

Diagrama de Clases Refactorizado

### Refactorización del Código

La refactorización se puede hacer en varias etapas siguiendo los principios SOLID:

## Refactorización para SRP (Separar responsabilidades)

- Clase Producto: Solo contiene la información del producto y su lógica de negocio relacionada (como calcular el precio).
  - Clase ProductoEspecRefactorización del Código
- La refactorización se puede hacer en varias etapas siguiendo los principios SOLID:
  - Refactorización para SRP (Separar responsabilidades):
- Clase Producto: Solo contiene la información del producto y su lógica de negocio relacionada (como calcular el precio).
- Clase Producto Especial: Deriva de Producto, pero agrega comportamientos especiales como descuentos.
- Clase Persistencia Productos: Se encarga únicamente de manejar la persistencia de datos (guardar y cargar productos desde archivos).

- Clase Sistema Productos: Gestiona la lógica de interacción con el usuario y la manipulación de productos, delegando la persistencia a Persistencia Productos.
- al: Deriva de Producto, pero agrega comportamientos especiales como descuentos.
- Clase Persistencia Productos: Se encarga únicamente de manejar la persistencia de datos (guardar y cargar productos desde archivos).
- Clase Sistema Productos: Gestiona la lógica de interacción con el usuario y la manipulación de productos, delegando la persistencia a Persistencia Productos

## Refactorización para OCP (Abierto/Cerrado)

Abstracciones: La clase Producto y su subclase Producto Especial permiten agregar nuevos tipos de productos sin modificar el código existente. Si se desean agregar más tipos de productos, simplemente se crea una nueva clase que derive de Producto.

## Refactorización para LSP (Sustitución de Liskov)

Las clases derivadas como Producto Especial pueden sustituir a Producto sin romper el funcionamiento del sistema, ya que ambas clases comparten la misma interfaz (método calcular precio)

## Violaciones identificadas en el código

- SRP: La clase Sistema Productos tenía múltiples responsabilidades.
- OCP: El código no permitía extensiones sin modificar el código base.
- LSP: Las clases derivadas no podían sustituir a sus clases base sin causar problemas

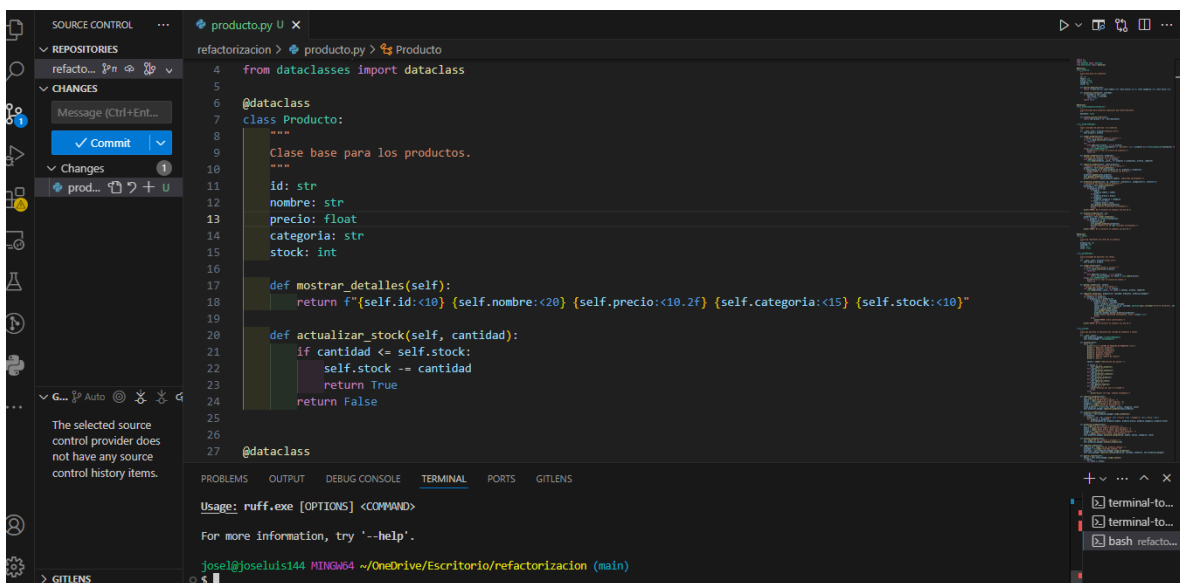
## Cambios realizados y justificación

- Se separaron las responsabilidades en diferentes clases (Producto, Producto Especial, Persistencia Productos, Sistema Productos).
- Se permitió la extensión del sistema sin modificar el código base utilizando la herencia de clases.

- Se aseguraron comportamientos consistentes entre clases base y derivadas para cumplir con LSP.

## Cómo se han aplicado los principios SOLID

- **SRP:** Cada clase tiene una única responsabilidad.
- **OCP:** Se pueden agregar nuevos tipos de productos sin modificar el código existente
- **LSP:** Las clases derivadas pueden sustituir a sus clases base sin problemas



The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with 'SOURCE CONTROL' and 'GITLENS' sections. The main editor area displays a Python file named 'producto.py'. The code defines a dataclass 'Producto' with attributes 'id', 'nombre', 'precio', 'categoria', and 'stock'. It includes two methods: 'mostrar\_detalle' and 'actualizar\_stock'. The terminal at the bottom shows the command 'ruff.exe' and the file path '~/OneDrive/Escritorio/refactorizacion (main)'.

```
4 from dataclasses import dataclass
5
6 @dataclass
7 class Producto:
8     """
9     Clase base para los productos.
10    """
11    id: str
12    nombre: str
13    precio: float
14    categoria: str
15    stock: int
16
17    def mostrar_detalle(self):
18        return f"{self.id:<10} {self.nombre:<20} {self.precio:<10.2f} {self.categoria:<15} {self.stock:<10}"
19
20    def actualizar_stock(self, cantidad):
21        if cantidad <= self.stock:
22            self.stock -= cantidad
23            return True
24        return False
25
26
27 @dataclass
```