

Trabajo Práctico 4: Programación Orientada a Objetos II

Caso Práctico

Sistema de Gestión de Empleados

Modelar una clase Empleado que represente a un trabajador en una empresa. Esta clase debe incluir constructores sobrecargados, métodos sobrecargados y el uso de atributos aplicando encapsulamiento y métodos estáticos para llevar control de los objetos creados.

CLASE EMPLEADO

Atributos:

- int id: Identificador único del empleado.
- String nombre: Nombre completo.
- String puesto: Cargo que desempeña.
- double salario: Salario actual.
- static int totalEmpleados: Contador global de empleados creados.

REQUERIMIENTOS

1. Uso de this: Utilizar this en los constructores para distinguir parámetros de atributos.

```
public class Empleado {
```

```
    // Atributos privados (encapsulamiento)
```

```
    private int id;
```

```
    private String nombre;
```

```
    private String puesto;
```

```
    private double salario;
```

```
    // Atributo estático
```

```
    private static int totalEmpleados = 0;
```

```
    // Constructor principal
```

```
    public Empleado(int id, String nombre, String puesto, double salario) {
```

```
        this.id = id;
```

```
this.nombre = nombre;

this.puesto = puesto;

this.salario = salario;

totalEmpleados++; // Incrementar contador estático
}

// Constructor sobrecargado (sin salario, valor por defecto)
public Empleado(int id, String nombre, String puesto) {
    this(id, nombre, puesto, 0.0); // Llama al constructor principal
}

// Constructor sobrecargado (solo con id y nombre)
public Empleado(int id, String nombre) {
    this(id, nombre, "Sin asignar", 0.0);
}

// Métodos getters y setters
public int getId() {
    return this.id;
}

public String getNombre() {
    return this.nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}
```

```
public String getPuesto() {  
    return this.puesto;  
}
```

```
public void setPuesto(String puesto) {  
    this.puesto = puesto;  
}
```

```
public double getSalario() {  
    return this.salario;  
}
```

```
public void setSalario(double salario) {  
    this.salario = salario;  
}
```

// Método estático para obtener el total de empleados creados

```
public static int getTotalEmpleados() {  
    return totalEmpleados;  
}
```

// Método sobrecargado: aumentar salario con porcentaje

```
public void aumentarSalario(double porcentaje) {  
    if (porcentaje > 0) {  
        this.salario += this.salario * porcentaje / 100;  
    }  
}
```

// Método sobrecargado: aumentar salario con monto fijo

```

public void aumentarSalario(double monto, boolean esMontoFijo) {
    if (esMontoFijo && monto > 0) {
        this.salario += monto;
    }
}

// Método toString para mostrar info del empleado
@Override
public String toString() {
    return "Empleado{id=" + id + ", nombre=" + nombre + ", puesto=" +
puesto + ", salario=" + salario + "}";
}

// Método main de prueba
public static void main(String[] args) {
    Empleado e1 = new Empleado(1, "Laura Gómez", "Ingeniera", 5000);
    Empleado e2 = new Empleado(2, "Carlos Ruiz", "Analista");
    Empleado e3 = new Empleado(3, "Ana Pérez");

    e2.setSalario(4000);
    e3.setPuesto("Diseñadora");
    e3.setSalario(3500);

    e1.aumentarSalario(10);        // Aumentar salario 10%
    e2.aumentarSalario(500, true); // Aumentar salario $500

    System.out.println(e1);
    System.out.println(e2);
    System.out.println(e3);

```

```
        System.out.println("Total empleados creados: " +  
Empleado.getTotalEmpleados());  
    }  
}
```

2. Constructores sobrecargados: ○ Uno que reciba todos los atributos como parámetros.

- Otro que reciba solo nombre y puesto, asignando un id automático y un salario por defecto.
- Ambos deben incrementar totalEmpleados.

```
public class Empleado {  
    // Atributos privados (encapsulados)  
    private int id;  
    private String nombre;  
    private String puesto;  
    private double salario;  
  
    // Atributos estáticos  
    private static int totalEmpleados = 0;  
    private static int siguienteId = 1; // Para generar IDs automáticos  
  
    // 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++;  
if (id >= siguienteId) {  
    siguienteId = id + 1; // Asegura que el siguiente ID no se repita  
}  
}
```

// Constructor parcial (nombre y puesto), con ID automático y salario por defecto

```
public Empleado(String nombre, String puesto) {  
    this.id = siguienteId++;  
    this.nombre = nombre;  
    this.puesto = puesto;  
    this.salario = 30000.0; // Salario por defecto  
    totalEmpleados++;  
}
```

// Métodos getter y setter

```
public int getId() {  
    return id;  
}
```

```
public String getNombre() {  
    return nombre;  
}
```

```
public void setNombre(String nombre) {  
    this.nombre = nombre;  
}
```

```
public String getPuesto() {  
    return puesto;  
}
```

```
public void setPuesto(String puesto) {  
    this.puesto = puesto;  
}
```

```
public double getSalario() {  
    return salario;  
}
```

```
public void setSalario(double salario) {  
    this.salario = salario;  
}
```

```
// Método estático para obtener total de empleados creados  
public static int getTotalEmpleados() {  
    return totalEmpleados;  
}
```

```
// Método sobrecargado: actualizar salario  
public void actualizarSalario(double nuevoSalario) {  
    this.salario = nuevoSalario;  
}
```

```
public void actualizarSalario(double porcentaje, boolean esPorcentaje) {  
    if (esPorcentaje) {  
        this.salario += this.salario * (porcentaje / 100);  
    }  
}
```

```

    }
}

// Método para mostrar información del empleado
public void mostrarInformacion() {
    System.out.println("ID: " + id);
    System.out.println("Nombre: " + nombre);
    System.out.println("Puesto: " + puesto);
    System.out.println("Salario: $" + salario);
    System.out.println("-----");
}
}

```

3. Encapsulamiento con la Clase Libro

- a. Crear una clase Libro con atributos privados: titulo, autor, añoPublicacion.

Métodos requeridos: Getters para todos los atributos. Setter con validación para añoPublicacion.

Tarea: Crear un libro, intentar modificar el año con un valor inválido y luego con uno válido, mostrar la información final.

```

public class Libro {

    // Atributos privados (encapsulamiento)
    private String titulo;
    private String autor;
    private int añoPublicacion;

    // Constructor
    public Libro(String titulo, String autor, int añoPublicacion) {
        this.titulo = titulo;
        this.autor = autor;
        setAñoPublicacion(añoPublicacion); // usamos el setter para validar
    }
}

```

```
}
```

```
// Getters
```

```
public String getTitulo() {  
    return titulo;  
}
```

```
public String getAutor() {  
    return autor;  
}
```

```
public int getAñoPublicacion() {  
    return añoPublicacion;  
}
```

```
// Setter con validación
```

```
public void setAñoPublicacion(int añoPublicacion) {  
    if (añoPublicacion > 0 && añoPublicacion <= 2025) {  
        this.añoPublicacion = añoPublicacion;  
    } else {  
        System.out.println("Año de publicación inválido: " + añoPublicacion);  
    }  
}
```

```
// Método para mostrar la información del libro
```

```
public void mostrarInfo() {  
    System.out.println("📖 Libro:");  
    System.out.println("Título: " + titulo);  
    System.out.println("Autor: " + autor);  
}
```

```

        System.out.println("Año de publicación: " + añoPublicacion);
        System.out.println("-----");
    }

    // Método principal para probar la clase
    public static void main(String[] args) {
        // Crear un libro
        Libro libro1 = new Libro("Cien años de soledad", "Gabriel García
Márquez", 1967);
        libro1.mostrarInfo();
        // Intentar modificar el año con un valor inválido
        libro1.setAñoPublicacion(3000); // inválido
        libro1.mostrarInfo();
        // Modificar el año con un valor válido
        libro1.setAñoPublicacion(1982); // válido
        libro1.mostrarInfo();
    }
}

```

4. Gestión de Gallinas en Granja Digital a. Crear una clase Gallina con los atributos: idGallina, edad, huevosPuestos.

Métodos requeridos: ponerHuevo(), envejecer(), mostrarEstado().

Tarea: Crear dos gallinas, simular sus acciones (envejecer y poner huevos), y mostrar su estado.

```

public class Gallina {
    // Atributos
    private int idGallina;
    private int edad;        // en meses o años, según prefieras
    private int huevosPuestos;

    // Constructor

```

```

public Gallina(int idGallina, int edadInicial) {
    this.idGallina = idGallina;
    this.edad = edadInicial;
    this.huevosPuestos = 0;
}

// Método para poner un huevo
public void ponerHuevo() {
    huevosPuestos++;
    System.out.println("La gallina " + idGallina + " ha puesto un huevo. Total: "
+ huevosPuestos);
}

// Método para envejecer
public void envejecer() {
    edad++;
    System.out.println(" La gallina " + idGallina + " ha envejecido. Nueva edad:
" + edad);
}

// Método para mostrar el estado de la gallina
public void mostrarEstado() {
    System.out.println("-----");
    System.out.println(" Estado de la Gallina " + idGallina + ":");
    System.out.println("Edad: " + edad);
    System.out.println("Huevos puestos: " + huevosPuestos);
    System.out.println("-----");
}

// Método principal para probar la clase

```

```

public static void main(String[] args) {
    // Crear dos gallinas
    Gallina gallina1 = new Gallina(1, 2);
    Gallina gallina2 = new Gallina(2, 1);

    // Simular acciones
    gallina1.envejecer();
    gallina1.ponerHuevo();
    gallina1.ponerHuevo();

    gallina2.envejecer();
    gallina2.ponerHuevo();

    // Mostrar estado final de cada gallina
    gallina1.mostrarEstado();
    gallina2.mostrarEstado();
}
}

```

5. Simulación de Nave Espacial

Crear una clase NaveEspacial con los atributos: nombre, combustible.

Métodos requeridos: `despegar()`, `avanzar(distancia)`, `recargarCombustible(cantidad)`, `mostrarEstado()`.

Reglas: Validar que haya suficiente combustible antes de avanzar y evitar que se supere el límite al recargar.

Tarea: Crear una nave con 50 unidades de combustible, intentar avanzar sin recargar, luego recargar y avanzar correctamente. Mostrar el estado al final.

```

public class NaveEspacial {

    // Atributos
    private String nombre;

```

```

private int combustible;

private final int LIMITE_COMBUSTIBLE = 100; // máximo permitido

// Constructor

public NaveEspacial(String nombre, int combustibleInicial) {

    this.nombre = nombre;

    if (combustibleInicial <= LIMITE_COMBUSTIBLE) {

        this.combustible = combustibleInicial;

    } else {

        this.combustible = LIMITE_COMBUSTIBLE;

        System.out.println("El combustible inicial excede el límite. Se ajustó a "
+ LIMITE_COMBUSTIBLE);

    }

}

// Método para despegar

public void despegar() {

    if (combustible >= 10) {

        combustible -= 10;

        System.out.println(" La nave " + nombre + " ha despegado. Combustible
restante: " + combustible);

    } else {

        System.out.println(" No hay suficiente combustible para despegar.");

    }

}

// Método para avanzar una distancia determinada

public void avanzar(int distancia) {

    int consumo = distancia / 2; // por ejemplo: 1 unidad de combustible por
cada 2 de distancia

```

```

        if (combustible >= consumo) {
            combustible -= consumo;

            System.out.println(" La nave avanzó " + distancia + " km. Combustible
            restante: " + combustible);
        } else {
            System.out.println(" No hay suficiente combustible para avanzar " +
            distancia + " km.");
        }
    }
}

```

```

// Método para recargar combustible
public void recargarCombustible(int cantidad) {
    if (combustible + cantidad <= LIMITE_COMBUSTIBLE) {
        combustible += cantidad;

        System.out.println(" Se recargaron " + cantidad + " unidades de
        combustible. Total: " + combustible);
    } else {
        combustible = LIMITE_COMBUSTIBLE;

        System.out.println("Se alcanzó el límite máximo de combustible (" +
        LIMITE_COMBUSTIBLE + ").");
    }
}
}

```

```

// Método para mostrar el estado actual de la nave
public void mostrarEstado() {
    System.out.println("-----");
    System.out.println(" Estado de la Nave Espacial");
    System.out.println("Nombre: " + nombre);
    System.out.println("Combustible: " + combustible + " / " +
    LIMITE_COMBUSTIBLE);
    System.out.println("-----");
}

```

```

    }

    // Método principal para probar la simulación
    public static void main(String[] args) {
        // Crear una nave con 50 unidades de combustible
        NaveEspacial nave1 = new NaveEspacial("Explorer", 50);

        // Intentar avanzar sin recargar
        nave1.despegar();
        nave1.avanzar(120); // debería fallar por falta de combustible

        // Recargar y avanzar correctamente
        nave1.recargarCombustible(40);
        nave1.avanzar(60);

        // Mostrar el estado final
        nave1.mostrarEstado();
    }
}

```

Nota: al querer subir las consignas realizadas a github me da un error. Dejo el link al repo que cree <https://github.com/lauoisin/UTN-TUPaD-P2>.