	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Dávila Pérez René Adrián

Asignatura: Programación Orientada a Objetos

Grupo: 1

No de Práctica(s): Proyecto 2

Integrante(s): 322089020

322089817

322151194

425091586

322085390

No. de lista o brigada: Equipo 4

Semestre: 2026-1

Fecha de entrega: 31 de octubre de 2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

1.1. Planteamiento del Problema

El problema a resolver consiste en el desarrollo de una aplicación en Java que simule el cálculo semanal de los ingresos de los empleados de una empresa. Dicha organización cuenta con dos tipos de trabajadores: empleados asalariados, que perciben un salario fijo semanal, y empleados por comisión, cuyos ingresos dependen de las ventas realizadas. La aplicación debe ser capaz de procesar esta información de forma automatizada, aplicando los principios de la Programación Orientada a Objetos para modelar correctamente las relaciones entre clases y garantizar una estructura de código modular, reutilizable y extensible.

1.2. Motivación

El proyecto "Sistema de Nómina" representa una oportunidad para aplicar los fundamentos teóricos de la Programación Orientada a Objetos en un contexto práctico. Su desarrollo permite comprender cómo conceptos como la abstracción, el encapsulamiento, la herencia y el polimorfismo se integran para crear software con un diseño ordenado y eficiente. Además, implementar un sistema de nómina es un ejercicio que emula un caso de uso real dentro del ámbito empresarial, donde la gestión de empleados y el cálculo de pagos son tareas esenciales para el funcionamiento de cualquier organización.

1.3. Objetivos

El objetivo general de este proyecto es implementar un sistema de nómina funcional que utilice las características de la POO para calcular los ingresos de distintos tipos de empleados.

Los objetivos específicos son:

- Diseñar una clase abstracta **Empleado** que sirva como base para la creación de subclases.

- Implementar las clases derivadas **EmpleadoAsalariado** y **EmpleadoPorComisión**, heredando los atributos y métodos de la clase base.
- Sobrescribir los métodos *ingresos()* y *toString()* para personalizar el comportamiento de cada tipo de empleado.
- Aplicar validaciones adecuadas a los atributos para evitar datos inválidos, como salarios o ventas negativas.
- Utilizar el polimorfismo en una clase principal **MainApp** para procesar distintos tipos de empleados bajo una misma estructura de código.

2. Marco Teórico

2.1. Abstracción

La abstracción es un proceso que permite identificar los elementos esenciales de un objeto del mundo real, seleccionando solo sus características y funciones más relevantes para representarlas en un modelo. En la programación orientada a objetos, la abstracción se utiliza para definir los atributos y métodos que comparten los objetos de una misma clase. Por ejemplo, los equipos de cómputo pueden describirse a través de propiedades como tipo, color, número de serie, memoria, disco duro o tecnología de almacenamiento. Además, es posible incluir otros componentes como la placa base, el procesador o el monitor, representando de manera general sus rasgos comunes.

2.2. Encapsulamiento

El encapsulamiento consiste en ocultar los datos internos y los métodos de una clase, exponiendo únicamente una interfaz pública que permita interactuar con el objeto de manera controlada. Este principio protege la integridad de los datos, facilita el mantenimiento del código y mejora su organización. En la práctica, el encapsulamiento se implementa mediante

métodos conocidos como *getters* y *setters*, que permiten obtener o modificar los valores de los atributos de forma segura, garantizando que el estado del objeto permanezca coherente.

2.3. Polimorfismo

El polimorfismo permite que un mismo método o símbolo adopte distintos comportamientos según el tipo de objeto que lo invoque. Este principio brinda flexibilidad al código y promueve la reutilización, ya que diferentes clases pueden responder de manera adecuada a la misma llamada de función. Gracias al polimorfismo, es posible trabajar con colecciones de objetos que comparten una misma interfaz o herencia, pero que ejecutan acciones específicas de acuerdo con su tipo particular.

2.4. Herencia

La herencia es un mecanismo que posibilita crear nuevas clases a partir de otras ya existentes, conocidas como clases base o superclases. A través de ella, las clases derivadas heredan atributos y métodos, lo que fomenta la reutilización del código y permite extender o modificar el comportamiento de las clases originales. Este principio facilita la creación de jerarquías de objetos, donde cada nivel puede añadir o redefinir funcionalidades sin afectar la estructura general del programa. [1]

3. Desarrollo

3.1. Implementación del Código

3.1.1. Empleado (Clase Base)

Esta clase abstracta define la plantilla general para cualquier tipo de empleado. No se puede crear directamente un objeto de esta clase, está diseñada para solo ser heredada.
Los atributos de está clase (nombre, apellido, número de seguro social) son private y final para que sean solo accesibles desde la misma clase y para que no se puedan cambiar despues de ser asginados en el constructor.
Los métodos de esta clase son:

- Constructor: Recibe los 3 atributos y los inicializa
- Getters para los atributos
- toString (método sobreescrito): para devolver una cadena de texto formateada con la información básica del empleado.
- ingresos(método abstracto): No contiene particularmente pero obliga a que cualquier clase que herede Empleado implemente este método y defina como calcula sus propios ingresos.

3.1.2. Empleado Asalariado (Clase Hija)

Esta clase representa un tipo específico de empleado. Hereda de Empleado, por lo que automáticamente tiene nombre, apellidoPaterno y nss. Otro atributo propio que contiene es uno privado denominado salarioSemanal, que almacena el salario fijo que recibe el empleado por semana.
Los métodos de esta clase son:

- Constructor:

    - super: Que llama al constructor de la clase padre para inicializar los atributos que heredó.
    - setSalarioSemanal: Que inicializa el atributo propio de esta clase usando su propio método setter.
- getSalarioSemanal: Que obtiene el salario semanal.
- setSalarioSemanal: Que establece el salario semanal, este incluye una validación para asegurar que el salario no sea negativo.
- ingresos (método sobreescrito): Que implementa el método abstracto que heredó, para un empleado asalariado, sus ingresos son simplemente su salario semanal.
- super.toString: Que llama al toString de la clase padre para obtener la información básica, y le añade la información específica de esta clase (Salario Semanal).

3.1.3. Empleado Por Comisión (Clase Hija)

Contiene 2 atributos privados más para el monto total de ventas que realizó el empleado y la tarifa de comisión que el empleado gana de sus ventas. Métodos:

- Constructor:
    - super: Que llama al constructo de la clase padre.
    - Setters para ventasNetas y tarifaComisión.
- Setter para Ventas Netas con validación de no negatividad.
- Setter para Tarifa de Comisión con validación de rango (0<comision<1).
- Getters para las ventas y la tarifa.
- ingresos (método sobreescrito): Que implementa el método abstracto que heredó para este empleado que se calcula multiplicando $getTarifaComisin * getVentasNetas$.

- toString (método sobreescrito): Que llama a super.toString y añade la información específica (ventas y tarifa).

3.1.4. MainApp (Clase Ejecutable)

Se crean las instancias de EmpleadoAsalariado y EmpleadoPorComisión, estos se guardan como tipo Empleado, aquí podemos ver el polimorfismo.
Tambien se crea un arreglo para los objetos tipo Empleado. Con un for-each se recorren todos los elementos del arreglo de empleado mostrando la información Nombre, Apellido, NSS y dependiendo del objeto el salario semanal o las ventas y la tarifa de comisión.

3.2. Pruebas

Output:

Empleado asalariado: Empleado: Daniel Contreras
Número de Seguro Social: 111-111-111
Salario semanal: 12000.50
Ingresos :12000.50

Empleado por Comisión: Empleado: Augusto Hori
Número de Seguro Social: 222-222-222
Ventas Netas: 8000.00
Tarifa de Comisión: 0.15
Ingresos: 1200.00

4. Resultados

```
carlo@Santantango:~/POO/POO/Proyectos/Proyecto2$ java mx.unam.fi.poo.g1.project2.MainApp
----------------
Empleado asalariado: Empleado: Daniel Contreras
Número de Seguro Social: 111-111-111
Salario semanal: $12000.50
Ingresos: $12000.50
----------------
Empleado por Comisión: Empleado: Augusto Hori
Número de Seguro Social: 222-222-222
Ventas Netas: 8000.00
Tarifa de Comisión: 0.15
Ingresos: $1200.00
----------------
```

Figura 1: Ejecución del código


```
carlo@Santantango:~/POO/POO/Proyectos/Proyecto2$ java mx.unam.fi.poo.g1.project2.MainApp
----------------
Empleado asalariado: Empleado: Daniel Contreras
Número de Seguro Social: 111-111-111
Salario semanal: $12000.50
Ingresos: $12000.50
----------------
Empleado por Comisión: Empleado: Augusto Hori
Número de Seguro Social: 222-222-222
Ventas Netas: 8000.00
Tarifa de Comisión: 0.15
Ingresos: $1200.00
----------------
```


Figura 2: Ejecución del código

5. Conclusiones

El desarrollo del sistema de nómina permitió comprender de manera práctica la aplicación de los principales pilares de la programación orientada a objetos. La implementación de la clase abstracta *Empleado* y sus subclases *EmpleadoAsalariado* y *EmpleadoPorComisión* demostró cómo la herencia y el polimorfismo facilitan la creación de estructuras flexibles y extensibles, capaces de adaptarse a diferentes tipos de empleados sin modificar la lógica principal del programa.

El control de acceso a los atributos mediante el encapsulamiento aseguró la integridad de los datos y reforzó la importancia de mantener un diseño seguro y mantenible. Asimismo, el uso de métodos sobrescritos permitió observar cómo el polimorfismo brinda dinamismo al sistema, posibilitando el manejo uniforme de objetos distintos a través de una misma interfaz.

Por último, este proyecto evidenció que la programación orientada a objetos es una herramienta esencial para el diseño de software modular y reutilizable. La implementación del sistema de nómina no solo resolvió el problema planteado de calcular los ingresos semanales de los empleados, sino que también consolidó las bases conceptuales necesarias para el desarrollo de aplicaciones más complejas en entornos reales.

6. UML

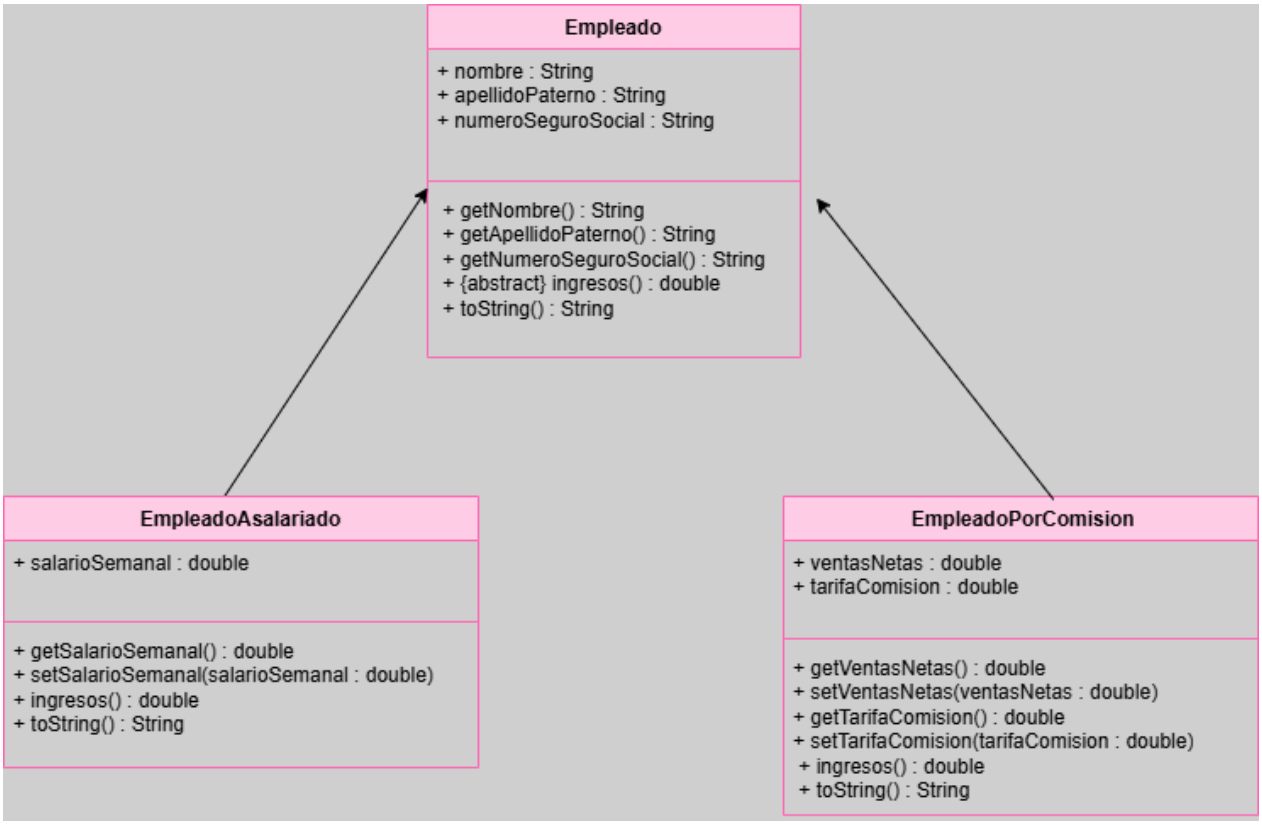

Figura 3: Diagrama de clases.

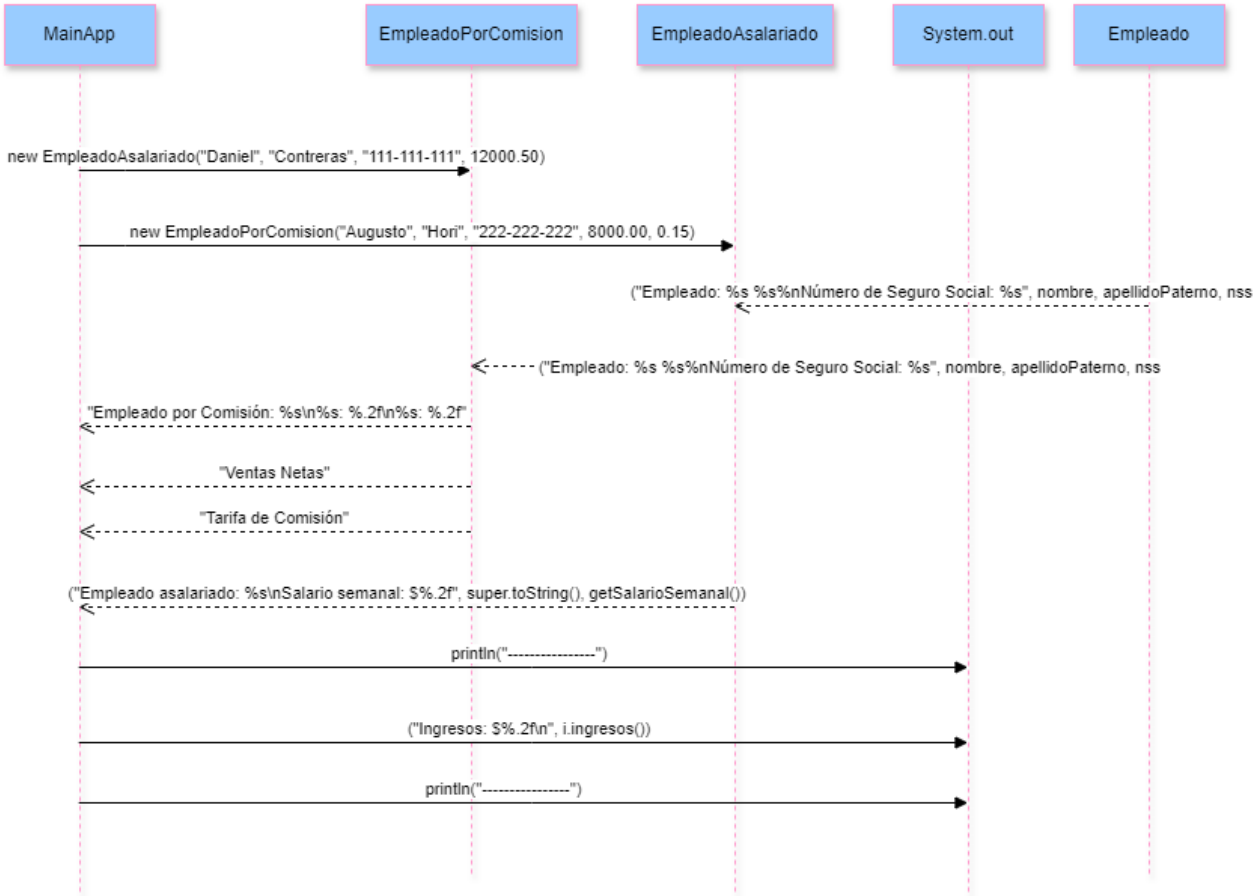

Figura 4: Diagrama de secuencia.

Referencias

[1] Jesus Enrique Guzman Livia. “LENGUAJE DE PROGRAMACIÓN ORIENTADO A OBJETOS Introducción, características principales, objetos, clases, herencia y polimorfismo. Lenguaje de programación java, principales sentencias, estructura. Comandos. Controles y sus propiedades. Estructura de control y procedimiento. Aplicaciones. Otros lenguajes de programación orientado a objetos.” En: (2022).