	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación
Salas A y B

Profesor(a): _____

Asignatura: _____

Grupo: _____

No de Práctica(s): Proyecto 2

Integrante(s): 322246320

322267567

322012051

322236688

322330872

No. de lista o brigada: _____

Semestre _____

Fecha de entrega: _____

Observaciones: _____

_____

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del Problema:** Se busca diseñar un programa que permita representar dos tipos de empleado con características y operaciones principales compartidas, como el nombre del empleado, su apellido y su seguro social; así como características únicas para cada tipo de empleado, como su forma de pago, empaquetando los códigos para organización.

- **Motivación:** La implementación de este código permitirá poner en practica la creación de una clase padre planeada desde un inicio para tener hijos siendo una clase abstracta, permitiendo implementar y practicar nuestros conocimientos en la creación de clases con herencia, sobrescritura y el polimorfismo; así como acostumbrándonos a empaquetar por formalidad y estructura y a la ejecución de códigos empaquetados.

- **Objetivos:** Implementar un programa en Java que mediante el uso de una sola referencia a una clase padre abstracta, se manejen diferentes tipos de empleados utilizando polimorfismo y herencia, mostrando en cada caso las características que tiene cada empleado. Asimismo se busca empaquetar las clases utilizadas, permitiendo manejar los archivos ordenadamente.

2. Marco Teórico

- **Encapsulamiento:** El *Encapsulamiento* es el principio de restringir el acceso directo a los atributos de una clase, de modo que solo puedan ser modificados mediante métodos controlados como getters y setters. Este enfoque nos ayuda a proteger los datos, evita inconsistencias y brinda mayor control del programa. [1]

- **Paquetes** Son el mecanismo que usa Java para facilitar la modularidad del código. Un paquete puede contener una o más definiciones de interfaces y clases, distribuyéndose habitualmente como un archivo. Para utilizar los elementos de un paquete es necesario importar este en el módulo de código en curso, usando para ello la sentencia **import**. [2]

- **Herencia:** Es un concepto que permite definir nuevas clases basadas en clases ya existentes, aprovechando y extendiendo su funcionalidad, reutilizando código. Para heredar una clase a otra, se usa la palabra reservada **extends** al declarar la clase. [3]

  Abstracción: Es la capacidad de representar problemas complejos de una forma mas simple, dejando solo las caracteristicas escenciales y ocultando los detalles innecesarios. [4]

  Clases abstractas e interfaces: Las clases abstractas son un tipo de clases de las cuales no se puede crear un objeto a partir de ellas; una clase abstracta puede contener métodos abstractos, que indican que se deben implementar en

la clase que herede, o métodos con comportamiento ya definido; la forma en la que se crea una clase abstracta es usando la palabra reservada **abstract**.

Una interfaz es un modelo del que tampoco se pueden crear objetos. Contiene solo métodos abstractos y atributos constantes estáticos, por lo que las clases que usen esta interfaz tienen la obligación de implementar todos los metodos heredados. Para crear una interfaz, es con la palabra reservada **interface** y para usarla con **implements**. [4]

- **Polimorfismo:** Es la capacidad de un objeto para adquirir diferentes comportamientos, mas específicamente, es el concepto de implementar métodos con el mismo nombre pero distinto comportamiento. [3]

3. Desarrollo

Para este proyecto se ha implementado el polimorfismo de una clase abstracta, sobrescritura de métodos y comunicación con el constructor de la clase base.

Todas las clases han sido empaquetadas siguiendo la sintaxis del **fullQualifiedName**: **/mx/unam/fi/poo/py2/**, agregando: **package mx.unam.fi.poo.py2;** al inicio de cada clase.

Comenzamos creando la clase abstracta **Empleado** que tiene los atributos privados **nombre, apellido y seguroSocial**, datos que el constructor recibe y llama a los métodos **setters** de cada atributo, además se generaron sus respectivos **getters**. Empleado tiene el método abstracto **ingresos** que se sobrescribirá en cada clase hija. También contiene el método sobrescrito **toString** que imprime los atributos en determinado formato.

La clase **EmpleadoPorComision**, heredada de la clase **Empleado**, tiene los atributos privados **tarifa** y **ventas**. Su constructor se comunica con el constructor de la clase base con la palabra reservada **super**, asignandole los valores corresponfientes; además, el contructor asgina los atributos propios de esta clase.

Para los **setters** de los dos atributos, hicimos la validación para que la tarifa de venta quede estrictamente entre 0 y 1 en el caso de **tarifa**, y para **ventas** validamos que la cantidad sea mayor a 0.

Después tenemos los métodos sobrescritos **toString** y **ingresos**. El primero extiende la implementación de **toString** de la clase base y además imprime los nuevos atributos siguiendo el formato ya creado. El segundo método multiplica las tarifas y las ventas para obtener un total de ingresos obtenidos.

La clase **EmpleadoAsalariado**, heredada de la clase **Empleado**, contiene el atributo privado **salarioSemanal**. Similar a la clase anterior, su constructor se comunica con el constructor de la clase base con **super**, recibiendo los atributos correspondientes; también el constructor asigna los atributos de su propia clase.

En **setSalarioSemanal** simplemente validamos que el salario sea mayor a 0. Igualmente tenemos los métodos sobrescritos **toString** e **ingresos**. El primero funciona de la misma manera que el de la clase anterior, sin embargo, este imprime también al atributo **salarioSemanal** en el formato ya acordado. El segundo método obtiene el salario mensual, al multiplicar **salarioSemanal** por 4.

Finalmente para la clase **MainApp**, en método **main** comenzamos creando una referencia polimórfica de tipo **Empleado**, la cual se 'transforma' en diferentes tipos de empleado a lo largo del programa. Para hacerlo más evidente, fuimos intercalando entre **EmpleadoPorComision** y **EmpleadoAsalariado**. Para cada empleado, se imprimen sus métodos **toString** e **ingresos**, mostrando las características de cada empleado.

4. Diagramas UML

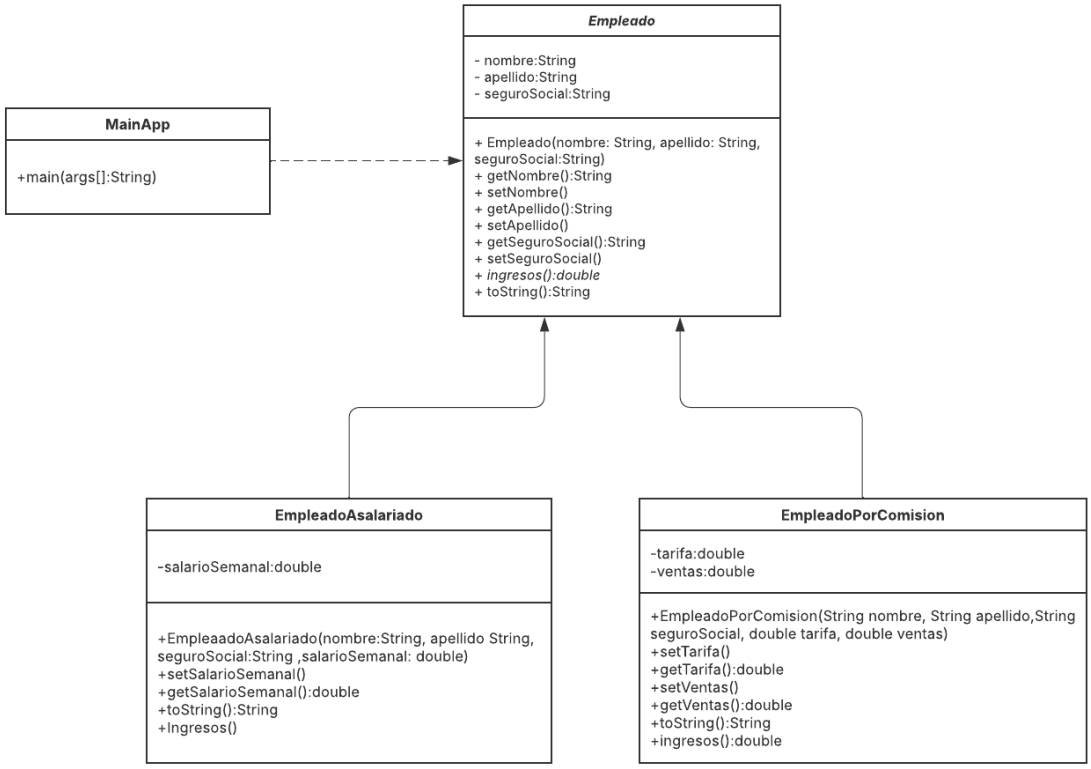

Diagrama UML de Clases

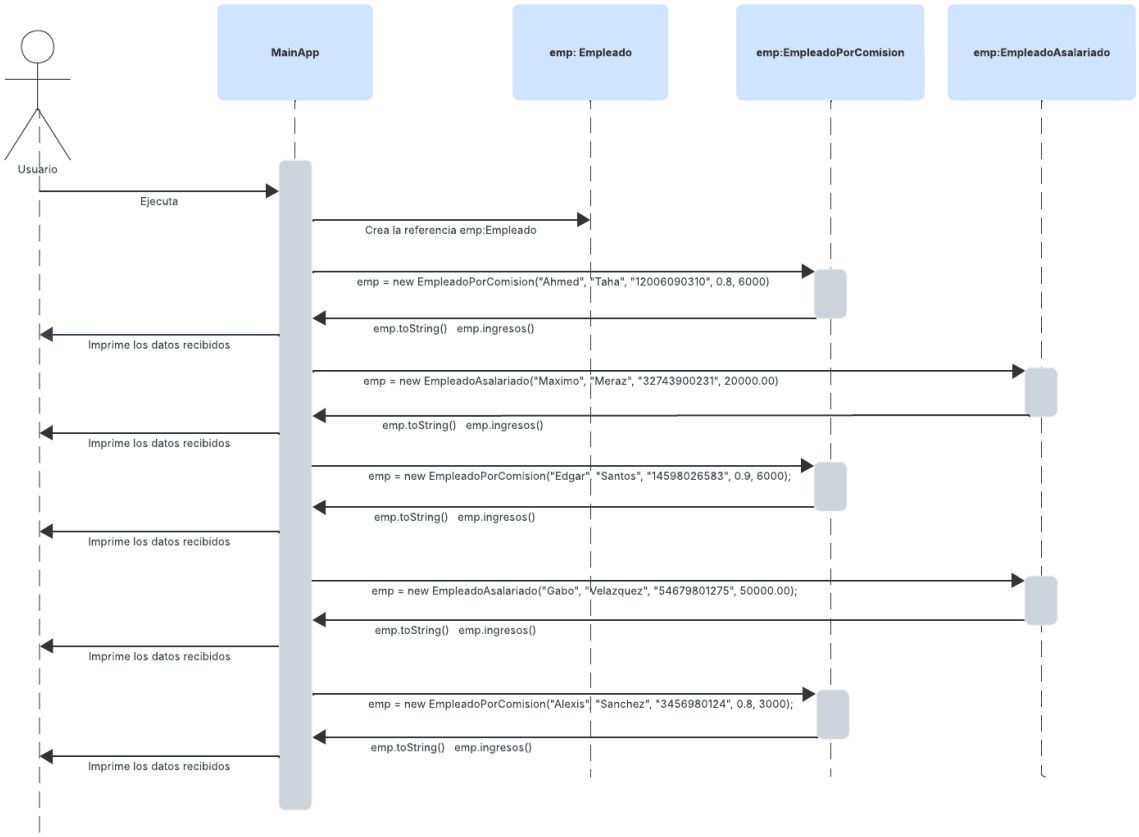

Diagrama UML de Secuencia

5. Resultados

```
maximogab@maximogab-Inspiron-3576:~/POO$ java mx.unam.fi.poo.py2.MainApp
Empleado Nombre= Ahmed, Apellido= Taha, SeguroSocial= 12006090310
Tarifa por comision: 0.8
Ventas netas: 6000.0
Ingreso semanal: $4800.0

Empleado Nombre= Maximo, Apellido= Meraz, SeguroSocial= 32743900231
Salario semanal: 20000.0
Ingreso mensual: $80000.0

Empleado Nombre= Edgar, Apellido= Santos, SeguroSocial= 14598026583
Tarifa por comision: 0.9
Ventas netas: 6000.0
Ingreso semanal: $5400.0

Empleado Nombre= Gabo, Apellido= Velazquez, SeguroSocial= 54679801275
Salario semanal: 50000.0
Ingreso mensual: $200000.0

Empleado Nombre= Alexis, Apellido= Sanchez, SeguroSocial= 3456980124
Tarifa por comision: 0.8
Ventas netas: 3000.0
Ingreso semanal: $2400.0
```

6. Conclusiones

El desarrollo de este proyecto permitió consolidar y aplicar de manera práctica conceptos de la Programación Orientada a Objetos, así como es la herencia para reutilizar código, las clases abstractas para definir un 'esqueleto' o súperclase común, la sobrescritura de métodos para especializar comportamientos en las clases derivadas, y el polimorfismo mediante referencias de la clase base abstracta *Empleado*,permitiendo manipular objetos de distintos tipos desde una misma referencia.

En esta practica, comprendimos la importancia que tiene la comunicaciones entre clases, así como la organización de varios archivos dentro de un mismo paquete para mantener un orden y estructura correcta. Este trabajo nos permitió integrar los principios de POO en un proyecto con un entorno bien estructurado.

Referencias

[1] *Introducción a POO en Java: Encapsulamiento*. 2023. URL: https://openwebinars.net/blog/introduccion-a-poo-en-java-encapsulamiento/.

[2] *Paquetes en Java: qué son, para qué se utilizan, y cómo se usan*. 2019. URL: https://www.campusmvp.es/recursos/post/paquetes-en-java-que-son-para-que-se-utilizan-y-como-se-usan.aspx (visitado 31-10-2025).

[3] *Introducción a POO en Java: Herencia y polimorfismo*. 2023-11-17. URL: https://openwebinars.net/blog/introduccion-a-poo-en-java-herencia-y-polimorfismo/ (visitado 31-10-2025).

[4] *Java Clases Abstractas y el polimorfismo*. 2024-09-28. URL: https://www.arquitecturajava.com/java-clases-abstractas-y-el-polimorfismo/ (visitado 17-10-2025).