Trabajo Práctico – POO

Alumno: *Fermin Alliot*
Carrera: Tecnicatura Universitaria en Programación
Materia: Programación II
Comisión: "3"
GitHub: `https://github.com/Falliot00/UTN-TUPaD-P2/tree/main/4.%20POO`

---

1) Enfoque y criterios de diseño

Para resolver el caso *Sistema de Gestión de Empleados* implementé una clase `Empleado` que:
- Usa `this` para desambiguar parámetros y atributos y para encadenar constructores (`this(...)`).
- Ofrece **constructores sobrecargados**: uno completo y otro simplificado (con `id` autoasignado y salario por defecto).
- Define **métodos sobrecargados** `actualizarSalario(...)`: uno interpreta el valor como **porcentaje** y otro como **monto fijo** (sobrecarga por **tipo** del parámetro: `double` vs `int`).
- Sobrescribe `toString()` para mostrar el estado de forma legible.
- Emplea **miembros estáticos** para llevar un contador global (`totalEmpleados`) y para generar `id` automáticos (`nextId`).

---

2) Implementación — Clase `Empleado`

```
public class Empleado {
    // Atributos de instancia
    private int id;
    private String nombre;
    private String puesto;
    private double salario;

    // Atributos estáticos (de clase)
    private static int totalEmpleados = 0;          // contador global
    private static int nextId = 1;                  // generador de IDs
automáticos
    private static final double SALARIO_POR_DEFECTO = 1000000.0;

    // Constructor completo
    public Empleado(int id, String nombre, String puesto, double salario)
{
        this.id = id;
        this.nombre = nombre;
        this.puesto = puesto;
```

```
        this.salario = salario;
        totalEmpleados++;
    }

    // Constructor simplificado: asigna id automático y salario por
defecto
    public Empleado(String nombre, String puesto) {
        this(nextId++, nombre, puesto, SALARIO_POR_DEFECTO);
    }

    // --- Métodos sobrecargados para actualizar salario ---
    // 1) Interpreta el parámetro como PORCENTAJE
    public void actualizarSalario(double porcentaje) {
        if (porcentaje < -100) {
            throw new IllegalArgumentException("El porcentaje no puede
ser menor a -100.");
        }
        this.salario += this.salario * (porcentaje / 100.0);
    }

    // 2) Interpreta el parámetro como MONTO FIJO (sobrecarga por tipo
int)
    public void actualizarSalario(int montoFijo) {
        double nuevo = this.salario + montoFijo;
        if (nuevo < 0) {
            throw new IllegalArgumentException("El salario no puede ser
negativo.");
        }
        this.salario = nuevo;
    }

    // Representación legible del objeto
    @Override
    public String toString() {
        return "Empleado{" +
               "id=" + id +
               ", nombre='" + nombre + '\'' +
               ", puesto='" + puesto + '\'' +
               ", salario=" + String.format("%.2f", salario) +
               '}';
    }

    // Método estático para consultar el total creado
    public static int mostrarTotalEmpleados() {
        return totalEmpleados;
    }

    // --- Getters y Setters ---
    public int getId() { return id; }
    public String getNombre() { return nombre; }
```

```
    public String getPuesto() { return puesto; }
    public double getSalario() { return salario; }

    public void setPuesto(String puesto) { this.puesto = puesto; }
    public void setSalario(double salario) {
        if (salario < 0) throw new IllegalArgumentException("El salario
no puede ser negativo.");
        this.salario = salario;
    }
}
```

---

3) Clase de prueba — `TrabajoPractico4`

```
public class TrabajoPractico4 {
    public static void main(String[] args) {
        // Instancias con ambos constructores
        Empleado e1 = new Empleado(100, "Ana Gómez", "Desarrolladora",
1350000.0);
        Empleado e2 = new Empleado("Carlos Díaz", "QA Analyst"); // id y
salario por defecto
        Empleado e3 = new Empleado("Lucía Pérez", "Tech Lead");

        // Aplicación de métodos sobrecargados
        e1.actualizarSalario(10.0);   // +10%
        e2.actualizarSalario(30000);  // +$30.000 (monto fijo)

        // Impresión de información (usando toString())
        System.out.println(e1);
        System.out.println(e2);
        System.out.println(e3);

        // Total de empleados creados (método estático)
        System.out.println("Total de empleados: " +
Empleado.mostrarTotalEmpleados());
    }
}
```

---

4) Compilación y ejecución

Salida de ejemplo

```
Empleado{id=100, nombre='Ana Gomez', puesto='Desarrolladora',
salario=1485000,00}
Empleado{id=1, nombre='Carlos Diaz', puesto='QA Analyst',
salario=1030000,00}
Empleado{id=2, nombre='Lucia Perez', puesto='Tech Lead',
```

```
salario=1000000,00}
Total de empleados: 3
```