PROGRAMACIÓN II

Trabajo Práctico 4: Programación Orientada a Objetos II

Alumno: Victor Barroeta

DNI: 95805903

Comisión: M2025 – 2

Matricula: 100393

Link de Github:
https://github.com/victormbar/UTN_Prog2_VictorBarroeta/tree/main/javaUTN_TPs/src/TPs_JAVA

OBJETIVO GENERAL

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

MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Uso de this	Referencia a la instancia actual dentro de constructores y métodos
Constructores y sobrecarga	Inicialización flexible de objetos con múltiples formas de instanciación
Métodos sobrecargados	Definición de varias versiones de un método según los parámetros recibidos
toString()	Representación legible del estado de un objeto para visualización y depuración
Atributos estáticos	Variables compartidas por todas las instancias de una clase
Métodos estáticos	Funciones de clase invocadas sin instanciar objetos
Encapsulamiento	Restringir el acceso directo a los atributos de una clase

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.
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**.
3. Métodos sobrecargados **actualizarSalario**:
   - Uno que reciba un porcentaje de aumento.
   - Otro que reciba una cantidad fija a aumentar.
4. Método **toString()**:
   - Mostrar id, nombre, puesto y salario de forma legible.
5. Método estático **mostrarTotalEmpleados()**:
   - Retornar el total de empleados creados hasta el momento.
6. Encapsulamiento en los atributos:
   - Restringir el acceso directo a los atributos de la clase.
   - Crear los métodos Getters y Setters correspondientes.

TAREAS A REALIZAR

1. Implementar la clase Empleado aplicando todos los puntos anteriores.
2. Crear una clase de prueba con método main que:
   - Instancie varios objetos usando ambos constructores.
   - Aplique los métodos **actualizarSalario()** sobre distintos empleados.
   - Imprima la información de cada empleado con **toString()**.
   - Muestre el total de empleados creados con **mostrarTotalEmpleados()**.

CONSEJOS

- Usá **this** en los constructores para evitar errores de asignación.
- Probá distintos escenarios para validar el comportamiento de los métodos sobrecargados.
- Asegúrate de que el método **toString()** sea claro y útil para depuración.
- Confirmá que el contador **totalEmpleados** se actualiza correctamente en cada constructor.

```java
package TPs_JAVA_TP4;


public class Empleado {

    private int id;
    private String nombre;
    private String puesto;
    private double salario;
    private static int totalEmpleados;

    public Empleado() {
        totalEmpleados++;
    }

    public Empleado(int id, String nombre, String puesto, double salario) {
        this.id = id;
        this.nombre = nombre;
        this.puesto = puesto;
        this.salario = salario;
        totalEmpleados++;
    }

    public Empleado(String nombre, String puesto) {
        this.nombre = nombre;
        this.puesto = puesto;
        this.id += Math.random()*100;
        this.salario = 2000;
        totalEmpleados++;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }
```

```java
    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;
    }

    public static int getTotalEmpleados() {
        return totalEmpleados;
    }

    public static void setTotalEmpleados(int totalEmpeados) {
        Empleado.totalEmpleados = totalEmpeados;
    }

    public void actualizarSalario(double aumento){
        this.salario = aumento * salario;
    }

    public void actualizarSalario(int aumento){
        this.salario += aumento;
```

```java
        return salario;
    }

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

    public static int getTotalEmpleados() {
        return totalEmpleados;
    }

    public static void setTotalEmpleados(int totalEmpeados) {
        Empleado.totalEmpleados = totalEmpeados;
    }

    public void actualizarSalario(double aumento){
        this.salario = aumento * salario;
    }

    public void actualizarSalario(int aumento){
        this.salario += aumento;
    }

    @Override
    public String toString() {
        return "Empleado(" + "id=" + id + ", nombre=" + nombre + ", puesto=" +
                puesto + ", salario=" + salario + '}';
    }

    public static void mostrarTotalEmpleados(){
        System.out.println("Total de Empleados: \n"+ getTotalEmpleados());
    }

}
```

```java
package TPs_JAVA_TP4;

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("TP #4 -  POO");

        Empleado e1 = new Empleado(1, "Victor Barroeta", "Developer", 2000.0);
        Empleado e2 = new Empleado("Carla Salinas", "Scrum Master");
        Empleado e3 = new Empleado("Andres Andrade", "QA");
        Empleado e4 = new Empleado();

        System.out.println("Ingresa los datos de un empleado:");
        e4.setId(20);
        e4.setNombre(scanner.nextLine());
        e4.setPuesto(scanner.nextLine());
        e4.setSalario(scanner.nextDouble());

        e1.actualizarSalario(0.10); //Se aplica un 10% de aumento
        e2.actualizarSalario(500);

        System.out.println("Empleados: ");
        System.out.println(e1.toString());
        System.out.println(e2.toString());
        System.out.println(e3.toString());
        System.out.println(e4.toString());
        System.out.println("Total de empleados: "+ Empleado.getTotalEmpleados());
    }

}
```

```
run:
TP #4 -  POO
Ingresa los datos de un empleado:
Pablo Cabrera
Administrador
1500
Empleados:
Empleado{id=1, nombre=Victor Barroeta, puesto=Developer, salario=200.0}
Empleado{id=7, nombre=Carla Salinas, puesto=Scrum Master, salario=2500.0}
Empleado{id=63, nombre=Andres Andrade, puesto=QA, salario=2000.0}
Empleado{id=20, nombre=Pablo Cabrera, puesto=Administrador, salario=1500.0}
Total de empleados: 4
BUILD SUCCESSFUL (total time: 18 seconds)
```

CONCLUSIONES ESPERADAS

- Comprender el uso de **this** para acceder a atributos de instancia.
- Aplicar constructores sobrecargados para flexibilizar la creación de objetos.
- Aplicar encapsulamiento en los atributos.
- Implementar métodos con el mismo nombre y distintos parámetros.
- Representar objetos con **toString()** para mejorar la depuración.
- Diferenciar y aplicar atributos y métodos estáticos en Java.
- Reforzar el diseño modular y reutilizable mediante el paradigma orientado a objetos.