EJERCICIO 1:

// Clase A

class A {

public void imprimir() {

System.*out*.println("a");

}

}

// Clase B

class B {

public void imprimir() {

System.*out*.println("b");

}

}

// Clase C

class C {

A a = new A();

B b = new B();

public void imprimir() {

a.imprimir(); // Imprime 'a'

b.imprimir(); // Imprime 'b'

System.*out*.println("c"); // Imprime 'c'

}

}

// Clase principal

public class Main {

public static void main(String[] args) {

// Crear tres objetos de la clase C

C obj1 = new C();

C obj2 = new C();

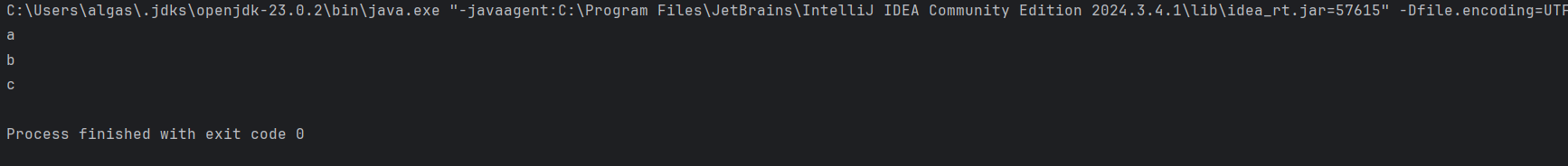
C obj3 = new C();

// Llamar a imprimir una vez (como pediste solo una secuencia a-b-c)

obj1.imprimir();

}

}



EJERCICIO 2:

import java.util.Scanner;

// Clase base con coordenadas 2D

class Punto2D {

protected double x;

protected double y;

public void leerDatos() {

Scanner scanner = new Scanner(System.*in*);

System.*out*.print("Ingrese la coordenada x: ");

x = scanner.nextDouble();

System.*out*.print("Ingrese la coordenada y: ");

y = scanner.nextDouble();

}

public void mostrarDatos() {

System.*out*.println("x = " + x);

System.*out*.println("y = " + y);

}

}

// Clase derivada que agrega coordenada z y color

class Punto3DColor extends Punto2D {

private double z;

private String color;

@Override

public void leerDatos() {

super.leerDatos(); // Leer x e y desde la clase base

Scanner scanner = new Scanner(System.*in*);

System.*out*.print("Ingrese la coordenada z: ");

z = scanner.nextDouble();

scanner.nextLine(); // Limpiar el buffer

System.*out*.print("Ingrese el color: ");

color = scanner.nextLine();

}

@Override

public void mostrarDatos() {

super.mostrarDatos(); // Mostrar x e y

System.*out*.println("z = " + z);

System.*out*.println("color = " + color);

}

}

// Clase principal para ejecutar el programa

public class Main {

public static void main(String[] args) {

Punto3DColor punto = new Punto3DColor();

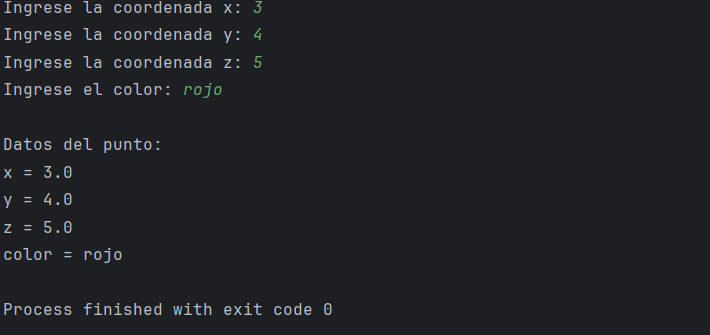
punto.leerDatos();

System.*out*.println("\nDatos del punto:");

punto.mostrarDatos();

}

}



EJERCICIO 3:

No tiene sentido que la clase Orden herede de Cliente, ya que Orden utiliza componentes de Cliente, pero no es un cliente. Al establecer esta relación de herencia, se le están asignando a Orden atributos y comportamientos innecesarios, lo cual va en contra de los principios de una buena arquitectura orientada a objetos. En este caso, sería más apropiado que Orden contenga una instancia de Cliente en lugar de heredar de ella.