

NOM:

GRUP:

1. 6 punts Es disposa de la classe **SaveGame**, que representa una partida guardada en una targeta de memòria d'una coneguda videoconsola. Cada *SaveGame* té associat el codi de regió del corresponent videojoc, un número que identifica al videojoc dins de la seua regió, la posició on s'emmagatzema el *SaveGame* i el percentatge de progrés aconseguit. Aquesta classe és coneguda d'usos previs i, a continuació, es mostra part de la seua documentació:

**Constructor Summary**

**Constructors**

**Constructor and Description**

**SaveGame**(java.lang.String r, int n, int p, **TimeInstant** t)  
Crea un objecte *SaveGame* a partir de la seua regió r, identificador del joc n, posició en l'àrea de guardat p i progrés indicat com un temps virtual t que internament es converteix en un percentage.

**Method Summary**

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method and Description	
int	<b>getIdentificador()</b> Torna l'identificador del joc.	
float	<b>getProgres()</b> Torna el progrés de la partida (en percentage).	
java.lang.String	<b>getRegio()</b> Torna la regió del joc.	
void	<b>setPosicio(int p)</b> Actualitza la posició de guardat de la partida a p.	

**Es demana:** implementar la classe tipus de dades **SaveArea** que representa el component de la consola on es guarden les partides, usant els següents atributs i mètodes:

- a) (0,5 punts) Atributs:
- **MAX\_GUARDADES**: atribut de classe públic constant de tipus enter que indica el màxim número de partides que poden guardar-se, sent 100 en aquest cas.
  - **nGuardades**: atribut d'instància privat de tipus enter que indica el número de partides guardades en un moment donat.
  - **guardades**: atribut d'instància privat de tipus array d'objectes **SaveGame** i capacitat **MAX\_GUARDADES**, per tal d'emmagatzemar les partides guardades en un moment donat. Cada instància de **SaveGame** s'emmagatzema en posicions consecutives de l'array, des de la 0 fins la **nGuardades** - 1. Una nova partida sempre s'emmagatzema a continuació de la darrera prèviament guardada. A més, l'atribut **posicio** de cada **SaveGame** ha de ser sempre igual a l'índex de l'element de l'array on es troba emmagatzemat. Açò últim s'ha de tenir especialment en compte en els mètodes **eliminarMesAntiga** i **guardar** que es descriuen més endavant.
- b) (0,5 punts) Un constructor per defecte (sense paràmetres) que crea l'array i inicialitza a 0 el número de partides guardades.
- c) (1 punt) Un mètode amb perfil:

```
private void eliminarMesAntiga()
```

que elimina la partida guardada més antiga, desplaçant la resta una posició cap a l'esquerra en l'array. No tindrà cap efecte si no hi ha cap partida guardada.

d) (1 punt) Un mètode amb perfil:

```
private boolean ambMajorIgualProgresQue(SaveGame s)
```

que torna **true** si ja existeix una partida guardada amb el mateix identificador i un progrés major o igual que el de **s** i, en cas contrari, torna **false**.

e) (1,5 punts) Un mètode amb perfil:

```
public boolean guardar(SaveGame s)
```

que afegeix **s** a les partides prèviament guardades. Per tal de poder guardar-la és necessari que:

1. el seu progrés siga superior al d'altres partides guardades prèviament amb el mateix identificador de joc. El mètode ha de comprovar-ho usant el mètode **ambMajorIgualProgresQue**.
2. hi haja espai disponible. Si l'array està ple, ha d'eliminar-se primer la partida més antiga usant el mètode **eliminarMesAntiga**, encara que siga d'un joc diferent al de **s**.

El mètode torna **true** quan s'ha guardat la partida amb èxit i, en cas contrari, torna **false**.

f) (1,5 punts) Un mètode amb perfil:

```
public SaveGame[] filtrarPerRegio(String regio)
```

que torna un array d'objectes **SaveGame** amb aquelles partides els jocs de les quals siguen de la regió indicada en el paràmetre **regio**. En cas de no haver-hi cap que acomplisca aquest criteri, torna un array buit.

### Solució:

```
public class SaveArea {
    public static final int MAX_GUARDADES = 100;
    private SaveGame[] guardades;
    private int nGuardades;

    public SaveArea() {
        guardades = new SaveGame[MAX_GUARDADES];
        nGuardades = 0;
    }

    private void eliminarMesAntiga() {
        if (nGuardades > 0) {
            for (int j = 0; j < nGuardades - 1; j++) {
                guardades[j] = guardades[j + 1];
                guardades[j].setPosicio(j);
            }
            nGuardades--;
            guardades[nGuardades] = null;
        }
    }

    private boolean ambMajorIgualProgresQue(SaveGame s) {
        int i = nGuardades - 1;
        while (i >= 0 && s.getIdentificador() != guardades[i].getIdentificador()) { i--; }
        return i != -1 && guardades[i].getProgres() >= s.getProgres();
    }

    public boolean guardar(SaveGame s) {
        // No guarda s si ja existeix una partida del mateix joc amb progres major o igual
        if (ambMajorIgualProgresQue(s)) { return false; }

        // Elimina la partida mes antiga si es necessari
        if (nGuardades == MAX_GUARDADES) { eliminarMesAntiga(); }

        // Guarda la nova partida
        guardades[nGuardades] = s;
        s.setPosicio(nGuardades);
        nGuardades++;
        return true;
    }

    public SaveGame[] filtrarPerRegio(String regio) {
        int cont = 0;
        for (int i = 0; i < nGuardades; i++) {
            if (guardades[i].getRegio().equals(regio)) { cont++; }
        }
    }
}
```

```

        SaveGame[] res = new SaveGame[cont];
        int j = 0;
        for (int i = 0; i < nGuardades && j < cont; i++) {
            if (guardades[i].getRegio().equals(regio)) {
                res[j] = guardades[i];
                j++;
            }
        }
        return res;
    }
}

```

2. 2 punts Un número poligonal és un número natural que pot recompondre's en un polígon regular de  $l$  costats. Per exemple, el número 9 és un número quadrat o el número 6 és un número triangular:

```

o o o      o
o o o      o o
o o o      o o o

```

En general, el  $n$ -èsim número poligonal es pot obtindre amb la fórmula:

$$\frac{n * [(l - 2) * n - (l - 4)]}{2}$$

on  $l$  és el número de costats del polígon. Per exemple, per a  $l = 3$ , els números 3-poligonals (triangulars) són 1 3 6 10 15 21 28...

**Es demana:** escriure un mètode estàtic que, donats un número  $k$  ( $k > 0$ ) i el número de costats del polígon  $l$  ( $l > 2$ ), torna **true** si  $k$  és un número  $l$ -poligonal i **false** en cas contrari. Per exemple, si  $k = 15$  i  $l = 3$ , el mètode torna **true** (15 és el 5-èsim número 3-poligonal) però, si  $k = 19$  i  $l = 3$ , el mètode torna **false** (19 no és un número 3-poligonal).

#### Solució:

```

/** Precondició: k > 0 i l > 2 */
public static boolean esNumeroPoligonal(int k, int l) {
    int numPol = 1;
    int i = 2;
    while (numPol < k) {
        numPol = i * ((l - 2) * i - (l - 4)) / 2;
        i++;
    }
    return numPol == k;
}

```

3. 2 punts **Es demana:** escriure un mètode estàtic que tinga com paràmetres un array d'int anomenat **limits** i un array de double anomenat **valors**. El resultat ha de ser un array d'enters amb la mateixa longitud que l'array **limits** i tal que l'element  $i$ -èsim ha de ser igual al número de valors de l'array **valors** que siga inferior a l'element  $i$ -èsim de l'array **limits**.

Per exemple, si els paràmetres són els següents:

- **limits** = {15, 35, 50, 37, 25, 70}
- **valors** = {10.0, 20.0, 50.0, 40.0, 30.0, 80.0}

l'array resultat seria {1, 3, 4, 3, 2, 5}, ja que hi ha:

- 1 valor menor que 15 (10.0)
- 3 valors menors que 35 (10.0, 20.0, 30.0)
- 4 valors menors que 50 (10.0, 20.0, 30.0, 40.0)
- 3 valors menors que 37 (10.0, 20.0, 30.0)
- 2 valors menors que 25 (10.0, 20.0)
- 5 valors menors que 70 (10.0, 20.0, 30.0, 40.0, 50.0)

**Solució:**

```
public static int[] freqAcumulada(int[] limits, double[] valors) {  
    int[] res = new int[limits.length];  
    for (int i = 0; i < valors.length; i++) {  
        for (int j = 0; j < limits.length; j++) {  
            if (valors[i] < limits[j]) { res[j]++; }  
        }  
    }  
    return res;  
}
```