Segon Parcial d'IIP (ETSInf)
8 de gener de 2018. Duració: 2 hores i 30 minuts

Nota: L'examen s'avalua sobre 10 punts, però el seu pes específic en la nota final d'IIP és **de 3,6 punts**

NOM:	GRUP:

1. 6 punts Es disposa de la classe `ProgramaRadio`, que representa un programa de ràdio d'una determinada cadena, i que s'emet en algun moment entre les 00:00 i les 23:59 d'un mateix dia. Entre les dades del programa, la classe inclou l'instant d'inici i final, i el tipus de programa. Se suposa que cap programa de ràdio acaba més enllà de la mitjanit del dia en què s'inicia. Aquesta classe és coneguda d'usos previs i a continuació es mostra un resum de la seua documentació, juntament amb la de la classe `Instant`:

Class ProgramaRadio

Field Summary

static int	**MAGAZINE** Constant per a codificar el tipus de programa magazine (0)
static int	**MUSICA** Constant per a codificar el tipus de programa musical (1)
static int	**NOTICIES** Constant per a codificar el tipus de programa de notícies (2)

Constructor Summary

ProgramaRadio(int tip, java.lang.String tit, int hora, int min, int duracio) Crea el programa a partir del tipus, del títol, de l'hora i minuts d'inici i de la duració en minuts.

Method Summary

int	**compareTo**(ProgramaRadio p) Retorna un valor > 0 si l'horari d'emissió de this és posterior al de p, < 0 si és anterior al de p, i 0 si coincideixen.
Instant	**getFi**() Retorna l'instant de finalització.
Instant	**getIni**() Retorna l'instant d'inici.
int	**getTipus**() Retorna el tipus de programa.

Class Instant

Constructor Summary

Instant(int h, int m) Crea un `Instant` amb `h` hores i `m` minuts.

Method Summary

int	**aMinuts**() Torna el número de minuts transcorreguts des de les 00:00 fins l'`Instant` en curs.

Es demana: Implementar la classe tipus de dades `Graella` que representa la graella o programació diària d'una cadena de ràdio en un dia determinat, usant els següents atributs i mètodes:

a) (0,5 punts) Atributs:

- `MAX_PROGS`: atribut de classe públic constant de tipus enter, que indica el màxim número de programes que pot haver-hi en un dia de programació, tenint en compte que la duració mínima permesa d'un programa és de 15 minuts.
- `programes`: atribut d'instància privat de tipus array de `ProgramaRadio`, i que emmagatzema els programes de la graella diària de la cadena; els programes s'han de disposar en posicions contigües de l'array des de la 0 en endavant i per ordre d'emissió, sense que hi haja solapament temporal entre ells (i.e., l'instant de finalització del programa en la posició i és menor o igual que el d'inici del programa en la posició $i+1$).
- `nProg`: atribut d'instància privat de tipus enter que indica el número de programes presents en la graella, açò és, el número de programes emmagatzemats en l'array `programes` en un moment donat.

b) (0,5 punts) Un constructor per defecte (sense paràmetres) que crea l'array i inicialitza a 0 el número de programes presents.

c) (2 punts) Un mètode amb capçalera o perfil:

`public boolean inserir(ProgramaRadio p)`

que intenta afegir `p` a la graella (i.e., inserir-lo en l'array `programes`), considerant com a precondició que `p` no se solapa amb cap dels programes presents en l'array. No s'inseriran programes massa breus, de manera que el número de programes presents no excedirà mai el màxim de programes que caben en la programació diària. El mètode ha d'actuar de la següent forma:

- Si el programa dura menys de 15 minuts, no s'ha d'inserir.
- En cas contrari, el programa s'ha d'inserir ordenadament, desplaçant una posició cap a la dreta els programes posteriors temporalment.

Si la inserció s'ha dut a terme, el mètode ha de retornar `true`, i `false` en cas contrari.

d) (1,5 punts) Un mètode amb capçalera o perfil:

`public int[] numDeCadaTipus()`

que retorne un array d'`int` que compte en cada component el número de programes d'un cert tipus (`MAGAZINE`, `MUSICA` i `NOTICIES`), que apareixen en la graella. S'entén que les components de l'array resultant vénen numerades pel tipus de programa (recordar que precisament els tipus de programa que existeixen es codifiquen per enters des de 0 endavant).

Per exemple, si els tipus dels successius programes emmagatzemats en l'array són `NOTICIES`, `MUSICA`, `MUSICA`, `MAGAZINE`, `NOTICIES`, i `MUSICA` ha de retornar l'array {`1,3,2`}.

e) (1,5 punts) Un mètode amb capçalera o perfil:

`public int progMesLlarg()`

que retorne la posició en l'array del programa de major duració. Com a precondició, se suposarà que en la graella hi ha almenys un programa. En cas de haver-ne més d'un amb la màxima duració, es retornarà la posició del primer d'ells.

Solució:

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

    public Graella() {
        programes = new ProgramaRadio[MAX_PROGS];
        nProg = 0;
    }

    /** Precondició: p no se solapa amb cap programa present en la graella. */
    public boolean inserir(ProgramaRadio p) {
        // Comprovació de que p cumpleix la duració mínima exigida:
        if ((p.getFi().aMinuts() - p.getIni().aMinuts()) < 15) { return false; }

        // Cerca de la posició en la que inserir p, desplaçant
        // cap a la dreta els programes que són posteriors temporalment.
        int i = nProg - 1;
        while (i >= 0 && p.compareTo(programes[i]) < 0) {
            programes[i + 1] = programes[i];
            i--;
        }
        // Inserció de p en la posició que li correspon:
        programes[i + 1] = p;
        nProg++;
        return true;
    }

    public int[] numDeCadaTipus() {
        int[] conts = new int[3];
        for (int i = 0; i < nProg; i++) {
            conts[programes[i].getTipus()]++;
        }
        return conts;
    }
```

```
    /**  Precondició: la graella conté almenys un programa. */
    public int progMesLlarg() {
        int pmax = 0, max = programes[0].getFi().aMinuts() - programes[0].getIni().aMinuts();
        for (int i = 1; i < nProg; i++) {
            int aux = programes[i].getFi().aMinuts() - programes[i].getIni().aMinuts();
            if (aux > max) {
                max = aux;
                pmax = i;
            }
        }
        return pmax;
    }
}
```

2. 2 punts Per a $n \geq 0$, les funcions enteres exponencial k^n i factorial $n!$, es poden calcular respectivament per les següents recurrències:

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

La funció factorial b_n creix més ràpidament que l'exponencial a_n, és a dir, a partir d'un cert n (major com més gran siga k), els termes $b_n > a_n$.

Es demana: escriure un mètode estàtic que reba com a paràmetre un enter $k > 1$, i que mostre en l'eixida, línia a línia, els successius termes:

a_0 b_0
a_1 b_1
a_2 b_2
...
fins a mostrar la primera línia en la qual el factorial supera a l'exponencial. Per exemple, per a $k = 3$, el mètode hauria d'escriure:

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

En la solució no es podran usar mètodes de la llibreria `Math` de Java.

Solució:

```
    /** Precondició: 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 punts **Es demana:** escriure un mètode estàtic que donat un array de `double` amb almenys una component, retorne `true` quan totes les parelles de components simètriques en l'array sumen el mateix, i `false` en cas contrari. En el cas en què l'array tinga longitud senar s'ha de considerar que l'element central és simètric d'ell mateix (i.e., s'ha de sumar amb ell mateix). Exemples:

Per als 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`} ha de retornar `true`;

per a {`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`} ha de retornar `false`.

Solució:

```
/** Precondició: 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);
}
```