	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: POO

Grupo: 01

No de Práctica(s): Proyecto 2

Integrante(s): 322059179

323629814

322113536

322125337

322080869

No. de lista o brigada: 06

Semestre: 2025-2026

Fecha de entrega: 21/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema**: Necesitamos un sistema de nómina para una compañía que paga a sus empleados semanalmente. Debemos modelar dos tipos distintos de empleados: Empleados Asalariados que reciben un salario semanal fijo y Empleados por Comisión que obtienen ingresos basados en un porcentaje de sus ventas. Tenemos que implementarlo utilizando una clase base abstracta (Empleado) y clases heredadas que representen a cada empleado.
- **Motivación**: Usar clases abstractas que en este caso es Empleado y la herencia son técnicas fundamentales para construir un software que tenga mayor calidad además de ser fácil su mantenimiento. Además que al usar clases y métodos abstractos se nos permite tratar a los diferentes tipos de empleados de manera polimórfica.
- **Objetivos**:
  - Aplicar la herencia para crear la clase EmpleadoAsalariado y EmpleadoPorComision tomando como base a la clase abstracta Empleado para entender mejor los usos de una clase abstracta y una heredada.
  - Reforzar la comprensión del polimorfismo, demostrando cómo una clase principal puede procesar objetos de ambas subclases de manera uniforme.

2. Marco Teórico

Encapsulamiento: Es un mecanismo para reunir datos y métodos dentro de una estructura ocultando la implementación del objeto, es decir, impidiendo el acceso a los datos por cualquier medio que no sean los servicios propuestos. La encapsulación permite, por tanto, garantizar la integridad de los datos contenidos en el objeto.[2]

Paquetes: Son el mecanismo de Java para agrupar clases e interfaces relacionadas en un espacio de nombres jerárquico, el cual corresponde directamente a una estructura de carpetas. Su principal objetivo es organizar el código y prevenir conflictos entre clases con el mismo nombre. El uso de paquetes, por tanto, es fundamental para mantener la modularidad, facilitar la reutilización del código y controlar el acceso a sus elementos.[1]

Herencia: Nos permite crear nuevas clases (subclases o clases hijas) a partir de clases existentes (superclases o clases padres). La clase hija hereda los atributos y métodos de la clase padre, lo que facilita la reutilización de código y la creación de una jerarquía de clases. Lo que significa que se pueden definir características y comportamientos comunes en una clase padre y luego crear clases hijas más especializadas que añaden o modifican esas características.[3]

Abstracción y Clases Abstractas: La abstracción consiste en ocultar los detalles complejos de implementación y mostrar solo las características esenciales de un objeto. Una clase abstracta es una clase que no se puede instanciar directamente (no se puede crear un objeto de ella) y sirve como una plantilla base para otras clases.[3]

Polimorfismo: Es la capacidad que tienen los objetos de diferentes clases de responder al mismo mensaje o método de manera específica para cada una. En Java, esto se logra comúnmente a través de la herencia y la sobrescritura de métodos (method overriding). La sobrescritura ocurre cuando una clase hija proporciona una implementación específica para un método que ya está definido en su clase padre. Para asegurar que la sobrescritura se realiza correctamente, es una buena práctica usar la anotación @Override. Esta anotación le indica al compilador que el método que sigue está destinado a anular un método de la superclase[4]

3. Desarrollo

Primero hicimos un repaso rápido guiandonos en las practicas pasadas, para no perdernos de algún punto importante sobrre los temas (pues el proyecto es basicamente una mezcla de todos los temas vistos anteriormente)

Despues de comprender lo que ibamos a hacer y con los conocimientos previos, empezamos creando la clase abstracta empleado. Esta clase tendria atributos protected como nombre, apellidoPaterno y numerroSeguroSocial (con sus respectivos setters y getters), y tambien contaria con el constructor y una clase abstrracta ingresos.

Una vez con la clase abstracta podiamos comenzar con las clases extendidas EmpleadoPorComision y EmpleadoAsalariado

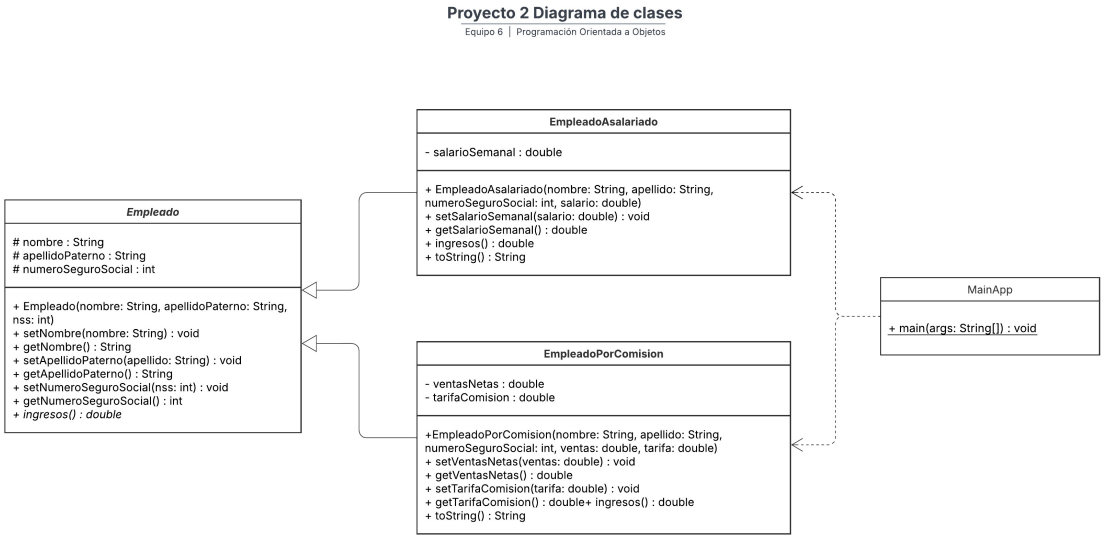
EmpleadoPorComision tendria nuevos atributos, los cuales son ventasNetas y tarifasComision (con sus getters y setters), con la peculiaridad de que el setter de TarifaComision debe tener la condición que el rango ea entre 0 y 1 Para su función ingresos (la abstracta) solo teniamos que devolver la multiplicación de los 2 atributos ventasNetas*tarifasComision (asi se calculan los ingresos de esta clase)

EmpleadoAsalariado tambien tendria un atributo nuevo, el cual es salarioSemanal (con su set y get), y su set tambien tendria una condición la cual es que el salario semanal no debe ser menor a 0 Para su función ingresos (la abstracta) solo era necesario devolvver el atributo salarioSemanal*4 (para que sea mensual)

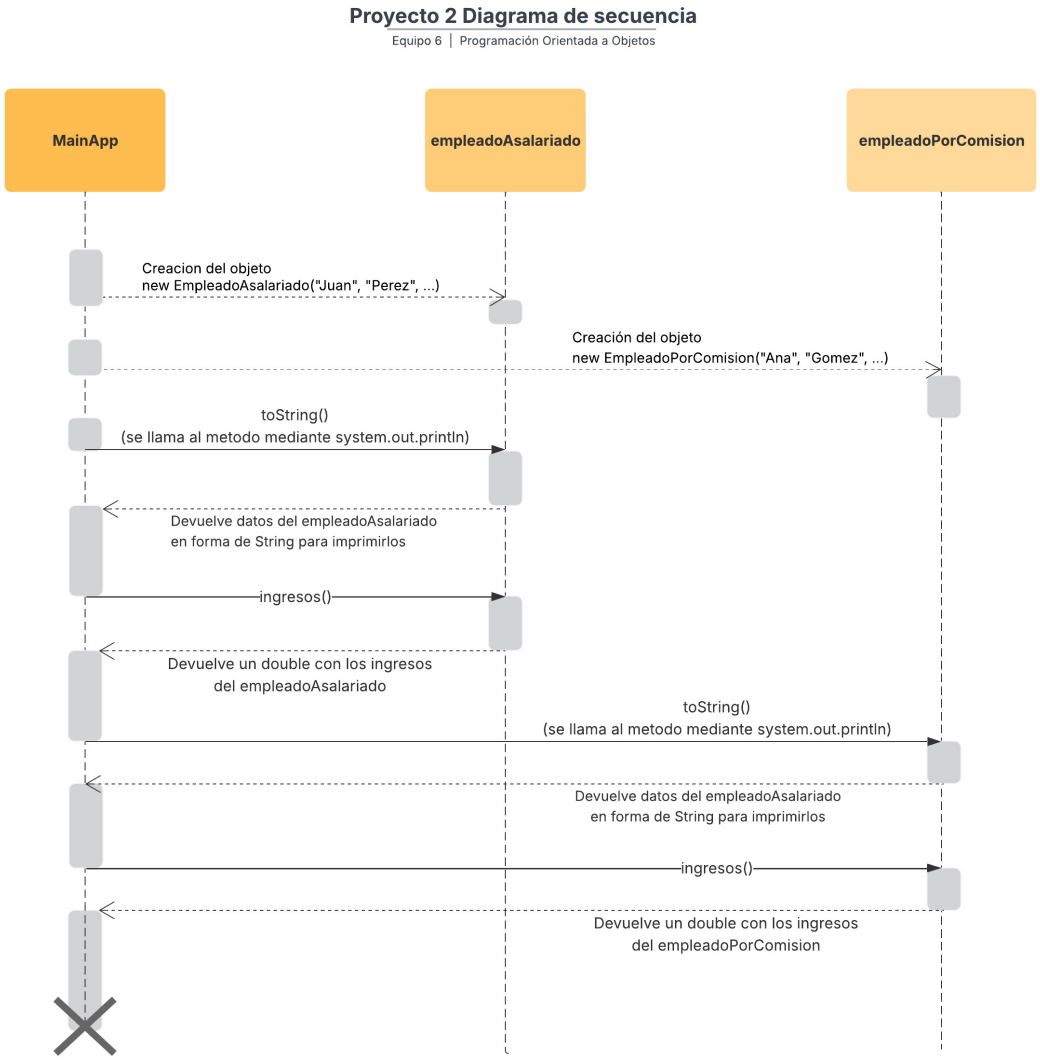
Ambas clases tienen el @ovevrrride en la función toString() para poder imprimir los atritutos de cada uno

Con esto ya teminarinamos las clases de los empleados y solamente fue necesario crear un MainApp para probar el programa, creando objetos de EmpleadoAsalariado y EmpleadoPorComision (de la clase Empleado no por que es abstracta)

-Diagrama de clases:

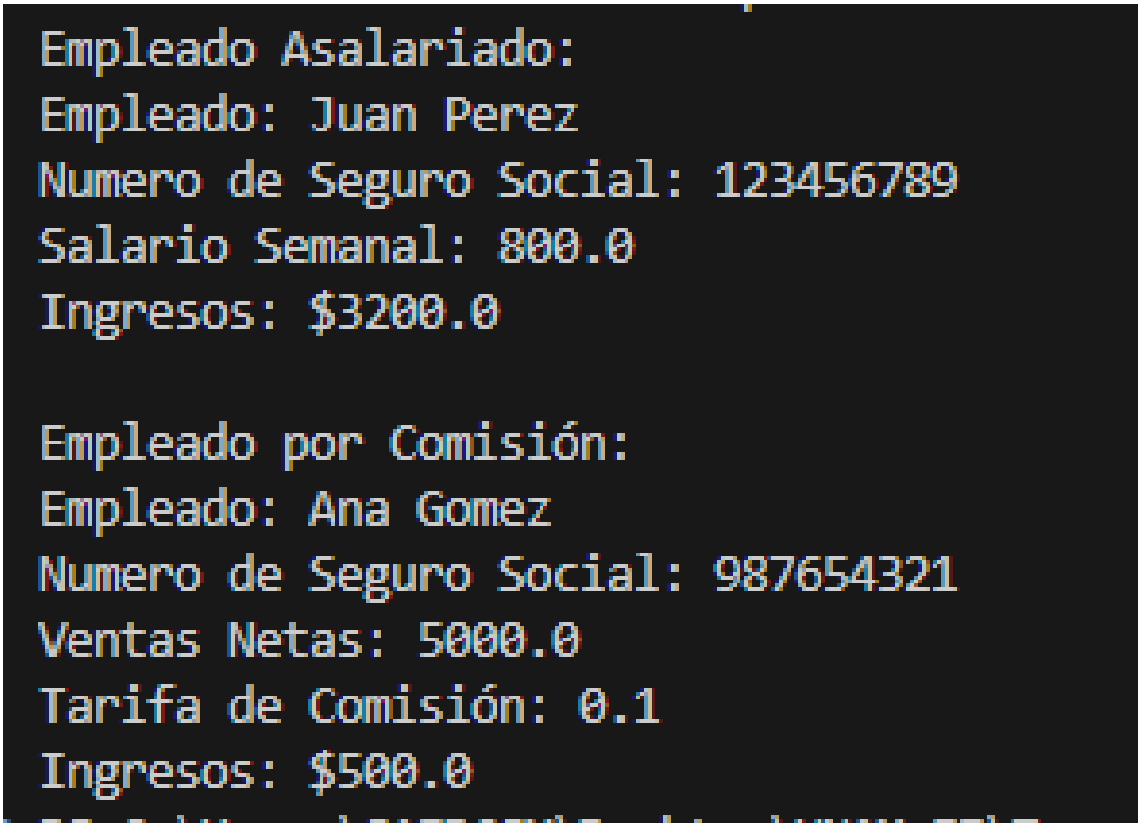


-Diagrama de secuencia:



4. Resultados

El programa se ejecuta correctamente



Como prueba se creó 1 objeto de cada clase, y los resultados esperados son los resultados obtenidos

Al imprimir el objeto de muestrar corrrectamente el toString() donde se especifica que tipo de empleado es y los atributos de cada uno

5. Conclusiones

Con el proyecto demostramos la aplicación práctica de la abstracción y la herencia para construir un sistema modular. Al definir la clase abstracta Empleado fue fundamental, ya que estableció el método ingresos()) que las clases concretas, EmpleadoAsalariado y EmpleadoPorComision, debían implementar. Esta estructura nos permitio usar polimorfismo, dejandonos que en la clase MainApp tratáramos los distintos tipos de empleados de forma genérica, pero invocando la lógica de cálculo específica de cada uno, como se comprueba en los resultados.

Adicionalmente, el encapsulamiento demostró ser crucial para garantizar la integridad de los datos. Al implementar validaciones directamente en los métodos set —como asegurar que el salario no fuera negativo y que la tarifa de comisión se mantuviera en un rango de 0.0 a 1.0 — se protegió la lógica de negocio de la aplicación. La ejecución y la precisión de los cálculos demuestran que la aplicación funciona y aplica estos principios orientados a objetos que hemos visto a lo largo del curso.

6. Referencias bibliográficas

[1] Alarcón, J. M. (2021, 27 de septiembre). Paquetes en Java: qué son, para qué se utilizan y cómo se usan. campusMVP. https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx

[2] David. (2025, 21 de agosto). Encapsulación: definición e importancia. DataScientest. https://datascientest.com/es/encapsulacion-definicion-e-importancia

[3] Oracle. (s.f.). Inheritance (The Java™ Tutorials >Learning the Java Language >Interfaces and Inheritance). Oracle Help Center. https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

[4] Oracle. (s.f.). Polymorphism (The Java™ Tutorials >Learning the Java Language >Interfaces and Inheritance). Oracle Help Center. https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html