Carátula para entrega de prácticas	
Facultad de Ingeniería	Laboratorio de docencia

Laboratorios de computación salas A y B

Profesor: René Adrián Dávila Pérez

Asignatura: Programación Orientada a Objetos

Grupo: 01

No. de proyecto: 02

Integrante(s):
322276824
322258516
425037384
320108116
322221415

No. de brigada: 01

Semestre: 2026-1

Fecha de entrega: 31 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción **2**

2. Marco Teórico **2**

3. Desarrollo **3**

4. Resultados **6**

5. Conclusiones **6**

1. Introducción

- **Planteamiento del problema:**
  Desarrollar una aplicación en Java que modele el sistema de nómina de una empresa en la que sus empleados se dividen en asalariados y por comisión mediante una clase abstracta y subclases para la construcción de cada tipo de empleado, implementando validaciones en los campos que lo requieran, y comprobando el concepto de polimorfismo en una clase de prueba.

- **Motivación:**
  Modelar correctamente diferentes tipos de objetos y sus métodos específicos mediante herencia y polimorfismo facilita la reutilización, extensión y mantenimiento del código. Además, aplicar validaciones en los métodos asegura robustez en un programa y lo vuelve menos susceptible a errores.

- **Objetivos:**
  Modelar una jerarquía de empleados mediante una clase abstracta y subclases concretas, utilizando validaciones en sus métodos para garantizar estados consistentes, así como evidenciar las ventajas del polimorfismo en un programa y documentar la solución con diagramas UML.

2. Marco Teórico

Clase abstracta

Una clase abstracta es aquella que no puede instanciarse directamente y que sirve como plantilla para sus subclases. Contiene métodos abstractos que deben ser implementados por las clases derivadas. Son útiles cuando se espera que las clases que extienden de la clase abstracta tengan muchos métodos o campos en común o cuando se desea compartir código entre varias clases estrechamente relacionadas. [1]

Herencia

La herencia es un mecanismo que permite que una clase reutilice y extienda los atributos y métodos de otra. Esto con el objetivo de fomentar la reutilización del código y la creación de jerarquías. De acuerdo con [2] “Una clase que se deriva de otra clase se denomina subclase (también clase derivada, clase extendida o clase hija). La clase de la que se deriva la subclase se denomina superclase (también clase base o clase padre).”.

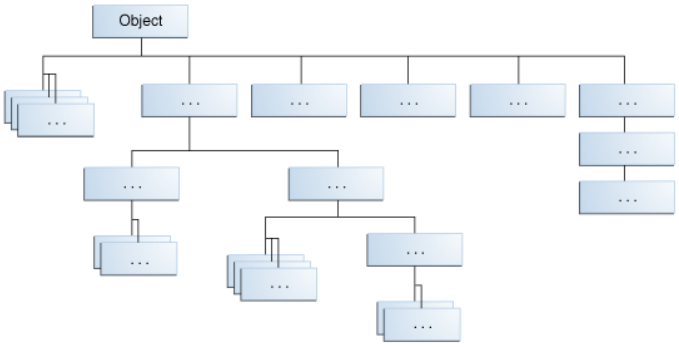


Figura 1: Todas las clases de Java descienden de `Object`. [2]

Polimorfismo

El polimorfismo es la capacidad de un mismo método de comportarse de manera distinta según el objeto que lo invoque. Esto permite que una referencia de un tipo pueda comportarse como uno diferente, ejecutando el método adecuado en cada caso. El polimorfismo es la habilidad de un objeto para tomar distintas formas y responder adecuadamente según su tipo real, depende directamente de la herencia, ya que sin herencia este no puede implementarse. [3]

3. Desarrollo

La implementación se organizó en el paquete `mx.unam.fi.poo.proyecto02` y consta de cuatro clases principales, a continuación se describen las clases y su implementación.

Clase Empleado

Es una clase abstracta que contiene los datos comunes de los empleados: `nombre`, `apellidoPaterno` y `numSeguroSocial`. Todos se declararon `private final` y de tipo `String`, se inicializan mediante un constructor que recibe los tres valores. Se declara el método abstracto `ingresos()`, mismo que será implementado por las subclases que extiendan de ella. Además, se sobreescribe el método `toString()` para devolver el nombre, apellido y número de seguro social con un formato determinado.

Clase EmpleadoAsalariado

Esta clase extiende de `Empleado` y se añade el atributo privado `salarioSemanal` de tipo `double`. Dispone de un constructor que hace uso del constructor de la superclase a través de `super` y llama a `setSalarioSemanal()` para inicializar el salario. El método `setSalarioSemanal()` hace una validación para que no se asignen valores negativos, en caso de recibir un valor inválido, el setter imprime un mensaje y asigna `0.0` como valor por defecto. La implementación de `ingresos()` devuelve el valor del

salario semanal, mientras que `toString()` añade información sobre el salario a la impresión del método heredado.

Clase EmpleadoPorComision

Esta clase también extiende de `Empleado`, en ella se declaran los atributos privados `ventasNetas` de tipo `int` y `tarifaComision` de tipo `double`. Su constructor usa al constructor de la superclase e inicializa ambos atributos mediante sus respectivos setters. El método `setVentasNetas()` valida que no haya ventas negativas y el método `setTarifaComision()` impide tarifas fuera del rango $[0.0 - 1.0]$, en caso de que la entrada sea inválida, los setters imprimen un mensaje por consola. El método `ingresos()` devuelve el producto `tarifaComision * ventasNetas`, finalmente el método `toString()` retorna una cadena que incluye los datos del empelado y las ventas, tarifa e ingresos totales calculados.

Clase MainApp

Es la clase que contiene el método principal con el que se demuestra el uso de polimorfismo. Se crea una referencia de tipo `Empleado` que primero se comporta como una instancia de `EmpleadoAsalariado` con datos de ejemplo y se imprime por consola, luego se comporta como una instancia de `EmpleadoPorComision` con datos de ejemplo para imprimirse de la misma manera. Esto para evidenciar cómo la misma referencia de tipo `Empleado` puede invocar los métodos adecuados según qué tipo de objeto sea en cada momento.

Diagrama de clases (UML estático)

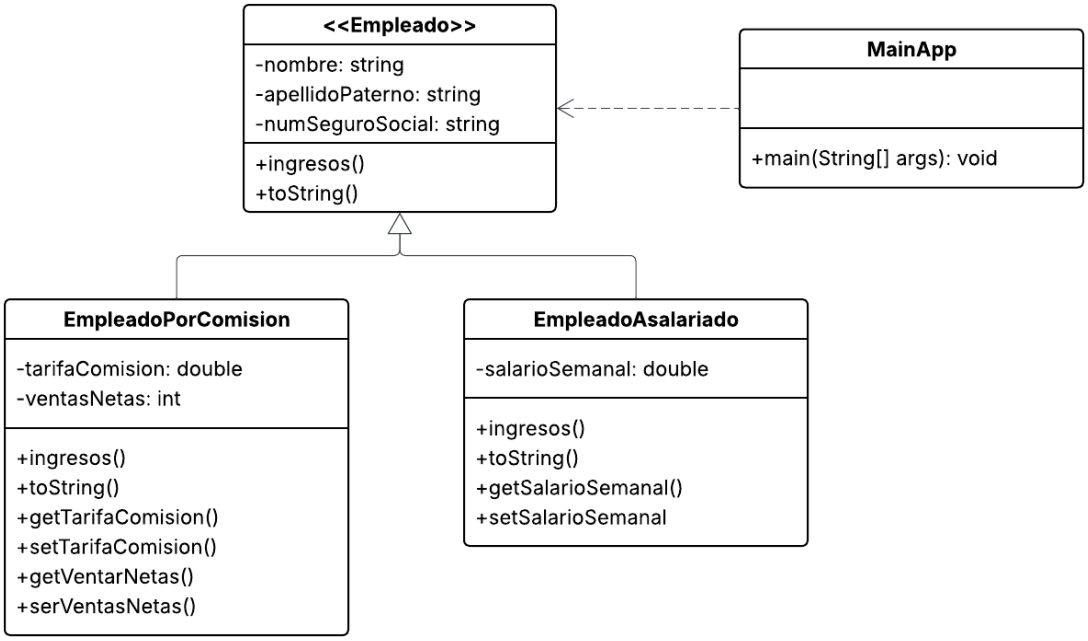
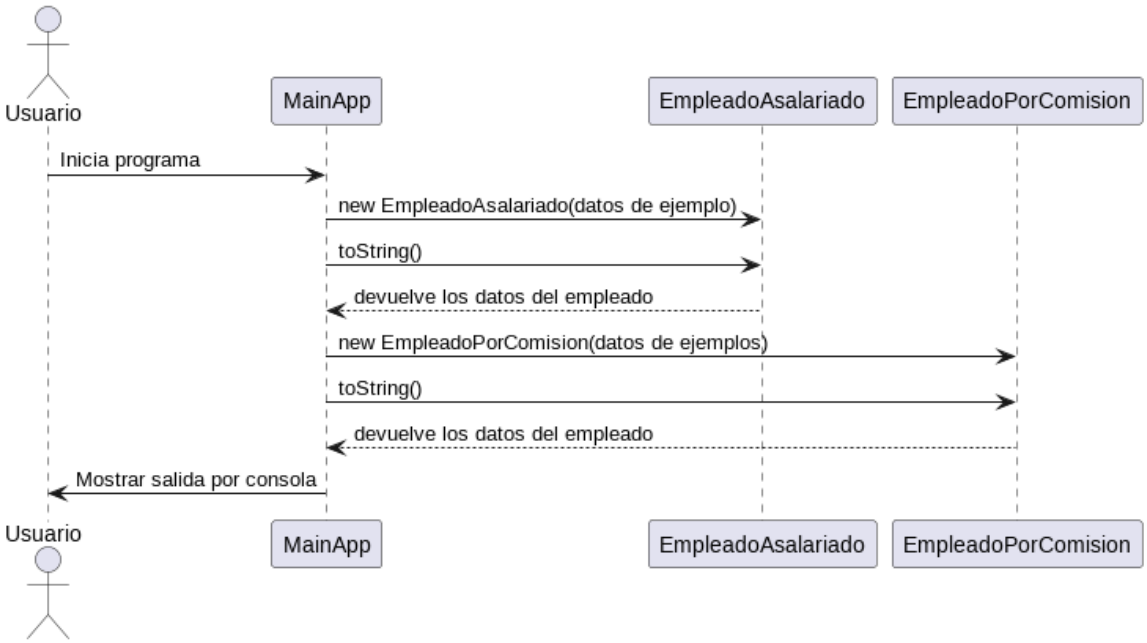


Diagrama de secuencia (UML dinámico)



4. Resultados

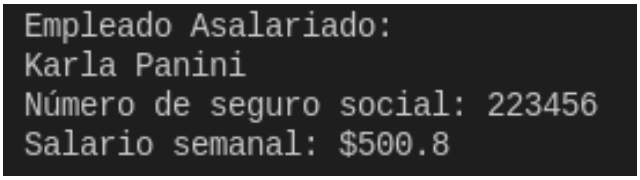


Figura 2: Ejemplo de empleado asalariado (entradas válidas).

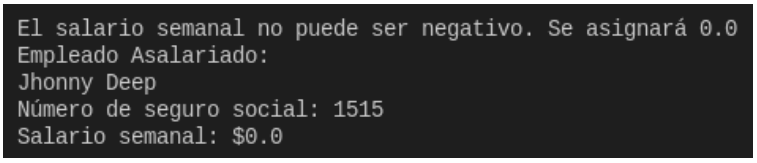


Figura 3: Ejemplo de empleado asalariado (entradas inválidas).

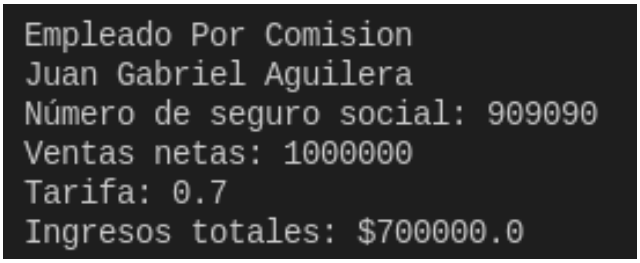


Figura 4: Ejemplo de empleado por comisión (entradas válidas).

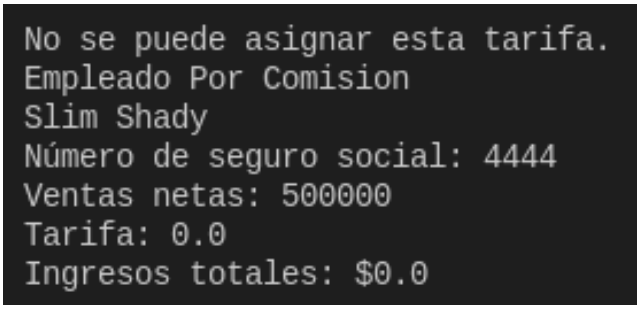


Figura 5: Ejemplo de empleado por comisión (entradas inválidas).

5. Conclusiones

El desarrollo de este proyecto permitió aplicar y verificar los conceptos de herencia y polimorfismo mediante un esquema de empleados, en el que, a través de una clase padre que definió el contrato común, se crearon subclases que se encargaron de

implementar los métodos para el cálculo específico de los ingresos. Las validaciones en los setters contribuyeron a la robustez del modelo, evitando valores inconsistentes en los objetos. Además, los diagramas UML facilitaron la comprensión del flujo y estructura del programa y mejoraron la estética de la documentación.

Referencias

[1] *Abstract Methods and Classes*. Oct. de 2025. URL: https://docs.oracle.com/javase/tutorial/java/IandI/abstract.html.

[2] *Inheritance*. Oct. de 2025. URL: https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html.

[3] *Polymorphism*. Oct. de 2025. URL: https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html.