

Primer Parcial de IIP (ETSIInf)
5 de Noviembre de 2018. Duración: 1 hora y 30 minutos

Nota: El examen se evalúa sobre 10 puntos, pero su peso específico en la nota final de IIP es de **2,4 puntos**

NOMBRE:

GRUPO:

1. **6 puntos** Se quiere diseñar una clase Tipo de datos denominada **SaveGame** para representar una partida guardada en una tarjeta de memoria de una conocida videoconsola. Cada *savegame* lleva asociado el código de región del correspondiente videojuego (un **String** de 4 caracteres), un número que identifica al videojuego dentro de su región (un **int** de 5 dígitos), la posición en la que se almacena el *savegame* (un **int** del 1 al 10), y el porcentaje de progreso logrado (de tipo **float**). Para representar el progreso en el juego, se dispone de un reloj virtual que sólo avanza con las acciones correctas del jugador; en otras palabras, cada instante del reloj virtual corresponde con un nivel de progreso concreto, de modo que un juego completado al 100 % tiene una duración virtual de 24 horas. A modo de ejemplo, se transcriben una serie de porcentajes de progreso y su equivalencia en horas y minutos correspondientes:

progreso	0 %	50 %	80 %
reloj	00:00	12:00	19:12

Asimismo, se dispone de la clase **TimeInstant** con la funcionalidad que se muestra en parte, a continuación, en su documentación:

Constructors	
Constructor and Description	
TimeInstant()	Crea un TimeInstant con el valor del instante actual UTC (tiempo universal coordinado).
TimeInstant(int iniHours, int iniMinutes)	Crea un TimeInstant con el valor de las horas y los minutos que recibe como argumentos, iniHours y iniMinutes, respectivamente.
All Methods	Instance Methods
Concrete Methods	
Modifier and Type	Method and Description
int	toMinutes() Devuelve el número de minutos transcurridos desde las 00:00 hasta el instante representado por el objeto en curso.
java.lang.String	toString() Devuelve el TimeInstant en el formato "hh:mm".

Se pide: implementar la clase **SaveGame**, considerando que está en el mismo directorio que la clase **TimeInstant**, con los atributos y métodos que se indican a continuación:

- a) (0.25 puntos) Atributos de clase públicos y constantes:

- **MINUTOS_POR_DIA**, de tipo **int**, con el valor 1440 ($24 * 60$) y que representa el total de minutos de un día.

Esta constante se debe utilizar siempre que se requiera (tanto en la clase **SaveGame** como en la clase **Gestor**).

- b) (0.5 puntos) Atributos de instancia privados: **region** (**String**), **identificador** y **posicion** (**int**), y **progreso** (**float**).
- c) (1.25 puntos) Un constructor general tal que, dados la región y el identificador de un videojuego, la posición de almacenamiento, y un objeto de la clase **TimeInstant** representando la duración virtual de la partida, inicialice todos los atributos de instancia (suponed que todos los datos recibidos son correctos).
- d) (1 punto) Un método **toHHMM()** que devuelva el progreso del objeto *savegame* actual en el formato "hh:mm". Por ejemplo, si el progreso es del 58,9 %, el resultado será "14:08".
- e) (1 punto) Un método **equals** (que sobrescribe el de **Object**) para comprobar si dos *savegames* son iguales, es decir, si tienen el mismo código de región, mismo identificador y mismo progreso, sin considerar sus posiciones.
- f) (1 punto) Un método **toString** (que sobrescribe el de **Object**) para que devuelva el resultado con una estructura como la mostrada en los siguientes ejemplos:

PAL: SCES_507.60 - 1 - 17.3%
USA: SCUS_971.13 - 2 - 24.3%
JAP: SLPS_204.01 - 3 - 58.9%

La estructura es la siguiente: "formato: **region_cociente** - **posicion** - **progreso**", donde **formato** puede ser: PAL (para las regiones SCES y SLES), USA (para las regiones SCUS y SLUS) o JAP, que integraría el resto de regiones; y **cociente** es el valor del atributo **identificador** al dividirlo por 100. El **progreso** se debe expresar con 1 solo decimal.

- g) (1 punto) Se establece una relación de orden entre todos los *savegames* mediante los siguientes criterios:

- En primer lugar, la relación de orden entre *savegames* queda determinada por el orden "alfabético" (lexicográfico) de sus regiones (atributo **region**), es decir, el derivado a partir del método **compareTo** de la clase **String**.
- Si ambos *savegames* tienen la misma **region**, debe ir delante aquel cuyo **identificador** es menor.
- Finalmente, a igualdad de **region** y **identificador**, debe ir delante aquel con un **progreso** menor.

Implementar un método **compareTo** que, dado un parámetro **sg** de tipo **SaveGame**, devuelva un **int** negativo si **this** debe ir delante de **sg**, positivo si **this** debe ir detrás de **sg**, o 0 si el orden de ambos objetos es el mismo.

Solución:

```
public class SaveGame {

    public static final int MINUTOS_POR_DIA = 1440;

    private String region;
    private int identificador, posicion;
    private float progreso;

    public SaveGame(String r, int n, int p, TimeInstant t) {
        region = r;
        identificador = n;
        posicion = p;
        int min = t.toMinutes();
        progreso = ((float) min / MINUTOS_POR_DIA) * 100;
    }

    public String toHHMM() {
        int minTotal = Math.round((progreso / 100) * MINUTOS_POR_DIA);
        int h = minTotal / 60;
        int m = minTotal % 60;
        TimeInstant t = new TimeInstant(h, m);
        return t.toString();
    }

    public boolean equals(Object o) {
        return o instanceof SaveGame
            && this.region.equals(((SaveGame) o).region)
            && this.identificador == ((SaveGame) o).identificador
            && this.progreso == ((SaveGame) o).progreso;
    }

    public String toString() {
        String res;
        if (region.equals("SCES") || region.equals("SLES")) {
            res = "PAL: ";
        }
        else if (region.equals("SCUS") || region.equals("SLUS")) {
            res = "USA: ";
        }
        else { res = "JAP: "; }

        double cociente = identificador / 100.0;
        double p = Math.round(progreso * 10) / 10.0;
        res = res + region + "_" + cociente + " - " + posicion + " - " + p + "%";
        return res;
    }

    public int compareTo(SaveGame sg) {
        int res = this.region.compareTo(sg.region);
        if (res == 0) {
            res = this.identificador - sg.identificador;
            if (res == 0) {
                res = Math.round(this.progreso - sg.progreso);
            }
        }
        return res;
    }
}
```

2. 2 puntos **Se pide:** implementar la clase Programa Gestor, en el mismo directorio que `SaveGame`, con un método `main` que realice las siguientes acciones:

- a) (0.25 puntos) Crear un objeto `t` de tipo `TimeInstant` con la información de la hora UTC actual. Puedes utilizar el constructor por defecto de la clase `TimeInstant`.
- b) (0.5 puntos) A continuación, crear un objeto `sg` de tipo `SaveGame` que tenga la región `SCES`, identificador `50760`, en la posición `1`, y con un nivel de progreso equivalente al instante representado por el objeto `t`.
- c) (1.25 puntos) Por último, imprimir el resultado de la llamada al método `toHHMM()` sobre el objeto `sg`, después, mostrar también por pantalla el resultado de la llamada al método `toString()` sobre el objeto `t`, y finalmente, mostrar por pantalla el resultado de comparar ambas cadenas de texto.

Solución:

```
public class Gestor {
    public static void main(String[] args) {
        TimeInstant t = new TimeInstant();
        SaveGame sg = new SaveGame("SCES", 50760, 1, t);

        String s1 = sg.toHHMM(), s2 = t.toString();
        System.out.println("El tiempo virtual transcurrido es el de " + s1);
        System.out.println("O lo que es lo mismo: " + s2);

        System.out.println("Ambas cadenas son iguales? " + s1.equals(s2));
    }
}
```

3. 2 puntos Se dispone de la clase `Point` que define un punto en un espacio bidimensional real (con dos atributos representando su abscisa y su ordenada), con la funcionalidad que se muestra en parte, a continuación, en su documentación:

Constructors	
Constructor and Description	
Point (double px, double py) Crea un Point con abscisa px y ordenada py.	

All Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description
double	getX()	Devuelve la abscisa del Point this.
double	getY()	Devuelve la ordenada del Point this.
void	setX (double px)	Actualiza la abscisa del Point this a px.
void	setY (double py)	Actualiza la ordenada del Point this a py.
java.lang.String	toString()	Devuelve un String que representa el Point this en el formato típico matemático, i.e., (abscisa,ordenada).

Dada la siguiente clase programa:

```
public class Ejercicio3 {
    public static void main(String[] args) {
        Point p = new Point(1.0, -1.0);
        double x = p.getX();
        double y = p.getY();
        System.out.print("Antes de llamar a cambiaCoords: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());

        cambiaCoords(x, y, p);
        System.out.print("Tras llamar a cambiaCoords 1 vez: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());

        x = p.getY();
        y = p.getX();
        cambiaCoords(x, y, p);
        System.out.print("Tras llamar a cambiaCoords 2 veces: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());
    }

    public static void cambiaCoords(double x, double y, Point p) {
        double z = x;
        x = y;
        y = z;
        p.setX(x);
        p.setY(y);
    }
}
```

Se pide: Completar qué se muestra por pantalla tras su ejecución.

Antes de llamar a `cambiaCoords`: x = 1.0, y = -1.0, p = (1.0, -1.0)

Tras llamar a `cambiaCoords` 1 vez: x = 1.0, y = -1.0, p = (-1.0, 1.0)

Tras llamar a `cambiaCoords` 2 veces: x = 1.0, y = -1.0, p = (-1.0, 1.0)