Segon Parcial d'IIP (ETSInf). Recuperació
21 de gener de 2020. 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 Recentment s'ha celebrat al Palau de Congressos de Madrid la COP25 sobre el canvi climàtic. Es desitja implementar la gestió d'aquesta cimera amb una aplicació en Java. Per a això, es disposa de la classe `Event` que representa cadascuna de les activitats proposades pels organitzadors que desitgen participar a la cimera. La informació d'un esdeveniment ve donada pel tipus de l'esdeveniment (exposició o debat), la seua hora d'inici, la seua hora de finalització, el títol de l'esdeveniment i qui l'organitza.

   Es mostra, a continuació, un resum de la seua documentació, amb les seues constants i un extracte del seus mètodes públics:

Fields

Modifier and Type	Field	Description
`static int`	**`DEBATE`**	Esdeveniment de tipus DEBAT amb valor 0
`static int`	**`EXPOSITION`**	Esdeveniment de tipus EXPOSITION amb valor 1

Constructors

Constructor	Description
`Event``(TimeInstant start, int dur, java.lang.String org, java.lang.String tit, int type)`	Crea un esdeveniment `Event` a partir de l'instant d'inici `start`, duració `dur` (en minuts), organitzador `org` , títol de l'esdeveniment `tit`, i tipus de l'esdeveniment (DEBATE o EXPOSITION).

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

Modifier and Type	Method	Description
`int`	**`getDuration`**`()`	Retorna la duració en minuts de l'esdeveniment.
`TimeInstant`	**`getEndTime`**`()`	Retorna l'instant de finalització de l'esdeveniment.
`TimeInstant`	**`getStartTime`**`()`	Retorna l'instant d'inici de l'esdeveniment.
`java.lang.String`	**`getTitle`**`()`	Retorna el títol de l'esdeveniment.
`int`	**`getType`**`()`	Retorna el tipus d'esdeveniment.
`void`	**`updateTime`**`(TimeInstant newStart)`	S'actualitza l'hora d'inici de l'esdeveniment a la nova hora indicada, reajustant-se també la seua hora de finalització, però mantenint-se la seua mateixa duració i resta d'atributs.

   Es demana: implementar la classe `Schedule` per a representar la programació d'esdeveniments del primer dia de la cimera. Els atributs i mètodes de la classe a implementar són els que s'indiquen a continuació:

   a) (0.5 punts) Atributs:

   - `MAX_EVENTS`, atribut de classe públic, estàtic i constant de tipus `int`, que representa el nombre màxim d'esdeveniments que es poden planificar en el dia i que val `30`.
   - `numEvents`, atribut d'instància privat de tipus `int` en l'interval `[0..MAX_EVENTS]` que indica el nombre total d'esdeveniments programats per al primer dia de la COP25.
   - `program`, atribut d'instància privat de tipus array d'objectes de la classe `Event` i grandària `MAX_EVENTS`, que emmagatzema els esdeveniments que s'han programat per al primer dia de la COP25, disposats en posicions consecutives de l'array des de la `0` fins la `numEvents - 1`. Un nou `Event` sempre es disposa en l'array a continuació del darrer prèviament guardat, coincidint la seua hora d'inici amb la de finalització d'eixe últim esdeveniment guardat. Cal tindre especialment en compte aquest aspecte en els mètodes `addEvent` i `deleteEvent` que es descriuen més endavant.

b) (0.5 punts) Un constructor per defecte que crea l'array `program` i inicialitza a 0 el nombre d'esdeveniments programats per al primer dia de la COP25.

c) (1 punt) Un mètode amb perfil:

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

que retorna la posició en l'array `program` de l'últim esdeveniment amb el títol indicat que s'haja planificat en la programació de la COP25. En cas de no existir cap esdeveniment amb eixe títol, retorna `-1`.

d) (1.5 punts) Un mètode amb perfil:

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

que intenta afegir un nou esdeveniment a la programació del primer dia de la COP25. Com a precondició s'assumeix que $0 \leq$ `type` ≤ 1. A més, cal tindre en compte les següents restriccions:

- Només es poden afegir debats que duren com a màxim 120 minuts, i exposicions que duren com a màxim 60 minuts.
- Si l'esdeveniment es pot afegir a la programació, s'afegirà sempre al final del programa, és a dir, en la primera posició lliure de l'array `program`. A més, la seua hora d'inici coincidirà amb l'hora de finalització de l'últim esdeveniment ja programat.
- Si fóra el primer esdeveniment de la programació, aleshores començarà a les 8:00h.

El mètode retorna `true` si s'ha aconseguit realitzar la inserció de l'esdeveniment en el programa de la COP25. En cas contrari, retorna `false`.

e) (1.5 punts) Un mètode amb perfil:

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

que elimina de la programació l'esdeveniment amb el títol indicat, desplaçant una posició cap a l'esquerra en l'array `program` tots els esdeveniments posteriors en aquest array, reajustant-se les seues hores d'inici i fi, per tal que així en la programació no quede cap buit en l'horari. El mètode retorna `true` si s'ha aconseguit realitzar l'eliminació de l'esdeveniment de la programació del primer dia de la COP25. En cas contrari, retorna `false`.

f) (1 punt) Un mètode amb perfil:

```
public int numExpositions()
```

que retorna el nombre d'esdeveniments de tipus `EXPOSITION` que s'han programat per al primer dia de la COP25.

Solució:

```
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;
    }

    /** Precondició: 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 punts Donat un número enter $n > 0$, es desitja mostrar per pantalla tots els seus divisors de menor a major. Per a això, l'algorisme proposat planteja realitzar successives divisions per 1, 2, 3, 4, etc., de manera que cada divisió exacta proporciona 2 divisors vàlids de n: el propi divisor i, al seu torn, també el quocient obtingut. El procés finalitza quan el quocient d'una determinada divisió és més xicotet que el divisor usat i, aleshores, ja no cal mirar si aquesta divisió és exacta o no.

   Per exemple, si n és 15, aleshores la sequència de divisions a fer seria:

$$\begin{array}{r|l} 15 & 1 \\ \hline 0 & 15 \end{array} \quad \begin{array}{r|l} 15 & 2 \\ \hline 1 & 7 \end{array} \quad \begin{array}{r|l} 15 & 3 \\ \hline 0 & 5 \end{array} \quad \begin{array}{r|l} 15 & 4 \\ \hline 3 & 3 \end{array} \qquad \text{FI (el procés es deté perquè } 3 < 4)$$

   De la primera divisió obtenim l'1 i el 15 com divisors de 15, de la tercera divisió obtenim el 3 i el 5. Per tant, ordenats de menor a major, els divisors de 15 són 1 3 5 15 i així és com s'han de mostrar per pantalla.

   Per a això, cal utilitzar un array de tipus `boolean` (de grandària $n+1$ per a indexar posicions en el rang $[1..n]$) que, seguint amb l'exemple de $n = 15$, i després d'aplicar l'algorisme proposat, presente la següent composició:

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

   indicant amb un valor `true` que la posició de la casella corresponent és un dels divisors vàlids de n.

   Es demana: implementar un mètode públic estàtic que, donat el paràmetre `n`, mostre per pantalla el resultat desitjat, i que, a més, retorne l'array de `boolean` construït amb els divisors vàlids de `n`.

Solució:

```
    /** Precondició: n > 0 */
    public static boolean[] divisors(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 punts] **Es demana:** implementar un mètode públic estàtic que determine si tots els caràcters d'una cadena donada `msg` pertanyen a cert alfabet `alf` representat com un array de `char`, sent tots dos paràmetres del mètode. El mètode retorna `true` en cas afirmatiu, i `false` en cas contrari.

   Per exemple, donat un alfabet `alf` = {'a', 'c', 'g', 't'}, si `msg` és `"gattaca"` el mètode ha de retornar `true`, però si `msg` és `"gattuca"` ha de retornar `false`.

   Nota: recorda que el mètode d'instància `charAt(int n)` retorna el caràcter de la posició `n` d'un `String`.

Solució:

```
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;
}
```