class Empleado {

private String nombre;

private String tipo;

private double salario;

public Empleado(String nombre, String tipo, double salario) {

this.nombre = nombre;

this.tipo = tipo;

this.salario = salario;

}

public double calcularPago() {

if (tipo.equals("Gerente")) {

return salario + 1000; // Bono para gerentes

} else if (tipo.equals("Desarrollador")) {

return salario;

} else if (tipo.equals("Practicante")) {

return salario \* 0.5; // Practicantes reciben medio salario

}

return salario;

}

public void guardarEnBaseDeDatos() {

System.out.println("Guardando empleado " + nombre + " en la base de datos...");

}

public void generarReporte() {

System.out.println("Generando reporte para el empleado " + nombre + "...");

}

}

class SistemaGestionEmpleados {

public void procesarEmpleado(Empleado empleado) {

double pago = empleado.calcularPago();

System.out.println("Pago calculado: " + pago);

empleado.guardarEnBaseDeDatos();

empleado.generarReporte();

}

}

public class Main {

public static void main(String[] args) {

Empleado gerente = new Empleado("Juan", "Gerente", 5000);

Empleado desarrollador = new Empleado("Ana", "Desarrollador", 3000);

Empleado practicante = new Empleado("Luis", "Practicante", 1000);

SistemaGestionEmpleados sistema = new SistemaGestionEmpleados();

sistema.procesarEmpleado(gerente);

sistema.procesarEmpleado(desarrollador);

sistema.procesarEmpleado(practicante);

}

}

Primer principio: "Single responsibility", la clase "Empleado", tiene muchísimas funciones y responsabilidades como manejar los datos del empleados, calcular pago, etc, por lo tanto no cumple.

Segundo principio: "Open/Closed", en calcularPago tenemos una estructura if-else que nos dificulta la extensión del código, puesto a que si quisiéramos agregar un nuevo tipo de empleado, tendríamos que modificar la clase, por lo tanto no cumple.

Tercer principio: "Liskov Substitution", al estar todo el código dentro de la clase Empleado, no podríamos siquiera reemplazar una clase hija por la clase padre, por lo tanto no cumple

Cuarto principio: "Interface Segregation", el código obliga a todas sus instancias a tener guardarEnBaseDeDatos() y generarReporte(), métodos que no usa, por lo tanto no cumple.

Quinto principio: "Dependency Inversión", en SistemaGestiónEmpleados se usa directamente la clase Empleados para funcionar, lo que hace que dependa de una implementación concreta, por lo tanto no cumple.

Soluciones

Principio 1: Separar el código en diferentes clases para no darle tanta responsabilidad a la clase Empleado.

Principio 2: Aplicar polimorfismo usando de base la clase Empleado, utilizando clases específicas para cada tipo de empleado.

Principio 3: Crear más clases para separar las funcionalidades.

Principio 4: Usar las lineas de codigo guardarEnBasedeDatos y generarReporte solo en los metodos que necesite, así mismo también usamos interfaces pequeñas.

Principio 5: Dejar de depender de empleados, utilizando interfaces en lugar de implementaciones(clase Empleado).