Trabajo Práctico 4: Programación Orientada a Objetos II

Materia: Programación II
Tecnicatura Universitaria en Programación a Distancia
Alumno: Emiliano Fabian Jara

Objetivo General

Comprender y aplicar conceptos de Programación Orientada a Objetos en Java, incluyendo el uso de this, constructores sobrecargados, métodos sobrecargados, encapsulamiento y miembros estáticos, para mejorar la modularidad, reutilización y diseño del código.

Marco Teórico Resumido

- **Uso de this**: Permite referenciar la instancia actual dentro de constructores y métodos.
- **Constructores sobrecargados**: Facilitan la creación flexible de objetos con diferentes parámetros.
- **Métodos sobrecargados**: Permiten definir varias versiones de un método con distintos parámetros.
- **Método toString()**: Proporciona una representación legible del estado de un objeto.
- **Atributos y métodos estáticos**: Compartidos por todas las instancias de una clase; útiles para contadores globales.
- **Encapsulamiento**: Protege los atributos de acceso directo, usando getters y setters.

Implementación de la clase Empleado

```
public class Empleado {

// Atributos privados (encapsulamiento)

private int id;

private String nombre;

private String puesto;
```

```
private double salario;

// Atributo estático para contar empleados
private static int totalEmpleados = 0;

// Constructor completo con todos los atributos
public Empleado(int id, String nombre, String puesto, double salario) {
this.id = id; // uso de this
this.nombre = nombre;
this.puesto = puesto;
this.salario = salario;
totalEmpleados++; // incremento del contador estático
}

// Constructor sobrecargado con nombre y puesto
public Empleado(String nombre, String puesto) {
totalEmpleados++; // incremento del contador estático
this.id = totalEmpleados; // asignación automática de ID
this.nombre = nombre;
this.puesto = puesto;
this.salario = 50000.0; // salario por defecto
}

// Método sobrecargado: actualizar salario por porcentaje
public void actualizarSalario(double porcentaje) {
this.salario += this.salario * (porcentaje / 100);
}
```

```
// Método sobrecargado: actualizar salario por monto fijo
public void actualizarSalario(int montoFijo) {
this.salario += montoFijo;
}

// Método toString para mostrar información legible
@Override
public String toString() {
return "Empleado [ID: " + id + ", Nombre: " + nombre + ", Puesto: " + puesto + ",
Salario: $" + salario + "]";
}

// Método estático para mostrar total de empleados
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 setNombre(String nombre) {
this.nombre = nombre;
}

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

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

Clase de prueba Main

```
public class Main {
public static void main(String[] args) {
// Crear empleados usando constructor completo
Empleado emp1 = new Empleado(101, "Ana Gómez", "Gerente", 120000.0);
Empleado emp2 = new Empleado(102, "Luis Pérez", "Analista", 85000.0);

// Crear empleados usando constructor con nombre y puesto
```

```
Empleado emp3 = new Empleado("María López", "Desarrolladora");
Empleado emp4 = new Empleado("Carlos Ruiz", "Soporte Técnico");

// Aplicar aumento por porcentaje
emp1.actualizarSalario(10); // 10% de aumento
emp3.actualizarSalario(5); // 5% de aumento

// Aplicar aumento por monto fijo
emp2.actualizarSalario(5000); // aumento de $5000
emp4.actualizarSalario(3000); // aumento de $3000

// Mostrar información de cada empleado
System.out.println(emp1.toString());
System.out.println(emp2.toString());
System.out.println(emp3.toString());
System.out.println(emp4.toString());

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

Conclusiones Esperadas

- Se aplicó correctamente el uso de this para distinguir atributos de parámetros.
- Se implementaron constructores sobrecargados para flexibilizar la creación de objetos.
- Se aplicó encapsulamiento mediante atributos privados y métodos públicos.
- Se definieron métodos sobrecargados para actualizar el salario.
- Se utilizó toString() para representar objetos de forma legible.
- Se aplicaron atributos y métodos estáticos para llevar control global de empleados.
- Se reforzó el diseño modular y reutilizable mediante el paradigma orientado a objetos.