P R O G R A M A C I O N

CASO PRACTICO I UD3

ALUMNO CESUR 24/25

Alejandro Muñoz de la Sierra

PROFESOR

María Carmen Buenestado Fernández

I N T R O D U C C I O N

Este documento presenta una solución para el caso práctico que se propuso, enfocándose en modelar una organización usando los principios básicos de la Programación Orientada a Objetos (POO). El proceso va desde el diseño de clases con notación UML hasta la implementación en Java, destacando conceptos como herencia, encapsulación y modularidad.

En el desarrollo, se toman decisiones clave, tales como:

Visibilidad: Asignación de niveles de acceso a atributos y métodos para mantener los datos seguros.
Relaciones: Creación de vínculos claros entre clases por medio de herencia.
Pruebas: Generación de instancias y verificación del sistema para asegurar su funcionamiento.
La solución sigue buenas prácticas en desarrollo de software, promoviendo un código ordenado, estructurado y reutilizable. Este ejercicio no solo aborda el problema, sino que también mejora la comprensión y aplicación de los principios de POO, estableciendo una base para proyectos más complicados en el futuro.

0 1

HACER CLASES CON NOTACIÓN UML Y CÓDIGO JAVA

El diseño UML tiene estas clases:

Empleado
(superclase): Tiene atributos y métodos que son comunes a todos los empleados.
Administrativo, Contable, Informático (subclases): Añaden funcionalidades a la superclase con atributos y métodos que son específicos.

Atributos que son comunes en la superclase Empleado:

- idEmpleado (int): Un número único.
- nombre (String): Nombre de la persona que trabaja.
- apellidos (String): Apellidos de la persona que trabaja.
- salario (double): Pagos de la persona que trabaja.

Atributos específicos en cada subclase:

Administrativo:
- áreaDeTrabajo (String): Departamento donde trabaja.
- nivelDeAcceso (int): Grado de privilegio dentro de la entidad.

Contable:
- tipoImpuesto (String): Impuesto que maneja.
- idProyectoAsignado (int): Proyecto financiero relacionado.

Informático:
- lenguajeDeProgramación (String): Lenguaje principal que usa.
- idProyectoAsignado (int): Proyecto tecnológico relacionado.

Constructores y Métodos

Constructores:

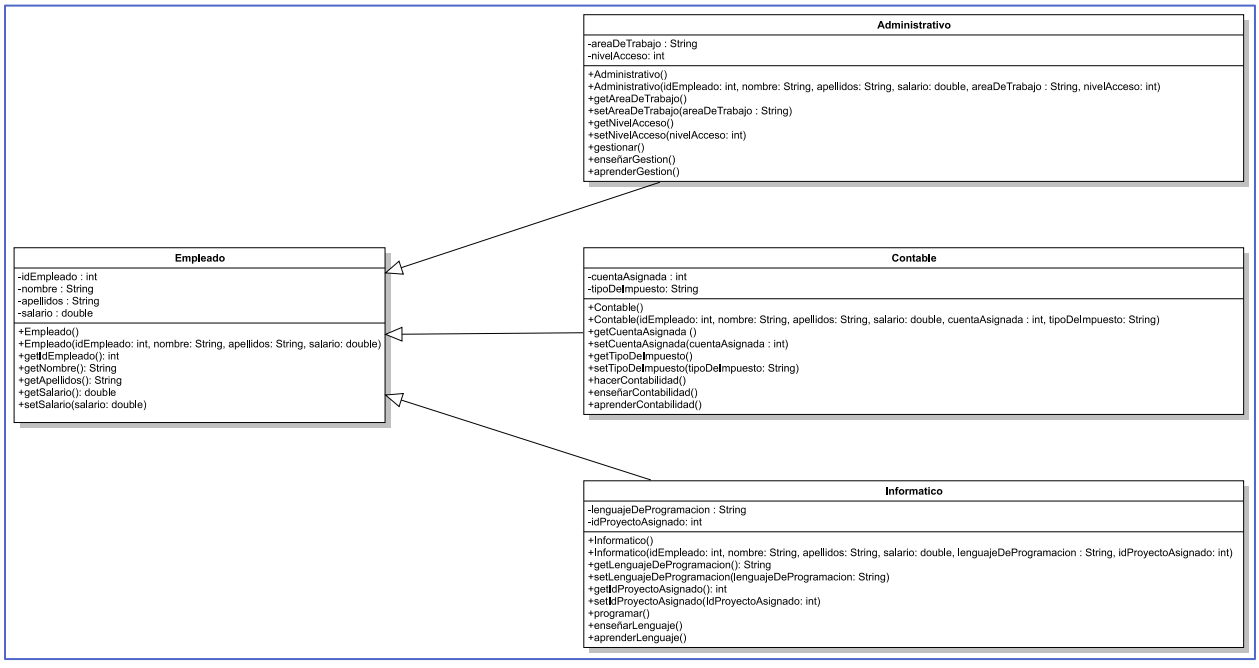
- Sin parámetros: Dan valores por defecto para evitar errores al crear objetos.
- Con parámetros: Permiten iniciar atributos directamente.

Getters y Setters:

- Controlan cómo se accede a atributos privados.
- Permiten validar datos (por ejemplo, no permitir salarios negativos).

Ejemplo de setSalario:

```java
public void setSalario(double nuevoSalario) {
    if (nuevoSalario > 0) {
        this.salario = nuevoSalario;
    } else {
        System.out.println("Error: El salario debe ser positivo.");
    }
}
```



0 2
E N C A P S U L A C I Ó N Y V I S I B I L I D A D

Decisiones de Encapsulación:

Atributos privados (private): Protegen la información interna y mantienen la integridad.

Métodos públicos (public): Permiten acceso controlado mediante getters y setters.

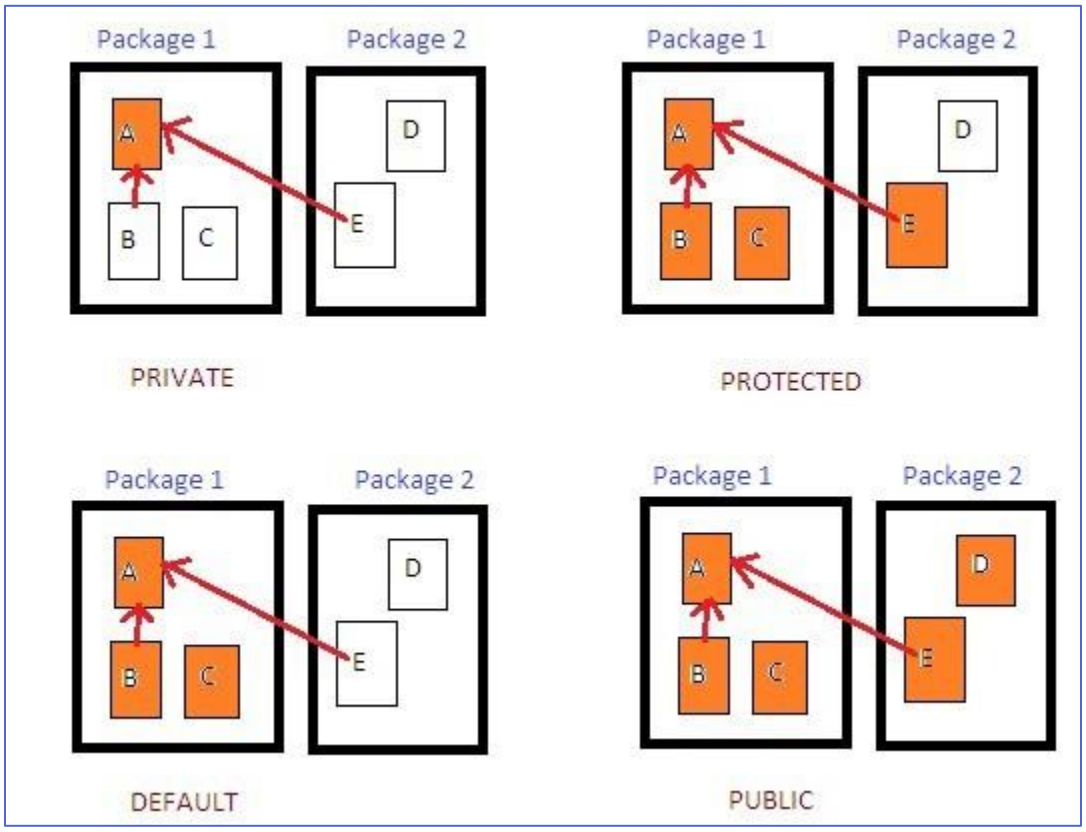
Métodos protegidos (protected): Permiten a subclases acceder a ellos.

Justificación:

Privacidad: Previene cambios no deseados.

Flexibilidad: Permite chequear datos antes de asignarlos.

Reutilización: Facilita el uso de las clases en varios contextos.



0 3

R E L A C I O N E S E N T R E C L A S E S

Relación de Herencia:

Las subclases (Administrativo, Contable, Informático) reciben atributos y métodos de la superclase Empleado.

Notación UML: Flecha con triángulo de subclases hacia la superclase.

Cardinalidad:

Relación Empleado → Subclases: 1..*

Cada empleado pertenece a una de las subclases.

Relación Subclases → Empleado: 1

Cada subclase deriva de Empleado.

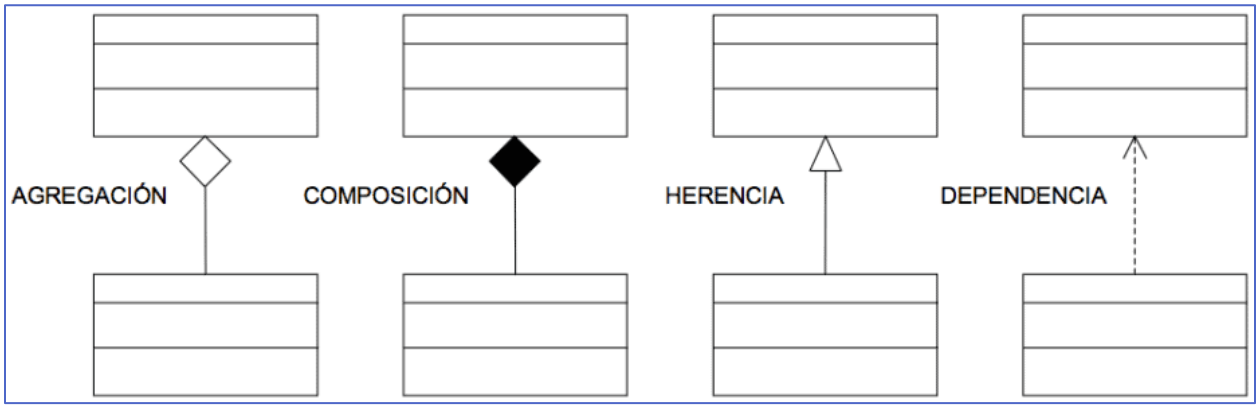
Agrupa atributos comunes en Empleado.

Permite añadir funciones específicas en cada subclase sin repetir código.

Ventajas de la Herencia:

Agrupa atributos comunes en Empleado.

Permite añadir funciones específicas en cada subclase sin repetir código.



0 4

C R E A R I N S T A N C I A S Y P R U E B A S

Instanciación de Objetos:

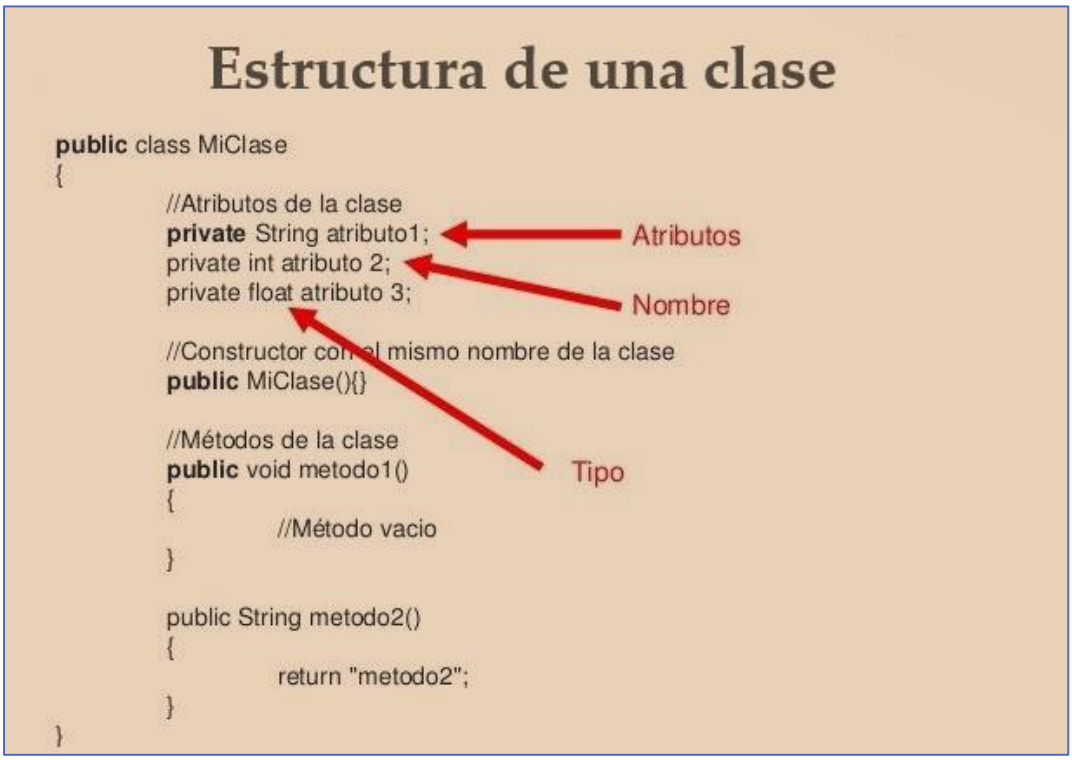
Constructor sin parámetros: Bueno para pruebas rápidas o inicializaciones simples.
Constructor con parámetros: Genera objetos completamente configurados.

Uso de Setters:

Los atributos de los objetos hechos con constructores sin parámetros se asignan después usando setters.

Pruebas:

Crear instancias de Empleado, Administrativo, Contable e Informático.
Utilizar getters para revisar valores asignados.
Probar métodos específicos de las subclases para verificar funciones extra.



Ejemplo de Implementación:

```java
public class Main {
    public static void main(String[] args) {
        // Crear instancias
        Administrativo admin = new Administrativo(1, "Ana", "López", 2500,
"Recursos Humanos", 5);
        Contable contable = new Contable(2, "Carlos", "Pérez", 3000, "IVA", 101);
        Informático informatico = new Informático(3, "Luis", "Martínez", 2800, "Java",
202);

        // Mostrar datos
        System.out.println(admin);
        System.out.println(contable);
        System.out.println(informatico);
    }
}
```

Salida esperada:

```
Administrativo: [ID: 1, Nombre: Ana López, Salario: 2500, Área: Recursos
Humanos, Nivel: 5]
Contable: [ID: 2, Nombre: Carlos Pérez, Salario: 3000, Impuesto: IVA, Proyecto:
101]
Informático: [ID: 3, Nombre: Luis Martínez, Salario: 2800, Lenguaje: Java,
Proyecto: 202]
```

Decisiones y Justificaciones

Encapsulación:
Protege la integridad de datos y impide accesos no permitidos.
Modularidad:
Separa el programa en clases individuales para ayudar en mantenimiento y crecimiento.
Herencia:
Permite usar código existente y mantiene el diseño organizado.
Pruebas:
Confirman que el sistema opera como se espera en situaciones reales.

0 4
C O N C L U S I Ó N

El desarrollo del caso práctico permitió aplicar los conceptos clave de la POO, cubriendo desde el diseño hasta la implementación y pruebas en Java. Aspectos importantes como:

La definición de atributos y métodos.
La implementación de encapsulación para proteger datos.
Las relaciones de herencia para evitar redundancias y fomentar la modularidad.
Fueron esenciales para crear un sistema organizado, escalable y fácil de mantener.

Las pruebas confirmaron que el sistema funciona correctamente y muestra su solidez, evidenciando cómo los principios de abstracción y modularidad simplifican el desarrollo y mantenimiento de aplicaciones complejas. Este caso práctico resalta lo importante que es una buena planificación inicial con UML, así como la necesidad de aplicar buenas prácticas para asegurar claridad y consistencia del código en situaciones reales.

En resumen, este ejercicio refuerza habilidades técnicas y también destaca la importancia de diseñar sistemas que sean funcionales y sostenibles a largo plazo.

0 5

R E F E R E N C I A S

https://www.youtube.com/watch?v=JioEGJIlg88&t=44s

https://desarrolloweb.com/articulos/499.php

https://www.microsoft.com/es-es/microsoft-365/business-insights-ideas/resources/guide-to-uml-diagramming-and-database-modeling

https://www.youtube.com/watch?v=Wr-z7dOzUwE

https://openwebinars.net/blog/introduccion-a-poo-en-java-encapsulamiento/

https://www.programarya.com/Cursos/Java/Modificadores-de-Acceso

https://sekthdroid.wordpress.com/2012/12/03/constructores-e-instanciacion-en-java/

https://blog.hubspot.es/website/que-es-constructor-java

https://openwebinars.net/blog/introduccion-a-poo-en-java-atributos-y-constructores/