

NOMBRE:

GRUPO:

1. 6 puntos Se dispone de la clase **SaveGame**, que representa una partida guardada en una tarjeta de memoria de una conocida videoconsola. Cada *SaveGame* lleva asociado el código de región del correspondiente videojuego, un número que identifica al videojuego dentro de su región, la posición en la que se almacena el *SaveGame* y el porcentaje de progreso logrado. Esta clase es conocida de usos previos y, a continuación, se muestra parte de su documentación:

Constructor Summary**Constructors****Constructor and Description****SaveGame**(java.lang.String r, int n, int p, TimeInstant t)Crea un objeto *SaveGame* a partir de su región r, identificador del juego n, posición en el área de guardado p y progreso indicado como un tiempo virtual t que internamente se convierte en un porcentaje**Method Summary****All Methods****Instance Methods****Concrete Methods****Modifier and Type****Method and Description**

int

getIdentificador()

Devuelve el identificador del juego

float

getProgreso()

Devuelve el progreso de la partida (en porcentaje)

java.lang.String

getRegion()

Devuelve la región del juego

void

setPosicion(int p)

Actualiza la posición de guardado de la partida a p

Se pide: Implementar la clase tipo de datos **SaveArea** que representa el componente de la consola donde se guardan las partidas, usando los siguientes atributos y métodos:

- a) (0,5 puntos) Atributos:

- **MAX_GUARDADAS:** atributo de clase público constante de tipo entero, que indica el máximo número de partidas que pueden guardarse, siendo 100 en este caso.
- **nGuardadas:** atributo de instancia privado de tipo entero que indica el número de partidas guardadas en un momento dado.
- **guardadas:** atributo de instancia privado de tipo array de objetos de la clase **SaveGame** y capacidad **MAX_GUARDADAS**, para almacenar las partidas guardadas en un momento dado. Cada instancia de **SaveGame** se almacena en posiciones consecutivas del array, desde la 0 hasta **nGuardadas - 1**. Una nueva partida siempre se guarda a continuación de la última previamente guardada. Además, el atributo **posicion** de cada **SaveGame** debe ser siempre igual al índice del elemento del array donde se encuentra guardado. Esto último debe tenerse especialmente en cuenta en los métodos **eliminaMasAntigua** y **guarda** que se describen más adelante.

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

- c) (1 punto) Un método con perfil:

```
private void eliminaMasAntigua()
```

que elimina la partida guardada más antigua, desplazando el resto una posición a la izquierda en el array. No tendrá ningún efecto si no hay ninguna partida guardada.

d) (1 punto) Un método con perfil:

```
private boolean conMayorIgualProgresoQue(SaveGame s)
```

que devuelve `true` si ya existe una partida guardada con el mismo identificador y un progreso mayor o igual que el de `s`, o `false` en caso contrario.

e) (1,5 puntos) Un método con perfil:

```
public boolean guarda(SaveGame s)
```

que añade `s` a las partidas previamente guardadas. Para poder guardarla es necesario que:

- 1 su progreso sea superior al de otras partidas guardadas previamente con el mismo identificador de juego. El método debe comprobarlo usando el método `conMayorIgualProgresoQue`.
- 2 haya espacio disponible. Si el array está lleno, se debe eliminar primero la partida más antigua usando el método `eliminaMasAntigua`, aunque sea de un juego distinto.

El método devuelve `true` cuando se ha guardado la partida con éxito y `false` en caso contrario.

f) (1,5 puntos) Un método con perfil:

```
public SaveGame[] filtrarPorRegion(String region)
```

que devuelve un array de objetos de la clase `SaveGame` con aquellas partidas cuyos juegos sean de la región indicada en el parámetro `region`. De no haber ninguna que cumpla dicho criterio, devuelve un array vacío.

Solución:

```
public class SaveArea {
    public static final int MAX_GUARDADAS = 100;
    private SaveGame[] guardadas;
    private int nGuardadas;

    public SaveArea() {
        guardadas = new SaveGame[MAX_GUARDADAS];
        nGuardadas = 0;
    }

    private void eliminaMasAntigua() {
        if (nGuardadas > 0) {
            for (int j = 0; j < nGuardadas - 1; j++) {
                guardadas[j] = guardadas[j + 1];
                guardadas[j].setPosicion(j);
            }
            nGuardadas--;
            guardadas[nGuardadas] = null;
        }
    }

    private boolean conMayorIgualProgresoQue(SaveGame s) {
        boolean mismoId = false;
        int i = nGuardadas - 1;
        while (i >= 0 && !mismoId) {
            if (s.getIdentificador() == guardadas[i].getIdentificador()) {
                mismoId = true;
            }
            else { i--; }
        }
        return mismoId && guardadas[i].getProgreso() >= s.getProgreso();
    }

    public boolean guarda(SaveGame s) {
        // No guarda s si ya existe una partida del mismo juego con progreso mayor o igual
        if (conMayorIgualProgresoQue(s)) { return false; }

        // Elimina la partida mas antigua si es necesario
        if (nGuardadas == MAX_GUARDADAS) { eliminaMasAntigua(); }

        // Guarda la nueva partida
        guardadas[nGuardadas] = s;
        s.setPosicion(nGuardadas);
        nGuardadas++;
        return true;
    }
}
```

```

public SaveGame[] filtrarPorRegion(String region) {
    int cont = 0;
    for (int i = 0; i < nGuardadas; i++) {
        if (guardadas[i].getRegion().equals(region)) { cont++; }
    }

    SaveGame[] res = new SaveGame[cont];
    int j = 0;
    for (int i = 0; i < nGuardadas && j < cont; i++) {
        if (guardadas[i].getRegion().equals(region)) {
            res[j] = guardadas[i];
            j++;
        }
    }
    return res;
}
}

```

2. 2 puntos Un número poligonal es un número natural que puede recomponerse en un polígono regular de l lados. Por ejemplo, el número 9 es un número cuadrado o el número 6 es un número triangular

```

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

```

En general, el n -ésimo número poligonal se puede obtener con la fórmula

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

donde l es el número de lados del polígono. Por ejemplo, para $l = 3$, los números 3-poligonales (triangulares) son 1 3 6 10 15 21 28...

Se pide: escribir un método estático que, dados un número k ($k > 0$) y el número de lados del polígono l ($l > 2$), devuelva **true** si k es un número l -poligonal y **false** en caso contrario. Por ejemplo, si $k = 15$ y $l = 3$, el método devuelve **true** (15 es el 5-ésimo número 3-poligonal) pero, si $k = 19$ y $l = 3$, el método devuelve **false** (19 no es un número 3-poligonal).

Solución:

```

/** Precondición: 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 puntos **Se pide:** escribir un método estático que tenga como parámetros un array de **int** llamado **limites** y un array de **double** llamado **valores**. El resultado debe ser un array de enteros con la misma longitud que el parámetro **limites** y cuyo i -ésimo elemento debe ser igual al número de valores del array **valores** que sea inferior al elemento i -ésimo del array **limites**.

Por ejemplo, si los parámetros son los siguientes

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

el array resultado sería {1, 3, 4, 3, 2, 5}, ya que hay

- 1 valor menor que 15 (10.0)
- 3 valores menores que 35 (10.0, 20.0, 30.0)

- 4 valores menores que 50 (10.0, 20.0, 30.0, 40.0)
- 3 valores menores que 37 (10.0, 20.0, 30.0)
- 2 valores menores que 25 (10.0, 20.0)
- 5 valores menores que 70 (10.0, 20.0, 30.0, 40.0, 50.0)

Solución:

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