Primer Parcial de IIP (ETSInf)
4 de Noviembre de 2021. 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 **3,75 puntos**

NOMBRE:	GRUPO:

1. 6 puntos Se quiere diseñar una clase Tipo de Datos denominada `VaccineDosis` para representar una dosis de vacuna COVID-19 preparada para inyectar en un centro de vacunación. Cada dosis de vacuna tiene asociados los siguientes elementos: tipo de vacuna, cantidad en ml. de la dosis, instante de tiempo en el que se ha preparado y SIP del paciente al que se le asigne la dosis.

   Para representar el instante de tiempo se dispone de la clase de usuario `TimeInstant`, cuya documentación se muestra –parcialmente– a continuación:

Constructor Summary

Constructors

Constructor	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.

Method Summary

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

Modifier and Type	Method	Description
`boolean`	`equals (java.lang.Object o)`	Devuelve `true` sii `o` es un objeto de la clase `TimeInstant` y sus horas y minutos coinciden con los del objeto en curso.
`java.lang.String`	`toString()`	Devuelve el `TimeInstant` en el formato "`hh:mm`".

Se pide: implementar la clase `VaccineDosis`, considerando que está en el mismo paquete que la clase `TimeInstant`, con los atributos y métodos que se indican a continuación:

a) (0.25 puntos) Tres atributos públicos, estáticos y constantes de tipo entero, para representar mediante un código numérico los tres tipos de vacunas a inyectar en el centro de vacunación. Sus identificadores y valores son, respectivamente: `JANSSEN`, con valor `1`; `ASTRAZENECA`, con valor `2` y `PFIZER`, con valor `3`. Estas constantes deberán ser utilizadas siempre que se requiera (tanto en la clase `VaccineDosis` como en la clase `Vaccines`).

b) (0.5 puntos) Cuatro atributos de instancia y privados, para representar los elementos asociados a una `VaccineDosis`. Siguiendo el orden en el que se han descrito previamente, sus identificadores (y tipos Java) son: `type` (`int`); `dosis` (`double`); `prepTime` (`TimeInstant`) y `sip` (`String`).

c) (0.75 puntos) Un constructor general tal que, dados el tipo de vacuna, su cantidad de dosis, las horas y los minutos en que ha sido preparada, y el SIP del paciente al que se le inyecta la vacuna, inicialice todos los atributos de instancia. Se supone como precondición que los valores de estos parámetros son correctos.

d) (0.5 puntos) Un constructor por defecto, que cree una `VaccineDosis` de tipo *Astrazeneca*, con `0.5` ml. de cantidad de dosis, preparada en el instante actual y sin paciente asignado (representado con la cadena vacía `""`).

e) (1.25 puntos) Método `equals`, que sobrescribe el de `Object` y comprueba si una `VaccineDosis` (`this`) es igual a otra, en concreto, si ambas son del mismo tipo, con la misma cantidad de dosis y se han preparado en el mismo instante de tiempo.

f) (1 punto) Método `compareTo`, que compara una `VaccineDosis` (`this`) con otra `VaccineDosis other`, en base a los criterios de efectividad que figuran a continuación, y devuelve un entero negativo si `this` tiene menos efectividad que `other`, un entero positivo si `this` tiene más efectividad que `other` y `0` si ambas vacunas tienen la misma efectividad.

   Criterios de efectividad:
   - Si se trata de vacunas de distinto tipo, tiene más efectividad `PFIZER`, luego `ASTRAZENECA` y, por último, `JANSSEN`.
   - Si se trata del mismo tipo de vacuna, entonces tiene mayor efectividad la dosis con mayor cantidad de ml.

g) (1.75 puntos) Método `toString`, que sobrescribe el de `Object` y que devuelve la descripción de la dosis de vacuna, es decir, el tipo de vacuna, su cantidad en ml. (con 1 decimal) y el instante de tiempo en el que se ha preparado (en el formato `"hh:mm"`). Además, si la dosis se ha inyectado a alguien, indica también el SIP del paciente. Ejemplos:

```
Vaccine: Astrazeneca 0.5 ml. prepared at 11:15
Vaccine: Pfizer 14.8 ml. prepared at 08:00 injected to SIP = 213422
```

Solución:

```
public class VaccineDosis {
    public static final int JANSSEN = 1, ASTRAZENECA = 2, PFIZER = 3;

    private int type;
    private double dosis;
    private TimeInstant prepTime;
    private String sip;

    public VaccineDosis(int t, double d, int h, int m, String s) {
        type = t;
        dosis = d;
        prepTime = new TimeInstant(h, m);
        sip = s;
    }

    public VaccineDosis() {
        type = ASTRAZENECA;
        dosis = 0.5;
        prepTime = new TimeInstant();
        sip = "";
        // o de manera equivalente:
        // this(ASTRAZENECA, 0.5, 0, 0, "");
        // prepTime = new TimeInstant();
    }

    public boolean equals(Object o) {
        return o instanceof VaccineDosis
            && type == ((VaccineDosis) o).type
            && dosis == ((VaccineDosis) o).dosis
            && prepTime.equals(((VaccineDosis) o).prepTime);
    }

    public int compareTo(VaccineDosis other) {
        int dif = type - other.type;
        if (dif == 0) {
            if (dosis < other.dosis) { dif = -1; }
            else if (dosis > other.dosis) { dif = 1; }
            // otra forma equivalente:
            //dif = (int) Math.signum(dosis - other.dosis);
        }
        return dif;
    }

    public String toString() {
        String res = "Vaccine: ";
        switch (type) {
            case JANSSEN:
                res += "Janssen ";
                break;
            case ASTRAZENECA:
                res += "Astrazeneca ";
                break;
            case PFIZER:
                res += "Pfizer ";
                break;
        }
        double dosisRounded = Math.round(dosis * 10) / 10.0;
        res += " " + dosisRounded + " ml. ";
        res += "prepared at " + prepTime.toString();
        if (!sip.equals("")) { res += " injected to SIP = " + sip; }
        return res;
    }
}
```

2. 2.25 puntos **Se pide**: implementar la clase Programa `Vaccines`, suponiendo que se ubica en el mismo paquete que `VaccineDosis` y `TimeInstant`, con un método `main` que realice las siguientes acciones:

   a) (0.25 puntos) Crear un objeto `vac1` de tipo `VaccineDosis`, que representa una dosis de la vacuna *Astrazeneca* de 0.5 ml. preparada en el instante actual.
   b) (0.75 puntos) Mostrar un mensaje por pantalla para solicitar el SIP de un paciente, así como la cantidad de dosis que se administrará. Leer por teclado ambos valores.
   c) (0.5 puntos) Crear un objeto `vac2` de tipo `VaccineDosis`, que representa una dosis de la vacuna *Pfizer* preparada a las 8:00, con la cantidad de dosis y el SIP del paciente solicitados anteriormente.
   d) (0.75 puntos) Comparar `vac1` con `vac2` usando el método `compareTo` y, en función de su resultado, mostrar un mensaje que indique si es más efectiva la primera dosis, si es más efectiva la segunda dosis, o si ambas dosis son igual de efectivas, según corresponda.

Solución:

```
import java.util.Locale;
import java.util.Scanner;

public class Vaccines {
    private Vaccines() { }

    public static void main(String[] args) {
        VaccineDosis vac1 = new VaccineDosis();

        Scanner scan = new Scanner(System.in).useLocale(Locale.US);
        System.out.print("Indique el SIP del paciente: ");
        String sip = scan.next().trim().toLowerCase();

        System.out.print("Indique la cantidad de dosis a suministrar: ");
        double quantity = scan.nextDouble();
        VaccineDosis vac2 = new VaccineDosis(VaccineDosis.PFIZER, quantity, 8, 0, sip);

        int dif = vac1.compareTo(vac2);
        String res;
        if (dif == 0) { res = "Ambas dosis son igual de efectivas."; }
        else if (dif > 0) { res = "Es más efectiva la primera dosis."; }
        else { res = "Es más efectiva la segunda dosis."; }
        System.out.println(res);
    }
}
```

3. 1.75 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:

Constructor Summary

Constructors

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

Method Summary

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

Modifier and Type	Method	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 tipico matemático, i.e., (abscisa,ordenada).

Se pide: indicar qué se muestra por pantalla tras la ejecución del siguiente código:

```
public class Exercise3 {
    private Exercise3() { }

    public static void main(String[] args) {
        Point p = new Point(3.0, -1.0);
        double x = p.getX();
        double y = p.getY();
        System.out.print("Inicialmente: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());
        myFirstMethod(x, y, p);
        System.out.print("Tras llamar a myFirstMethod: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());
        double a = p.getY();
        double b = p.getX();
        mySecondMethod(a, b, p);
        System.out.print("Tras a llamar a mySecondMethod: ");
        System.out.println("x = " + x + ", y = " + y + ", p = " + p.toString());
    }

    public static void myFirstMethod(double x, double y, Point p) {
        p.setX(y);
        p.setY(x);
    }

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

Solución:

```
Inicialmente: x = 3.0, y = -1.0, p = (3.0, -1.0)
Tras llamar a myFirstMethod: x = 3.0, y = -1.0, p = (-1.0, 3.0)
Tras llamar a mySecondMethod: x = 3.0, y = -1.0, p = (-1.0, 3.0)
```