cambio climático. Se desea implementar la gestión de esta cumbre con una aplicación en Java. Para ello, se dispone de la clase `Event` que representa cada una de las actividades propuestas por los organizadores que desean participar en la cumbre. La información de un evento viene dada por el tipo del evento (exposición o debate), su hora de inicio, su hora de finalización, el título del evento y quién lo organiza.

   Se muestra, a continuación, un resumen de su documentación, con sus constantes y un extracto de sus métodos públicos:

Fields

Modifier and Type	Field	Description
`static int`	**`DEBATE`**	Evento de tipo DEBATE con valor 0
`static int`	**`EXPOSITION`**	Evento de tipo EXPOSITION con valor 1

Constructors

Constructor	Description
`Event``(TimeInstant start, int dur, java.lang.String org, java.lang.String tit, int type)`	Crea un evento `Event` a partir del instante de inicio `start`, duración `dur` (en minutos), organizador `org`, título del evento `tit` y tipo del evento (DEBATE o EXPOSITION).

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

Modifier and Type	Method	Description
`int`	**`getDuration`**`()`	Devuelve la duración en minutos del evento.
`TimeInstant`	**`getEndTime`**`()`	Devuelve el instante de finalización del evento.
`TimeInstant`	**`getStartTime`**`()`	Devuelve el instante de inicio del evento.
`java.lang.String`	**`getTitle`**`()`	Devuelve el título del evento.
`int`	**`getType`**`()`	Devuelve el tipo de evento.
`void`	**`updateTime`**`(TimeInstant newStart)`	Se actualiza la hora de inicio del evento a la nueva hora indicada, reajustándose también su hora de finalización, pero manteniéndose su misma duración y resto de atributos.

   Se pide: implementar la clase `Schedule` para representar la programación de eventos del primer día de la cumbre. Los atributos y métodos de la clase a implementar son los que se indican a continuación:

   a) (0.5 puntos) Atributos:

   - `MAX_EVENTS`, atributo de clase público, estático y constante de tipo `int`, que representa el número máximo de eventos que se pueden planificar en el día y que vale `30`.
   - `numEvents`, atributo de instancia privado de tipo `int` en el intervalo `[0..MAX_EVENTS]` que indica el número total de eventos programados para el primer día de la cumbre.
   - `program`, atributo de instancia privado de tipo array de objetos de la clase `Event` y tamaño `MAX_EVENTS`, que almacena los eventos que se han programado para el primer día de la cumbre, dispuestos en posiciones consecutivas del array desde la `0` hasta la `numEvents - 1`. Un nuevo evento `Event` siempre se dispone en el array a continuación del último previamente guardado, coincidiendo su hora de inicio con la de finalización de ese último evento guardado. Este aspecto debe tenerse especialmente en cuenta en los métodos `addEvent` y `deleteEvent` que se describen más adelante.

b) (0.5 puntos) Un constructor por defecto que crea el array `program` e inicializa a 0 el número de eventos programados para el primer día de la cumbre.

c) (1 punto) Un método con perfil:

```
public int searchTitle(String title)
```

que devuelve la posición en el array `program` del último evento con el título indicado que se haya planificado en la programación de la COP25. En caso de no existir ningún evento con dicho título, devuelve el valor `-1`.

d) (1.5 puntos) Un método con perfil:

```
public boolean addEvent(String org, int type, int duration, String title)
```

que intenta añadir un nuevo evento a la programación del primer día de la COP25. Como precondición se supone que $0 \leq$ `type` ≤ 1. Además, se deben tener en cuenta las siguientes restricciones:

- Solamente se pueden añadir debates que duren como máximo 120 minutos, y exposiciones que duren como máximo 60 minutos.
- Si el evento se puede añadir a la programación, se añadirá siempre al final del programa, es decir, en la primera posición libre del array `program`. Además, su hora de inicio coincidirá con la hora de finalización del último evento ya programado.
- Si fuera el primer evento de la programación, entonces empezará a las 8:00h.

El método devuelve `true` si se ha conseguido realizar la inserción del evento en el programa de la COP25. En caso contrario, devuelve `false`.

e) (1.5 puntos) Un método con perfil:

```
public boolean deleteEvent(String title)
```

que elimina de la programación el evento con el título indicado, desplazando una posición a la izquierda en el array `program` todos los eventos posteriores en dicho array, reajustándose sus horas de inicio y final, para que así en la programación no quede ningún hueco de horario. El método devuelve `true` si se ha conseguido realizar la eliminación del evento de la programación del primer día de la COP25. En caso contrario, devuelve `false`.

f) (1 punto) Un método con perfil:

```
public int numExpositions()
```

que devuelve la cantidad de eventos de tipo `EXPOSITION` que se han programado para el primer día de la COP25.

Solución:

```
public class Schedule {
    public static final int MAX_EVENTS = 30;
    private int numEvents;
    private Event[] program;

    public Schedule() {
        program = new Event[MAX_EVENTS];
        numEvents = 0;
    }

    public int searchTitle(String title) {
        int i = numEvents - 1;
        while (i >= 0 && !program[i].getTitle().equals(title)) { i--; }
        return i;
    }

    /** Precondición: 0 <= type <= 1 */
    public boolean addEvent(String org, int type, int duration, String title) {
        if (numEvents == MAX_EVENTS) { return false; }
        if ((type == Event.DEBATE && duration > 120)
            || (type == Event.EXPOSITION && duration > 60)) { return false; }

        TimeInstant start;
        if (numEvents > 0) { start = program[numEvents - 1].getEndTime(); }
        else { start = new TimeInstant(8, 0); }

        program[numEvents] = new Event(start, duration, org, title, type);
        numEvents++;
        return true;
    }
```

```
    public boolean deleteEvent(String title) {
        int pos = searchTitle(title);
        if (pos == -1) { return false; }
        TimeInstant start = program[pos].getStartTime();
        for (int i = pos; i < numEvents - 1; i++) {
            program[i] = program[i + 1];
            program[i].updateTime(start);
            start = program[i].getEndTime();
        }
        numEvents--;
        program[numEvents] = null;
        return true;
    }

    public int numExpositions() {
        int num = 0;
        for (int i = 0; i < numEvents; i++) {
            if (program[i].getType() == Event.EXPOSITION) { num++; }
        }
        return num;
    }
}
```

2. 2 puntos Dado un número entero $n > 0$, se desea mostrar por pantalla todos sus divisores de menor a mayor. Para ello, el algoritmo propuesto plantea realizar sucesivas divisiones por 1, 2, 3, 4, etc., de manera que cada división exacta proporciona 2 divisores válidos de n: el propio divisor y, a su vez, también el cociente obtenido. El proceso termina cuando el cociente de una determinada división es más pequeño que el divisor empleado, y ya no hace falta mirar si dicha división es exacta o no.

   Por ejemplo, si n es 15, entonces la secuencia de divisiones a realizar sería:

$$\begin{array}{r|l} 15 & 1 \\ \hline 0 & 15 \end{array} \quad \begin{array}{r|l} 15 & 2 \\ \hline 1 & 7 \end{array} \quad \begin{array}{r|l} 15 & 3 \\ \hline 0 & 5 \end{array} \quad \begin{array}{r|l} 15 & 4 \\ \hline 3 & 3 \end{array} \quad \text{FIN (el proceso se detiene porque } 3 < 4)$$

   De la primera división nos salen el 1 y el 15 como divisores de 15, de la tercera división obtenemos el 3 y el 5. Por tanto, ordenados de menor a mayor, los divisores de 15 son 1 3 5 15 y así es como deben salir en pantalla.

   Para ello, se debe usar un array de tipo `boolean` (de tamaño $n+1$ para manejar índices en el rango $[1..n]$) que, siguiendo con el ejemplo de $n = 15$, y tras aplicar el algoritmo propuesto, presente la siguiente composición:

false	true	false	true	false	true	false	false	false	false	false	false	false	false	false	true
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

   indicando con un valor `true` que la posición de la casilla correspondiente es uno de los divisores válidos de n.

   Se pide: implementar un método público estático que, dado el parámetro `n`, muestre por pantalla el resultado deseado, y que, además, devuelva el array booleano construido con los divisores válidos de `n`.

Solución:

```
    /** Precondición: n > 0 */
    public static boolean[] divisores(int n) {
        boolean[] a = new boolean[n + 1];

        int d = 1, c = n;
        while (d <= c) {
            if (n % d == 0) {
                a[d] = a[c] = true;
            }
            d++;
            c = n / d;
        }

        for(int i = 1; i <= n; i++) {
            if (a[i]) { System.out.print(i + " "); }
        }
        return a;
    }
```

3. 2 puntos **Se pide:** implementar un método público estático que determine si todos los caracteres de una cadena dada `msg` pertenecen a cierto alfabeto `alf` representado como un array de `char`, siendo ambos parámetros del método. El método devuelve `true` en caso afirmativo, y `false` en caso contrario.

   Por ejemplo, dado un alfabeto `alf = {'a', 'c', 'g', 't'}`, si `msg` es `"gattaca"` el método debe devolver `true`, pero si `msg` es `"gattuca"` debe devolver `false`.

   Nota: recuerda que el método de instancia `charAt(int n)` devuelve el carácter de la posición `n` de un `String`.

Solución:

```
public static boolean matches(String msg, char[] alf) {
    boolean res = true;
    int i = 0;
    while (i < msg.length() && res) {
        char c = msg.charAt(i);
        int j = 0;
        while (j < alf.length && alf[j] != c) { j++; }
        if (j >= alf.length) { res = false; }
        i++;
    }
    return res;
}
```