



Carátula para entrega de prácticas

Facultad de Ingeniería

Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): Proyecto 2

Integrante(s): 322184022

322238682

321570789

322069093

322249723

*No. de lista o
brigada:* 5

Semestre: 2026-1

Fecha de entrega: 31/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción	2
1.0.1. Planteamiento del problema	2
1.0.2. Motivación	2
1.0.3. Objetivos	2
2. Marco Teórico	2
2.1. Herencia	2
2.2. Polimorfismo	3
2.3. Sobrescritura	3
2.4. Diagrama UML (Lenguaje Unificado de Modelado)	4
2.4.1. Diagrama de Clases	4
2.4.2. Diagrama de Secuencia	4
3. Desarrollo	5
4. Resultados	8
5. Conclusiones	8
6. Referencias	9

1. Introducción

1.0.1. Planteamiento del problema

Se requiere desarrollar un sistema para la gestión de personal de una empresa. El desafío principal es modelar de forma eficiente a los distintos tipos de empleados, como pueden ser Operarios, Directivos, Oficiales y Técnicos. Estos empleados comparten atributos y comportamientos comunes (como un nombre), pero a la vez poseen características únicas y formas de actuar específicas de su puesto.

1.0.2. Motivación

La implementación de este sistema es una oportunidad para aplicar conceptos clave de la Programación Orientada a Objetos (POO). El proyecto permitirá emplear el encapsulamiento para proteger la integridad de la información de los empleados (como sus datos personales o salario), restringiendo el acceso directo a sus atributos. Se utilizará la herencia para crear una jerarquía de clases, partiendo de una superclase Empleado y derivando subclases específicas, lo cual promueve la reutilización de código. También, se aplicará el polimorfismo para que un objeto pueda comportarse como una instancia de su propia clase o de cualquiera de sus subclases, facilitando el manejo de los distintos tipos de empleado de manera homogénea. Finalmente, el uso de diagramas UML será fundamental para el modelado abstracto del sistema, permitiendo visualizar la estructura y relaciones entre clases antes de la implementación.

1.0.3. Objetivos

- Aplicar los conceptos teóricos de herencia y polimorfismo para modelar la relación jerárquica entre diferentes tipos de empleados.
- Diseñar código reutilizable, aprovechando las superclases para definir atributos y métodos comunes.
- Proteger la integridad de los datos del sistema mediante la correcta aplicación de los modificadores de acceso (encapsulamiento).
- Implementar un sistema capaz de gestionar objetos de distintas clases de forma polimórfica.
- Implementar distintos tipos de diagramas UML para observar cómo se comporta el orden y secuencia de los programas.

2. Marco Teórico

2.1. Herencia

[2] En términos simples, se puede definir como contruir "algo" a partir de cimientos ya existentes. Permite definir nuevas clases basadas en clases existentes, aprovechando

y extendiendo su funcionalidad.

- **Super Clase**

Se define una sola clase base, la cual contendrá los atributos y métodos comunes que deseamos compartir con las clases derivadas.

- **Clases derivadas**

Se crea la clase hija, la cual como su nombre lo indica, será la que hereda de la clase padre.

Con la "herencia" ahora se pueden crear objetos de la clase derivada y acceder a los atributos y métodos tanto de la clase derivada como de la clase base.

- **Sintaxis**

modificador class nombreSubClase
extends nombreSuperClase

Ventajas

- Reutilización del código
- Abstracción y generalización
- Uso de **polimorfismo**

2.2. Polimorfismo

[2] Es la habilidad de una clase para implementar métodos con el mismo nombre, pero con comportamientos distintos dependiendo de la clase con la que se este interactuando.

- **Crear clase base/interfaz**

De igual forma que con la herencia, se definen los métodos que deamos que las clases derivadas implementen.

- **Clase derivada**

Se crean las clases que implementarán la interfaz o hereden de la clase padre. Cada clase debe proporcionar su propia implementación del método definido en la interfaz/clase padre.

Ahora se pueden crear objetos de las clases derivadas y tratarlos como objetos de la interfaz/clase base.

2.3. Sobrescritura

[3] Permite que una sub-clase proporcione su propia implementación del método para adaptarlo a su comportamiento.

Ocurre cuando una sub-clase redefine un método heredado de su clase padre con:

- Mismo nombre

- Mismo tipo y número de parámetros.
- Mismo nivel de acceso.
- Mismo tipo de valor de retorno.

@Override (*se indica que se está sobrescribiendo un método*)

Para acceder:

La palabra clave **super** se utiliza para comunicar métodos y atributos de la clase base a una clase derivada, permitiendo llamar a los constructores, métodos o acceder a los atributos de la clase padre desde la clase hija.

Esto es útil si deseamos agregar funcionalidad adicional en la sub-clase, pero también queremos preservar el comportamiento original del método de la clase base.

2.4. Diagrama UML (Lenguaje Unificado de Modelado)

[1] El **Lenguaje Unificado de Modelado** (UML, por sus siglas en inglés) es un lenguaje gráfico estándar y formal para la industria del software. No es un lenguaje de programación, sino un conjunto de *diagramas* que actúan como planos. Se utiliza para visualizar, especificar, construir y documentar los diferentes artefactos y vistas de un sistema de software complejo, facilitando la comunicación entre desarrolladores, analistas y clientes.

2.4.1. Diagrama de Clases

Es el diagrama más común de UML y pertenece a los diagramas de **estructura** (estáticos). Describe la estructura del sistema mostrando sus clases, atributos (datos), operaciones (métodos) y las relaciones estáticas entre ellas (como herencia, asociación, agregación o composición). [1]

2.4.2. Diagrama de Secuencia

Este es un diagrama de **comportamiento** (dinámico), enfocado en la interacción. Muestra cómo los objetos colaboran entre sí a lo largo del *tiempo*. Se centra en el orden cronológico (la secuencia) de los mensajes que se envían y reciben entre diferentes objetos (representados como "líneas de vida") para realizar una tarea o un caso de uso específico. [1]

3. Desarrollo

1. Clase Empleado:

El programa comienza con la clase **Empleado**, la cual se declara **abstracta** para definir la estructura que van a tener todos los empleados. Dentro de esta clase se encuentran declarados tres atributos:

- **nombre** — tipo String
- **apellidoPaterno** — tipo String
- **numeroSeguroSocial** — tipo String

Además son declarados con **protected final** para que tengan herencia, pero no sean modificables después de inicializar el objeto. Y aparte de esto se declara un método abstracto llamado **ingresos()** el cual devolvera los ingresos dependiendo del empleado. El método se sobrescribira por las clases heredadas de **Empleado**. Para terminar esta clase se implemento **toString()** para imprimir la información del empleado.

2. Clase EmpleadoAsalariado:

Para este tipo de empleado se creó la clase **EmpleadoAsalariado**, que hereda de **Empleado**. En esta clase se declaró un atributo **private** para trabajar con encapsulamiento, junto con sus métodos **get** y **set**.

- **salarioSemanal** — tipo double

También se agrego una condición para evitar los números negativos. Después se sobrescribió el método heredado **ingresos** para que devuelva ahora el salario semanal y **toString()** para mostrar los datos del empleado.

3. Clase EmpleadoPorComision:

La clase **EmpleadoPorComision** también hereda de **Empleado**. Para este tipo se implementaron dos atributos:

- **ventasNetas** — tipo double
- **tarifaComision** — tipo double

Ambos atributos se declararon como **private**, con sus respectivos métodos de acceso. Se implementaron condiciones para evitar valores negativos y porcentajes fuera del rango. A continuación se sobreesciben los métodos heredados **ingresos()** y **toString()**, ingresos ahora devuelve la multiplicación de ventas-Netas * tarifaComision. Mientras que **toString()** muestra los datos y calcula el porcentaje de comisión.

4. MainApp:

En esta clase los empleados **EmpleadoAsalariado** y **EmpleadoPorComision** se almacenan dentro de un **ArrayList** llamado **nomina**, empleando la

herencia para tratarlos como **Empleado**. A continuación se recorre con un ciclo **for-each** para imprimir la información de cada empleado, y al llamar al método **ingresos()**, se ejecuta dependiendo del tipo de objeto, es decir, se emplea el polimorfismo.

A continuación se muestran las representaciones de la estructura de los programas en diagrama UML estático (clases) y dinámico (secuencia).

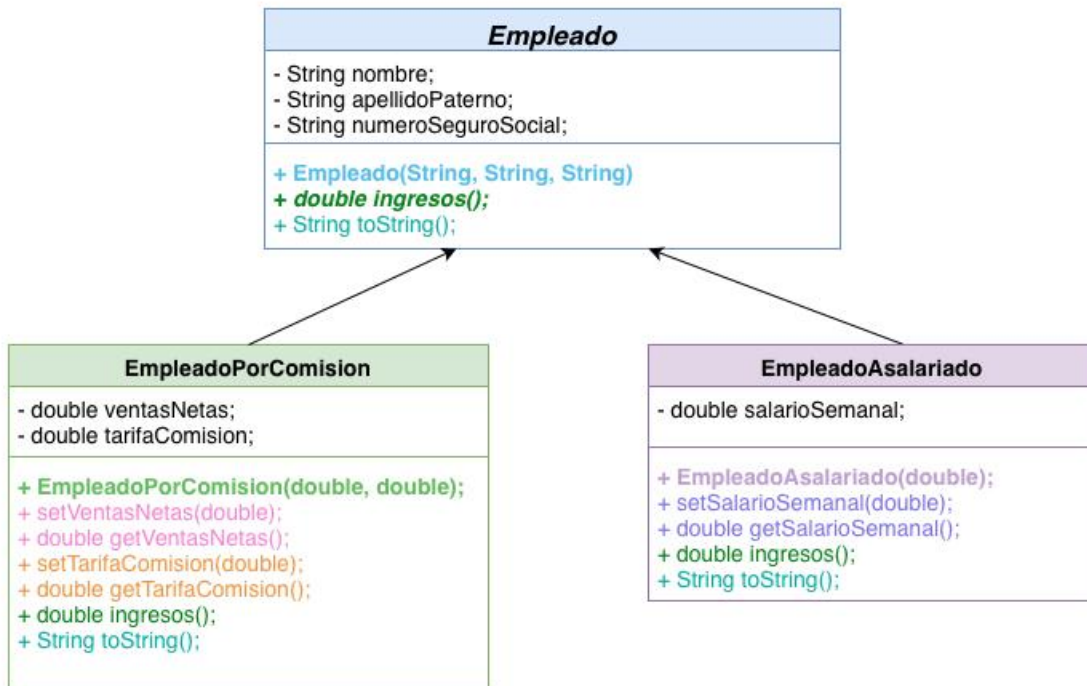


Figura 1: Diagrama de clases

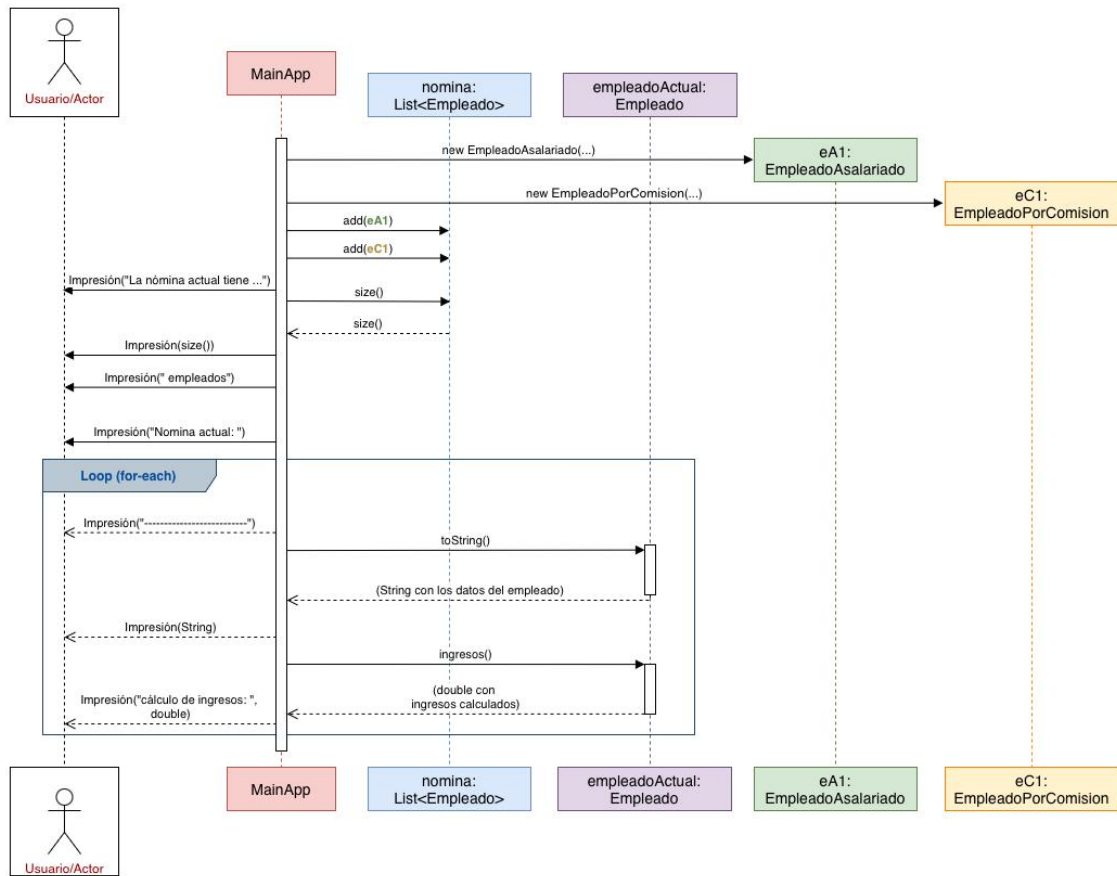
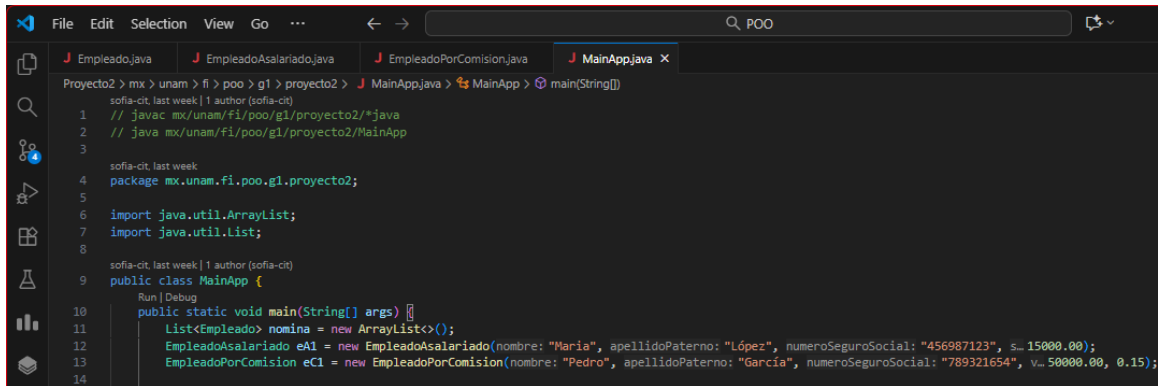


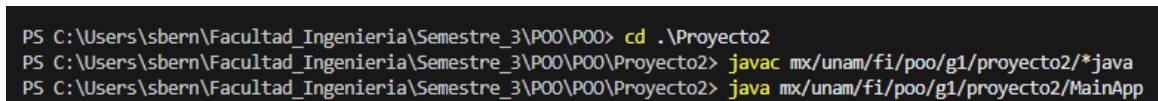
Figura 2: Diagrama de secuencia

4. Resultados



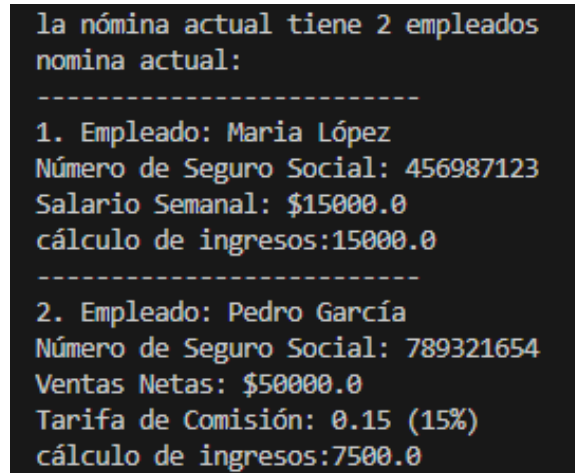
```
Project2 > mx > unam > fi > poo > g1 > proyecto2 > J MainApp.java > MainApp > main(String[])
sofia-cit, last week | 1 author (sofia-cit)
1 // javac mx/unam/fi/poo/g1/proyecto2/*.java
2 // java mx/unam/fi/poo/g1/proyecto2/MainApp
3
4 sofia-cit, last week
5 package mx.unam.fi.poo.g1.proyecto2;
6
7 import java.util.ArrayList;
8 import java.util.List;
9
10 sofia-cit, last week | 1 author (sofia-cit)
11 public class MainApp {
12     Run | Debug
13     public static void main(String[] args) {
14         List<Empleado> nomina = new ArrayList<>();
15         EmpleadoAsalariado eA1 = new EmpleadoAsalariado(nombre: "Maria", apellidoPaterno: "López", numeroSeguroSocial: "456987123", s_ 15000.00);
16         EmpleadoPorComision eC1 = new EmpleadoPorComision(nombre: "Pedro", apellidoPaterno: "García", numeroSeguroSocial: "789321654", v_ 50000.00, 0.15);
```

Figura 3: Creación de las instancias empleado asalariado y empleado por comisión



```
PS C:\Users\sbern\Facultad_Ingenieria\Semestre_3\P00\P00> cd .\Proyecto2
PS C:\Users\sbern\Facultad_Ingenieria\Semestre_3\P00\P00\Proyecto2> javac mx/unam/fi/poo/g1/proyecto2/*.java
PS C:\Users\sbern\Facultad_Ingenieria\Semestre_3\P00\P00\Proyecto2> java mx/unam/fi/poo/g1/proyecto2/MainApp
```

Figura 4: Comandos de compilación y ejecución



```
la nómina actual tiene 2 empleados
nomina actual:
-----
1. Empleado: Maria López
Número de Seguro Social: 456987123
Salario Semanal: $15000.0
cálculo de ingresos:15000.0
-----
2. Empleado: Pedro García
Número de Seguro Social: 789321654
Ventas Netas: $50000.0
Tarifa de Comisión: 0.15 (15%)
cálculo de ingresos:7500.0
```

Figura 5: Ejecución del programa con los empleados agregados

5. Conclusiones

Consideramos que el proyecto cumple con los objetivos fundamentales de aplicar y demostrar los conceptos de **herencia y polimorfismo**. Esto se logró mediante el diseño de una jerarquía de clases, poniendo en práctica dichos conocimientos en un sistema de nómina que cuenta con la clase base abstracta Empleado y sub clases como EmpleadoAsalariado y EmpleadoPorComision.

Específicamente el concepto de herencia se implementó al definir la clase base con atributos y métodos a compartir, mientras que la sobrescritura nos permitió que un método común como (`ingresos()`) tuviera un trabajo distinto y adaptado en cada clase derivada. Por otra parte, el polimorfismo se implementó en la clase principal para poder tratar a objetos de diferentes tipos como objetos de la clase padre, ejecutando la versión correcta del método sobrescrito (`ingresos()`) en tiempo de ejecución.

Igualmente pudimos implementar los diagramas UML de estructura y de comportamiento para aportar una visión fija del sistema (en el caso de diagrama de clases), y visualizar la comunicación entre elementos del sistema para un proceso específico (en el caso de diagrama de secuencia)

6. Referencias

- [1] J. Avila, J. y Bailón. Diagramas uml. en análisis y diseño en poo. (<https://portalacademico.cch.unam.mx/cibernetica1/analisis-y-diseno-en-poo/diagramas-UML>), 2022. Consultado el 28-10-2025.
- [2] J.L. Blasco. Introducción a poo en java: Herencia y polimorfismo. (<https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/>), 2023. Consultado el 28-10-2025.
- [3] Nivardo. Sobrescritura de métodos en java. (<https://oregoom.com/java/sobrescritura-de-metodos/>), 2024. Consultado el 28-10-2025.