Segundo Parcial de IIP (ETSInf)
8 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 `ProgramaRadio`, que representa un programa de radio de una determinada cadena, y que se emite en algún momento entre las 00:00 y las 23:59 de un mismo día. Entre los datos del programa, la clase incluye el instante de inicio y final, y el tipo de programa. Se supone que ningún programa de radio termina más allá de la medianoche del día en que se inicia. Esta clase es conocida de usos previos y a continuación se muestra un resumen de su documentación, junto con la de la clase `Instante`:

Class ProgramaRadio

Field Summary

static int	MAGAZINE Constante para codificar el tipo de programa magazine (0)
static int	MUSICA Constante para codificar el tipo de programa musical (1)
static int	NOTICIAS Constante para codificar el tipo de programa de noticias (2)

Constructor Summary

ProgramaRadio(int tip, java.lang.String tit, int hora, int min, int duracion) Crea el programa de radio a partir del tipo, título, la hora y minutos de inicio, y la duración en minutos.

Method Summary

int	compareTo(ProgramaRadio p) Devuelve un valor > 0 si el horario de emisión de this es posterior al de p, < 0 si es anterior al de p, y 0 si coinciden.
Instante	getFin() Devuelve el instante de finalización.
Instante	getIni() Devuelve el instante de inicio.
int	getTipo() Devuelve el tipo de programa.

Class Instante

Constructor Summary

Instante(int h, int m) Crea un Instante con h horas y m minutos.

Method Summary

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 `Parrilla` que representa la parrilla o programación diaria de una cadena de radio en un día determinado, usando los siguientes atributos y métodos:

a) (0,5 puntos) Atributos:

- `MAX_PROGS`: atributo de clase público constante de tipo entero, que indica el máximo número de programas que puede haber en un día de programación, teniendo en cuenta que la duración mínima permitida de un programa es de 15 minutos.
- `programas`: atributo de instancia privado de tipo array de `ProgramaRadio`, y que almacena los programas de la parrilla diaria de la cadena; los programas se deben disponer en posiciones contiguas del array desde la 0 en adelante y por orden de emisión, sin que haya solapamiento temporal entre ellos (i.e., el instante de fin del programa en la posición i es menor o igual que el de inicio del programa en la posición $i + 1$).
- `nProg`: atributo de instancia privado de tipo entero que indica el número de programas presentes en la parrilla, esto es, el número de programas almacenados en el array `programas` en un momento dado.

b) (0,5 puntos) Un constructor por defecto (sin parámetros) que crea el array e inicializa a 0 el número de programas presentes.

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

`public boolean insertar(ProgramaRadio p)`

que intenta añadir `p` en la parrilla (i.e., insertarlo en el array `programas`), considerando como precondición que `p` no se solapa con ninguno de los programas presentes en el array. No se insertarán programas demasiado breves, de modo que el número de programas presentes no excederá nunca el máximo de programas que caben en la programación diaria. El método debe actuar de la siguiente forma:

- Si el programa dura menos de 15 minutos, no se debe insertar.
- En caso contrario, el programa se debe insertar ordenadamente, moviendo una posición hacia la derecha los programas posteriores temporalmente.

Si la inserción se ha llevado a cabo, el método debe devolver `true`, y `false` en caso contrario.

d) (1,5 puntos) Método con cabecera o perfil:

`public int[] numDeCadaTipo()`

que devuelva un array de `int` que cuente en cada componente el número de programas de un cierto tipo (`MAGAZINE`, `MUSICA` y `NOTICIAS`), que aparecen en la parrilla. Se entiende que las componentes del array resultante vienen numeradas por el tipo de programa (recordar que precisamente los tipos de programa que existen se codifican por enteros de 0 en adelante).

Por ejemplo, si los tipos de los sucesivos programas almacenados en el array fuesen `NOTICIAS`, `MUSICA`, `MUSICA`, `MAGAZINE`, `NOTICIAS`, y `MUSICA` debe devolver el array {`1,3,2`}.

e) (1,5 puntos) Un método con cabecera o perfil:

`public int progMasLargo()`

que devuelva la posición en el array del programa de mayor duración. Como precondición, se supondrá que en la parrilla hay al menos un programa. En caso de haber más de uno con la máxima duración, se devolverá el primero de ellos.

Solución:

```
public class Parrilla {
    public static final int MAX_PROGS = 24 * 4;
    private ProgramaRadio[] programas;
    private int nProg;

    public Parrilla() {
        programas = new ProgramaRadio[MAX_PROGS];
        nProg = 0;
    }

    /** Precondición: p no se solapa con ningún programa presente en la parrilla. */
    public boolean insertar(ProgramaRadio p) {
        // Comprobación de que p cumple la duración mínima exigida:
        if ((p.getFin().aMinutos() - p.getIni().aMinutos()) < 15) { return false; }

        // Búsqueda de la posición en la que insertar p, desplazando
        // a la derecha los programas que son posteriores temporalmente.
        int i = nProg - 1;
        while (i >= 0 && p.compareTo(programas[i]) < 0) {
            programas[i + 1] = programas[i];
            i--;
        }
        // Inserción de p en la posición que le corresponde:
        programas[i + 1] = p;
        nProg++;
        return true;
    }

    public int[] numDeCadaTipo() {
        int[] conts = new int[3];
        for (int i = 0; i < nProg; i++) {
            conts[programas[i].getTipo()]++;
        }
        return conts;
    }
```

```
    /**  Precondición: la parrilla contiene al menos un programa. */
    public int progMasLargo() {
        int pmax = 0, max = programas[0].getFin().aMinutos() - programas[0].getIni().aMinutos();
        for (int i = 1; i < nProg; i++) {
            int aux = programas[i].getFin().aMinutos() - programas[i].getIni().aMinutos();
            if (aux > max) {
                max = aux;
                pmax = i;
            }
        }
        return pmax;
    }
}
```

2. [2 puntos] Para $n \geq 0$, las funciones enteras exponencial k^n y factorial $n!$, se pueden calcular respectivamente por las siguientes recurrencias:

$$\begin{array}{ll} a_0 = 1 & b_0 = 1 \\ a_n = k \cdot a_{n-1}, \quad n > 0 & b_n = b_{n-1} \cdot n, \quad n > 0 \end{array}$$

La función factorial b_n crece más rápidamente que la exponencial a_n, es decir, a partir de un cierto n (mayor cuanto más grande sea k), los términos $b_n > a_n$.

Se pide: escribir un método estático que reciba como parámetro un entero $k > 1$, y que muestre en la salida, línea a línea, los sucesivos términos:

a_0 b_0
a_1 b_1
a_2 b_2
...

hasta mostrar la primera línea en la que el factorial supera a la exponencial. Por ejemplo, para $k = 3$, el método debería escribir:

```
1       1
3       1
9       2
27      6
81      24
243     120
729     720
2187    5040
```

En la solución no se podrán usar métodos de la librería `Math` de Java.

Solución:

```
    /** Precondición: k > 1. */
    public static void compara(int k) {
        int n = 0;
        int a = 1, b = 1;
        while (a >= b) {
            System.out.println( a + "\t\t" + b);
            n++;
            b = b * n;
            a = k * a;
        }
        System.out.println(a + "\t\t" + b);
    }
```

3. [2 puntos] **Se pide:** escribir un método estático que dado un array de `double` con al menos una componente, devuelva `true` cuando todas las parejas de componentes simétricas en el array sumen lo mismo, y `false` en caso contrario. En el caso en que el array tenga longitud impar se debe considerar que el elemento central es simétrico de sí mismo (i.e., se debe sumar consigo mismo). Ejemplos:

Para los arrays {`3.0,2.9,2.1,2.0`}, {`-1.0,2.0,-1.0,-4.0,-1.0`}, {`3.0,-2.0,-1.0,-6.0`}, {`1.3,4.5`} o {`4.5`} tiene que devolver `true`;

para {`1.0,2.5,5.0`}, {`1.0,2.5,4.0,2.0,3.0`}, {`1.0,2.4,4.6,-3.0`}, o {`-1.0,2.0,-2.0,-4.0,-1.0`} debe devolver `false`.

Solución:

```
/** Precondición: a.length >= 1. */
public static boolean sumSimetric(double[] a) {
    int i = 0,  j = a.length - 1;
    double sum = a[i] + a[j];
    while (i <= j && sum == a[i] + a[j]) {
        i++;
        j--;
    }
    return (i > j);
}
```