Cooperativa Empresarial

Programación Orientada a Objetos

Práctica 2022-2023

Memoria

Juan R. Barranco Esteban

Índice

1. Objetivo

El objeto de este proyecto es el diseño y desarrollo de una aplicación en Java para la gestión de una cooperativa agrícola. Su ámbito y alcance es el de un proyecto académico, como complemento práctico a la asignatura de Programación Orientada a Objetos, del 1er curso del Grado de Ingeniería en Tecnologías de la Información de la UNED.

2. Especificaciones

El dominio de esta aplicación es, como se indica, la gestión empresarial de una cooperativa agraria, destacando principalmente las siguientes áreas funcionales:

- Productores y productos
- Clientes
- Logística
- Creación de pedidos y mantenimiento del registro de éstos

Se resumen a continuación los aspectos fundamentales del enunciado de la práctica, que describe en detalle todos los requerimientos que debe cumplir el sistema.

Productores

Se llama productores a los actores que proporcionan productos a la cooperativa para su venta y que cobran el precio estipulado por ellos cuando éstos se venden. Hay tres tipos fundamentales de productores:

- Pequeños productores: aquellos que tienen una extensión de cultivos inferior a un límite establecido y producen menos de 5 productos diferentes. El límite de superficie se define anualmente y tiene aplicación durante todo el año fiscal. El valor para el año en curso es de 5 Ha.
- Grandes productores: aquellos cuya suma de extensiones de cultivo supera el límite mencionado o producen más de 5 productos diferentes.
- Productores federados: son uniones de productores en torno a la explotación de un producto determinado. Cada productor individual aporta al federado la extensión de cultivo que tiene dedicada al producto en cuestión. Sólo puede haber un productor federado por cada producto. Sólo se pueden federar pequeños productores, y siempre y cuando la suma de extensiones de explotación del producto federado no supere el límite establecido para pequeños productores.

Productos

Un producto tiene un rendimiento por hectárea definido, que se usa como referencia para calcular la producción a partir de la superficie de cultivo, y un precio de referencia, que se actualiza semanalmente, pudiendo experimentar tanto subidas como bajadas.

Logística

En cuanto a la logística se considera como tal a aquellas empresas que proveen servicios de transporte. Hay dos tipos de logística según se detalla a continuación, y cada empresa proporciona servicios de uno u otro tipo:

- Gran logística: cubre las distancias grandes, desde la cooperativa hasta la capital de la provincia de destino.
- Pequeña logística: cubre las distancias desde la capital de provincia hasta el destino final.

Del mismo modo, los costes de logística dependen de si los productos transportados son perecederos o no, siendo más caro el transporte de perecederos. La forma en que se realiza este cálculo se detalla en un apartado posterior.

Clientes

En cuanto a los clientes, los hay de dos tipos:

- Distribuidores: compran al por mayor, teniendo un mínimo de compra en cada pedido de 1000 kg. El beneficio para la cooperativa de las ventas a estos clientes es del 5% sobre el precio del producto, y no están obligados a pagar IVA.
- Consumidores finales: compran al por menor, teniendo un límite máximo de compra en cada pedido de 100 kg. El beneficio para la cooperativa en este caso es del 15% sobre el precio del producto, y pagan un 10% de IVA (sobre el precio total, incluido transporte).

2.0.1. Interpretaciones y resoluciones

Se explican a continuación, agrupadas por área funcional, una serie de interpretaciones y resoluciones que se han tomado respecto a aquellos aspectos del enunciado que no quedaban suficientemente claros o no están suficientemente definidos.

Consideraciones generales

- Para los temas derivados de temporalidad y fechas, se ha considerado para la carga de datos de ejemplo y prueba el contexto de un único año fiscal completo (2023).
- Es decir, por ejemplo, para el umbral de diferenciación entre pequeños y grandes productores se ha establecido un único valor fijo, correspondiente a dicho año fiscal, y para la evolución semanal de precios de productos se han predefinido los correspondientes a cada una de las semanas de un año completo.
- En relación con el punto anterior, las fechas de creación de elementos (pedidos, etc.) se introducirán manualmente dentro de los márgenes de este plazo de un año fiscal, al tratarse de una simulación en la que no se toma en consideración la fecha actual en que se está ejecutando el sistema.

Productor y producto

- Los distintos productos que gestiona la cooperativa son predefinidos. No se puede registrar un productor que trabaja un producto distinto a los registrados.
- La evolución de precios se maneja de forma individualizada para cada producto, mediante una lista de precios indexada por semana.
- Se interpreta la limitación de 5 cultivos a pequeños productores como un umbral más a partir del que serán considerados grandes productores. Un productor es considerado gran productor si pasa uno de los dos umbrales, el de superfice o el de número de cultivos (no necesariamente ambos a la vez).
- Se interpreta que un cultivo produce una cantidad de producto anual igual a su extensión multiplicada por el rendimiento por Ha. del producto. El stock inicial de un producto se establece como la suma de las producciones de todos los cultivos de este producto.

Logística

- Se entiende como logística lo que comúnmente llamamos transportista, es decir, aquella empresa que se dedica al transporte de mercancías. En el sistema los hay de dos tipos, los que hacen envíos a larga distancia (gran logística), y los que lo hacen a pequeña distancia (pequeña logística).
- Para un pedido puede haber una o dos lineas de envío, dependiendo de si se necesita gran logística, pequeña logística, o una combinación de ambas.
- Todos los envíos tienen como origen las instalaciones centrales de la cooperativa, que a efectos de tablas de distancias hemos localizado arbitrariamente en Albacete. No se tendrán en cuenta en el sistema las localizaciones de los productores ni la posibilidad de envíos desde éstas sin pasar por la central de la cooperativa.
- El cálculo de las líneas de envío necesarias para un pedido y sus kilometrajes se basa en las siguientes reglas:
    - El envío en gran logística se determina a partir de la provincia de la dirección de envío, leida desde los dos primeros dígitos del código postal, y según tablas de distancias guardadas en los sets de datos de prueba. Sólo se hacen envíos al territorio español peninsular, por lo que no se aceptarán pedidos cuyo destino esté fuera de este territorio.
    - Posteriormente se calcula la distancia desde la capital de la provincia de destino hasta la localización exacta dividiendo entre 10 el valor numérico de los 3 últimos dígitos del código postal. (Esta lógica claramente no se corresponde con la realidad, pero es una forma de que este dato sea un valor calculado a partir de la dirección de envío, simplificándolo de acuerdo con el alcance y ámbito de este proyecto).
    - Si este resultado es superior a 5km o se ha calculado que no es necesaria una linea de gran logística, se añade una linea de pequeña logística con esta distancia. En cualquier otro caso, se suma esta distancia a la linea de gran logística creada previamente.
- No se establecen diferencias, en cuanto al cálculo de lineas de envío y kilometrajes, para envíos de productos perecederos y no perecederos. Las diferencias entre envíos de tipos

de productos radicarán únicamente en el cálculo de sus costes, teniendo precios por km diferentes para cada uno de los dos tipos.

- Tras análisis del procedimiento definido en el enunciado para el cálculo de precios de logística, se infieren las siguientes reglas equivalentes simplificadas:
  - Precio en pequeña logística: precio por km, según tarifa establecida.
  - Precio en gran logística:
    - El precio en gran logística tiene dos componentes:
      - Precio por km, según tarifa establecida.
      - Precio por km y kg: 0,01 * precio de referencia del producto. El resultado de este cálculo es equivalente al precio por tramos detallado en el enunciado.
    - El precio final es la suma de ambos componentes, multiplicado cada uno de ellos por los kg de mercancía y/o km de distancia, según corresponda.
- Para los precios por km se establecerá una única tarifa para perecederos y una única para no perecederos, a aplicar tanto en pequeña como gran logística por todos los transportistas como precio base.*
- Cada transportista al generar una oferta, y para que haya variedad entre ellas y competencia, puede aplicar incrementos o descuentos sobre este cálculo base, según los criterios que estime oportunos. En nuestra simulación generaremos estas variaciones de forma aleatoria, dentro de unos márgenes acotados entre el -15% y el +15% sobre el precio base.
- *Nota: durante el desarrollo se observó que aplicando de forma directa estos algoritmos se daban situaciones muy dispares, como precios muy elevados en gran logística (del orden de 10 veces el precio del producto) y muy bajos en pequeña (del orden de la centésima parte), de modo que se han añadido en diferentes partes de los cálculos una serie de coeficientes adicionales, ajustados por ensayo y error hasta que se observaron empíricamente resultados más razonables.

Pedido

- Un pedido sólo puede contener un único producto, para evitar la problemática que generaría con los envíos la mezcla de productos perecederos y no perecederos.
- Un pedido puede tener una o varias líneas de envío, según si se necesita pequeña logística, gran logística o una combinación de ambas para poder hacer llegar el envío a destino.
- Se adopta, respecto al requisito del enunciado referente al plazo de entrega de 10 días y el recálculo de precios 10 días antes de la entrega si el cliente solicitara un plazo mayor, la siguiente estrategia: siempre se pedirá por defecto al cliente, durante la creación del pedido, la fecha en que desea que se realice la entrega, y se establecerá como fecha de formalización del pedido y cálculo de precios la fecha del décimo día anterior a la de entrega.

2.1. Especificaciones funcionales

Casos de uso a desarrollar en la aplicación:

- Productores:
  - Registrar productor
  - Listar productores
- Logística
  - Registrar empresa de logística
  - Listar empresas de logística
- Clientes
  - Registrar cliente
  - Listar clientes
- Pedidos
  - Registrar pedido
- Carga de datos de prueba
  - Productores
  - Empresas de logística
  - Clientes
- Generación de informes
  - Ventas totales por producto
  - Beneficios de cada productor por producto
  - Beneficios de cada empresa de logística
  - Beneficios de la cooperativa por producto
  - Evolución semanal de precios por producto

2.2. Especificaciones no funcionales

El proyecto debe cumplir las siguientes especificaciones no funcionales:

- Lenguaje de programación: Java
- IDE a usar en el desarrollo: BlueJ

3. Fases

Se piden tres fases de la entrega, acumulativas (cada una parte de la anterior) pero con entregables independientes, que se interpretan de las siguiente manera:

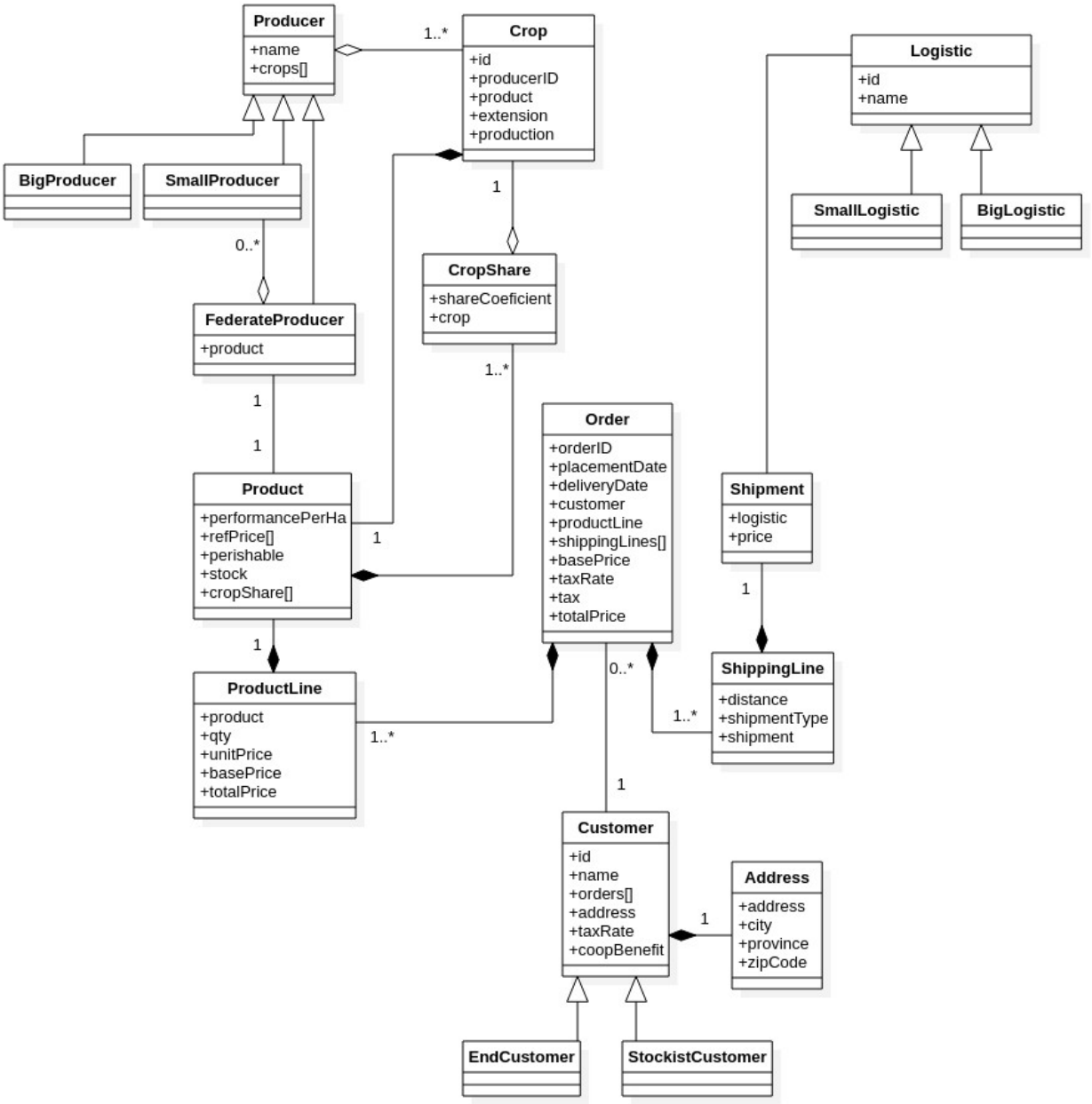
1. Modelo del dominio y diseño de clases.

    - Modelado de los objetos de la realidad que forman el sistema, así como las relaciones entre ellos. Se consideran por tanto sólo las clases de estos conceptos de la realidad, y no así cualesquiera otras clases auxiliares que puedan ser necesarias en el desarrollo de la aplicación a nivel de software (vistas, controladores, clases necesarias para la aplicación de determinados patrones, etc.).
    - El desarrollo de esta fase se presenta en el apartado 3.1. de la presente memoria.
2. Desarrollo del sistema.
    - Diseño de las clases software necesarias para el funcionamiento completo de la aplicación, incluyendo tanto las del modelo como, ahora sí, cualesquiera otras que puedan ser necesarias. Desarrollo a código de la aplicación.
    - El desarrollo de esta fase se presenta en el apartado 3.2. de la presente memoria, junto con el código del archivo adjunto L2_cooperativa_base.zip
3. Funcionalidad de generación de informes.
    - Desarrollo, una vez hecha la aplicación, de una funcionalidad para generar informes que cubra como mínimo los listados en la sección “Generación de informes” de los casos de uso enumerados en el apartado 2.1. de especificaciones funcionales.
    - El desarrollo de esta fase se presenta en el apartado 3.3. de la presente memoria, junto con el código del archivo adjunto L3_cooperativa_informes.zip

3.1. Modelo del dominio y diseño de clases

Se ha modelado el dominio del problema de tal forma que podemos agrupar las clases definidas según las siguientes áreas:

- Productor y producto
- Logística
- Cliente
- Pedido



En las siguientes subsecciones se listan y detallan las clases incluidas.

3.1.1. Productor y producto

- Productor (Producer). Clase principal del grupo. Representa al productor. Tiene los siguientes subtipos:
  - Gran productor (BigProducer)
  - Pequeño productor (SmallProducer)
  - Productor federado (FederateProducer). Aparte de los atributos heredados de productor, tiene un atributo producto que define el producto sobre el que versa la federación, y además agrega objetos pequeño productor.

- Producto (Product). Representa al producto como definición general, con sus cualidades tales como rendimiento, precio de referencia, si es perecedero, stock, y tabla de participaciones de los distintos cultivos en que se produce (CropShare).
- Cultivo (Crop). Define una explotación de un producto perteneciente a un productor. Tiene una superficie y una producción que es resultado de la superficie y el rendimiento del producto explotado.
- Participación de cultivo (CropShare). Se asocia a un producto y a un cultivo de ese producto, definiendo la participación (porcentaje, coeficiente) de dicho cultivo en la producción total del producto, que será necesaria para el cálculo de beneficios de cada productor.

3.1.2. Logística

- Logística (Logistic). Representa a una empresa de logística. Tiene dos subtipos:
  - Gran logística (BigLogistic)
  - Pequeña logística (SmallLogistic)
- Envío (Shipment). Representa un envío de la cantidad de producto de un pedido. Se asocia con la logística que lo sirve y con la linea de envío del pedido en cuestión.

3.1.3. Cliente

- Cliente (Customer). Representa al cliente y guarda su información, dirección, lista de pedidos, así como el tipo de IVA y el porcentaje de beneficio aplicable según subtipo de cliente. Tiene dos subtipos:
  - Cliente final (EndCustomer)
  - Distribuidor (StockistCustomer)
- Dirección (Address). Guarda los datos de la dirección del cliente.

3.1.4. Pedido

- Pedido (Order). Guarda los datos de un pedido: id, fecha de realización del pedido, fecha de entrega, cliente, linea de producto, lineas de envío, precio base, tipo de IVA, IVA y precio total.
- Linea de producto (ProductLine). Asocia un pedido con el producto vendido. Un pedido tiene una linea de producto. Guarda el producto, cantidad, precio unitario, precio base, que corresponde al precio de referencia del producto por la cantidad, y el precio total, correspondiente al precio base más el beneficio de la cooperativa.
- Líneas de envío (ShippingLIne). Asocia un pedido con sus envíos. Un pedido puede tener una o dos líneas de envío dependiendo de si necesita envío en gran logística, envío en pequeña logística o ambos. Cada linea de envio guarda el envío, el tipo de éste, y la distancia.

3.2. Desarrollo del sistema base

Se adjunta el proyecto BlueJ, con el código de todas las clases desarrolladas en el archivo L2_cooperativa_base.zip. En los siguientes apartados se da una descripción funcional de la aplicación, así como de las clases desarrolladas para su implementación, tanto del modelo descrito anteriormente como complementarias.

3.2.1. Descripción funcional

Desde el punto de vista de la usabilidad, la aplicación se ha desarrollado como un programa de consola (si bien la arquitectura realizada permite desacoplar la vista y adaptarla a otro tipo de aplicación reusando el mismo núcleo).

La interfaz está estructurada en una serie de menús que se corresponden con las áreas funcionales del sistema: productores, logística, clientes y pedidos (y adicionalmente informes, una vez implementada la funcionalidad del apartado 3.3.).

Cada uno de estos menús contiene una serie de elementos los cuales se corresponden con cada uno de los casos de uso. Estos elementos se ejecutan introduciendo por teclado el índice numérico asociado a cada opción.

La estructura global de menús y acciones ejecutables es la siguiente:

- Menú principal
    - Menú productores
        - Vista: listado de productores (se muestra directamente en la vista de menú, junto al listado de acciones ejecutables)
        - Registrar productor
        - Volver atrás (menú superior)
    - Menú logística
        - Vista: listado de empresas de logística
        - Registrar empresa de logística
        - Atrás (menú superior)
    - Menú clientes
        - Vista: listado de clientes
        - Registrar cliente
        - Crear pedido para cliente
        - Atrás (menú superior)
    - Menú pedidos
        - Crear pedido (redirige a menú clientes, ya que necesitamos poder ver el listado de éstos para referenciar el cliente para el que se hace el pedido)
        - Atrás (menú superior)
    - Salir de la aplicación

3.2.2. Casos de uso

A continuación se describen los casos de uso desarrollados, y los flujos que siguen.

- Listado de productores: imprime por pantalla un listado de todos lo productores registrados en el sistema, incluyendo ID, tipo, nombre y cultivos
- Registrar productor: registra un productor en el sistema.
    - Datos de entrada: ID, nombre, productos y extension de cultivo de cada uno de ellos
    - Validaciones: los productos introducidos existen en el sistema
    - Comprueba el número de productos distintos introducidos y la suma total de las extensiones introducidas y determina el subtipo de productor a crear
    - Crea objetos Producer (del subtipo que corresponda) y Crop, los asocia y los guarda en la colección de productores.
- Listado de empresas de logística: imprime por pantalla un listado de empresas de logística, incluyendo ID, tipo, nombre
- Registrar empresa de logística: registra una empresa de logística en el sistema.
    - Datos de entrada: ID, nombre, tipo
    - Validaciones: el tipo introducido existe
    - Crea objeto Logistic (del subtipo que corresponda) y lo guarda en la colección de logística.
- Listado de clientes: imprime por pantalla un listado de clientes, incluyendo ID, tipo, nombre, ciudad y provincia.
- Registrar cliente: registra un cliente en el sistema.
    - Datos de entrada: ID, nombre, tipo, dirección, ciudad, provincia, código postal
    - Validaciones: el tipo introducido existe, el código postal introducido pertenece al territorio penisular de España.
    - Si se detecta un código de postal no válido, aparte de pedir corregirlo da la opción de cancelar el registro.
    - Crea objetos Customer (del subtipo que corresponda) y Address, los asocia convenientemente y los guarda en la colección de clientes.
- Crear pedido para cliente: crea un pedido para el cliente designado.
    - La creación del pedido se hace en tres fases:
        - En primer lugar se piden al cliente los datos básicos y se crea el pedido con ellos, sin confirmar (estado created).
        - Posteriormente se generan ofertas de logística y se le dan a elegir al cliente. Se guarda/n en el pedido la/s opción/es de envío elegida/s, se calculan y actualizan los totales.
        - Se pide confirmación al cliente.

        - Si éste confirma, se confirma el pedido (estado confirmed), se guarda en la colección de pedidos, y se registra en los historiales correspondientes (historial de pedidos del cliente, historial de pedidos del producto, historial/es de envíos de la/s empresa/s de logística).
        - En caso de que el cliente no confirme, el pedido se deja sin pasar a estado confirmado y no se asocia a las colecciones e historiales, de forma que, aunque el objeto se haya creado, no consta como registrado en el sistema ni computa en posteriores cálculos. Asumimos que, al quedar sin ningún puntero que lo referencie, el garbage collector se encargará de destruirlo.
    - Datos de entrada: ID de cliente, fecha de entrega, producto, cantidad, envíos (elección entre ofertas generadas).
    - Validaciones: el cliente existe, la fecha es válida, el producto existe, la cantidad de producto está disponible, la empresa de logística elegida (ID introducido) existe.
- Crear pedido (menú pedidos): redirige a menú clientes para creación de pedido desde éste.

3.2.3. Clases del sistema

Además de las clases explicadas en el modelo del dominio, se han desarrollado un gran número de otras clases atendiendo a diversas necesidades técnicas. Aunque las decisiones de diseño que han llevado la arquitectura de la aplicación que se ha definido se explicarán en el apartado 4., se da a continuación un esbozo de esta arquitectura, el listado de paquetes en que se ha estructurado, y las clases contenidas por cada uno de ellos, junto con una descripción de la finalidad y funcionamiento de estas.

Modelo-Vista-Controlador

Tal como se desarrolla en el apartado 4.1., la aplicación está estructurada según una arquitectura modelo-vista-controlador. La parte de la vista además utiliza un sistema de menús con elementos asociados a comandos (patrón de diseño Command).

Hay por tanto, aparte de los modelos ya descritos, las siguientes categorías de clases de objetos en el sistema:

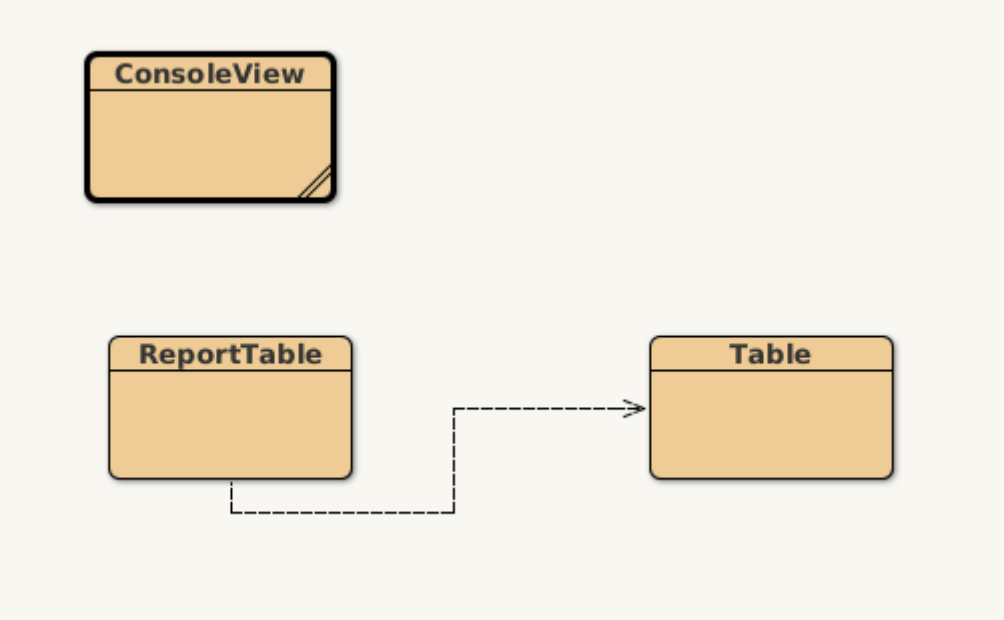
- Clase que contiene el main (Cooperativa)
- En la capa de vista:
    - Clase principal de la vista (ConsoleView)
    - Clases de menús y elementos de menú
    - Comandos asociados a los elementos de menú
    - Clases auxiliares de la vista
- En capa controlador:
    - Un controlador por cada área funcional (productor, producto, logística, cliente, pedido)
- En capa modelo:

    - Por cada área funcional:
        - El tipo padre y los subtipos del modelo principal (p. ej. Logistic, BigLogistic, SmallLogistic)
        - Colección de objetos del modelo principal del área funcional (siguiendo el ejemplo anterior, LogisticCollection)
        - Modelos auxiliares (Shipment, OrderShipment, etc.)

Paquetes

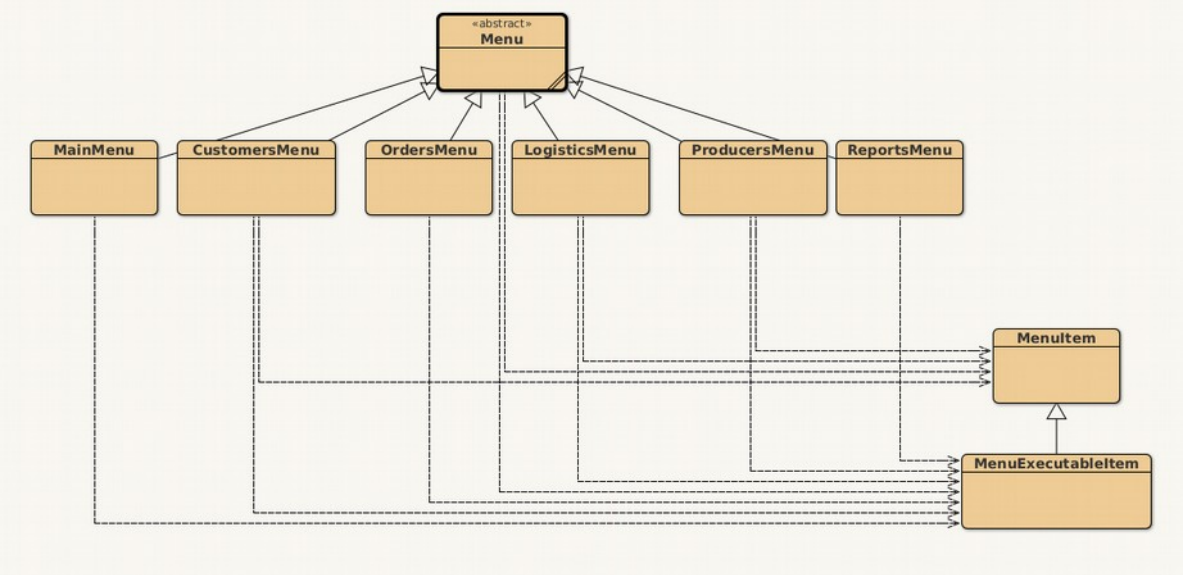
Los paquetes de la aplicación se han estructurado principalmente según capas de MVC, y en alguno de los niveles adicionalmente por funcionalidad. Se listan a continuación todos los paquetes existentes en la aplicación, junto con las clases que contienen y una breve descripción de cada una de ellas, así como el diagrama de clases a nivel de paquete generado por BlueJ.

- (no contenido en ningún paquete)
    - Cooperativa. Clase principal del sistema, contiene el método main()
- **view**
    - ConsoleView. Clase principal de la vista
    - Table. Clase auxiliar para la generación de tablas
    - ReportTable. Contenedor de tabla para informe

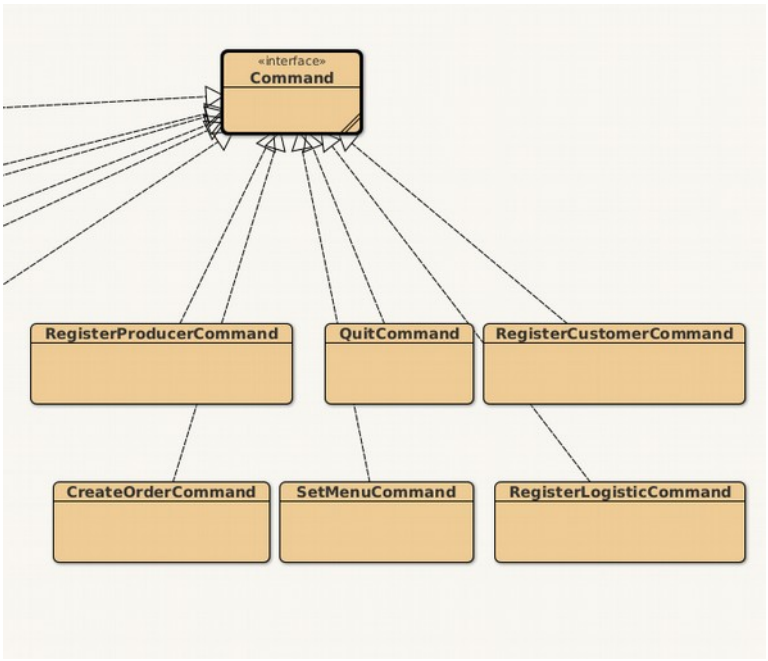


- **view.menu**
    - Menu. Abstracta. Clase padre de todas las clases menú.
    - MainMenu. Clase del menú principal
    - CustomersMenu. Menú de clientes

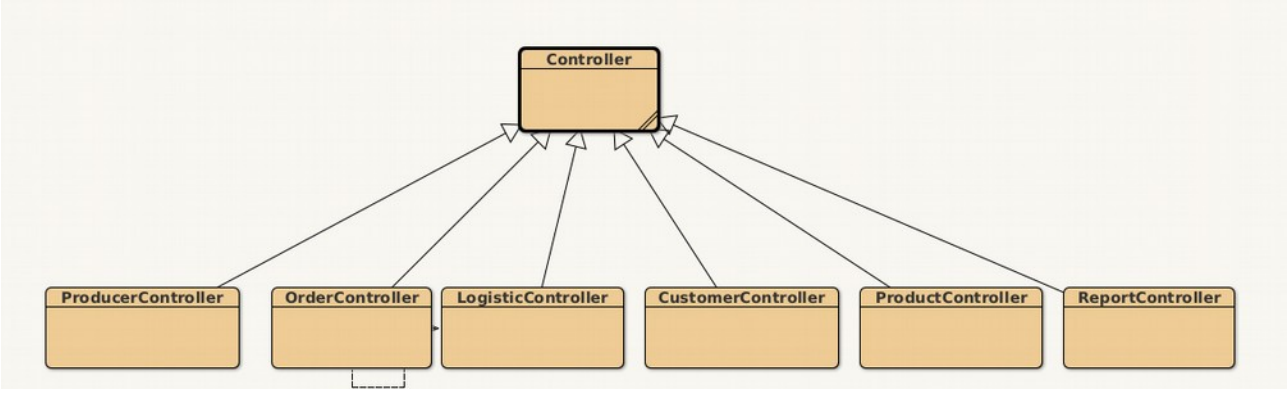
- OrdersMenu. Menú de pedidos
- LogisticsMenu. Menú de logística
- ProducersMenu. Menú de productores



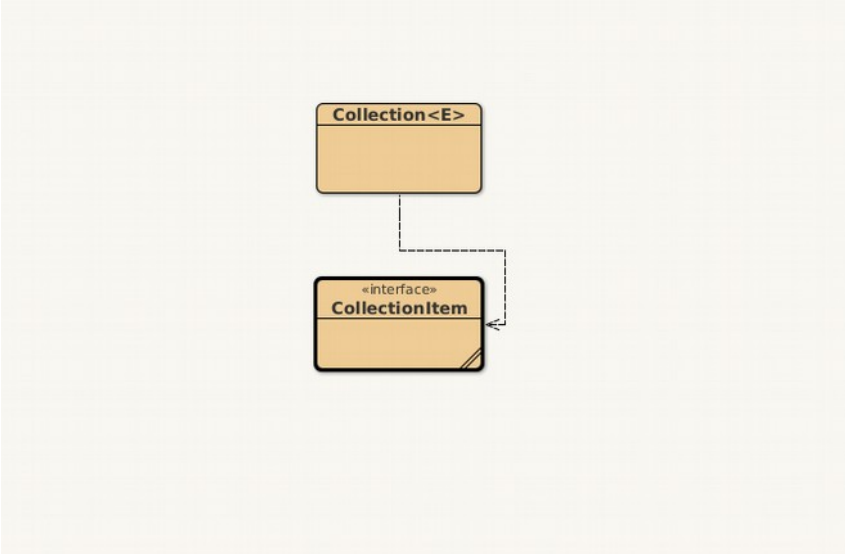
- **view.command**
  - Command. Interfaz, implementada por todas las clases Command.
  - RegisterProducerCommand. Ejecuta el registro de productor.
  - QuitCommand. Envía a la vista un mensaje para cerrar la aplicación.
  - RegisterCustomerCommand. Registra un cliente
  - CreateOrderCommand. Crea un pedido
  - SetMenuCommand. Envía a la vista un mensaje para establecer el menú actual.
  - RegisterLogisticCommand. Registra una empresa de logística.



- **controller**
  - ProducerController. Controlador de productores.
  - OrderController. Controlador de pedidos.
  - LogisticController. Controlador de logística.
  - CustomerController. Controllador de clientes.
  - ProductController. Controlador de productos.

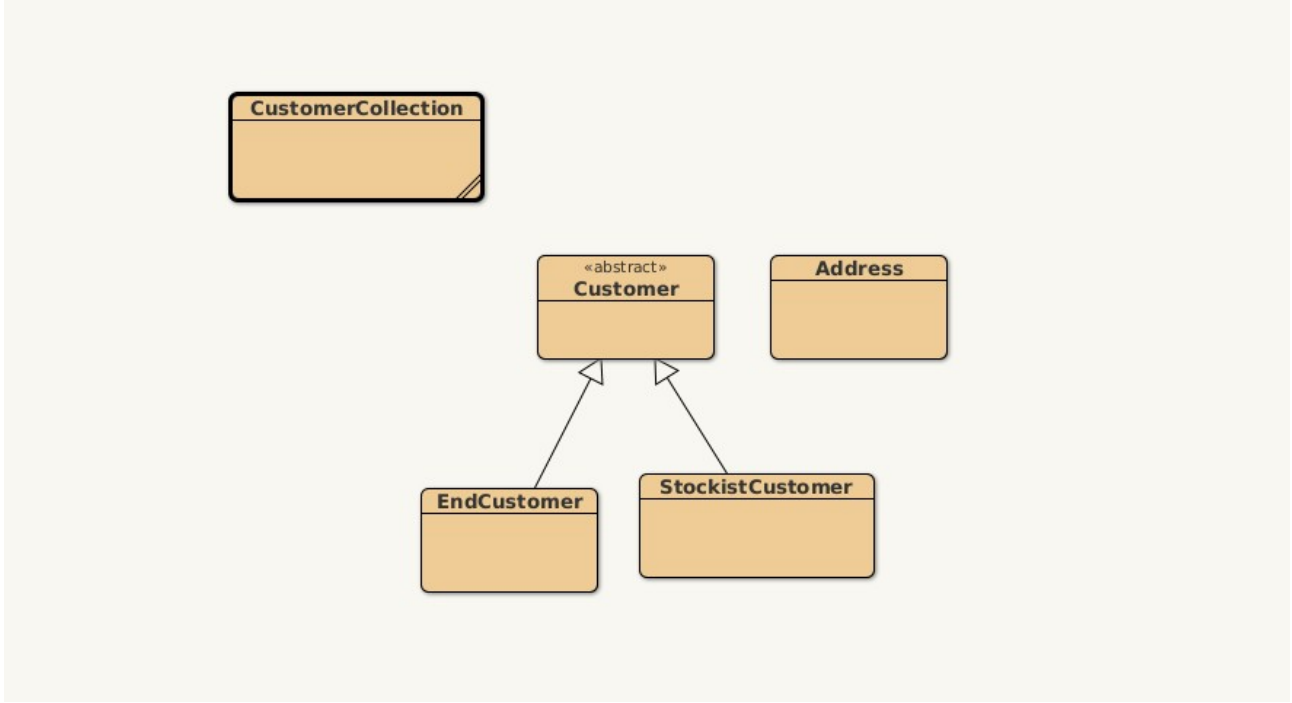


- **model**
  - Collection<E>. Clase padre de las colecciones. Parámetrizada con el tipo de los elementos contenidos en la colección.
  - CollectionItem. Interfaz a implementar por los elementos contenidos en la colección.

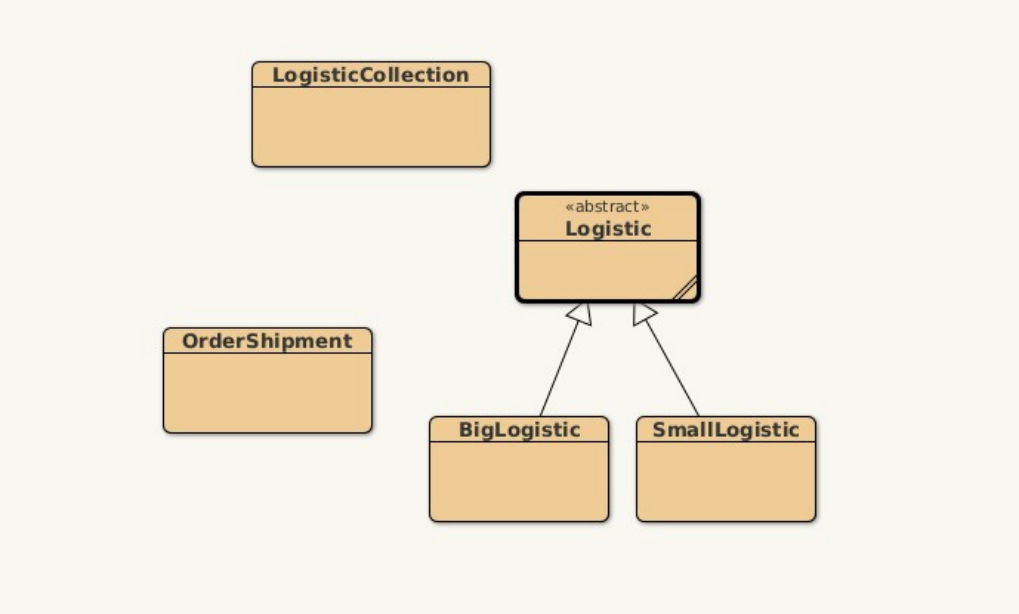


- **model.customer**
  - CustomerCollection. Colección de clientes
  - Customer. Abstracta. Clase padre cliente

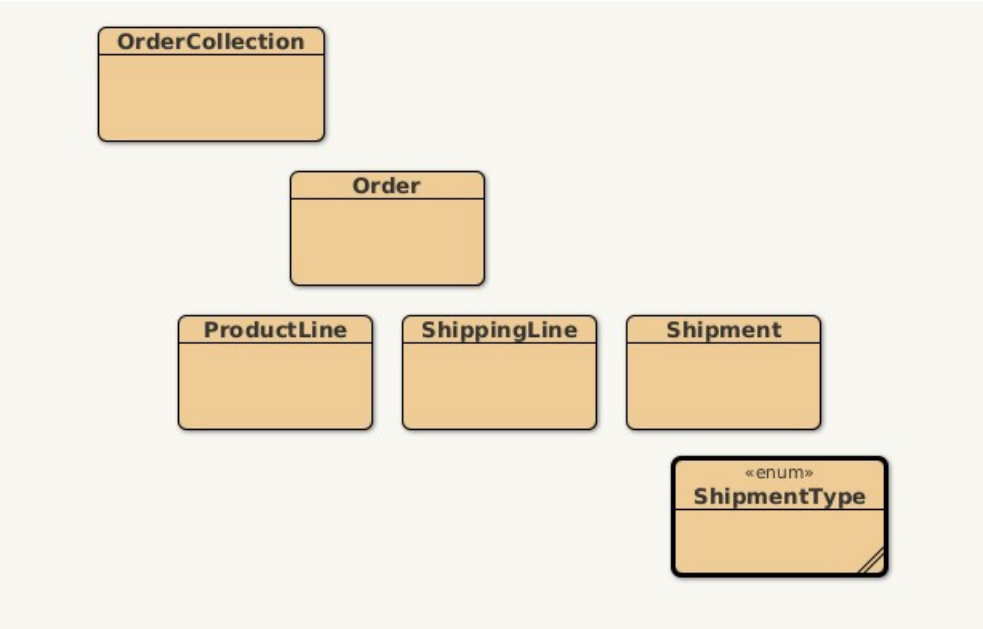
- EndCustomer. Cliente final
- StockistCustomer. Distribuidor.
- Address. Dirección



- **model.logistic**
  - LogisticCollection. Colección de empresas de logística.
  - Logistic. Abstracta. Clase padre empresa de logística
  - BigLogistic. Gran logística.
  - SmallLogistic. Pequeña logística.
  - OrderShipment. Asociación entre pedido y envío, para guardar en el historial de envíos.

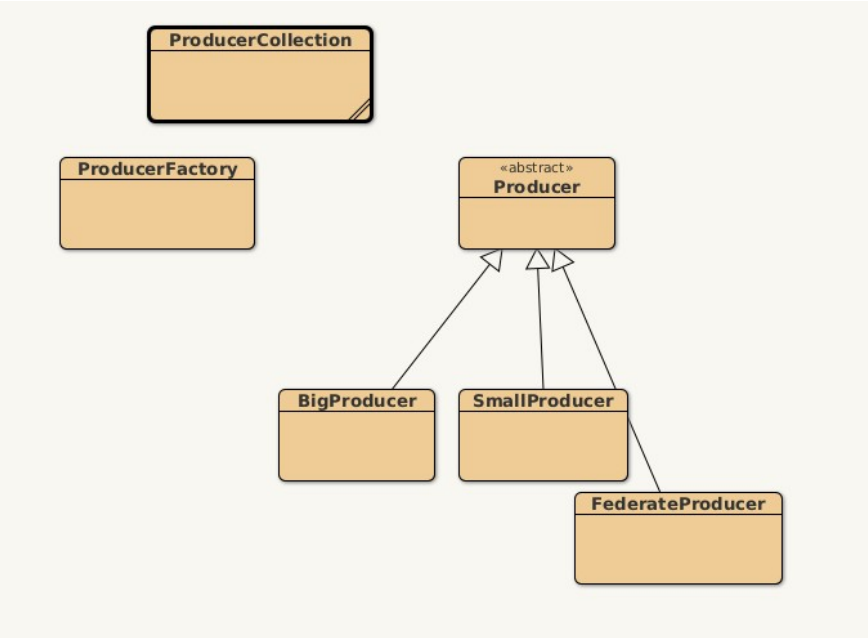


- **model.order**
  - OrderCollection. Colección de pedidos
  - Order. Clase pedido
  - ProductLine. Linea de producto
  - ShippingLine. Linea de envío
  - Shipment. Envío
  - ShipmentType. Enum, utilizado en Shipment para hacer referencia al tipo de envío que es (no se consideró necesaria herencia de subtipos para este objeto)

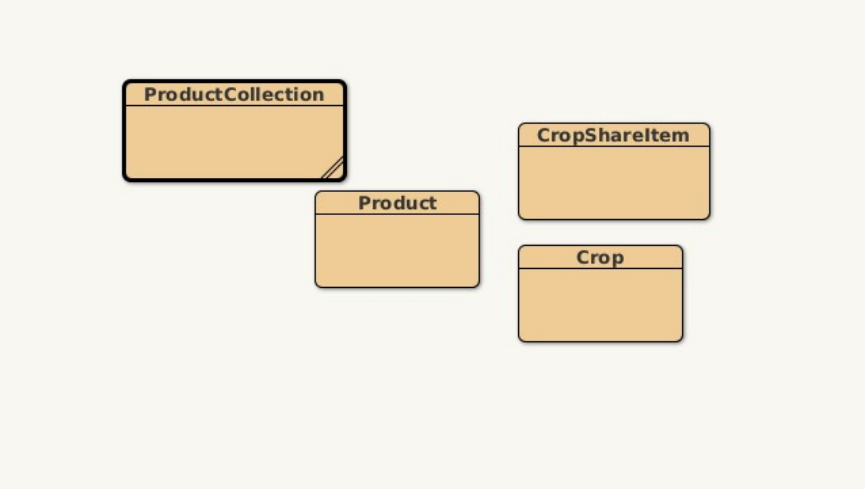


- **model.producer**

- ProducerCollection. Colección de productores.
- ProducerFactory. Factoría de productores. Genera productores de uno u otro subtipo según las condiciones que determinen los parámetros recibidos.
- Producer. Abstracta. Clase padre productor.
- BigProducer. Gran productor.
- SmallProducer. Pequeño productor.
- FederateProducer. Productor federado.



- **model.product**
  - ProductCollection. Colección de productos.
  - Product. Clase producto.
  - Crop. Cultivo, define una explotación de un producto.
  - CropShareItem. Define la participación de un cultivo en la producción total de un producto.



- **utils**
  - DistanceUtils. Útiles para el cálculo de distancias de envío.
- **sampledata –** Clases auxiliares para la carga de datos de prueba.
  - DistanceTable. Tabla de distancias respecto a la cooperativa.
  - ProductPrices. Evolución de precios de productos.
  - ProductType. Productos registrados en el sistema.
  - SampleCusomer. Clientes de prueba.
  - SampleLogistic. Empresas de logística de prueba.
  - SampleOrder. Pedidos de prueba.
  - SampleProducer. Productores de prueba.
  - ShippingRate. Tarifas de envío.
  - SystemData. Datos generales del sistema.

3.2.4. Datos de prueba

Se han generado una serie de sets de datos de prueba con los que inicializar la aplicación. Para evitar el uso de persistencia de datos, que queda fuera del alcance de esta práctica, se han definido mediante clases con datos predefinidos en el paquete sampledata (referenciados y listados ya en el apartado anterior). En la mayoría de casos se trata de enumerados donde los distintos valores representan el conjunto de los datos de prueba a cargar. Al inicializarlos se crean objetos plantilla a partir de los que se cargan los datos en los objetos reales del sistema.

3.3. Funcionalidad de generación de informes

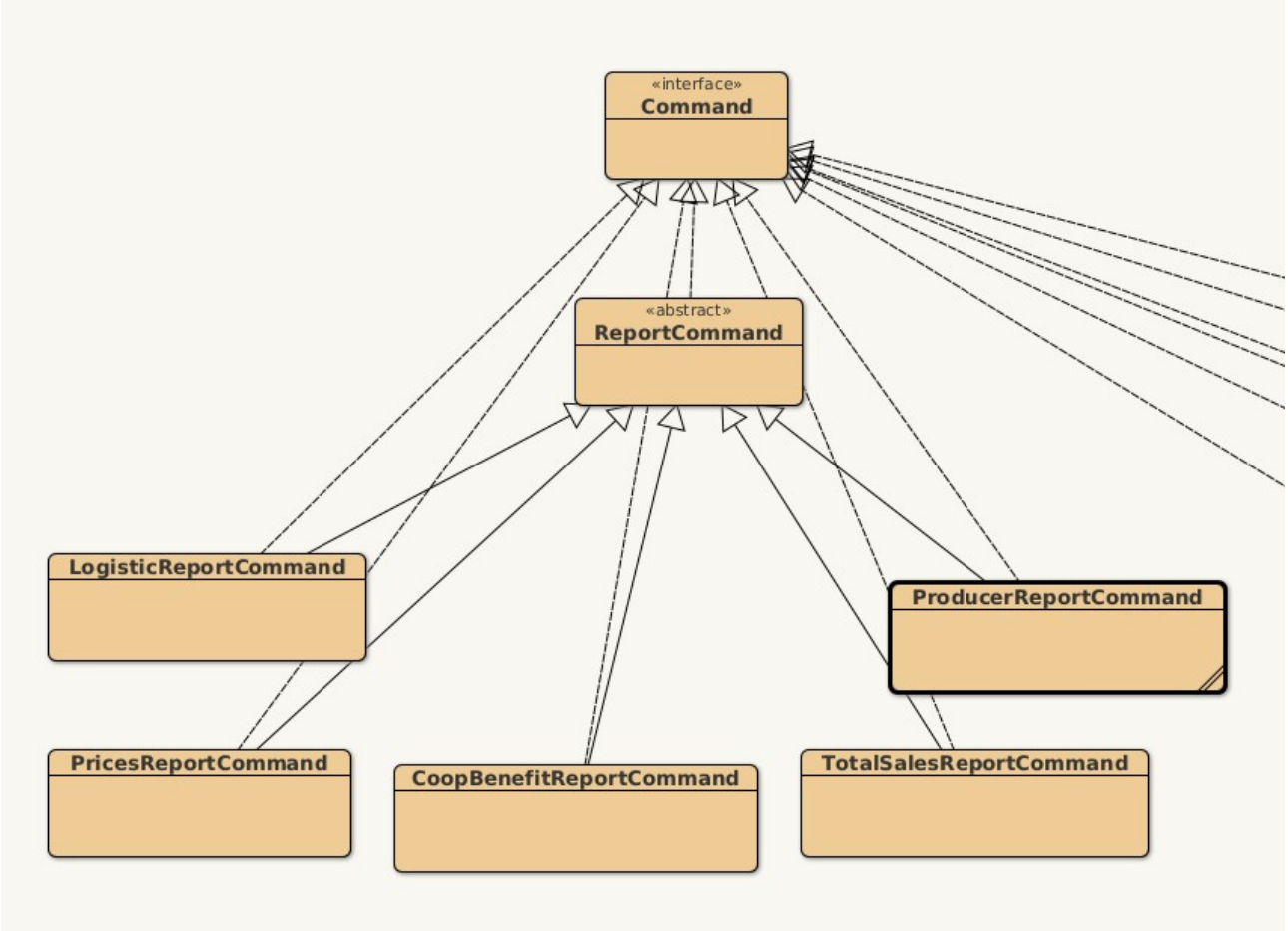
Se pide en el tercer apartado de la práctica el desarrollo, partiendo de la fase anterior, de una funcionalidad de informes que proporcione, al menos, los siguientes:

- Ventas totales por producto
- Beneficios de cada productor por producto
- Beneficios de cada empresa de logística
- Beneficios de la cooperativa por producto
- Evolución semanal de precios por producto

Dada la arquitectura de la aplicación, esta operación consistirá simplemente en añadir un controlador, un menú y sus respectivos commands. Los modelos guardan ya la información a consultar con lo cual no son necesarias modificaciones en ellos. Se añaden también, en la capa de vista, las clases Table y ReportTable en las que se implementa la generación de tablas y su renderizado en consola. Se adjunta la resolución de esta fase del proyecto en el archivo L3_cooperativa_informes.zip

En lineas generales la ejecución de cada informe consíste fundamentalmente en llamar a la colección correspondiente, recorrer los objetos pertenecientes a esta, leer los datos relevantes y agregarlos, y construir con el agregado un objeto tabla que devolvemos a la vista para su renderizado.

Cabe mencionar aquí que los distintos objetos Command heredan de una única clase ReportCommand que realiza las operaciones de renderizado, encargándose los hijos de poblar los datos (título, tablas) a partir de los que efectuar este renderizado.



Clases introducidas en esta fase del proyecto:

- **controller**
  - ReportController
- **view**
  - Table
  - ReporTable
- **view.menu**
  - ReportsMenu
- **view.command**
  - ReportCommand

- CoopBenefitReportCommand
- LogisticReportCommand
- PricesReportCommand
- ProducerReportCommand
- TotalSalesReportCommand

4. Diseño

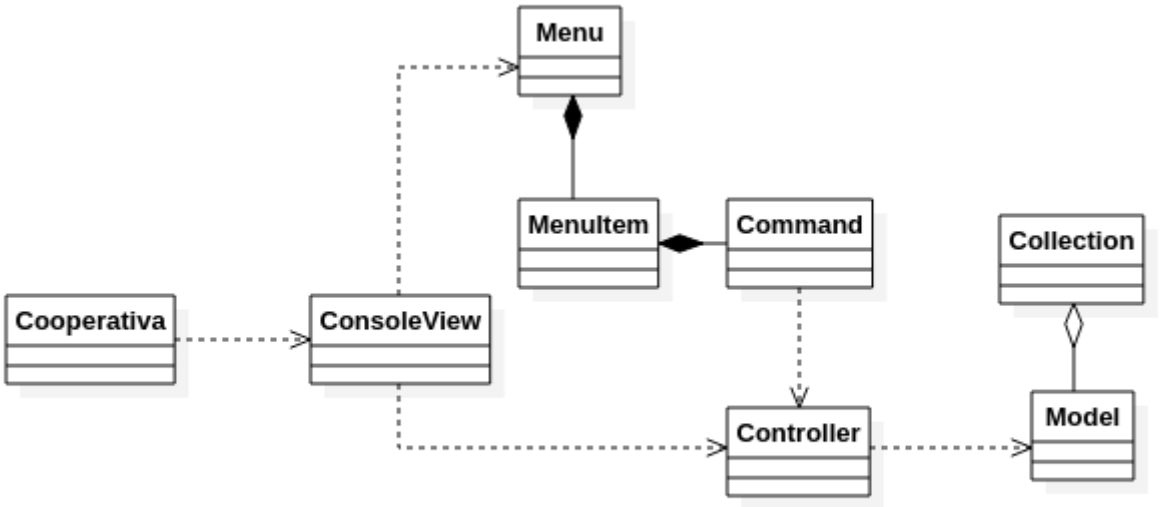
4.1. Arquitectura MVC

Como se ha venido mencionando en la sección anterior, la aplicación sigue una arquitectura modelo-vista-controlador, de tal forma que esté desacoplada la lógica del sistema de su representación.

La capa vista además utiliza un sistema de menús con elementos asociados a comandos (patrón de diseño Command). De esta manera, cada elemento de menú, al ser accionado, ejecuta la acción definida en el objeto Command que tiene asociado. Éste pide los datos de entrada y hace las validaciones necesarias para preparar y efectuar una llamada a un método determinado de un controlador por el que se define el caso de uso (en determinados casos puede llamar de forma auxiliar a más de un método del controlador, o incluso de distintos controladores).

Hay tantos controladores como áreas funcionales tiene el sistema, y tantos métodos en ellos como casos de uso (más algunos auxiliares adicionales según necesidades).

El controlador entonces llama a los modelos y colecciones de éstos implicados en el caso de uso y les pasa los mensajes que sean necesarios en cada situación.



En todo caso, siempre se sigue la secuencia Menu → Comando → Controlador → Modelo. De esta forma se mantiene el acoplamiento siempre lo más bajo posible, y se evita de cualquier manera que los modelos se puedan acoplar a la vista, lo cual sería completamente indeseable.

4.2. Paquetes

Ya se ha hecho referencia en la sección 3 al listado de paquetes que se han creado, y las clases que contiene cada uno de ellos. La estructura, recordemos, es:

- view
- view.menu
- view.command
- controller
- model
- model.customer
- model.logistic
- model.order
- model.producer
- model.product
- utils
- sampledata

Como se puede observar, la división principal es por capas MVC. De esta manera se facilita una posible implementación a futuro de otro tipo de vista distinto, posibiltando la sustitución de un paquete vista por otro.

En un segundo nivel, vemos que el modelo tiene subpaquetes por funcionalidad. De esta manera se facilita la adición de áreas funcionales nuevas.

4.3. Uso de herencia y polimorfismo

Además de en las clases en que es evidente que se esperaba el uso de la herencia (superclases y subclases de los modelos: productores, logística, clientes), se ha intentado facilitar y usar lo máximo posible la extensibilidad de clases, tanto desde el punto de vista de la herencia de propiedades como de su uso polimórfico, abstrayendo en la medida de lo posible al cliente del subtipo del servidor (principio de sustitución de Liskov).

Se han utilizado para esto tanto superclases como clases abstractas e interfaces en diversas casuísticas: menús, comandos, colecciones, etc.

4.4. Estructuras de datos y parametrización de clases

Se han usado estructuras de datos abstractas (en particular colecciones) definiéndolas mediante clases parametrizadas e interfaces. Para que esto fuera aplicable a cualquier tipo, se ha parametrizado el contenedor (Collection<E>) y se ha definido la interfaz CollectionItem para los

elementos contenidos. De esta forma se pueden crear colecciones de cualquier tipo que implemente CollectionItem, lo que nos da una gran flexibilidad.

Adicionalmente, dado que las colecciones de los distintos modelos pueden necesitar compartir sólo los métodos propios de la estructura de datos Colección, pero tener otros que no sean compatibles entre unas y otras, se ha optado por que en lugar de heredar de Collection la usen por composición (es decir, conteniendo una colección internamente, a la que se redirigen los métodos propios de la estructura de datos, mientras los no propios se resuelven en el objeto envolvente).

4.5. Patrones de diseño

Se han resuelto algunos problemas de diseño mediante el uso de determinados patrones, según se detalla a continuación.

Factory Method:

El patrón Factory Method se usa cuando se necesita instanciar objetos de distintos subtipos de un mismo tipo, donde el subtipo a instanciar depende de una determinada lógica, pero manteniendo el tratamiento polimórfico en el cliente y sin que éste necesite conocer esta lógica ni el subtipo que recibe.

Se ha utilizado en particular para la instanciacion de objetos productor, cuyo subtipo depende de la superficie y número de productos cultivados.

Singleton:

Se usa cuando se necesita asegurar que exista una única instancia de un objeto, y que esta sea accesible desde cualquier parte, sin necesidad de que se esté pasando la referencia de unos objetos a otros durante la secuencia de ejecución.

Para ello se implementa mediante un atributo y un getter estáticos de la clase.

En nuestro caso, es crucial asegurar que las instancias de las colecciones sean únicas. Si bien en la mayoría de los casos se accede a ellas desde los controladores, hay algunos casos en que necesitan estar accesibles también desde otras clases de los modelos, con lo cual es idóneo que sean Singleton.

Command:

Ya se ha explicado extensamente el uso que se hace en la aplicación del patrón Command. En general, este patrón se usa cuando un objeto sabe que en algún momento se va a ejecutar una determinada acción, y quién la va a recibir, pero no quién la va a ejecutar ni cuándo.

Consiste fundalmentalmente en convertir esta acción en un objeto (encerrando la acción en el método execute de dicho objeto) de forma que se pueda pasar por parámetro para que sea otro objeto el que la ejecute más adelante.

Es muy común utilizar este patrón en elementos de interfaces de usuario, así como en elementos de menú, tal y como lo hemos usado en este proyecto.

5. Conclusiones

En primer lugar se hace necesario mencionar algunos aspectos de la práctica que, si bien se habían planteado originalmente, se han quedado fuera del proyecto por limitación de tiempo:

- Productores federados. Se planteó la clase pero no se ha llegado a desarrollar la lógica para federar productores.
- Vista de detalle de cada uno de los modelos (productor, empresa de logística, cliente, pedido), con toda su información al completo.
- Listado de pedidos, general y por cliente (si bien en los informes del tercer apartado, aunque no se listen, se accede a ellos y se recorren para generar los agregados de datos).
- Aplicación de las tablas desarrolladas en la fase 3, con mejor aspecto visual y las columnas bien alineadas, a los listados generados en los menús de la fase 2.

Dicho esto, y como valoración personal, es este un proyecto bastante extenso e interesante en el que se afrontan una gran variedad de problemas relacionados con la programación orientada a objetos, donde aplicar los conocimientos adquiridos en la asignatura.

Cabe hacer mención sin embargo a lo poco equilibrado que parece el esfuerzo requerido respecto al peso que tiene la práctica en la evaluación global de la asignatura. No obstante, es mucho lo que se aprende con un ejercicio de este calado. En particular me quedo con lo aprendido del lenguaje Java, con el que no había trabajado nunca antes, la aplicación estricta de los conceptos de POO, el tipado estático, etc.

El aprendizaje es amplio y sólido, la cantidad de conceptos vistos es bastante elevada y personalmente lo valoro muy positivamente.

Juan Barranco