Introducción a la Programación Orientada a Objetos
PRIMER PARCIAL – 10/5/2025

Apellido y nombre:
LU:
Cantidad de hojas entregadas(sin enunciado).
Ejercicio 1:

Triangulo
<<Atributos de instancia>> p1,p2,p3: Punto
<<Constructor>> Triangulo(punto1,punto2,punto3:Punto) Triangulo() **<<Comandos>>** establecerP1(p: Punto) establecerP2(p: Punto) establecerP3(p: Punto) copy(t: Triangulo) **<<Consultas>>** obtenerP1(): Punto obtenerP2(): Punto obtenerP3(): Punto clone(): Triangulo equals(t: Triangulo):boolean perímetro():real perimetroMayor(t:Triangulo):Triangulo

Punto
<<Atributos de instancia>> x: entero y: entero
<<Constructor>> Punto(coord,ord: entero) **<<Comandos>** establecerX(coord: entero) establecerY(ord: entero) copy(p: Punto) **<<Consultas>>** obtenerX(): entero obtenerY(): entero clone(): Punto equals(p: Punto): boolean distancia(p:Punto):real

Dadas los diagramas de clases de Punto y Triangulo impleméntelas completas en Java teniendo en cuenta la siguiente especificación:

Para la clase Punto

- El método *distancia(p:Punto):real* calcula la distancia entre el punto que recibe el mensaje y el punto pasado por parámetro. La fórmula para calcular la distancia entre dos puntos (x1, y1) y (x2, y2) es: $d = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$. Para calcular la raíz cuadrada puede utilizar el método Math.sqrt(x).

Para la clase Triángulo

- El constructor *Triangulo(punto1,punto2,punto3:Punto)* requiere que los puntos estén ligados y sean válidos para formar un triángulo en el plano cartesiano.
- El constructor *Triangulo()* crea un nuevo triangulo con los puntos (0,0),(1,1),(2,0).
- Los métodos *clone, copy e equals* se implementan en profundidad.
- En los comandos *establecer*, si el parámetro no está ligado el método no deberá hacer nada.
- El método *perimetroMayor(t:Triangulo):Triangulo* retorna el triángulo que tenga mayor perímetro entre el triángulo receptor del mensaje y el pasado por parámetro. Si tienen el mismo perímetro retornará null.
- Todos los métodos que reciben un Triángulo requieren que el parámetro esté ligado.

Ejercicio 2: Considerando las clases del ejercicio 1, con el clone y el copy implementados superficial y el equals en profundidad, dibuje el diagrama de objetos al completarse la ejecución de:

1. Punto p1,p2,p3;
2. Triangulo t1,t2,t3
3. p1= new Punto(1,2);
4. p2= new Punto(1,5);
5. p3= new Punto(8,2);
6. t1= new Triangulo(p1,p2,p3);
7. t2= t1.clone();
8. t3= new Triangulo(new Punto(34,67), p2.clone(),p3.clone());
9. Triangulo t4= new Triangulo();
10. t4.copy(t3);
11. t4.obtenerP1().establecerX(120);
12. Punto p4=p3;

Muestre la salida de las siguientes instrucciones:

1. t1.obtenerP2()==t2.obtenerP2()
2. p2.equals(t3.obtenerP2())
3. t3==t4
4. t1.equals(t2)
5. t3.equals(t4)

Ejercicio 3:

RegistroTemperaturas
<<atributos de instancia>> reg: float []
<<constructor>> RegistroTemperaturas() **<<comandos>>** establecerTemp(mes:int, temp:float) **<<consultas>>** mesMasCaluroso(): entero exactamenteNbajoCero(n:entero):boolean

La clase **RegistroTempraturas** lleva un registro del promedio de temperatura de cada mes del año. Enero es considerado como el mes cero. Implemente esta clase completa teniendo en cuenta que:

- El constructor inicializa el arreglo con todos los meses con un registro de 0.
- El comando *establecerTemp(mes:entero,temp:float)* requiere *mes* válido.
- La consulta *mesMasCaluroso(): entero* retorna el mes en que se registró el promedio de temperaturas más altas, en caso de repetirse deberá devolver el último mes donde se registró la temperatura.
- La consulta *exactamenteNbajoCero(n:entero):boolean* retorna verdadero si el año registró exactamente n promedios de temperaturas menores a 0. Requiere 1<=n<=12.