Análisis sistema supermercado “Supercomprín”

Según representado en el modelo de clases aquí abajo, el sistema del supermercado está constituido por dos tablas (clases Java) con atributos de tipo private a los cuales se puede acceder desde la otra clase a través de métodos getters.

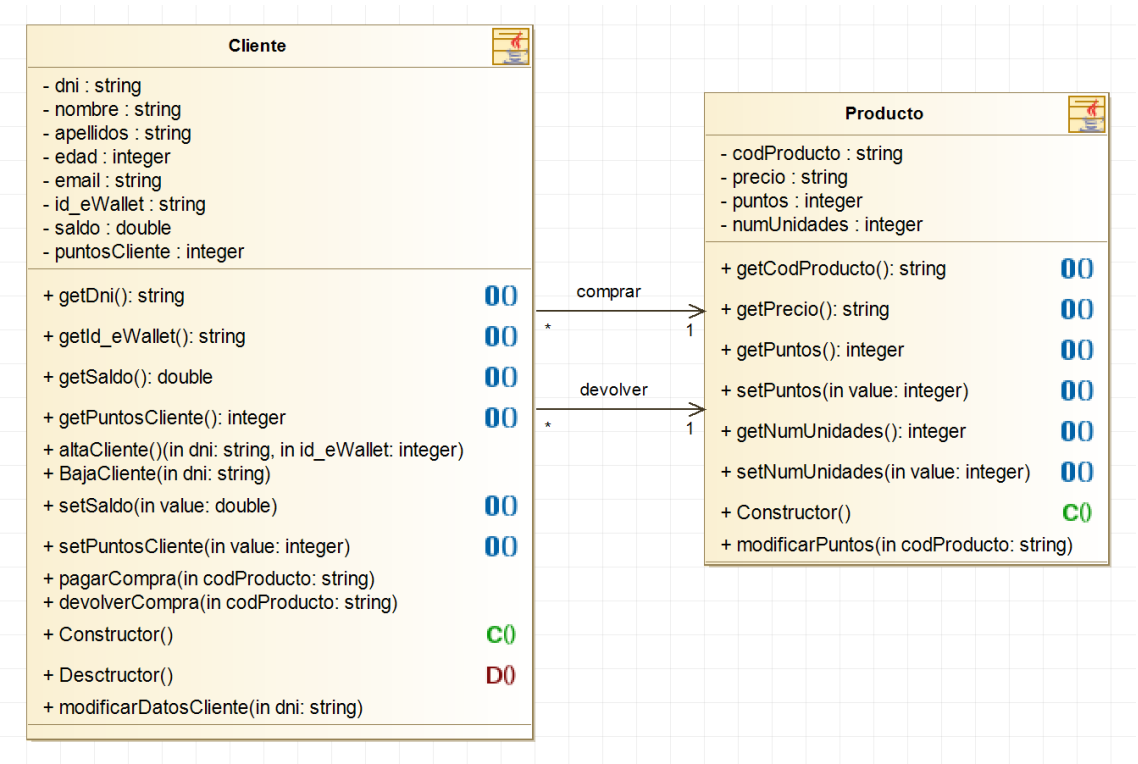
Las operaciones a implementar dentro de la clase Cliente son:

- Alta cliente (sentencia INSERT);
- Baja cliente (sentencia DELETE);
- Modificar datos clientes (sentencia UPDATE);
- Pagar compra;
- Devolver compra;

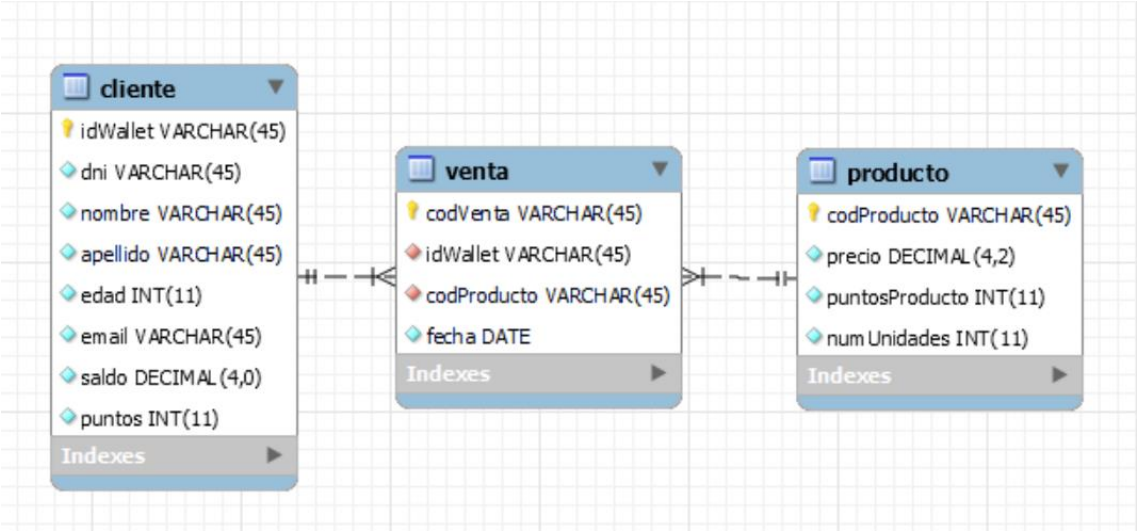
La única operación a implementar dentro de la clase Producto es “modificar puntos del producto”, porque solo se pide modificar los puntos de un producto. Todavía sería útil añadir dos opciones más, para poder insertar y borrar productos.

Las dos tablas están relacionadas entre sí, a través de las operaciones de compra y devolución. Este modelo representado es simplificado e incorrecto porque realmente falta una tabla donde mover los productos desde el stock (podría llamarse “venta”), una vez comprados, y de la tabla venta a producto, una vez devueltos.

Un cliente puede, en cada operación, comprar o devolver como mucho un producto (0-1), y un producto podría, teóricamente, ser comprado por más clientes a la vez (0-*), ya que cada producto se identifica con un código y puede tener más unidades (es un poco inexacto porque sería ideal que cada producto, aunque del mismo tipo, tuviese id único).



El modelo que he implementado en MySQL es este aquí abajo, con tres tablas, pero, de momento que en el ejercicio se pide implementar unos métodos que se pueden desarrollar de forma más sencilla, en Java no he creado las clases Venta y VentaDAO.



DDL BD MySLQ

```
CREATE DATABASE supermercatín;

USE supermercatín;

CREATE TABLE `cliente` (
  `idWallet` varchar(45) NOT NULL,
  `dni` varchar(45) NOT NULL,
  `nombre` varchar(45) NOT NULL,
  `apellido` varchar(45) NOT NULL,
  `edad` int(11) NOT NULL,
  `email` varchar(45) NOT NULL,
  `saldo` decimal(4,0) NOT NULL,
  `puntos` int(11) NOT NULL,
  PRIMARY KEY (`idWallet`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

CREATE TABLE `producto` (
  `codProducto` varchar(45) NOT NULL,
  `precio` decimal(4,2) NOT NULL,
```

```
  `puntosProducto` int(11) NOT NULL,
  `numUnidades` int(11) NOT NULL,
  PRIMARY KEY (`codProducto`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
CREATE TABLE `venta` (
  `codVenta` varchar(45) NOT NULL,
  `idWallet` varchar(45) NOT NULL,
  `codProducto` varchar(45) NOT NULL,
  `fecha` date NOT NULL,
  PRIMARY KEY (`codVenta`),
  KEY `codProducto_idx` (`codProducto`),
  KEY `idWallet` (`idWallet`),
  CONSTRAINT `codProducto` FOREIGN KEY (`codProducto`) REFERENCES `producto`
(`codProducto`),
  CONSTRAINT `idWallet` FOREIGN KEY (`idWallet`) REFERENCES `cliente` (`idWallet`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Aclaraciones sobre algunas decisiones tomadas

- El cliente tiene que ser mayor de edad: Para no complicar el manejo de datos, no he trabajado con fechas, decidiendo registrar la edad como atributo de Cliente y sobre su inserción poner una condición restrictiva.

- Asignación del e-Wallet: He creado una única clase Cliente que contiene los datos básicos del cliente más los de su e-Wallet porque realmente el dato que identifica un cliente como tal (PK) es exclusivamente el id del e-Wallet; sin este id no vamos a guardar las personas en la base de datos porque no son clientes.

- Los productos se identifican por código de producto (PK) y al tener número de unidades está claro que hay más de un producto con el mismo código, aunque sería más exacto identificar cada pieza individualmente.

- Una vez empezado a desarrollar el código, he decidido poner los métodos de “comprarProducto” y “devolverProducto” en la clase ProductoDAO, aunque, como ya explicado, es incorrecto porque realmente tendría que existir una tabla intermedia donde sí tendría sentido desarrollar estos dos métodos. Nuestro caso es simplificado.

- He añadido unos métodos a la clase ClienteDAO (“buscarCliente” para refactorizar la búsqueda común a muchos métodos y “recargarSaldo”) y a la clase ProductoDAO (“seleccionar”, “buscarProducto”, “comprarProducto” y “devolverProducto”).

Ruta JavaDoc

View Generated javadoc at “TransaccionesJDBC\target\site\apidocs\index.thml”

Enlace GitHub

https://github.com/alibar92/Supermercatin.git