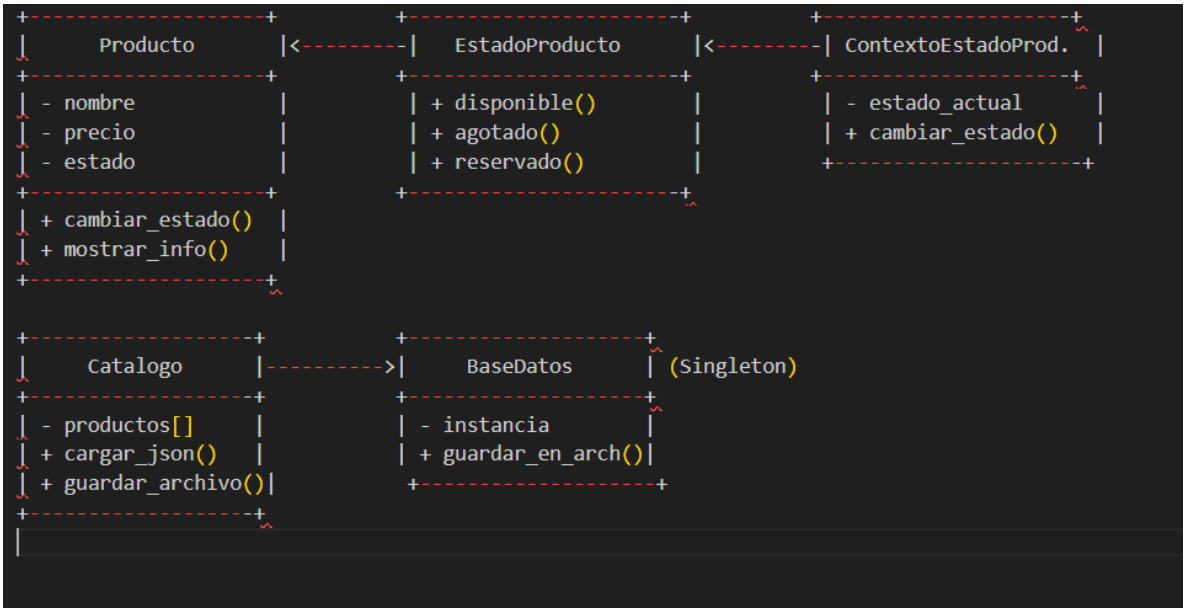
Melany Veronica Gomez Rodriguez
7690-23-1041
Proyecto 2
Ingenieria en sistemas de la informacion
Seccion A

DIAGRAMA DE CLASES (UML)



CLASE ESTADOPRODUCTO

```python
class EstadoProducto:
    def disponible(self):
        pass

    def agotado(self):
        pass

    def reservado(self):
        pass

class Producto:
    def __init__(self, nombre, precio, categoria):
        self.nombre = nombre
        self.precio = precio
        self.estado = None  # Se inicializa como None, cambiará con State
        self.categoria = categoria

    def cambiar_estado(self, nuevo_estado):
        self.estado = nuevo_estado

    def mostrar_informacion(self):
        print(f"Producto: {self.nombre}, Precio: {self.precio}, Estado: {self.estado}")
```

CLASE CONTEXTOESTADO

```python
class ContextoEstadoProducto:
    def __init__(self, estado_inicial):
        self.estado_actual = estado_inicial

    def cambiar_estado(self, nuevo_estado):
        self.estado_actual = nuevo_estado
```

CATALOGO

```python
import json

class Catalogo:
    def __init__(self):
        self.productos = []

    def cargar_desde_json(self, archivo_json):
        with open(archivo_json, 'r') as archivo:
            datos = json.load(archivo)
            for item in datos:
                producto = Producto(item['nombre'], item['precio'], item['categoria'])
                self.productos.append(producto)

    def guardar_en_archivo(self, archivo_salida):
        with open(archivo_salida, 'w') as archivo:
            productos_dict = [producto.__dict__ for producto in self.productos]
            json.dump(productos_dict, archivo, indent=4)
```

BASE DE DATOS

```python
class BaseDatos:
    _instancia = None

    def __new__(cls):
        if cls._instancia is None:
            cls._instancia = super(BaseDatos, cls).__new__(cls)
            cls._instancia.catalogo = Catalogo()
        return cls._instancia

    def guardar_en_archivo(self, archivo_salida):
        self.catalogo.guardar_en_archivo(archivo_salida)
```

Explicación de los Patrones Utilizados:

1. **Patrón State** : Para manejar los diferentes estados de los productos (disponible, agotado, reservado), se utiliza un contexto que permite cambiar entre diferentes estados de manera dinámica.
2. **Patrón Singleton** : La clase `BaseDatos`se asegura de que existe solo una instancia de la base de datos en todo el sistema, facilitando la gestión global del catálogo.

Próximos pasos:

- Poder ajustar las clases `EstadoProducto`para que manejen comportamientos específicos de cada estado (por ejemplo, no permitir compras si el producto está agotado).
- Agregar validaciones y mejoras según las necesidades de tu sistema.

Este código combina los patrones de diseño, la carga de datos desde un archivo JSON y el almacenamiento de información actualizado en archivos.