Examen de Programación Orientada a Objetos (Plan 2014)

7 de noviembre de 2020

Nº matrícula:_____ **Grupo:**_____ **Nombre:** _____

Apellidos: _____

NOTA: La duración del examen es de 60 minutos.

Teniendo en cuenta la siguiente implementación de la clase `Intervalo`, que permite gestionar intervalos de dos valores y que se considera completamente funcional:

```
class Intervalo {
	// Atributos
	private double extremoInferior;
	private double extremoSuperior;

	// Constructores
	public Intervalo() {
		// nuevo intervalo vacío [0.00, 0.00]
		…
	}
	public Intervalo(double extremoInferior, double extremoSuperior) {
		// nuevo intervalo [extremoInferior, extremoSuperior]
		…
	}
	public Intervalo(Intervalo intervalo) {
		// nuevo intervalo con extremos iguales a los del pasado por parámetro
		…
	}
	public Intervalo(double extremo) {
		// nuevo intervalo [0.00, extremo] o [extremo, 0.00]
		…
	}

	// Métodos públicos
	public double longitud() {
		// devuelve la longitud del intervalo
		return this.extremoSuperior - this.extremoInferior;
	}

	public double puntoMedio() {
		// devuelve el punto medio del intervalo
		return this.extremoInferior + this.longitud() / 2;
	}

	public boolean incluye(double punto) {
		// indica si el punto pasado por parámetro está incluido en el intervalo
		return this.extremoInferior <= punto && punto <= this.extremoSuperior;
	}
}
```

1. (**0.5 puntos**) Definir una propiedad `ORIGEN` que tenga como valor fijo `0.00`, que pueda ser accedida sin instanciar un objeto de la clase y únicamente por cualquier subclase que herede de `Intervalo` o que no heredando de `Intervalo` se encuentre en su mismo paquete.

```
protected static final double ORIGEN = 0.00;
```

2. (**0.5 puntos**) ¿Sería posible convertir el método `puntoMedio` en estático, sin variar su firma? En caso afirmativo: ¿Qué cambios serían necesarios? Justifica la respuesta.

No se puede puesto que necesita de la información dinámica proporcionada por el constructor de la clase.

3. (**1 puntos**) Se pide reescribir el método `toString` heredado de la clase `Object`. La salida del método deberá mostrar el nombre de la clase y los extremos del intervalo con el formato:

   `<nombre de la clase> [<extremo inferior>, <extremo superior>]`

   NOTA: El nombre de una clase se puede obtener con la llamada `getClass().getName()` disponible en la clase `Object`.

```
@Override
public String toString() {
        return this.getClass().getName() +
                " [" + this.extremoInferior + ", " +
                this.extremoSuperior + "]";
}
```

Se considera válido el uso de métodos *getter*.

4. (**1 punto**) Sobrecargar el método `incluye`, para que compruebe si el Intervalo actual incluye/contiene a un Intervalo pasado por parámetro. Se deberá comprobar que el parámetro pasado no es `null`.

```
public boolean incluye(Intervalo intervalo) {
   boolean incluye = false;
   if(intervalo != null) {
        incluye = this.incluye(intervalo.extremoInferior) &&
                  this.incluye(intervalo.extremoSuperior);
        }
   return incluye;
}
```

Se considera válido el uso de métodos *getter*.

Teniendo en cuenta la siguiente implementación de la clase `Intervalo3P`, que permite gestionar intervalos con tres valores:

```
public class Intervalo3P extends Intervalo {
    private double puntoIntermedio;

    public static void main(String[] args) {
        Intervalo3P i1 = new Intervalo3P();
        System.out.println(i1);
    }
}
```

5. (**1 punto**) ¿El código propuesto funcionaría o daría error? Justificar la respuesta en caso de error o indicar la salida en caso de considerar que funcionaría.

```
Intervalo3P [0.00, 0.00]
```

6. (**2 puntos**) Se pide añadir los siguientes constructores a la clase `Intervalo3P`:
   a) Que reciba los dos extremos y el punto intermedio.

```
public Intervalo3P(double extremoInferior,
                   double extremoSuperior,
                   double puntoIntermedio) {
   super(extremoInferior, extremoSuperior);
   this.puntoIntermedio = puntoIntermedio;
}
```

b) Que reciba como parámetro un `Intervalo` y asigne al punto intermedio el punto medio de dicho `Intervalo`.

```
public Intervalo3P(Intervalo intervalo) {
     super(intervalo);
     //this.puntoIntermedio = super.puntoMedio();
     //this.puntoIntermedio = this.puntoMedio();
     this.puntoIntermedio = intervalo.puntoMedio();
}

O

public Intervalo3P(Intervalo intervalo) {
     this(intervalo.getExtremoInferior(),
          intervalo.getExtremoSuperior(),
          intervalo.puntoMedio());
}
```

7. (**1 punto**) Se pide reescribir el método `equals` heredado de la clase `Intervalo`.

```
@Override
public boolean equals(Object obj) {

        return super.equals(obj) &&
               this.puntoIntermedio == ((Intervalo3P)obj).puntoIntermedio;




}

Se considera válido el uso de métodos getter.
```

8. (**1 punto**) ¿Qué cambios serían necesarios realizar en la clase `Intervalo` para convertirla en una clase abstracta? Razonar de forma justificada.

```
Habría que declararla como de tipo abstracto (public abstract class Intervalo) y es
conveniente que al menos uno de sus métodos no se implementara, declarándose también
como abstracto.
```

Teniendo en cuenta la siguiente implementación de la clase `Test`, que permite comprobar el funcionamiento de las clases `Intervalo` e `Intervalo3P`:

```
public class Test {
   public static void main(String[] args) {
        Intervalo i0 = new Intervalo();
        Intervalo i1 = new Intervalo(5.0);
        Intervalo i2 = new Intervalo(4.0, 8.0);
        Intervalo3P i3 = new Intervalo3P();
        Intervalo3P i4 = new Intervalo3P(i2);
        Intervalo3P i5 = new Intervalo3P(4.0, 8.0, 7.0);

        Intervalo lista[] = new Intervalo[]{i0, i1, i2, i3, i4, i5};

        for(Intervalo i: lista){
            System.out.println(i);
        }
    }
}
```

Y teniendo en cuenta que la clase `Intervalo3P`, cuenta con el siguiente método:

```
@Override
public String toString() {
    return super.toString().replaceFirst(",", ", "+this.puntoIntermedio +",");
}
```

Dónde el método `replaceFirst()` reemplaza la primera aparición de la subcadena proporcionada como primer parámetro con la subcadena proporcionada como segundo parámetro.

9. (**1 punto**) ¿El código propuesto funcionaría o daría error? Justificar la respuesta en caso de error o indicar la salida en caso de considerar que funcionaría.

```
Considerando que todos los constructores existen

Intervalo [0.00, 0.00]
Intervalo [0.00, 5.00] o [5.00, 0.00]
Intervalo [4.00, 8.00]
Intervalo3P [0.00, 0.00, 0.00]
Intervalo3P [4.00, 6.00, 8.00]
Intervalo3P [4.00, 7.00, 8.00]

Considerando que sólo existen los constructores definidos en el enunciado

Error porque el constructor vacío no existe.
```

10. (**1 punto**) Teniendo en cuenta que la interfaz `MismaClase`, que define los métodos necesarios para determinar si dos intervalos dados son de la misma clase, se implementa en `Intervalo` e `Intervalo3P`. **Se pide** codificar los métodos en ambas clases.

```
public interface MismaClase {
	boolean mismaClase(Intervalo intervalo);
	boolean esUn(Intervalo intervalo);
	boolean esUn(Intervalo3P intervalo3P);
}
```

NOTAS:

- Un intervalo tiene la `mismaClase` que otro cuando es una instancia de su misma clase.
- No está permitido utilizar el operador `instanceof`.

En `Intervalo`:

```
public boolean mismaClase(Intervalo intervalo){
	return intervalo.esUn(this);
}
public boolean esUn(Intervalo intervalo){
	return true;
}
public boolean esUn(Intervalo3P intervalo3P){
	return false;
}
```

En `Intervalo3P`:

```
public boolean mismaClase(Intervalo intervalo){
	return intervalo.esUn(this);
}
public boolean esUn(Intervalo intervalo){
	return false;
}
public boolean esUn(Intervalo3P intervalo3P){
	return true;
}
```