Proyecto

Sistema de Gestión de Pedidos
Parte 1

Nombre: González Gómez, Juan Francisco

Fecha: 30/11/2025

Módulo: Entorno de desarrollo

Curso: 2025/2026

IES Mar de Cádiz - El Puerto de Santa María
Desarrollo de Aplicaciones Web

Índice

1. Introducción

Se ha encargado el desarrollo de un sistema de gestión de pedidos en el lenguaje de programación Java. El siguiente documento abarca las decisiones de diseño tomadas y expone pormenorizadamente su arquitectura por medio de definiciones y diagramas de clase. Los objetivos de este proyecto son la aplicación de los principios de la OOP y la obtención de su correcta representación por medio de diagramas UML.

2. Descripción del diseño

Se solicita que el sistema sea capaz de llevar a cabo las siguientes tareas:

1. Gestionar productos físicos y productos digitales. Ambos tipos de producto deben heredar de una clase padre común.
2. Registrar la información clientes con información básica como nombre, correo o dirección.
3. Gestionar pedidos, donde cada pedido está asociado a un cliente y puede contener uno o varios productos.
4. Calcular el importe total del pedido teniendo en cuenta factores como el coste de envío de los productos físicos.
5. Mostrar un resumen del pedido, incluyendo: datos del cliente, productos comprados e importe total.

Los requisitos mínimos de implementación son seguir la estructura de clases a continuación incluídas y representar tal, y como se piden, las relaciones entre las mismas.

Estructura de clases

- Clase Producto (abstracta o base)
    - Atributos comunes: nombre, precio.
    - Métodos: calcularPrecioFinal(), toString() o equivalente.
    - Subclases:
        - ProductoFisico: incluye atributo costeEnvio.
        - ProductoDigital: incluye atributo tamañoDescarga o licencia.
- Clase Cliente: con atributos básicos y métodos de acceso.
- Clase Pedido: que contenga una lista de productos y un cliente.
    - Métodos: agregarProducto(), calcularTotal(), mostrarResumen().
- Clase Main o App: para crear objetos y probar el sistema.

Relaciones a representar

- Herencia (Producto → ProductoFisico, ProductoDigital).
- Asociación (Pedido → Cliente).
- Agregación o composición (Pedido → Producto).

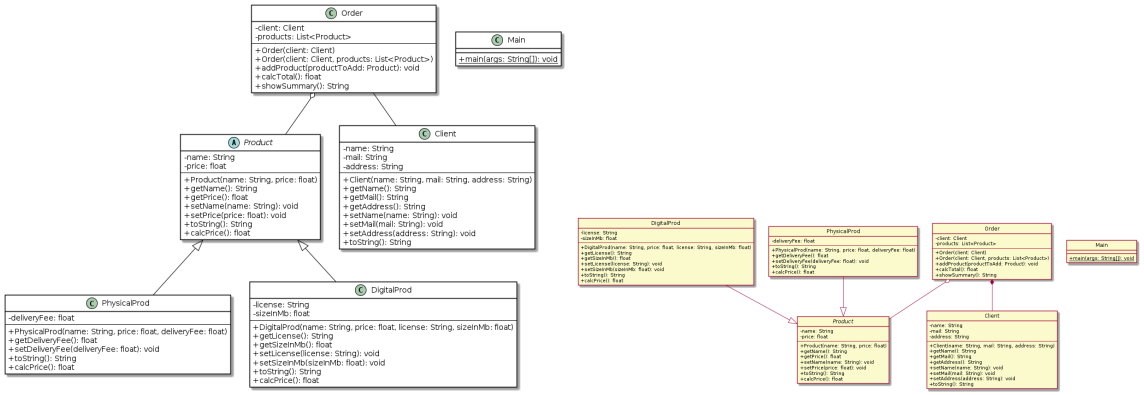
Se procede al diseño del diagrama según las instrucciones recibidas. Se toman las siguientes decisiones:

- El código será escrito en inglés.

- Siguiendo los estándares de la programación orientada a objetos, el ámbito de los atributos será privado mientras que el de los métodos será público.
- Teniendo en cuenta que la clase producto no debería poder ser instanciada, ya que los productos solo serán físicos o digitales, se decide definir la clase **Product** como abstracta.
- En todas las clases se implementará un método **.toString()** para mejorar la modularidad y la legibilidad del código.
- El método **.calcPrice()** de la clase **Product** será sobrescrito en las clases hijas, ya que el cálculo en cada caso es distinto.
- El método constructor de la clase **Order** será sobrecargado para permitir que los parámetros puedan ser un objeto de la clase **Client** o este mismo objeto y un objeto **List<Product>**. En caso de solo usar un objeto Client como parámetro, se inicializará un objeto **List<Product>** vacío.
- La relación entre las clases **Client** o **Product** y **Order** es de agregación, ya que la existencia de los dos primeros no debería depender de la existencia del último.
- La relación entre las clases **PhysicalProd** o **DigitalProd** y **Product** son de herencia, por solo se han definido en estas dos primeras clases los métodos y atributos no definidos en la clase padre o aquellos cuya funcionalidad difiera de esta. Esta relación permite tener un código más limpio y menos redundante, ya que se evita repetir líneas de código y la información común queda relegada al archivo de la clase padre.

3. Diagramas de clase

En este apartado se muestran tanto el diagrama UML diseñado previamente al desarrollo del software, como el generado por la herramienta CASE UML Generator for Java, que genera diagramas a partir de código.



Se puede encontrar una versión ampliada de estos diagramas en el primer anexo de este documento.

A la izquierda, el diagrama original. A la derecha, el diagrama obtenido tras aplicar ingeniería directa al primer diagrama, desarrollar el sistema de gestión y aplicar ingeniería inversa al resultado. Si bien las clases en realidad son las mismas, la relación entre **Client** y **Order** se interpreta como composición. Esto no es correcto, ya que **Order** solo usa una instancia **Client** como parámetro, lo que significa que este podría existir aún si la instancia de **Order** a la que pertenece dejase de hacerlo.

4. Descripción de la estructura del código

Cada clase se escribe en un archivo diferente que compartirá su nombre. Todos estos archivos se encuentran dentro de la misma carpeta. Todas las clases constan de los siguientes métodos:

- Constructor o constructores. Inicializa los atributos.
- *Getters* y *Setters*. Permiten acceso a los atributos, que son privados.
- *.toString*. Permite obtener información del objeto en forma de cadena.

La clase **Order** incluye código algo más complejo que el resto de clases, ya que trabaja con estructuras de datos. Los métodos .**calcTotal()** y .**showSummary()** iteran sobre esta estructura para poder calcular el precio total y mostrar los artículos en una lista itemizada respectivamente.

Además, el método **.calcTotal()** aprovecha el polimorfismo al llamar al método **.calcPrice**, aplicándose automáticamente la suma de los gastos de envío en los productos físicos sin necesidad de aplicar estructuras condicionales.

En la clase **Main** se han desarrollado una serie de pruebas para comprobar el correcto funcionamiento del sistema. Punto por punto se van probando los métodos de cada clase. El texto debe ser comprobado por el usuario, mientras que los cálculos se comprueban con una estructura condicional simple.

5. Conclusiones y posibles mejoras

Habiendo llevado a cabo el desarrollo del sistema de gestión tras haber diseñado el UML, es notable la facilidad con la que este tipo de herramienta puede ayudar a visualizar la manera en la que interactúan las clases.

Si bien el proyecto se adecua al encargo recibido, podría ser interesante mejorarlo con la implementación de alguna de las siguientes ideas:

- Integrar una clase por medio de la cual el cliente pudiese guardar un pedido. Esto implicaría que los datos deberían almacenarse en un archivo o base de datos, obteniendo así persistencia de datos entre sesiones. Para esto también resultaría interesante incluir un id para el pedido.
- Escribir un método en la clase **Order** que permitiese eliminar objetos del pedido en base a **.name** o un atributo nuevo que podría ser **.id**.
- También sería una mejora importante el poder comprobar la cantidad que existe de cada producto en el pedido para poder representarlos agrupados en lugar de individualmente.

Anexo I - Diagramas de clase

Diagrama diseñado

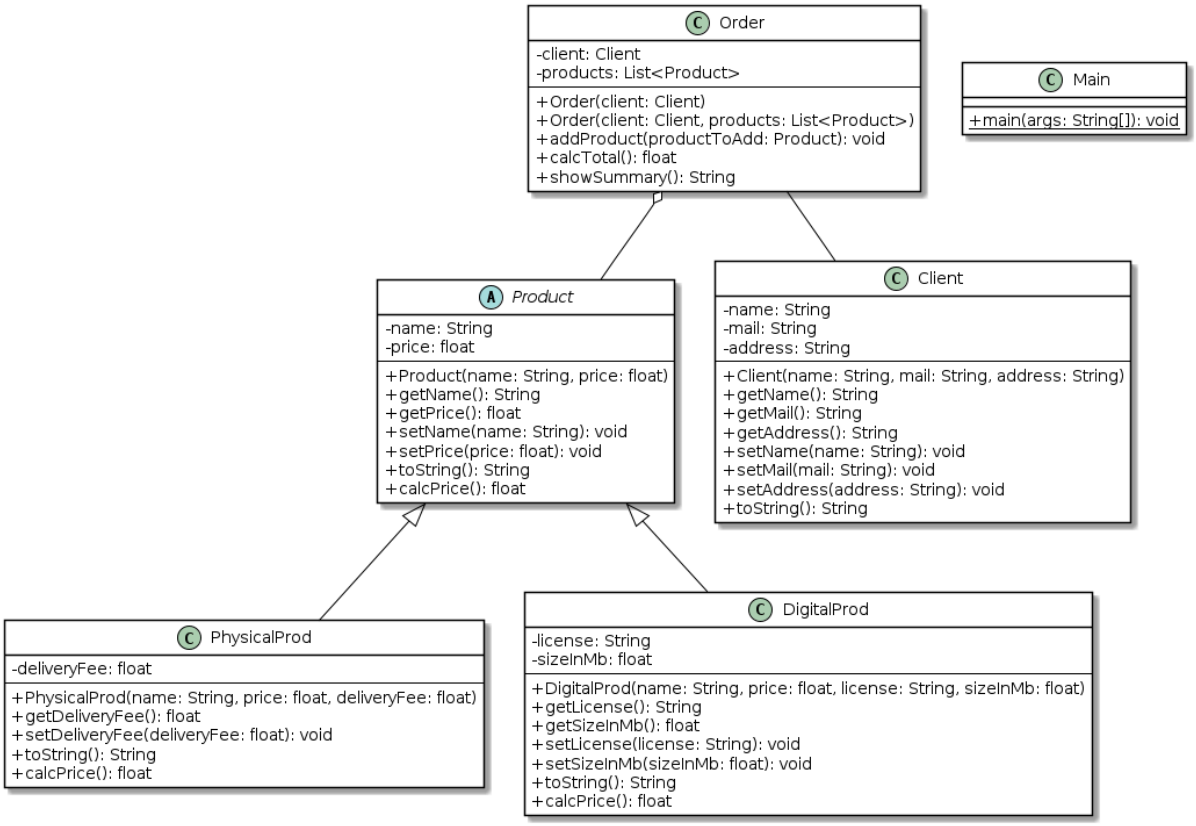


Diagrama obtenido