Primer Parcial de IIP (ETSInf)
5 de Novembre de 2018. Duració: 1 hora i 30 minuts

Nota: Aquest examen s'avalua sobre 10 punts, però el seu pes específic en la nota final de IIP és de **2,4 punts**

NOM:	GRUP:

1. 6 punts Es vol dissenyar una classe de tipus de dades anomenada `SaveGame` per representar una partida emmagatzemada en la targeta de memòria d'una coneguda videoconsola. Cada *savegame* (objecte de la classe `SaveGame`) porta associat un codi de regió on correspon el videojoc (una `String` de 4 caràcters), un nombre que identifica al videojoc dins de la regió (un `int` de 5 dígits), la posició del *savegame* dins de la targeta de memòria (un `int` de l'1 al 10), i el percentatge de progrés aconseguit (un `float` $\in [0, 100]$). Per representar el progrés en el joc, es disposa d'un rellotge virtual que sols avança amb les accions correctes del jugador; en altres paraules, cada instant del rellotge virtual correspon amb un nivell de progrés concret, de manera que un joc completat fins al 100% té una duració virtual de 24 hores. En la següent taula es mostren tres exemples de la equivalència entre el percentatge de progrés i les hores i minuts corresponents:

progrés	0%	50%	80%
rellotge	00:00	12:00	19:12

Així mateix, es disposa de la classe `TimeInstant` amb la funcionalitat que es mostra en part, a continuació, en la seua documentació:

Constructors

Constructor and Description
`TimeInstant()` Crea un objecte de la classe TimeInstant amb el el valor de l'instant actual en UTC (temps universal coordinadat).
`TimeInstant(int iniHours, int iniMinutes)` Crea un objecte de la classe TimeInstant amb el valor de les hores i els minuts que rep com a paràmetres iniHours i iniMinutes, respectivament.

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

Modifier and Type	Method and Description
int	`toMinutes()` Torna el nombre de minuts transcorreguts des de les 00:00 fins l'instant representat per l'objecte en curs.
java.lang.String	`toString()` Torna el TimeInstant en el format "hh:mm".

Es demana: implementar la classe `SaveGame`, considerant que està en el mateix directori que la classe `TimeInstant`, amb els atributs i mètodes que s'indiquen a continuació:

a) (0.25 punts) Constants i variables de classe:

- `MINUTS_PER_DIA`, de tipus `int`, amb el valor `1440` (`24 * 60`) i que representa el total de minuts d'un dia.

Aquesta constant ha d'utilitzar-se sempre que calga, tant en la classe `SaveGame` com en la classe `Gestor`.

b) (0.5 punts) Atributs privats (variables d'instància): `regio` (`String`), `identificador` i `posicio` (`int`), i `progres` (`float`).

c) (1.25 punts) Un constructor genèric tal que, donades la regió i l'identificador d'un videojoc, la posició d'emmagatzemament, i un objecte de la classe `TimeInstant` representant la duració virtual de la partida, inicialitze tots els atributs (variables d'instància). Suposeu que tots els paràmetres rebuts són correctes.

d) (1 punt) Un mètode `toHHMM()` que torne el progrés de l'objecte *savegame* actual en el format `"hh:mm"`. Per exemple, si el progrés és del 58.9%, el resultat serà `"14:08"`.

e) (1 punt) Un mètode `equals` (que sobreescriga el de la classe `Object`) per a comprovar si dos *savegames* són iguals, és a dir, si tenen el mateix codi de regió, identificador numèric i progrés, sense considerar les seues posicions.

f) (1 punt) Un mètode `toString` (que sobreescriga el de la classe `Object`) per tal que torne el resultat amb una estructura com la mostrada en els següents exemples:

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

L'estructura és la següent: `"format: regio_quocient - posicio - progres%"`, on `format` pot ser: `PAL` (per a les regions `SCES` i `SLES`), `USA` (per a les regions `SCUS` i `SLUS`) o `JAP`, que integraria la resta de regions; i `quocient` és el valor de l'atribut `identificador` al dividir-lo per 100. El `progres` ha de mostrar-se amb 1 sol decimal.

g) (1 punts) Es vol definir una relació d'ordre entre tots els *savegames* mitjançant els següents criteris:

- En primer lloc, la relació d'ordre entre *savegames* queda determinada per l'ordre "alfabètic" (lexicogràfic) de les seues regions (atribut `regio`), és a dir, el derivat a partir del mètode `compareTo` de la classe `String`.
- Si ambdós *savegames* tenen la mateixa `regio`, aleshores es considera que va davant aquell *savegame* amb `identificador` menor.
- Finalment, quan `regio` i `identificador` són iguals, es considera que va davant aquell *savegame* amb `progres` menor.

Implementar un mètode `compareTo` que, donat un paràmetre `sg` de tipus `SaveGame`, torne un `int` negatiu si `this` ha d'anar davant de `sg`, positiu si `this` ha d'anar darrere de `sg`, i zero quan l'ordre que els correspon és el mateix.

Solució:

```
public class SaveGame {

    public static final int MINUTS_PER_DIA = 1440;

    private String regio;
    private int identificador, posicio;
    private float progres;

    public SaveGame(String r, int n, int p, TimeInstant t) {
        regio = r;
        identificador = n;
        posicio = p;
        int min = t.toMinutes();
        progres = ((float) min / MINUTS_PER_DIA) * 100;
    }

    public String toHHMM() {
        int minTotal = Math.round((progres / 100) * MINUTS_PER_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.regio.equals(((SaveGame) o).regio)
            && this.identificador == ((SaveGame) o).identificador
            && this.progres == ((SaveGame) o).progres;
    }

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

        double quocient = identificador / 100.0;
        double p = Math.round(progres * 10) / 10.0;
        res = res + regio + "_" + quocient + " - " + posicio + " - " + p + "%";
        return res;
    }

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

2. 2 punts **Es demana**: implementar la classe programa `Gestor`, al mateix directori on estiga `SaveGame`, amb un mètode `main` que faça les següents accions:

   a) (0.25 punts) Crear un objecte `t` de tipus `TimeInstant` amb la informació de l'hora UTC actual. Pots fer ús del constructor per defecte de la classe `TimeInstant`.

   b) (0.5 punts) Tot seguit, crear un objecte `sg` de tipus `SaveGame` amb la regió `SCES`, identificador 50760, en la posició 1 dins la targeta de memòria, i amb un nivell de progrés equivalent a l'instant de temps representat per l'objecte `t`.

   c) (1.25 punts) Per acabar, imprimir el resultat d'invocar al mètode `toHHMM()` sobre l'objecte `sg`, després, mostrar també per pantalla el resultat d'invocar al mètode `toString()` sobre l'objecte `t`, i, finalment, mostrar per pantalla el resultat de comparar ambdues cadenes de text.

Solució:

```
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 temps virtual transcorregut és el de " + s1);
        System.out.println("O el que és el mateix: " + s2);

        System.out.println("Ambdues cadenes de caràcters són iguals? " + s1.equals(s2));
    }
}
```

3. [2 punts] Es disposa de la classe `Point` que defineix un punt en l'espai bidimensional real (amb dos atributs que representen la seua abscissa i la seua ordenada), amb la funcionalitat que es mostra en part, a continuació, en la seua documentació:

Constructors

Constructor and Description
`Point(double px, double py)` Crea un `Point` amb abscissa `px` i ordenada `py`.

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

Modifier and Type	Method and Description
`double`	`getX()` Torna l'abscissa del `Point this`.
`double`	`getY()` Torna l'ordenada del `Point this`.
`void`	`setX(double px)` Actualitza l'abscissa del `Point this` a `px`.
`void`	`setY(double py)` Actualitza l'ordenada del `Point this` a `py`.
`java.lang.String`	`toString()` Torna un `String` que representa el `Point this` en el format típic matemàtic, i.e., (abscissa,ordenada).

Donat el següent programa Java:

```
public class Exercici3 {
    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("Abans d'invocar a canviaCoords: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());

        canviaCoords(x, y, p);
        System.out.print("Després de la primera crida a canviaCoords: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());

        x = p.getY();
        y = p.getX();
        canviaCoords(x, y, p);
        System.out.print("Després de la segona crida a canviaCoords: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());
    }

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

Es demana: Completar què es mostra per pantalla després de executar-lo.

Abans d'invocar a canviaCoords: x = __1.0__, y = __-1.0__, p = (__1.0__, __-1.0__)

Després de la primera crida a canviaCoords: x = __1.0__, y = __-1.0__, p = (__-1.0__, __1.0__)

Després de la segona crida a canviaCoords: x = __1.0__, y = __-1.0__, p = (__-1.0__, __1.0__)