Definir estándares de codificación de acuerdo a plataforma de desarrollo elegida

David Felipe Morales

María Fernanda Ibáñez Benavides

Dilan Alexander Robayo

Juan Sebastian Florez

Julio Roberto Galvis

Servicio Nacional de Aprendizaje

Análisis y desarrollo de Software

2024

Bogotá D.C

Tabla de contenidos

Introduccion

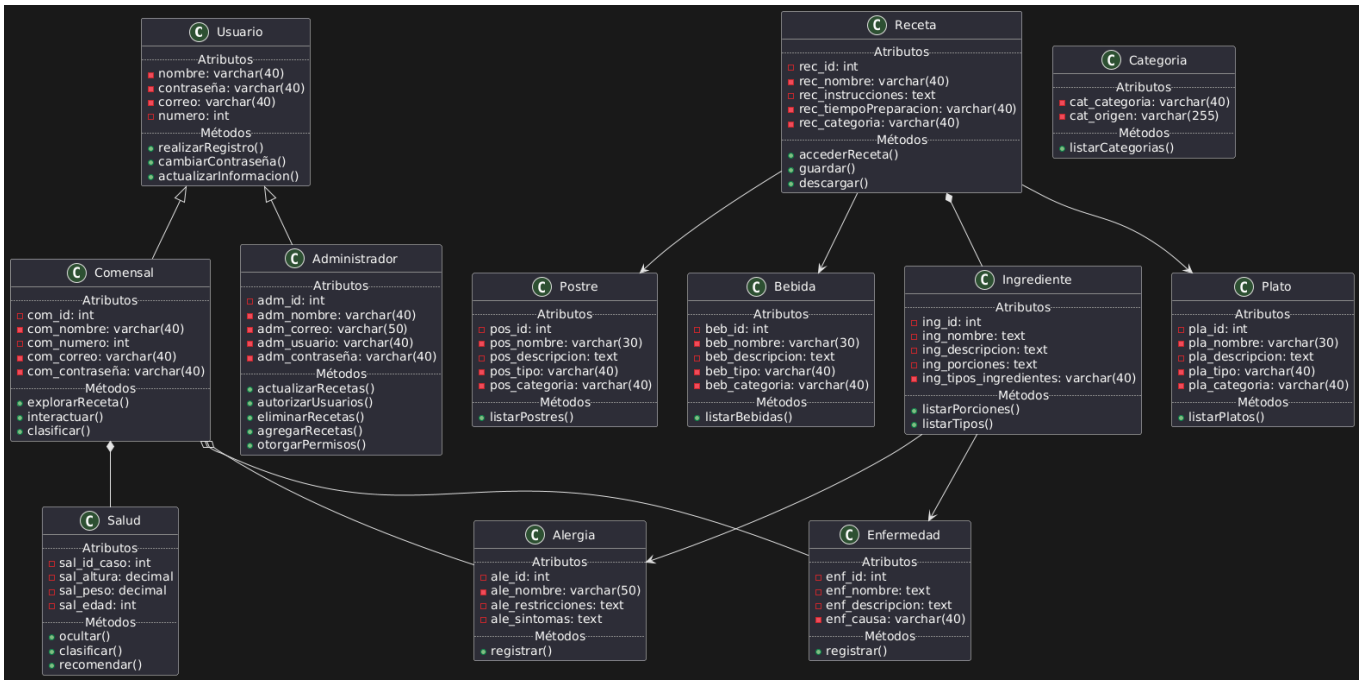
La Programación Orientada a Objetos se basa en conceptos como abstracción, encapsulación, herencia y polimorfismo para organizar y gestionar el código de forma eficiente. En POO, los objetos agrupan datos y comportamientos relacionados y se comunican entre sí mediante interfaces claras, manteniendo ocultos sus detalles internos. Este ocultamiento de información es esencial, ya que permite que los objetos interactúen sin conocer su implementación, lo que facilita la modularidad y mejora el mantenimiento del software.

Objetivo

Explorar y comprender las características fundamentales de la programación orientada a objetos (POO), enfocándose en la implementación de clases y objetos, así como en el manejo de sus atributos, métodos, constructores, destructores y sobrecarga de métodos. Además, se busca analizar cómo se lleva a cabo la comunicación entre clases mediante asociaciones, composición/agrupación y herencia, destacando la importancia del ocultamiento de información para una buena ingeniería de software.

The World Food

Diagrama de clases



Clases

Atributos:

- Se declaran al inicio de la clase
- Usar **private** para encapsulamiento
- Seguir el formato: **private tipoVariable nombreVariable;**
- Los tipos pueden ser:
  - Para texto: **String**
  - Para números enteros: **int**
  - Para números decimales: **decimal** o **double**
  - Para textos largos: **text** se implementa como **String**

Constructores:

- Tienen el mismo nombre que la clase
- Se pueden tener múltiples constructores:
  - Constructor vacío: sin parámetros
  - Constructor con parámetros: incluye todos los atributos
- Usar **this** para referenciar los atributos de la clase

Métodos:

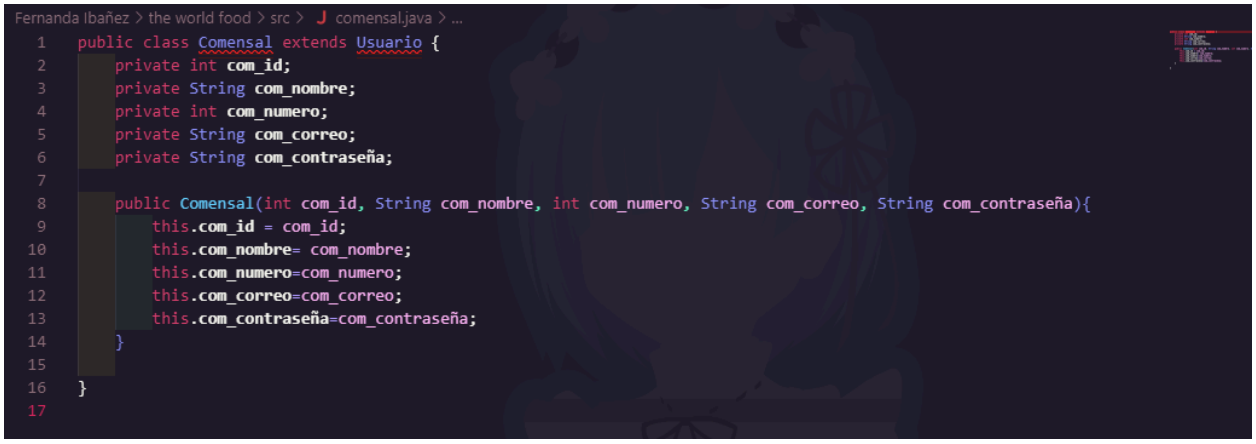
- Generalmente son **public**
- Formato: **public tipoRetorno nombreMetodo(parametros)**
- Si no retornan nada, usar **void**
- Ejemplos del diagrama:
  - **public void realizarReceta()**
  - **public void guardar()**
  - **public void listarPostres()**

Convenciones de nombres:

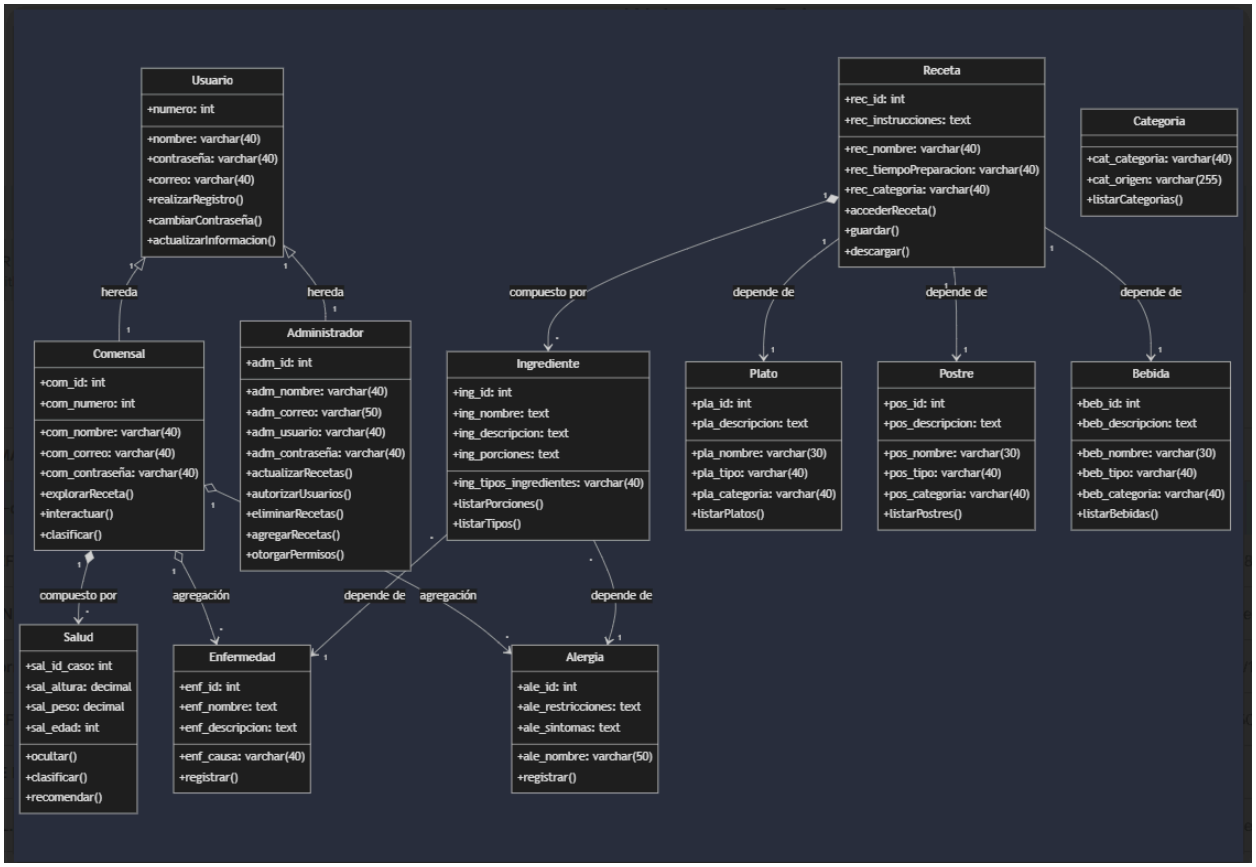
- **Clases:** Primera letra mayúscula **(PascalCase)**

- **Métodos y atributos:** Primera letra minúscula **(camelCase)**
- **Constantes:** Todo en mayúsculas con guiones bajos

Ejemplo :



Relación



Herencia (representada por las flechas con triángulo vacío

- Usuario es la clase padre/superclase de:
  - **Comensal**
  - **Administrador** Esto significa que tanto Comensal como Administrador

Receta es la clase central que se relaciona con:

- **Postre** (una Receta puede ser un Postre)
- **Bebida** (una Receta puede ser una Bebida)
- **Plato** (una Receta puede ser un Plato)

- **Categoría** (una Receta pertenece a una Categoría)
- **Ingrediente** (una Receta contiene Ingredientes)

Relaciones con la clase Salud:

- **Comensal** se relaciona con **Salud** (un Comensal puede tener datos de Salud)
- **Salud** se relaciona con:
  - **Alergia** (los datos de Salud pueden incluir Alergias)
  - **Enfermedad** (los datos de Salud pueden incluir Enfermedades)

Este tipo de estructura sugiere que:

- Un Usuario puede ser de tipo Comensal o Administrador
- Una Receta puede clasificarse como Postre, Bebida o Plato
- Cada Receta tiene una Categoría y contiene Ingredientes
- Los Comensales tienen información de Salud asociada
- La información de Salud puede incluir Alergias y Enfermedades

Cardinalidad

Relaciones con tbl_recetas (tabla central):

tbl_recetas → tbl_bebida: (1:1) Una receta puede ser una bebida

tbl_recetas → tbl_platos: (1:1) Una receta puede ser un plato

tbl_recetas → tbl_postres: (1:1) Una receta puede ser un postre

tbl_recetas → tbl_categorias: (N:1) Muchas recetas pueden pertenecer a una categoría

tbl_recetas → tbl_ingredientes: (1:N) Una receta puede tener muchos ingredientes

Relaciones con tbl_comensal:

tbl_comensal → tbl_rol: (N:1) Muchos comensales pueden tener un rol

tbl_comensal → tbl_salud: (1:1) Un comensal tiene una información de salud

Relaciones con tbl_salud:

- tbl_salud → tbl_alergias: (1:N) Un registro de salud puede tener múltiples alergias
- tbl_salud → tbl_enfermedad: (1:N) Un registro de salud puede tener múltiples enfermedades

Relaciones con tbl_rol:

tbl_rol → tbl_administradores: (1:N) Un rol puede tener múltiples administradores

Relaciones con tbl_ingredientes:

tbl_ingredientes → tbl_ingredientes_audit: (1:1) Un ingrediente tiene un registro de auditoría

Conclusion

Con base al material de apoyo anterior, se ha realizado de manera detallada este informe en el cual se explicaron conceptos básicos de JAVA, conociendo todos estos conceptos se puede deducir que la implementación de JAVA en cualquier proyecto no solamente ayuda a que el programa sea más funcional y lógico posible.