A

Introducción a la Programación Orientada a Objetos
DCIC - UNS

PRIMER PARCIAL – 07/10/2025

APELLIDO Y N: ..... **LU:** .

CANTIDAD DE HOJAS ENTREGADAS (SIN ENUNCIADO): .

Importante: La interpretación del enunciado es parte del examen.

LA EVALUACIÓN SE REALIZARÁ CONSIDERANDO LA CORRECTITUD, LEGIBILIDAD Y EFICIENCIA DE LAS SOLUCIONES PROPUESTAS.

NO RESUELVA LOS EJERCICIOS EN EL ENUNCIADO. REALICE LOS PROBLEMAS EN HOJAS SEPARADAS.

PROBLEMA 1

EstacionMeteorologica
<<Atributos de instancia>> codigo: entero activa: booleano sensor: Sensor
<<Constructor>> EstacionMeteorologica (cod:entero, s:Sensor) *<<Comandos>>* establecerCodigo(cod:entero) establecerSensor(s:Sensor) copy(e: EstacionMeteorologica) activar() desactivar() *<<Consultas>>* obtenerCodigo(): entero obtenerSensor():Sensor estaActiva():booleano clone():EstacionMeteorologica equals(e: EstacionMeteorologica):booleano estacionMasCalida(e: EstacionMeteorologica): EstacionMeteorologica

Sensor
<<Atributos de instancia>> temperatura: entero humedad: entero viento: entero
<<Constructor>> Sensor(temp, hum, viento:entero) *<<Comandos>* establecerTemperatura(temp:entero) establecerHumedad(hum:entero) establecerViento(v:entero) copy(s:Sensor) *<<Consultas>>* obtenerTemperatura():entero obtenerHumedad():entero obtenerViento():entero equals(s:Sensor):booleano clone():Sensor

Implemente en Java la clase *EstacionMeteorologica* completa, asumiendo que cuenta con la clase *Sensor* totalmente implementada.

- El **constructor** crea una estación meteorológica, asociado al sensor *s*. Requiere *s* ligado y *cod*>0. Una estación está siempre activa cuando se crea.
- **establecerCodigo(cod:entero).** Requiere *cod*>0.
- **establecerSensor(s: Sensor).** Requiere *s* ligado.
- **copy(e: EstacionMeteorologica).** Si el parámetro *e* está ligado modifica el estado interno de la estación que recibe el mensaje; sino no provoca ningún cambio. Los métodos **equals** y **copy** se implementan en profundidad y **clone** de forma superficial.
- **estacionMasCalida(e: EstacionMeteorologica): EstacionMeteorologica.** Retorna la estación cuyo sensor registró la temperatura más alta. En caso de que ambos hayan registrado la misma temperatura, retorna la estación cuyo sensor registró el mayor nivel de humedad. Si también hay coincidencia, devuelve la estación que recibe el mensaje. Requiere *e* ligado.

PROBLEMA 2

a. A partir del diagrama de clases del **PROBLEMA 1**, y considerando que los métodos **equals** y **copy** se implementaron en profundidad, y el método **clone** de forma superficial, dibuje el diagrama de objetos al completarse la ejecución, **indicando el N° de instrucción en cada paso**:

```
1.  Sensor s1, s2, s3, s4;
2.  EstacionMeteorologica e1, e2, e3, e4, e5;
3.  s1 = new Sensor(25, 75, 30);
4.  s2 = new Sensor(27, 65, 30);
5.  e1 = new EstacionMeteorologica(123, s1);
6.  e2 = new EstacionMeteorologica(126, s2);
7.  s3 = s1.clone();
8.  s4 = s3;
9.  e2.desactivar();
10. s4.establecerHumedad(80);
11. e3 = new EstacionMeteorologica(123, s4);
12. e4 = e1;
13. e5 = e2.clone();
14. e1.copy(e3);
15. e2.obtenerSensor().establecerHumedad(50);
```

b. Muestre los valores que computan las siguientes expresiones:

1. e1.equals(e3)
2. e1.obtenerSensor().equals(e2.obtenerSensor())
3. e2.equals(e5)
4. e1==e3
5. e2.obtenerSensor()==e5.obtenerSensor()
6. e1.obtenerSensor()==e3.obtenerSensor()

PROBLEMA 3

PulseraFitness
<<Atributos de clase>> minPasos = 10000 *<<Atributos de instancia>>* registro: entero [ ]
<<Constructor>> PulseraFitness(n: entero) *<<Comandos>>* establecerCantPasos(d, pasos: entero) *<<Consultas>>* obtenerCantDias(): entero obtenerCantPasos(d: entero): entero alMenosNDiasCumplidos(n: entero):booleano diaMasActivo():entero

1. Implemente la clase *PulseraFitness* completa. La clase *PulseraFitness* encapsula un arreglo de enteros, donde cada posición corresponde al total de pasos registrados por una persona en un día. De esta manera, el arreglo permite almacenar y consultar la evolución de la actividad física a lo largo de un período de tiempo. A partir de estos registros, la clase ofrece funcionalidades para analizar el rendimiento del usuario.

   - **PulseraFitness(n: entero).** Requiere *n*>0. La clase cliente ve las posiciones **a partir de 0.**
   - **establecerCantPasos(d, pasos: entero).** Establece la cantidad de *pasos* registrados en el día *d*. Si *d* no es un día válido, o *pasos* es negativo, no produce ningún efecto.
   - **obtenerCantPasos(d: entero): entero.** Retorna la cantidad de pasos registrados en el día *d*. Si *d* no es un día válido, retorna -1.
   - **alMenosNDiasCumplidos(n: entero):booleano.** Retorna verdadero si, y solo si, existen al menos *n* días con una cantidad de pasos mayor o igual a *minPasos*. Requiere *n* > 0.
   - **diaMasActivo():entero.** Retorna el índice del día con la mayor cantidad de pasos registrados. En caso de existir más de un día con el mismo valor máximo, retorna el primero de ellos.

2. Implemente la clase *TesterPulsera* con el objetivo de probar el correcto funcionamiento de la clase *PulseraFitness*:
   a. Cree una instancia de *PulseraFitness* para registrar los pasos correspondientes a **7 días** (una semana) e inicialice el arreglo con valores de pasos diarios a elección.
   b. Verifique el método **alMenosNDiasCumplidos(n: entero): booleano** con **al menos 3 casos significativos.**