Taller de POO

Aprendiz:

Sergio Andrés Bustos Mondragon

Instructor:

Mayron Salazar

Competencia:

Programación

Ficha:

3210115

Programa:

Técnico en Programación de Software

Fecha:

25 de octubre del 2025

Preguntas del Taller

Parte 1 — Investigación Teórica

1. ¿Qué es la Programación Orientada a Objetos (POO) y cuáles son sus ventajas

frente a la programación estructurada?

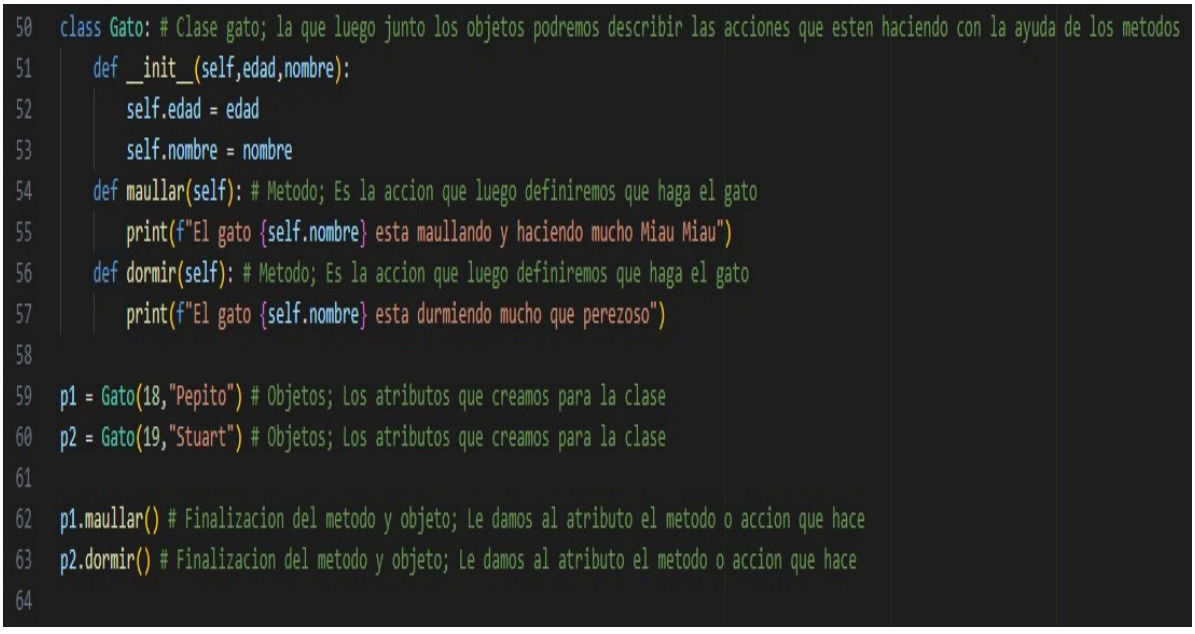
La Programación orientada a objetos o POO es un paradigma de programación, que permite organizar el código como una relación entre objetos, como en resumen es una estructura que permite optimizar el rendimiento de un aplicativo.

Frente a la programación estructurada la POO tiene ciertas ventajas como:

- Los cambios se hacen en una clase y se reflejan en los objetos relacionados,ósea,se puede optimizar o mantener más facil
- Ofrece la posibilidad de heredar de unas clases a otras (clase padre a clase hija)
- Podemos hacer miles de ideas de cómo se puede comportar la clase.
- El codigo se divide en clases y objetos, no bloques de codigo suelto, siendo más organizado
- Permite reusar código ya escrito gracias a la herencia.
- No se encapsulan los atributos, ni el acceso a los atributos desde las funciones que también están encapsuladas dentro de la clase.

2. ¿Qué es una clase? ¿Qué es un objeto? Explica con tus palabras y da un ejemplo.

Clase: Es una entidad o lo que podemos describir y asignarle una serie de adjetivos. Define atributos (características) y métodos (acciones).
Objeto: Representa una instancia específica de un tipo de objeto, con su propio "estado" (los valores de sus atributos o caracteristicas) y "comportamiento" (sus métodos).
Ejemplo:



```
50  class Gato: # Clase gato; la que luego junto los objetos podremos describir las acciones que esten haciendo con la ayuda de los metodos
51      def __init__(self,edad,nombre):
52          self.edad = edad
53          self.nombre = nombre
54      def maullar(self): # Metodo; Es la accion que luego definiremos que haga el gato
55          print(f"El gato {self.nombre} esta maullando y haciendo mucho Miau Miau")
56      def dormir(self): # Metodo; Es la accion que luego definiremos que haga el gato
57          print(f"El gato {self.nombre} esta durmiendo mucho que perezoso")
58
59  p1 = Gato(18,"Pepito") # Objetos; Los atributos que creamos para la clase
60  p2 = Gato(19,"Stuart") # Objetos; Los atributos que creamos para la clase
61
62  p1.maullar() # Finalizacion del metodo y objeto; Le damos al atributo el metodo o accion que hace
63  p2.dormir() # Finalizacion del metodo y objeto; Le damos al atributo el metodo o accion que hace
64
```

3. Define los conceptos de atributo, método y constructor en POO.

Atributo: Son las propiedades o características que tiene un objeto.

-Ejemplo: En el codigo anterior los atributos son las propiedades de la clase;

```
50  class Gato: # Clase gato; la que luego junto los objetos podremos describir las acciones que esten haciendo con la ayuda de los metodos
51      def __init__(self,edad,nombre):
52          self.edad = edad
53          self.nombre = nombre
54      def maullar(self): # Metodo; Es la accion que luego definiremos que haga el gato
55          print(f"El gato {self.nombre} esta maullando y haciendo mucho Miau Miau")
56      def dormir(self): # Metodo; Es la accion que luego definiremos que haga el gato
57          print(f"El gato {self.nombre} esta durmiendo mucho que perezoso")
58
59  p1 = Gato(18,"Pepito") # Objetos; Los atributos que creamos para la clase
60  p2 = Gato(19,"Stuart") # Objetos; Los atributos que creamos para la clase
61
62  p1.maullar() # Finalizacion del metodo y objeto; Le damos al atributo el metodo o accion que hace
63  p2.dormir() # Finalizacion del metodo y objeto; Le damos al atributo el metodo o accion que hace
64
```

Por ejemplo en el primer objeto o gato pusimos:

18,”Pepito” ...

indicando que tiene 18 años y “Pepito” es su nombre.

Al igual que con el segundo objeto o gato...

Método: son las acciones o comportamientos de los objetos.

Ejemplo: El gato puede dormir o maullar teniendo en cuenta el ejemplo anterior

Constructor: es un método especial que se ejecuta al crear el objeto, y sirve para darle valores iniciales a los atributos

4. Explica qué significa encapsulación, herencia, polimorfismo y abstracción.

- Encapsulación: La encapsulación protege los datos de un objeto para que no se modifiquen directamente desde fuera de la clase.
En Python se logra usando guiones bajos (_atributo o __atributo) y métodos getter y setter.

Se hace usando modificadores como private o protected (en Python se usa _atributo).

Ej: Una cuenta de un banco no permite cambiar el saldo directamente, sino con métodos como depositar() o retirar().

-Herencia: Permite que una clase hija herede atributos y métodos de otra padre.

Esto ahorra código y realiza la reutilización de este.

Ej: Una clase Perro y una clase Gato pueden heredar de Animal, porque ambos tienen nombre y edad.

-Polimorfismo: Un mismo método puede funcionar de distintas formas según el objeto.

Ej: hacersonido() puede imprimir "Guau" en un perro y "Miau" en un gato o”Muu” en una vaca,esto según el objeto

-Abstracción: Consiste en mostrar solo lo esencial de un objeto y ocultar los detalles internos.

Ej: Cuando se maneja una bicicleta solo se usa el manubrio y los pedales de abajo, no se necesita saber cómo funciona las ruedas, las cadenas etc para que la bicicleta ruede

5. ¿Qué es UML y para qué sirven los diagramas de clases, de objetos, de secuencia y de actividad?

¿ Que es UML?

UML (Unified Modeling Language o Lenguaje Unificado de Modelado) es un lenguaje visual que se usa en programación y diseño de software para representar gráficamente cómo funciona un sistema.

No es un lenguaje de programación, sino un conjunto de diagramas que sirven para planear, organizar y entender un sistema antes de programarlo.

Su propósito principal es facilitar la comunicación entre programadores, diseñadores y clientes, organizar mejor las cosas, prevenir errores de y dejar una documentación clara del sistema.

Ejemplo: Se piensa en UML como por ejemplo los planos eléctricos de un cajero: Antes de construir, el arquitecto dibuja planos para ver la estructura, las conexiones y los espacios de los cableados.

Ejemplo: Se piensa igual pero en el desarrollo de software, antes de escribir el código, se usan diagramas UML para entender qué clases habrá, cómo se relacionan, qué pasos sigue el sistema y cómo interactúan sus partes, es como un diagrama de flujo pero muy centrado en prevenir errores y mejorar los procesos y no describirlo como lo hace el diagrama de flujo, concluyendo que no se puede usar solo en el desarrollo de software, tambien en la vida real

¿Para qué sirven los diagramas de clases, de objetos, de secuencia y de actividad?

- Diagrama de clases

El diagrama de clases es uno de los más importantes en UML porque muestra la estructura del sistema.
Representa las clases con sus atributos (características) y métodos (acciones), además de las relaciones entre ellas, como la herencia o las asociaciones.

Sirve para organizar y planificar el sistema antes de programarlo, mostrando qué clases existen, qué hacen y cómo se relacionan.

Ejemplo:

En un zoológico se puede tener una clase llamada Animal, que tiene el atributo tipoDeAnimal y el método hacerSonido().
A partir de ella pueden heredar las clases Perro y Gato, que son tipos específicos de animal.
Cada una sobrescribe el método hacerSonido() para emitir sonidos diferentes: el perro dirá “Guau” y el gato “Miau”.
Así, el diagrama de clases muestra claramente la relación entre estas clases y la herencia que existe entre ellas.

- Diagrama de objetos

El diagrama de objetos es parecido al diagrama de clases, pero en lugar de mostrar la estructura general, muestra ejemplos concretos de las clases, es decir, los objetos creados a partir de ellas.

Sirve para visualizar cómo se ven y funcionan los objetos reales del sistema en un momento determinado, mostrando los valores reales que tienen sus atributos.

Ejemplo:

Siguiendo el mismo caso, podemos tener un objeto perro1 de la clase Perro con el atributo tipoDeAnimal = “Canino” y otro objeto gato1 de la clase Gato con el atributo tipoDeAnimal = “Felino”. Este diagrama permite ver que, aunque ambos provienen de la clase Animal, cada uno tiene sus propios valores y comportamientos específicos.

- Diagrama de secuencia

El diagrama de secuencia muestra cómo los objetos se comunican entre sí a lo largo del tiempo, mediante el envío de mensajes o llamadas a métodos.
Permite entender el orden en que ocurren las acciones dentro de un proceso o caso de uso.
Sirve para:

Visualizar la comunicación entre los objetos.

Analizar el flujo de mensajes paso a paso y comprender el comportamiento dinámico del sistema.

Ejemplo:
En el caso del zoológico, si el Usuario solicita que el perro haga un sonido:

1. El usuario envía una solicitud al objeto perro1.
2. El objeto perro1 recibe el mensaje y ejecuta el método hacerSonido().
3. El sistema responde mostrando el resultado: “Guau”.

Este diagrama ayuda a visualizar la interacción entre el usuario y los objetos del sistema en un orden cronológico.

- Diagrama de actividad

El diagrama de actividad representa el flujo de trabajo o de acciones que se realizan dentro de un proceso. Muestra paso a paso qué actividades se ejecutan, qué decisiones se toman y cómo avanza el control del sistema de una acción a otra.

Sirve para:

Entender la lógica de un proceso y analizar el orden de las actividades

Identificar decisiones, bifurcaciones o condiciones dentro del flujo.

Ejemplo:
Si el usuario quiere escuchar el sonido de un animal, el proceso sería:

1. El usuario inicia la acción “Hacer sonido”.
2. El sistema identifica qué tipo de animal es (por ejemplo, un perro o un gato).
3. Se verifica el tipo de animal.
4. Si es un Perro, el sistema ejecuta la acción hacerSonido() y muestra “Guau”.
5. Si es un Gato, ejecuta hacerSonido() y muestra “Miau”.
6. El proceso termina una vez que el sonido ha sido mostrado.

De esta manera, el diagrama de actividad muestra de forma clara cómo fluye el proceso desde que el usuario inicia una acción hasta que el sistema genera el resultado.

6. ¿Cuáles son las diferencias principales al implementar POO en Python, Java y C++?

Python;

o Muy fácil de aprender para principiantes

o No exige definir tipos de variables como Java

o No se necesita declarar todo dentro de las clases (más fácil).

o Constructor: __init__

Java;

o Más robusto pero largo y todo el código debe estar dentro de clases.

o Se debe definir el tipo de las variables (int, double,String, boolean).

o El constructor tiene el mismo nombre de la clase.

C++;

oLenguaje más complejo, mezcla programación estructurada y POO,siendo mas dificil

oSe debe gestionar la memoria (con new y delete).

oEl constructor tiene el mismo nombre de la clase.

oTiene herencia múltiple (una clase puede heredar de varias).

Parte 2 — Diagramas UML

7. Elige un mini-proyecto (ejemplo: biblioteca, gestión de cursos o tienda online) y responde:

- ¿Qué clases identificas en el problema?

- ¿Qué atributos y métodos tendría cada clase?

- ¿Qué relaciones existen entre ellas?

Mini proyecto; Gestor de productos

- ¿Qué clases identificas en el problema?

1) Productos

Representa productos generales que no son alimenticios con atributos básicos (nombre, precio, marca, fechas) y métodos de gestión (comprar, registrar, vender).

2) ProductoAlimenticio

Representa productos alimenticios. Hereda de Productos y agrega un atributo extra (calorias) y redefine el método comprar().

- ¿Qué atributos y métodos tendría cada clase?

1) Productos

Atributos (públicos):

nombre → nombre del producto

precio → precio del producto

marca → marca del producto

fechaenquesehizo → fecha de fabricación

fechaenquesevence → fecha de vencimiento

Métodos (públicos):

comprar() → imprime información del producto al comprarlo

registrar() → imprime información al registrarlo en la tienda

vender() → imprime información al venderlo

2) ProductoAlimenticio (hereda de Productos)

Atributos adicionales:

calorias → información nutricional del producto alimenticio

Métodos redefinidos / polimórficos:

comprar() → imprime un mensaje distinto, incluyendo calorias

Métodos heredados sin cambios:

registrar()

vender()

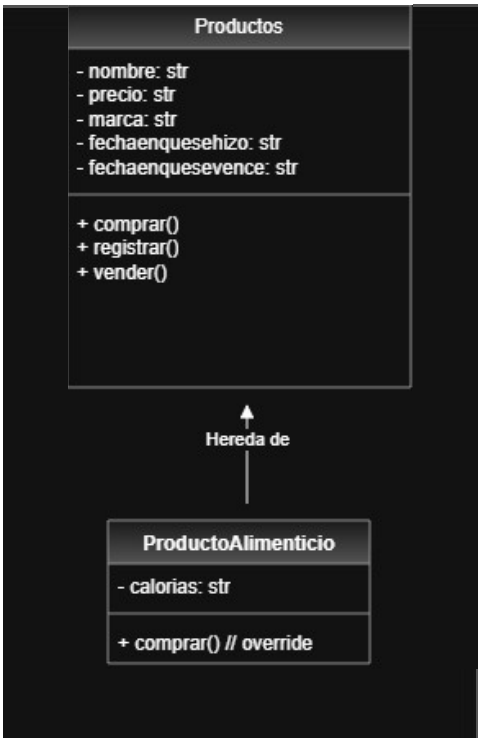
- ¿Quérelaciones existen entre ellas? Herencia (ProductoAlimenticio → Productos):

ProductoAlimenticio es un tipo de Productos. Hereda atributos y métodos de Productos.

Puede redefinir métodos heredados (polimorfismo), como comprar(). Polimorfismo: El

método comprar() funciona de manera diferente según el tipo de objeto (Productos o ProductoAlimenticio).

8. Realiza un diagrama de clases para el problema elegido.



9. Crea un diagrama de objetos con instancias concretas de tus clases.

10. Dibuja un diagrama de secuencia que muestre cómo interactúan los objetos en un caso de uso (ej.: inscribir estudiante, prestar libro, hacer compra).

Diagrama en el caso de comprar():

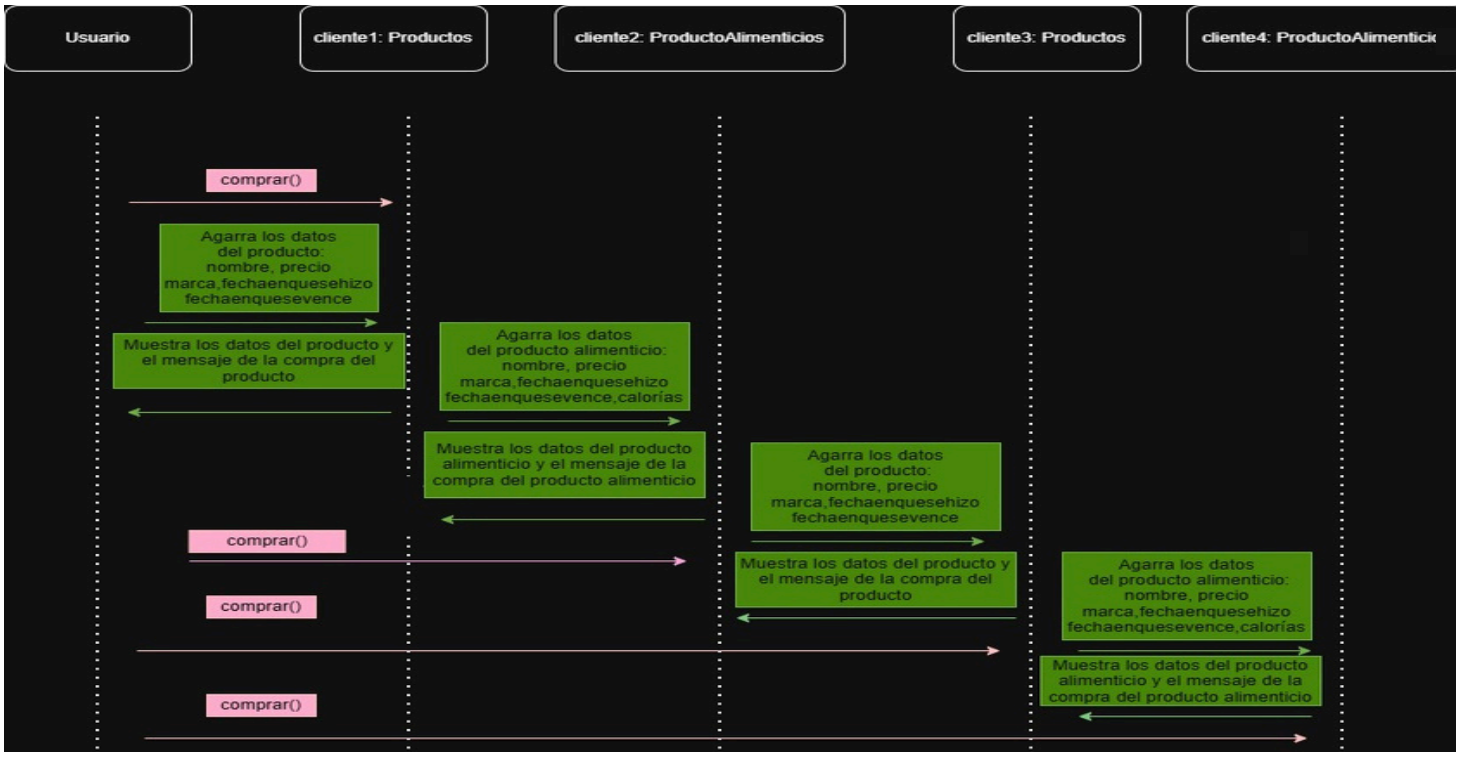


Diagrama en el caso de registrar(): Diferente ya que el método registrar() solo esta en la clase productos y no en la clase ProductoAlimencicio

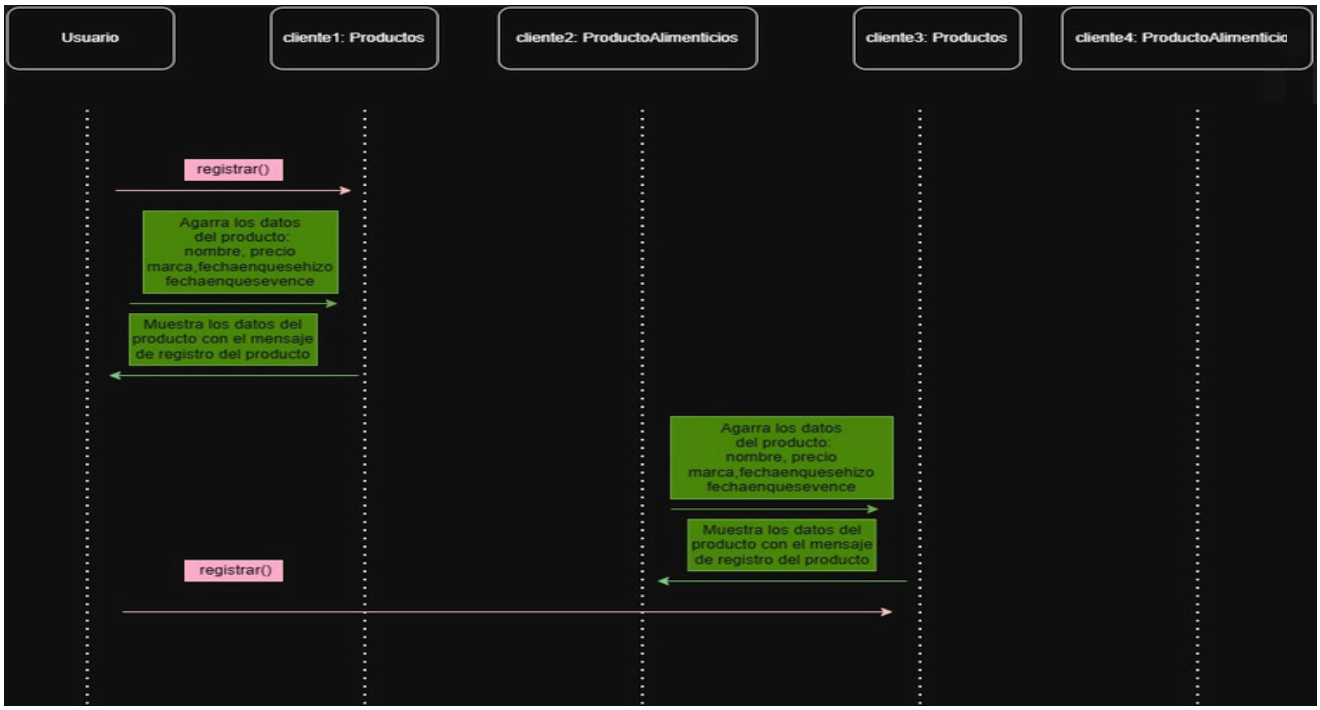
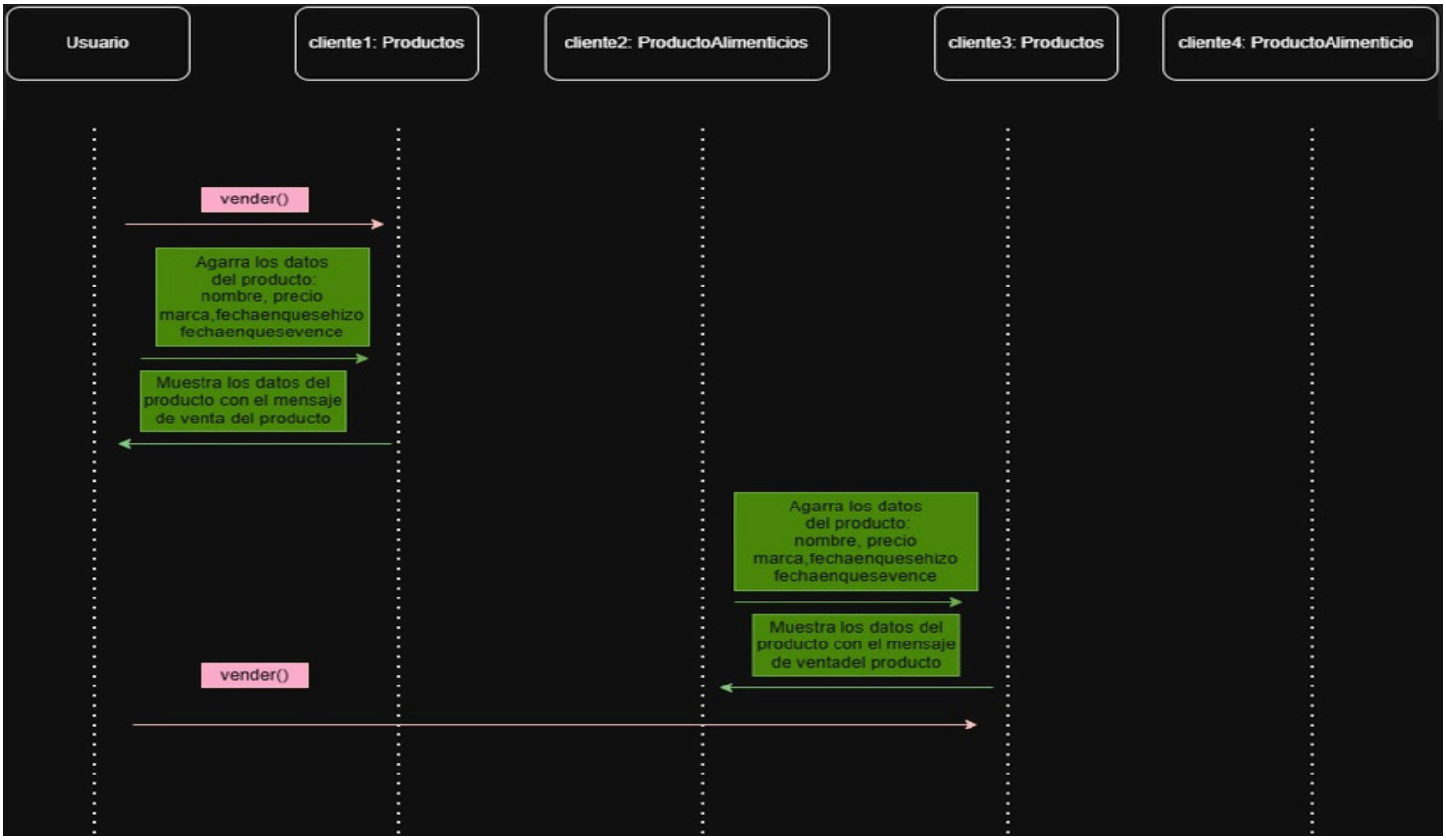
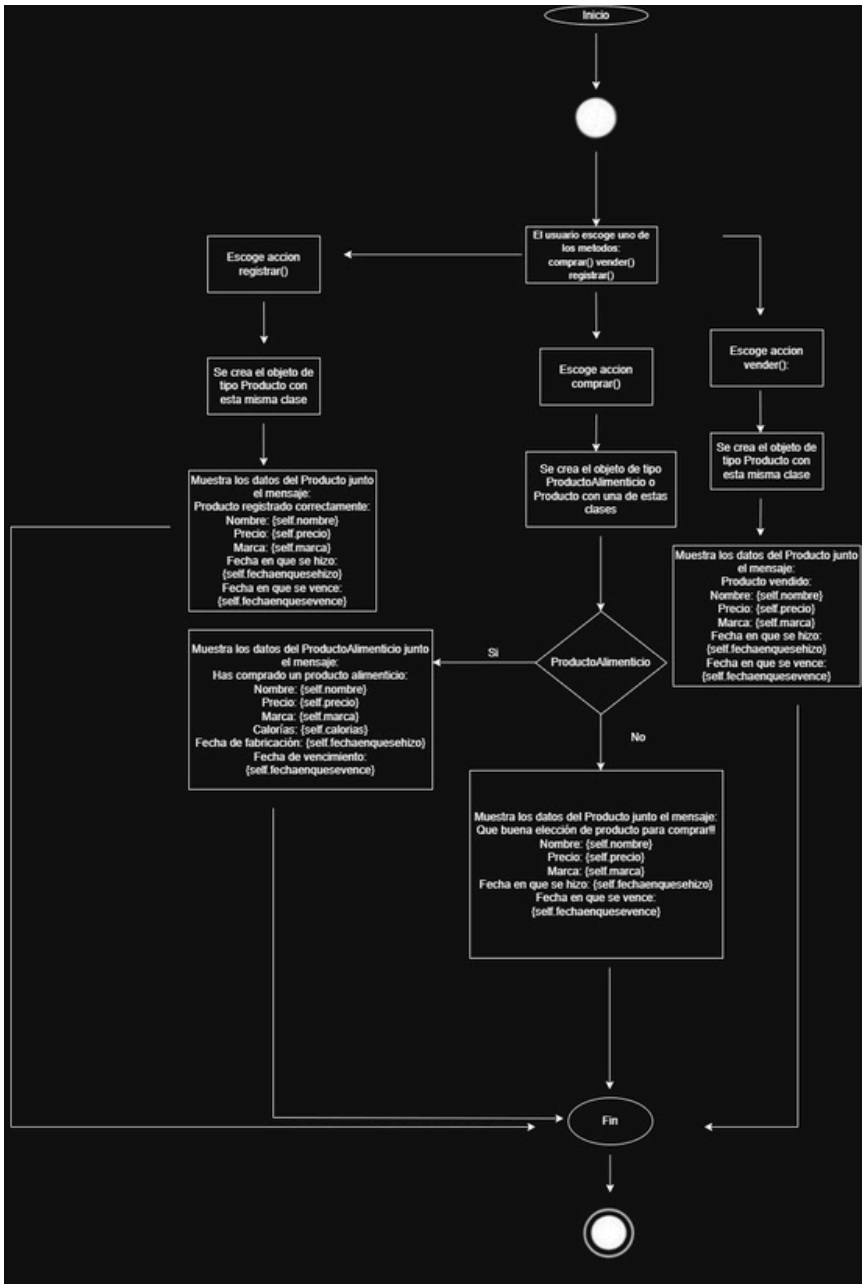


Diagrama en el caso de vender(): Diferente ya que el método vender() solo esta en la clase productos y no en la clase ProductoAlimencicio



11. Dibuja un diagrama de actividad con el flujo de pasos del caso de uso elegido.



Parte 3 — Implementación en Python

12. Implementa en Python las clases de tu diagrama.

13. Agrega atributos, métodos y una relación de herencia.

14. Muestra un ejemplo de polimorfismo en tu código.

15. Ejecuta tu programa y muestra la salida

Codigo:

```python
# Gestor de productos con herencia y polimorfismo

class Productos:
    def __init__(self, nombre, precio, marca, fechaenquesehizo, fechaenquesevence):
        self.nombre = nombre
        self.precio = precio
        self.marca = marca
        self.fechaenquesehizo = fechaenquesehizo
        self.fechaenquesevence = fechaenquesevence

    def comprar(self):
        print(f"""
Que buena elección de producto para comprar!!
Nombre: {self.nombre}
Precio: {self.precio}
Marca: {self.marca}
Fecha en que se hizo: {self.fechaenquesehizo}
Fecha en que se vence: {self.fechaenquesevence}
        """)

    def registrar(self):
        print(f"""
Producto registrado correctamente:
Nombre: {self.nombre}
Precio: {self.precio}
Marca: {self.marca}
Fecha en que se hizo: {self.fechaenquesehizo}
Fecha en que se vence: {self.fechaenquesevence}
        """)

    def vender(self):
        print(f"""
Producto vendido:
Nombre: {self.nombre}
Precio: {self.precio}
Marca: {self.marca}
Fecha en que se hizo: {self.fechaenquesehizo}
Fecha en que se vence: {self.fechaenquesevence}
        """)


# Clase hija que hereda de Productos
class ProductoAlimenticio(Productos):
    def __init__(self, nombre, precio, marca, fechaenquesehizo, fechaenquesevence, calorias):
        # Heredamos atributos del padre
        super().__init__(nombre, precio, marca, fechaenquesehizo, fechaenquesevence) # Super permite reutilizar el código de la clase padre sin volver a escribirlo.
        self.calorias = calorias  # Atributo nuevo de la clase hija

    # Polimorfismo: redefinimos el método comprar()
    def comprar(self):
        print(f"""
Has comprado un producto alimenticio:
Nombre: {self.nombre}
Precio: {self.precio}
Marca: {self.marca}
Calorías: {self.calorias}
Fecha de fabricación: {self.fechaenquesehizo}
Fecha de vencimiento: {self.fechaenquesevence}
        """)


# --- Ejemplos de instancias ---
cliente1 = Productos("Perfume", "45000", "Aromax", "10/09/2025", "10/09/2030")
cliente2 = ProductoAlimenticio("Yogurt", "2500", "Alpina", "20/10/2025", "27/10/2025", "120 kcal")
cliente3 = Productos("Iphone", "14.500.000","Apple","10/08/2024","10/08/2030")
cliente4 = ProductoAlimenticio("Leche","15.000","Alqueria","10/09/2005","10/06/2025","120 kcal")

# --- Ejemplo de polimorfismo ---
# Cada cliente actua de diferente forma dependiendo del producto que compro; Si es alimenticio o si es un producto electronico
cliente1.comprar()
cliente2.comprar()
cliente3.comprar()
cliente4.comprar()
```

Salida:

```
Página de códigos activa: 65001

C:\Users\usuario\Downloads\Certificados,titulos\SENA TITULOS\TECNICO PROGRAMACION DE SOFTWARE>C:/Users/usuario/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/usuario/Downloads/Certificados,t
itulos/SENA TITULOS/TECNICO PROGRAMACION DE SOFTWARE/TallerPOO/Codigo Python/codigogestordeproductos.py"

Que buena elección de producto para comprar!!
Nombre: Perfume
Precio: 45000
Marca: Aromax
Fecha en que se hizo: 10/09/2025
Fecha en que se vence: 10/09/2030


Has comprado un producto alimenticio:
Nombre: Yogurt
Precio: 2500
Marca: Alpina
Calorías: 120 kcal
Fecha de fabricación: 20/10/2025
Fecha de vencimiento: 27/10/2025


Que buena elección de producto para comprar!!
Nombre: Iphone
Precio: 14.500.000
Marca: Apple
Fecha en que se hizo: 10/08/2024
Fecha en que se vence: 10/08/2030


Has comprado un producto alimenticio:
Nombre: Leche
Precio: 15.000
Marca: Alqueria
Calorías: 120 kcal
Fecha de fabricación: 10/09/2005
Fecha de vencimiento: 10/06/2025


C:\Users\usuario\Downloads\Certificados,titulos\SENA TITULOS\TECNICO PROGRAMACION DE SOFTWARE>
```

Parte 4 — Implementación en Java o C++

16. Implementa el mismo ejercicio en Java o C++.

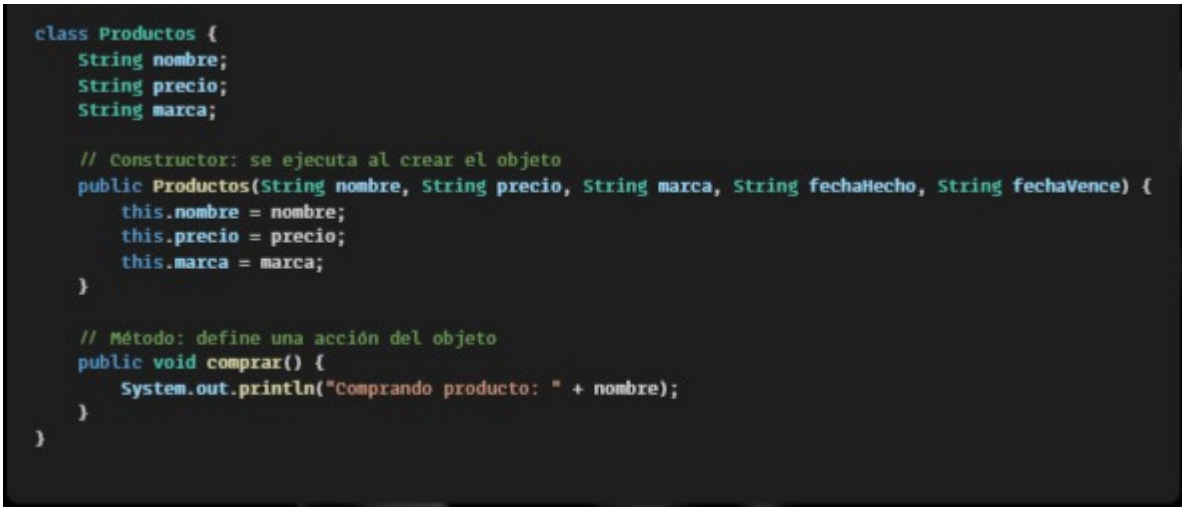
17. Explica cómo se definen las clases y métodos en el lenguaje que elegiste.

En Java, las clases son moldes que permiten crear objetos y se definen con la palabra clave class, seguida del nombre de la clase y un bloque de código {}.
Dentro de una clase se pueden definir:

Atributos: describen las características del objeto (como nombre, precio o marca).

Métodos: describen las acciones o comportamientos que puede realizar el objeto.

Ejemplo de comprar() en la clase producto:



```java
class Productos {
    String nombre;
    String precio;
    String marca;

    // Constructor: se ejecuta al crear el objeto
    public Productos(String nombre, String precio, String marca, String fechaHecho, String fechaVence) {
        this.nombre = nombre;
        this.precio = precio;
        this.marca = marca;
    }

    // Método: define una acción del objeto
    public void comprar() {
        System.out.println("Comprando producto: " + nombre);
    }
}
```

18. Muestra cómo aplicaste herencia y polimorfismo en tu programa.

Herencia:

La clase ProductoAlimenticio hereda de Productos. Esto significa que no fue necesario volver a escribir los atributos ni los métodos de la clase padre, ya que los hereda automáticamente.

Polimorfismo:

El método comprar() se comporta de manera distinta dependiendo del tipo de objeto. En Productos muestra los datos generales, y en ProductoAlimenticio muestra también pero con las calorías.

```java
// Gestor de productos con herencia y polimorfismo

// Clase base (padre)
class Productos {
    // Atributos
    String nombre;
    String precio;
    String marca;
    String fechaEnQueSeHizo;
    String fechaEnQueSeVence;

    // Constructor
    public Productos(String nombre, String precio, String marca, String fechaEnQueSeHizo, String fechaEnQueSeVence) {
        this.nombre = nombre;
        this.precio = precio;
        this.marca = marca;
        this.fechaEnQueSeHizo = fechaEnQueSeHizo;
        this.fechaEnQueSeVence = fechaEnQueSeVence;
    }

    // Método comprar
    public void comprar() {
        System.out.println("\nQue buena elección de producto para comprar!!" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }

    // Método registrar
    public void registrar() {
        System.out.println("\nProducto registrado correctamente:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }

    // Método vender
    public void vender() {
        System.out.println("\nProducto vendido:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }
}

// Clase hija que hereda de Productos
class ProductoAlimenticio extends Productos {
    String calorias; // Atributo nuevo de la clase hija

    // Constructor con herencia
    public ProductoAlimenticio(String nombre, String precio, String marca,
                               String fechaEnQueSeHizo, String fechaEnQueSeVence, String calorias) {
        super(nombre, precio, marca, fechaEnQueSeHizo, fechaEnQueSeVence); // Llama al constructor del padre
        this.calorias = calorias;
    }

    // Polimorfismo: redefinimos el método comprar()
    @Override
    public void comprar() {
        System.out.println("\nHas comprado un producto alimenticio:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nCalorías: " + calorias +
                "\nFecha de fabricación: " + fechaEnQueSeHizo +
                "\nFecha de vencimiento: " + fechaEnQueSeVence);
    }
}

// Clase principal con método main
public class GestorProductos {
    public static void main(String[] args) {
        // --- Ejemplos de instancias ---
        Productos cliente1 = new Productos("Perfume", "45000", "Aromax", "10/09/2025", "10/09/2030");
        ProductoAlimenticio cliente2 = new ProductoAlimenticio("Yogurt", "2500", "Alpina", "20/10/2025", "27/10/2025", "120 kcal");
        Productos cliente3 = new Productos("Iphone", "14.500.000", "Apple", "10/08/2024", "10/08/2030");
        ProductoAlimenticio cliente4 = new ProductoAlimenticio("Leche", "15.000", "Alqueria", "10/09/2005", "10/06/2025", "120 kcal");

        // --- Ejemplo de polimorfismo ---
        // Cada cliente actúa de forma diferente dependiendo del tipo de producto
        cliente1.comprar();
        cliente2.comprar();
        cliente3.comprar();
        cliente4.comprar();
    }
}
```

19. Ejecuta tu código y muestra la salida.

Codigo:

```java
// Gestor de productos con herencia y polimorfismo

// Clase base (padre)
class Productos {
    // Atributos
    String nombre;
    String precio;
    String marca;
    String fechaEnQueSeHizo;
    String fechaEnQueSeVence;

    // Constructor
    public Productos(String nombre, String precio, String marca, String fechaEnQueSeHizo, String fechaEnQueSeVence) {
        this.nombre = nombre;
        this.precio = precio;
        this.marca = marca;
        this.fechaEnQueSeHizo = fechaEnQueSeHizo;
        this.fechaEnQueSeVence = fechaEnQueSeVence;
    }

    // Método comprar
    public void comprar() {
        System.out.println("\nQue buena elección de producto para comprar!!" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }

    // Método registrar
    public void registrar() {
        System.out.println("\nProducto registrado correctamente:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }

    // Método vender
    public void vender() {
        System.out.println("\nProducto vendido:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nFecha en que se hizo: " + fechaEnQueSeHizo +
                "\nFecha en que se vence: " + fechaEnQueSeVence);
    }
}

// Clase hija que hereda de Productos
class ProductoAlimenticio extends Productos {
    String calorias; // Atributo nuevo de la clase hija

    // Constructor con herencia
    public ProductoAlimenticio(String nombre, String precio, String marca,
                               String fechaEnQueSeHizo, String fechaEnQueSeVence, String calorias) {
        super(nombre, precio, marca, fechaEnQueSeHizo, fechaEnQueSeVence); // Llama al constructor del padre
        this.calorias = calorias;
    }

    // Polimorfismo: redefinimos el método comprar()
    @Override
    public void comprar() {
        System.out.println("\nHas comprado un producto alimenticio:" +
                "\nNombre: " + nombre +
                "\nPrecio: " + precio +
                "\nMarca: " + marca +
                "\nCalorías: " + calorias +
                "\nFecha de fabricación: " + fechaEnQueSeHizo +
                "\nFecha de vencimiento: " + fechaEnQueSeVence);
    }
}

// Clase principal con método main
public class GestorProductos {
    public static void main(String[] args) {
        // --- Ejemplos de instancias ---
        Productos cliente1 = new Productos("Perfume", "45000", "Aromax", "10/09/2025", "10/09/2030");
        ProductoAlimenticio cliente2 = new ProductoAlimenticio("Yogurt", "2500", "Alpina", "20/10/2025", "27/10/2025", "120 kcal");
        Productos cliente3 = new Productos("Iphone", "14.500.000", "Apple", "10/08/2024", "10/08/2030");
        ProductoAlimenticio cliente4 = new ProductoAlimenticio("Leche", "15.000", "Alqueria", "10/09/2005", "10/06/2025", "120 kcal");

        // --- Ejemplo de polimorfismo ---
        // Cada cliente actúa de forma diferente dependiendo del tipo de producto
        cliente1.comprar();
        cliente2.comprar();
        cliente3.comprar();
        cliente4.comprar();
    }
}
```

Salida:

```
Microsoft Windows [Versión 10.0.19045.6456]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\usuario\Downloads\Certificados,titulos\SENA TITULOS\TECNICO PROGRAMACION DE SOFTWARE> cmd /C ""C:\Program Files\Java\jre1.8.0_461\bin\java.exe" -cp "C:\Users\usuario\AppData\Roaming\Code\User\wor
kspaceStorage\c4bce0504ed0bfd3ce0587adaee527b7\redhat.java\jdt_ws\TECNICO PROGRAMACION DE SOFTWARE_3f6aeca4\bin" Main "

¡Qué buena elección de producto para comprar!
Nombre: Perfume
Precio: 45000
Marca: Aromax
Fecha en que se hizo: 10/09/2025
Fecha en que se vence: 10/09/2030

Has comprado un producto alimenticio:
Nombre: Yogurt
Precio: 2500
Marca: Alpina
Calorías: 120 kcal
Fecha de fabricación: 20/10/2025
Fecha de vencimiento: 27/10/2025

¡Qué buena elección de producto para comprar!
Nombre: Iphone
Precio: 14.500.000
Marca: Apple
Fecha en que se hizo: 10/08/2024
Fecha en que se vence: 10/08/2030

Has comprado un producto alimenticio:
Nombre: Leche
Precio: 15.000
Marca: Alqueria
Calorías: 120 kcal
Fecha de fabricación: 10/09/2005
Fecha de vencimiento: 10/06/2025

C:\Users\usuario\Downloads\Certificados,titulos\SENA TITULOS\TECNICO PROGRAMACION DE SOFTWARE>
```