Clases avanzadas: Especificación e implementación de un proyecto

Vamos a ver un ejemplo solucionado de un proyecto con una cierta envergadura, en el que vamos a seguir todos los pasos lo más explicado posible desde su especificación hasta su implementación final, en el proyecto veremos ejemplos prácticos de los conceptos vistos en los temas relativos a clases: clases abstractas, métodos abstractos, colecciones, excepciones anidadas y uso de fechas.

Parte 1: Especificación de requerimientos

En la especificación el cliente nos cuenta qué datos necesita en su aplicación y que funciones necesita sobre esos datos. Dado que estamos en el módulo de programación, no nos vamos a centrar demasiado en esta primera parte que consiste en pasar de desde la especificación al diseño, no obstante al saber la especificación podemos tener una visión general de qué es lo que quiere el cliente y saber el porqué se ha definido de esa manera la estructura de clases que mostraremos más abajo.

*Nuestra aplicación consiste en la **gestión de ventas de un almacén**, en dicho almacén. En primer lugar deseamos gestionar la información de nuestros productos, de cada producto queremos saber su identificador de producto (que será calculado de forma automática por el programa), el nombre del producto, el precio del producto (el precio del producto depende del tipo de producto que sea y se calcula a partir de un precio base), las ventas que se han realizado con ese producto y también queremos saber el total de productos que tenemos dados de alta.*

Hay dos tipos básicos de productos, los muebles y los electrodomésticos, de los muebles queremos saber el año de fabricación el tipo de mueble y el estilo. De los electrodomésticos queremos saber la marca, el tamaño y el fabricante.

Además tenemos dos subtipos más de electrodomésticos: Los televisores y las lavadoras.

De los televisores queremos saber las pulgadas y el tipo de de televisor que es (LED, Plasma..). De las lavadoras queremos saber dos datos: las revoluciones y la carga.

En nuestra empresa esos son todos los tipos de productos que comercializamos, es decir que nuestros productos siempre van a ser o mueble, o televisor o lavadora, no tenemos ningún producto más y por lo tanto no se permite dar de alta productos que no estén incluidos en una de esas tres categorías.

Como se ha dicho, el precio del producto dependerá del tipo de producto que sea, el precio del producto se calcula siempre a partir de un precio de base y estas son las normas para calcular el precio definitivo:

- *Si es un mueble:*
  - *Si el tipo de madera es ROBLE, su precio será el precio base introducido por el usuario más el 10%*
  - *Si el tipo de madera es PINO, su precio será el precio base menos el 10%*
  - *Si es de otro tipo el precio será el mismo que el precio base*
- *Si es un televisor*
  - *Si tiene más de 40 pulgadas, su precio es el precio base + el 10%*
  - *Si es de PLASMA su precio es el precio base menos el 10%*
  - *o En cualquier otro caso, su precio es igual al precio base*

- *Si es una lavadora:*
  - *Si las revoluciones son más de 500 su precio es el precio base mas el 10%*
  - *Si la carga es menor que 8 kilos su precio es el precio base menos el 15%*
  - *En cualquier otro caso su precio es igual al precio de base.*

Como se ha dicho de cada producto queremos saber sus ventas, será necesario gestionar los siguientes datos de cada venta: el identificador de la venta que se calculará automáticamente, el nombre del vendedor, el cliente y producto y el importe final de su compra tras aplicarle el descuento en caso de que el cliente fuese mayorista.

También es necesario saber la siguiente información sobre los clientes:

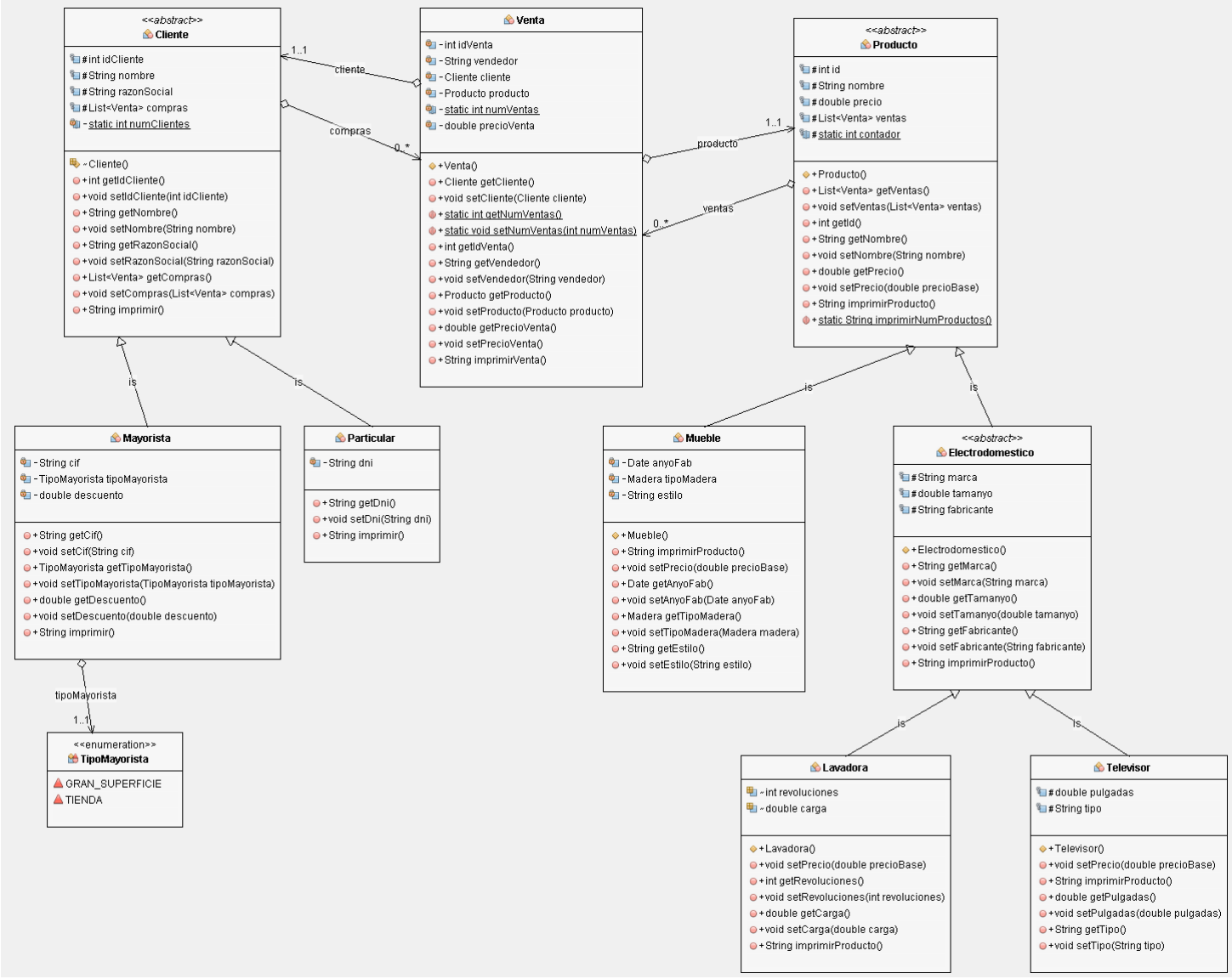
El id de cliente, calculado automáticamente, su nombre, su razón social y las compras que nos ha realizado. Todos nuestros cliente tienen que ser de dos tipos : Particulares y mayoristas, en el caso de que sean particulares queremos además saber su dni, en el caso de que sea mayorista queremos saber su cif, el descuento que tiene y el tipo de mayorista que es (tiene que ser forzosamente Gran superficie o tienda)

Queremos una aplicación capaz de dar de alta un cliente, producto o mayorista, mostrar por pantalla un cliente, venta o mayorista concreto a partir de su id, ver todos los clientes ventas o mayoristas en un listado, eliminar un producto (en cuyo caso habrá que eliminar también sus correspondientes ventas) , eliminar una venta y eliminar un Cliente(en cuyo caso habrá que eliminar también las ventas asociadas a él).

A partir de esta especificación de requerimientos **el diseñador** de la aplicación deberá de ser capaz de realizar el siguiente modelo de datos y especificación de clases a realizar, con sus correspondientes métodos, **el programador** debe de, a partir del modelo de abajo, realizar la implementación.

Parte 2: Modelado de clases

A partir de la especificación anterior, vamos a construir el esquema de clases que vamos a usar, para ello en primer lugar definimos las clases del modelo, es decir que clases necesitamos para guardar los datos básicos y los métodos que operan sobre dichos datos básicos. Para expresar los datos del modelo lo más apropiado es usar un diagrama UML veamos como quedaría:



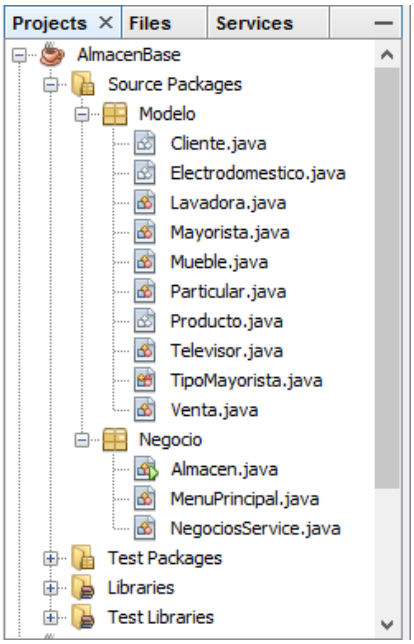
Algunas directrices para realizar el diseño:

1. Identifica las clases, cuando un subtipo tenga atributos propios, decláralo como clase hija.
2. Identifica los atributos de dichas clases.
3. Recuerda que los métodos de las clases operan sobre sus atributos, intenta que la mayoría (o todas) las operaciones sobre los atributos estén dentro de su clase.
4. Si un método no se puede resolver hasta que no sepamos el subtipo de la clase a la que pertenece, decláralo abstracto.
5. Intenta realizar la mayor parte de código posible en las clases padres, para que el código pueda ser reutilizado.
6. En las clases hijas usa los métodos de las clases padres.

Una vez llegados a este punto pueden surgir las siguientes dudas:

- **¿Por qué se ha definido Producto y Electrodoméstico como abstracto?** Porque en la especificación quedó bien claro que nuestros productos tienen que ser necesariamente del subtipo Mueble, Televisor y Lavadora.
- **¿Se podría haber realizado una única clase producto en el que estuvieran incluidos todos los atributos de sus subclases?** Si pero es totalmente ineficiente ya que tendríamos muchos atributos que no se usan nunca, además podríamos tener datos inconsistentes (Muebles con pulgadas p.e.), también tendríamos un atributo TIPO, que condicionaría todo el funcionamiento de la clase, es decir, imprimir dependería del tipo, en resumen cuando una clase sea un subtipo de otra y el subtipo tenga atributos propios, hay que relacionarlas mediante herencia.
- **¿Cuándo se crea un nuevo televisor, que datos puedo guardar en él?** TODOS (menos los static del padre) los propios y los de los padres , un televisor es un producto(y un electrodoméstico) y por lo tanto también tiene nombre, precio, fabricante…etc.
- **¿Y qué métodos puedo usar?** Todos también
- **¿Por qué en Mueble, Televisor y lavadora tengo el método imprimir? ¿No tengo ya definido el del padre?** Como clases hijas, pueden usar el método imprimir ya definido en el padre, sin embargo si uso el del padre, solo me imprime los datos de producto. Si quiero que me imprima también los datos de Televisor debo redefinir el método, cuando lo hago se pone la anotación @Override que indica que el método está redefinido en la clase hija.
- **¿Al redefinir el método debo volver a hacer el código para imprimir los datos de producto?** NO, si haces referencia a super.imprimir(); se te ejecuta el método imprimir de la clase padre y no tienes que volver a escribirla, además si introduces cambios a la hora de imprimir un producto, solo tienes que cambiar código en un sitio(en la clase padre donde está definido).
- **¿Por qué el método setPrecio(doublé precioBase) se ha definido como abstracto?** Como se decía en la especificación, el precio del producto depende del tipo que sea, por lo tanto no puedo implementarlo en Producto ya que a priori no puedo saber el tipo de producto que es. Al definirlo como abstracto me ahorro implementarlo y además fuerzo a las clases hijas no abstractas a definirlo, asegurándome que todos los productos implementan el método setPrecio()
- **¿En el atributo List<Venta> ventas de producto están incluidas las ventas de mi aplicación?** NO, el atributo ventas es un atributo de Producto, es decir cada uno de los productos que demos de alta tiene su propia Lista de ventas que refleja las ventas concretas que hemos tenido de ese producto, no olvides que los atributos de producto hacen referencia a un único producto. Necesitaremos un objeto List<Venta> para meter todas las ventas de nuestra aplicación pero ese objeto no estará en el modelo sino en un nivel superior.

Las clases anteriores pertenecen al modelado y las vamos a guardar todas en el **package modelo** como se muestra a continuación:



Fíjate en lo siguiente: TipoMayorista no es una clase, es una enumeración mediante la cual forzamos a que el tipo de mayorista sea forzosamente o tienda o gran_almacen o null.

Pues bien una vez tenemos ya hecho el paquete modelo, podemos realizar ahí todas nuestras clases del modelo, pero sin embargo el modelo solo guarda datos de forma individual, es decir podemos crear un producto pero **¿Dónde guardamos la lista de todos nuestros productos, clientes y ventas? ¿Y donde ejecutamos el menú principal? ¿Y todos los submenús del menú principal?** Podemos estar tentados a realizar todas esas cosas en el método main pero, en cuanto empecemos a realizar código nos daremos cuenta de que será un método muy grande, además como normal general , cuando programemos con un lenguaje orientado a objetos hay que agrupar en clases todo lo que se pueda, evitando un main cargado.

Si pensamos un poco nos damos cuenta que necesitamos un List<Venta> ventas (donde guardamos todas las ventas) un List<Producto> productos(donde guardamos todos los productos) y un List<Cliente> clientes (donde guardamos todos los clientes). Además necesitamos : añadir, mostrar y eliminar datos de esas listas, es decir necesitamos un sitio donde agrupar UN CONJUNTO DE DATOS Y LOS MÉTODOS NECESARIOS PARA GESTIONAR ESOS DATOS, estamos hablando de la propia definición de clase, esa clase se va a llamar **NegociosService**. El paquete donde vamos a realizar esas clases en las que gestionar los datos de nuestro modelo lo vamos a llamar negocio y consta de las siguientes clases:

- **Almacen.java:** Clase que contiene el método main, no tiene atributos, en dicho método se declara una variable de la clase menúPrincipal y se ejecuta el método iniciar aplicación

- **MenuPrincipal.java:**
  - Atributos;
    - NegociosService servicio
  - Métodos:
    - menuPrincipal() inicializa atributo servicio
    - IniciarAplicacion() -> método que tiene el menú principal, a partir de aquí se deriva al menú clientes, productos o ventas

- menuClientes() se implementan todas las opciones referentes a clientes, llamando a los métodos implementados en NegociosService, es decir, si el usuario decide dar de alta un cliente, llama al método de NegociosService correspondiente (servicio.nuevoCliente())
- menuProductos() idem para productos
- menuVentas() idem para ventas.
- datosProducto() recoge los datos como producto, y llama al metodo según el tipo de producto
- pedirCliente() recoge datos del cliente
- pedirMayorista() recoge datos si es un mayorista
- pedirParticular() recoge datos si es particular
- pedirMueble()recoge datos del mueble
- pedirLavadora() recoge datos de la lavadora
- pedirTelevisor() recoge datos del televisor
- pedirTipoM() recoge tipo de cliente
- pedirMadera() recoge tipo de madera
- validarFecha() valida fecha introducida

- **NegociosService.java:**
  - Atributos:
    - List<Producto> productos -> Lista que contiene todos los productos de nuestra aplicación
    - List<Venta> ventas-> Lista que contiene todas las ventas de nuestra aplicación
    - List<Cliente> clientes-> lista que contiene todos los clientes de nuestra aplicación
  - Métodos:
    - Void introducirProducto()-> introduce un cliente en nuestra lista
    - Void introducirVenta() ->introduce una venta en nuestra lista, recibe un id de Producto y otro de cliente y el vendedor
    - Void introducirCliente()->Introduce un cliente en nuestra lista
    - Producto buscarProducto()-> recibe un id de producto y devuelve ese producto.
    - Cliente buscarCliente() -> recibe un id de cliente y devuelve ese cliente.
    - Venta buscarVenta() -> recibe un id de Venta y devuelve los datos de esa venta.
    - boolean eliminarProducto() -> recibe un id de producto y elimina ese producto de nuestra lista, también elimina sus compras asociadas.
    - boolean eliminarCliente() -> recibe un id de Cliente y elimina dicho cliente y sus compras asociadas.
    - boolean eliminarVenta()->recibe un id de venta y la elimina.
    - String imprimirTodosProductos()-> devuelve en forma de tabla todos los productos que tenemos en la lista
    - String imprimirTodosCliente() -> devuelve en forma de tabla todos nuestros clientes
    - String imprimirTodasVentas() -> devuelve en forma de tabla todas nuestras ventas.

Llegados a este punto podemos plantearnos si **¿no hubiera sido más a apropiado hacer una clase service para Ventas, otra para Productos y otra para Clientes, cada una de las cuales gestionara su lista?** La respuesta a esa cuestión es que **SI,** es más apropiado pero hay que tener en cuenta que al eliminar Productos también hay que acceder a la lista de Ventas, es decir deberías implementar también getters y setters en esas clases de servicio. Adaptar el programa a esa nueva estructura lo haremos en ejercicios próximos.