	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): M.C. RENE ADRIAN DAVILA PEREZ

Asignatura: PROGRAMACIÓN ORIENTADA A OBJETOS

Grupo: 01

No de Proyecto(s): 02

Integrante(s): 322118311

322094028

322092842

322078673

322067738

No. de lista o brigada: 03

Semestre: 2026-1

Fecha de entrega: 31/10/2025

Observaciones:

CALIFICACIÓN: _____

Índice

1. Introducción

- **Planteamiento del problema.** Para este proyecto se requiere desarrollar un sistema de nómina para una compañía que gestiona a sus empleados los cuales tienen dos modalidades de pago distintas.**Empleados asalariados** que perciben un salario semanal fijo, independientemente de otros factores Y **Empleados por comisión**, cuyos ingresos se calculan como un porcentaje variable de las ventas que han generado. El problema es diseñar una aplicación que pueda manejar estos dos tipos de empleados al mismo tiempo, pero que a su vez aplique el cálculo indicidual de ingresos de forma correcta para cada uno.

- **Motivación.**Este proyecto nos servira para poner en practica lo aprendido en esta unidad sobre Herencia y Polimorfismo, los cuales nos permitiran crear un codigo mas limpio y eficiente, practica indispensable en el ambito profesional.

- **Objetivos.** Al finalizar este proyecto esperamos desarrollar un programa que contenga las siguientes características:

  - Implementar una clase abstracta `Empleado` que encapsule los atributos y comportamientos de los objetos.
  - Definir los métodos abstractos `ingresos()` y `toString()`que despues se sobreescribiran en las subclases
  - Crear las clases `EmpleadoAsalariado` y `EmpleadoPorComision` que hereden de `Empleado` y sobrescriban los métodos abstractos.
  - Aplicar validaciones en los métodos `set` de las subclases para asegurar la integridad de los datos (salarios no negativos y tarifas de comisión en rangos válidos).

2. Marco Teórico

2.1. Herencia:

Es apartir de una clase a la que denominamos clase padre y apartir de esta desarrollar otra que posea las mismas caracteristicas, sin embargo, extendiendo nuevos metodos o funcionalidades. La clase nueva nos ahorra código y que reutiliza código ya definido de en una clase padre y que ademas lo extiende. Dicho de otra forma "La herencia es el mecanismo de basar un objeto o clase en otro objeto (herencia basada en prototipos) o clase (herencia basada en clases), conservando una implementación similar. También se define como derivar nuevas clases (subclases) de las existentes, como superclase o clase base, y luego formarlas en una **jerarquía de clases**.[1]

Listing 1: Sintaxis de herencia

```
class Figura {
    // Clase padre
}
```

```
4
5 class Circulo extends Figura {
6     // Subclase que hereda de Figura
7 }
```

2.2. Polimorfismo:

De una clase padre podemos derivar varias subclases, en ese sentido el polimorfismo es que cada clase padre sera diferente con respecto a cada subclase. Denominamos polimorfismo al "mecanismo que nos permite tener un método en una clase padre como vimos en la herencia y sobrescribirlo en la clase hija".

Esto quiere decir que tendremos el mismo método en ambas clases, pero en la clase hija realizara diferentes acciones. Por lo que el polimorfismo es también denominado **sobreescritura de métodos.**

Hay dos formas de polimorfismo:

En tiempo de ejecución, que tiene que ver con las **interfaces**. La segunda llamada **polimorfismo estático**, la cual determina que método se va a ejecutar durante la compilación. [2]

Listing 2: Sintaxis de polimorfismo

```
1 Figura f;           // Referencia de tipo Figura
2 f = new Circulo(); // Puede apuntar a cualquier subclase de
    Figura
```

2.3. Clases abstractas:

Son clases en las que no se pueden crear objetos, su funcion principal es servir como plantilla para las clases que se heredaran de una clase padre, estas clases pueden contener metodos pero unicamente se les colocara la firma, pues estos metodos deberan definirse en las subclases. [3]

Listing 3: Sintaxis de clase abstracta

```
1 abstract class Figura {
2     abstract void calcularArea(); // Metodo abstracto
3 }
4
5 class Circulo extends Figura {
6     void calcularArea() {
7         // Implementacion del metodo
8     }
9 }
```

2.4. Diagramas UML:

Los diagramas UML en Java son la representación gráfica y estandarizada de la arquitectura de una aplicación antes de codificarla, o para documentarla después. Un diagrama de clases de UML, por ejemplo, funciona como un "plano"que se traduce casi directamente a código [4]

2.5. Diagrama de Clases:

Es un diagrama de estructura estático que describe la arquitectura de un sistema. Muestra las clases del sistema representadas como cajas divididas en tres partes (nombre, atributos y métodos) y, de forma crucial, visualiza las relaciones estáticas entre ellas, como la herencia (generalización), asociación (una clase usa a otra), agregación (una clase tiene a otra) y composición (una clase posee a otra). [5]

2.6. Lucidchart:

Lucidchart es una herramienta de diagramación basada en la web, que permite a los usuarios colaborar y trabajar juntos en tiempo real, creando diagramas de flujo, organigramas, esquemas de sitios web, diseños UML, mapas mentales, prototipos de software y muchos otros tipos de diagrama. [6]

2.7. Funciones de Lucidchart:

- Generar un diagrama de secuencia en UML a partir del marcado de texto.
- Crear un organigrama a partir de un archivo CSV
- Importar la infraestructura de AWS para diagramas de red.
- Importar una base de datos, cuadros y esquemas para ERD.
- Conectar figuras de diagramas con datos en vivo en Google Sheets.[6]

3. Desarrollo

3.1. Diagrama de Clases:

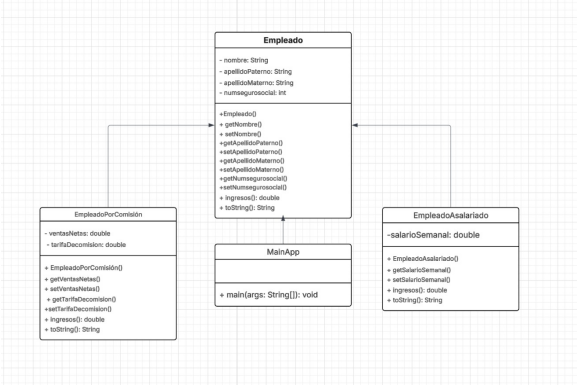

Figura 1: Diagrama de clases

3.2. Diagrama de Secuencia:

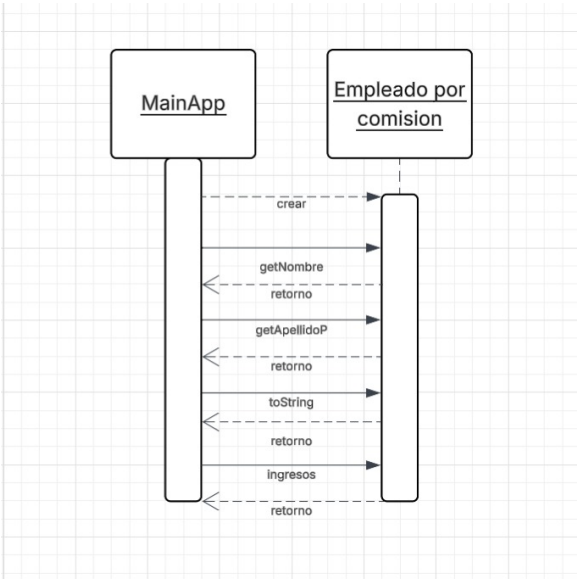

Figura 2: Diagrama de secuencia

3.3. Clase abstracta Empleado:

En esta clase definiremos los atributos principales que tendran todos los tipos de empleado: Iniciaremos encapsulando los atributos que tendran todos los empleados, de tipo String el nombre, apellido paterno, apellido materno y finalmente de tipo entero, el numero de seguro social. Posteriormente inicializamos los atributos mediante un metodo constrcutor. Esta clase declara ademas dos metodos abstractos: **toString()** e **ingresos()**, obligando así a cualquier subclase concreta que herede de Empleado a implementar su propia logica.

3.4. Clase EmpleadoAsalariado:

Esta clase hereda de la clase abstracta Empleado y definira a un Empleado Asalariado el cual tendra los atributos: **salarioSemanal** y utilizara el constructor de la clase padre (vía super()) para inicializar los datos que tienen en comun (nombre, apellidos, seguro social).Posteriormente se sobrescribiran e implementaran métodos abstractos establecidos en la clase Empleaod: **ingresos()**, que define que el ingreso de este tipo de empleado es su salario semanal, y **toString()**, que simplemente devuelvora la cadena **Empleado Asalariado.**

3.5. Clase EmpleadoPorComision:

Esta clase tambie heredara de la clase abstracta Empleado y ademas de los atributos establecidos en la clase padre, tendra tambien los atributos **ventasNetas** y **tarifaDecomision**. Su constructor utilizara super() para pasar la información personal básica al padre (nombre, apellidos, seguro social). Posteriormente mediante sobreesciritura se implementaran los metodos **ingresos()**, que implementaran la lógica de negocio para este tipo de empleado calculando el pago como el producto de las ventas netas por la tarifa de comisión, y mediante **toString()**, que identificara al objeto como ."Empleado por comisión". tambien se implementa la lógica de validación en sus métodos setters para asegurar que las ventas netas no sean negativas y que la tarifa de comisión se mantenga en un rango porcentual válido (entre 0.0 y 1.0).

3.6. MainApp:

Finalmente en el main declaramos una variable "godin"que sera del tipo Empleado (clase abstracta), despues se generara aleatoriamente un numero(1 o 0) y mediante un if haciendo uso de este número, instanciara un objeto de una de las subclases concretas **EmpleadoAsalariado** si es 0, o **EmpleadoPorComision** si es 1 y lo asigna a la variable godin. Finalmente, imprimira todos los detalles del empleado.

4. Resultados

```
PS C:\Users\marti\Desktop> javac mx/unam/fi/poo/proyecto2/*.java
PS C:\Users\marti\Desktop> java mx.unam.fi.poo.proyecto2.MainApp
-----Empleado consultado-----
Alexei
Ramírez
Pedroza
456789
es un Empleado por comisión
210.0
```

Figura 3: Datos de Empleado por Comisión

```
PS C:\Users\marti\Desktop> java mx.unam.fi.poo.proyecto2.MainApp
-----Empleado consultado-----
Edgar
Ruiz
Miguel
123456
es un Empleado Asalariado
20000.0
```

Figura 4: Datos de Empleado Asalariado

5. Conclusiones

Logramos darle solucion al problema mediante el uso de la abstracción en la clase Empleado la cual mediante los metodos ingresos() y toString() forzó a las subclases a sobreescribir estos metodoso lo cual nos permitió que la clase MainApp aplicara el polimorfismo de manera efectiva: al invocar godin.ingresos() sobre una referencia de la superclase, el sistema ejecutel método sobrescrito correspondiente (EmpleadoAsalariado o EmpleadoPorComision). Este mecanismo, complementado por la herencia para reutilizar atributos y el encapsulamiento para validar los datos, fue nuestra una solución teórica para manejar distintas lógicas de negocio en un mismo programa.

6. Bibliografía

- **[1]** AcademiaLab, "Herencia (programación orientada a objetos)", AcademiaLab, 2025. [Enlace]. Disponible en: `https://academia-lab.com/enciclopedia/herencia-programacion-orientada-a-objetos/\#google_vignette`
- **[2]** NetMentor, "Polimorfismo en programación orientada a objetos", NetMentor, 14 sep. 2019. [Enlace]. Disponible en: `https://www.netmentor.es/Entrada/polimorfismo-poo/\#mcetoc_1ehncnesj4`

- **[3]** Arturo Parra, .Entendiendo las Clases Abstractas en Java", Programando-Java, 23 ago. 2023. [Enlace]. Disponible en: `https://www.programandojava.com/blog/clases-abstractas-java/`
- **[4]** C. Larman, Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development, 3rd ed. Upper Saddle River, NJ: Prentice Hall PTR, 2005.
- **[5]** G. Booch, J. Rumbaugh, and I. Jacobson, The Unified Modeling Language User Guide, 2nd ed. Upper Saddle River, NJ: Addison-Wesley Professional, 2005
- **[6]** TodosLosHechos.es, "¿Para que nos sirve el programa lucidchart?," TodosLosHechos.es, 16 de enero de 2022. [En línea]. Disponible en: https://todosloshechos.es/para-que-nos-sirve-el-programa-lucidchart/. [Consultado: 31 de octubre de 2025].