Olima, Nicolás Pablo

nicolima200@gmail.com

[Link Repo GitHub](https://github.com/nicolima200/TPs_Progra2doCuatrimestre/tree/main/TP%204)

Resumen

Este trabajo práctico resume conceptos clave de la programación orientada a objetos en Java: uso de this, constructores y sobrecarga, métodos sobrecargados y toString() para representar objetos. También aborda atributos y métodos estáticos y la importancia del encapsulamiento para proteger los datos internos.

PROGRAMACIÓN II: Programación Orientada a Objetos II

TRABAJO PRÁCTICO N°4

**PROGRAMACIÓN II**

**Trabajo Práctico 4: Programación Orientada a Objetos II**

# 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:
   1. Utilizar **this** en los constructores para distinguir parámetros de atributos.
2. Constructores sobrecargados:
   1. 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**.

1. Métodos sobrecargados **actualizarSalario**:
   1. Uno que reciba un porcentaje de aumento.

○ Otro que reciba una cantidad fija a aumentar.

1. Método **toString()**:
   1. Mostrar id, nombre, puesto y salario de forma legible.
2. Método estático **mostrarTotalEmpleados()**:
   1. Retornar el total de empleados creados hasta el momento.
3. Encapsulamiento en los atributos:
   1. 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.

# 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.

***Clase Principal:***

A computer code on a dark background

AI-generated content may be incorrect.

***Clase Empleado***

A screen shot of a computer program

AI-generated content may be incorrect.

A computer code with text

AI-generated content may be incorrect.

***Salida por consola:***

A screenshot of a computer program

AI-generated content may be incorrect.