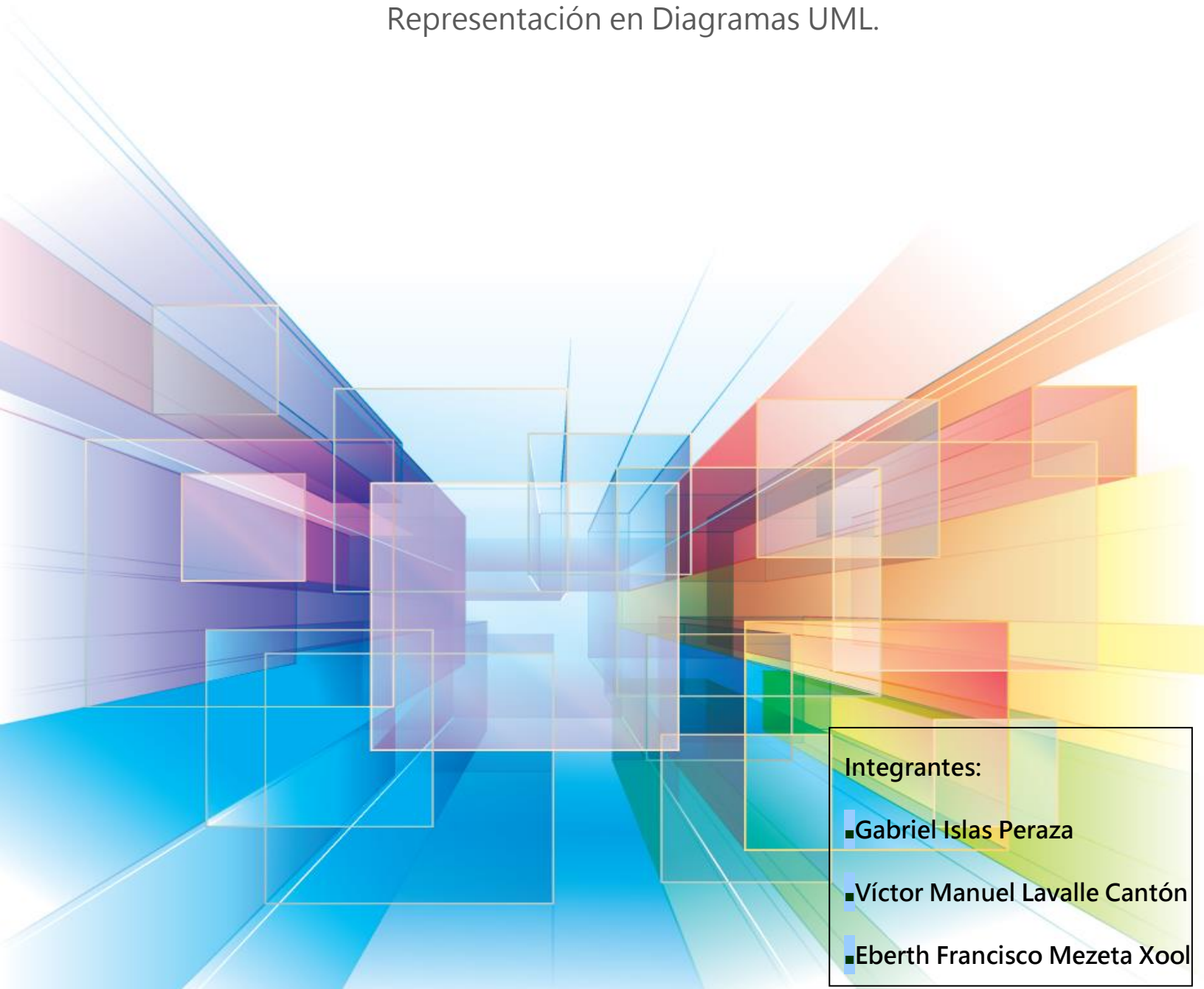
PROYECTO FINAL

Diseño de Software

Aplicación de Los Patrones de Diseño, Principios SOLID Y Representación en Diagramas UML.



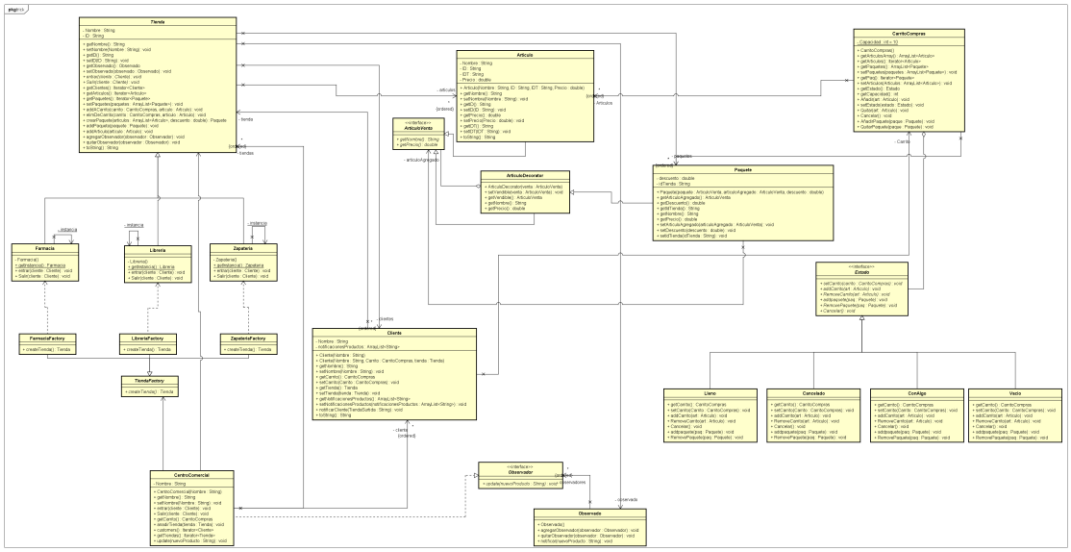
Integrantes:

- **Gabriel Islas Peraza**
- **Víctor Manuel Lavalle Cantón**
- **Eberth Francisco Mezeta Xool**

Diagrama de Clases

Se adjunta link y dentro de la carpeta del proyecto, en la subcarpeta “Diagramas > Clases” se encuentra la imagen de este diagrama (y de los otros diagramas)”, ya que, al tratar de ponerlo en el presente documento, no se apreciaba con claridad este diagrama.

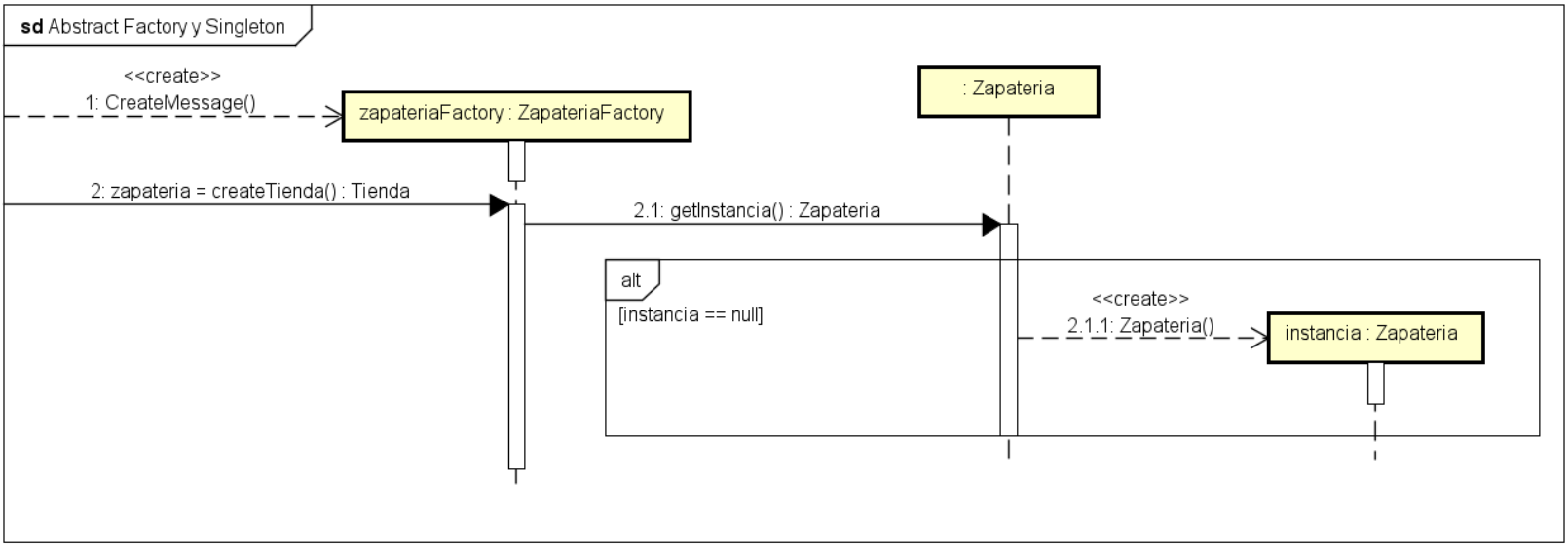
https://i.imgur.com/OhcMJGy.png



Diagramas de Secuencia

Abstract Factory & Singleton

Este patrón se aplicó con la finalidad de facilitar la creación de tiendas que se encuentren en el centro comercial.



Paso 1: Se crea una *ZapateriaFactory*, clase que nos permite fabricar una instancia de la tienda Zapatería, gracias al patrón *Singleton*.

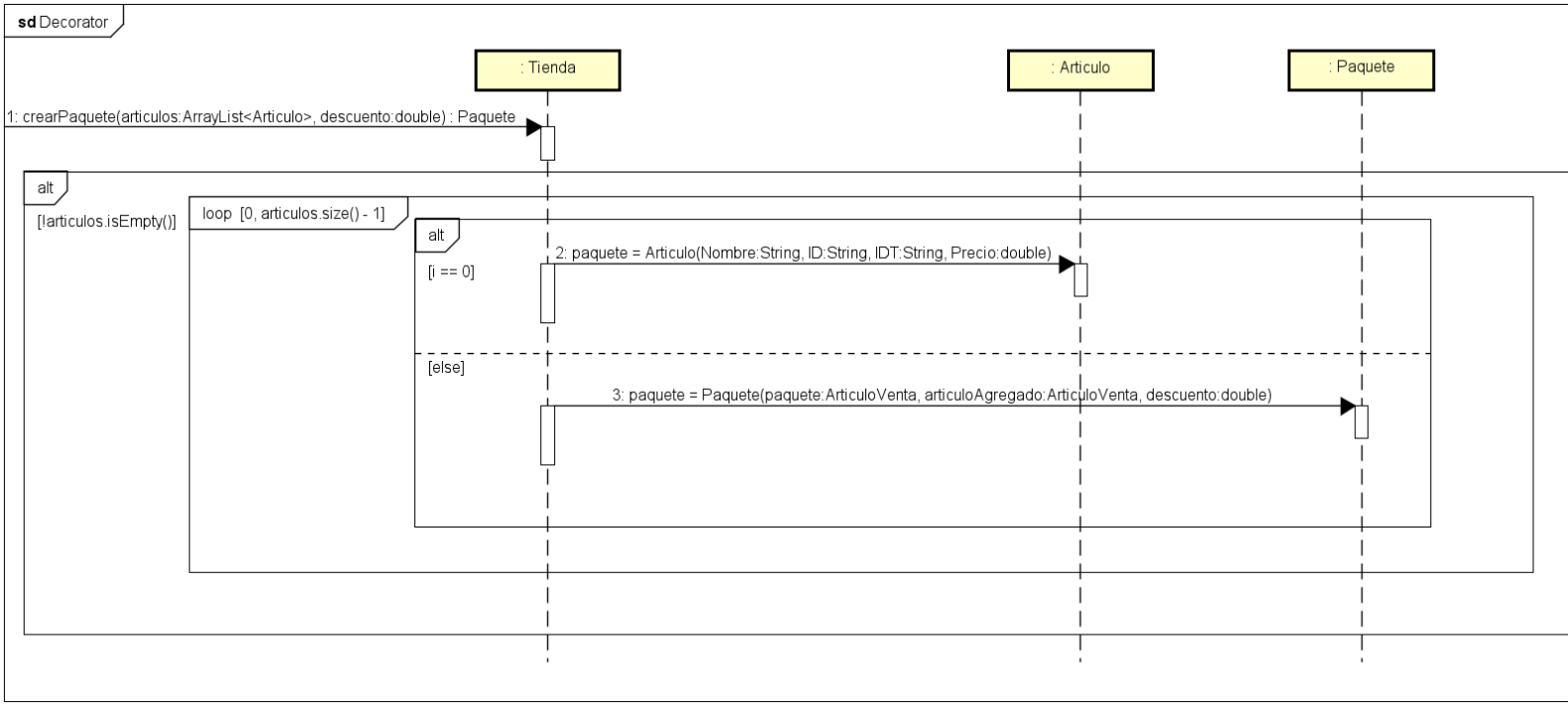
Paso 2: Procedemos a crear una *Zapateria*, utilizando la función *CrearTienda*

Paso 2.1: Por consecuencia, esta función llama a la funcion *getInstance* que nos retornara la instancia única de la *Zapateria* Para hacer esto, hay una estructura *if,* solo se puede entrar si instancia == null. Esto significa que solo se puede entrar al *if* si la instancia no ha sido creada. Si se cumple:

> **Paso 2.1.1**: A instancia se le asigna una nueva Zapateria y se retorna el valor. De lo contrario: Se retorna el valor ya establecido de instancia.

Decorator

El uso de este patrón fue útil para poder crear los paquetes con los artículos de las tiendas.Se puede ver la implementación con la función "*CrearPaquete"* la cual sería el decorado.



Paso 1: Para crear un paquete en una Tienda, se llama a su funcion *CrearPaquete*, la cual recibe una lista de artículos con la cual formar el paquete.

-Utilizando esta lista, se forma un paquete, pero primero se debe pasar un *if*, que revisa si dicha lista no está vacía, con *!articulos.isEmpty()* -Si se cruza dicho *if*, entramos en un loop que va de i = 0 y continua mientras i < tamano de la lista de *articulos (articulos.size)* -En este loop, hay una estructura *if - else*, si i == 0, o sea el primer elemento de la lista.

Paso 2: En la variable paquete, se asigna el primer artículo.

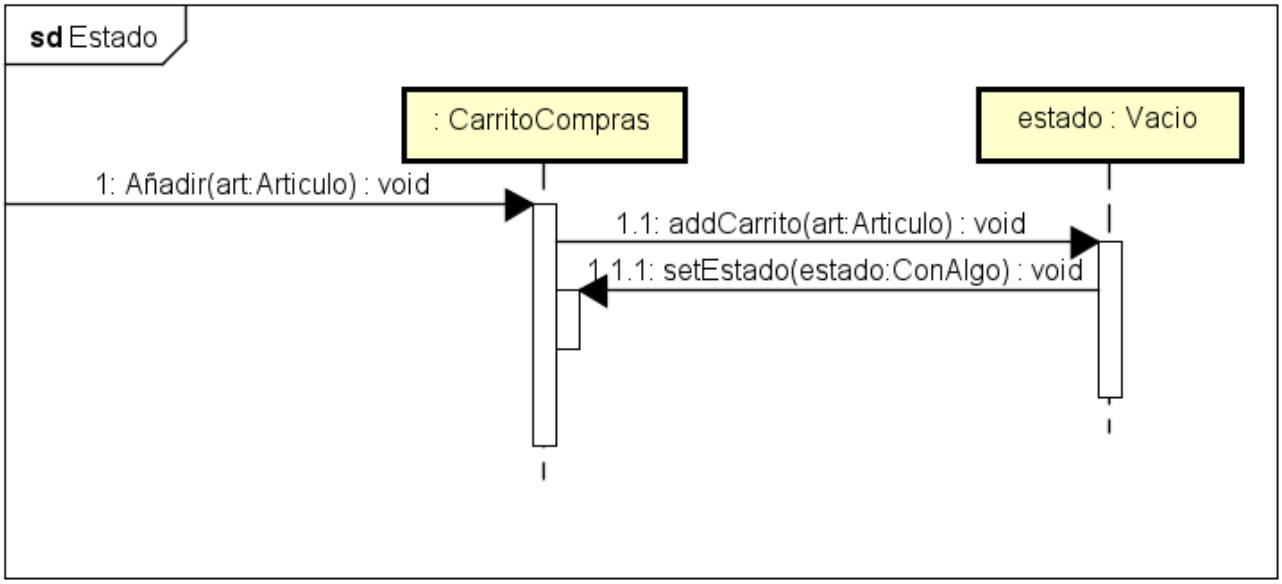
-Si no se da este caso, o sea el primer elemento de la lista ya paso.

Paso 3: En la variable paquete, se asigna un paquete, formado con los contenidos anteriores de la variable paquete, lo cual nos permite construir el paquete con todos los elementos de la lista.

Nota: La variable paquete es de tipo *ArticuloVenta*, que es una interfaz implementada por Articulo y Paquete, lo cual nos permite combinar ambos en dicha variable.

Estado

El uso de este patrón fue útil para poder determinar el estado del carrito de compras mediante las acciones que se realizaban agregar o eliminar un artículo o paquete.



Paso 1: Se añade un artículo al carrito de compras utilizando la función *Anadir*.

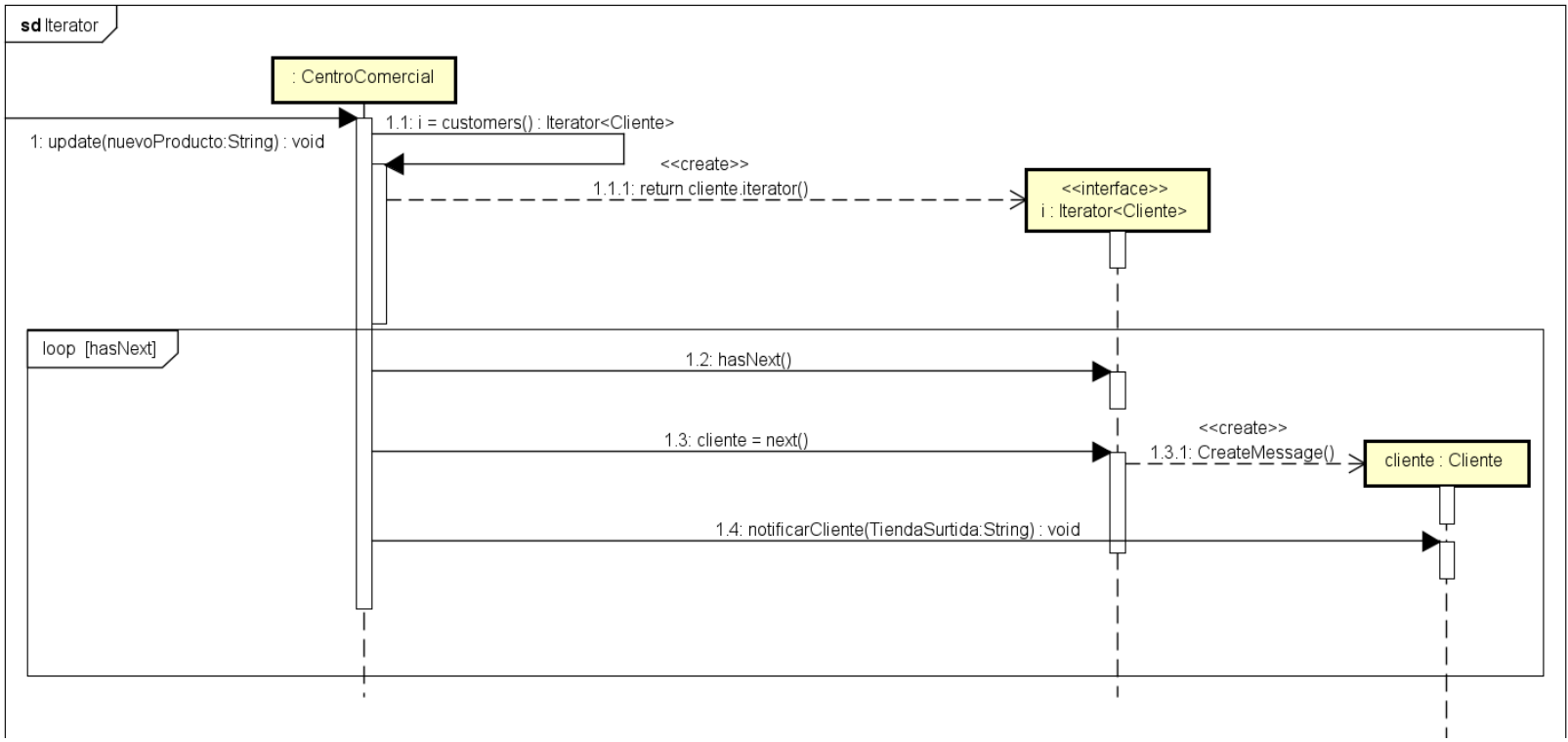
Paso 1.1: Ya que se está usando el patrón de Estados, el carrito de compras posee un atributo del estado actual, en este caso Vacío. Se llama a la función *addCarrito* de este estado.

Paso 1.1.1: El estado está encargado de finalmente agregar al carrito de compras el artículo, pero aparte, llama a la función *setEstado* en el carrito de compras, pasando un estado *ConAlgo*.

Iterator

Este patrón de diseño fue el más útil en el programa, ya que con este pudimos manejar la lista de artículos y paquetes con todas sus características disponibles y manejar sus datos a favor.

Lo implementamos para la creación y uso de las listas de artículos, clientes, paquetes y tiendas. Así que este patrón fue de los más aprovechados en el proyecto.



Paso 1: Se realiza la función *update* al centro comercial, para poder notificar a los clientes dentro del centro comercial acerca de los nuevos productos.

> **Paso 1.1**: Utilizamos la función customers para recibir un Iterador en los clientes presentes en el centro comercial.
>
> En el **paso 1.1.1** podemos notar como retorna este *Iterator* la función.

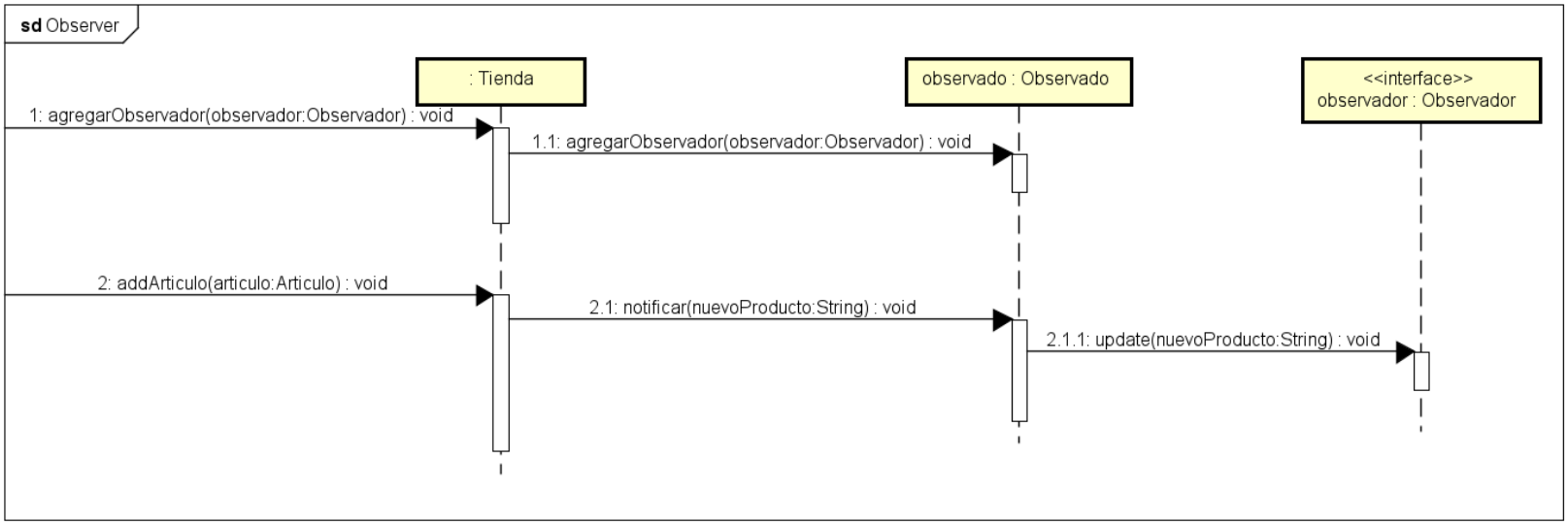
Se entra en un *loop*, que utiliza la función del **Paso 1.2** *hasNext*, que nos indica si hay o no un siguiente elemento en el iterador de clientes Mientras esto sea verdad**.**

Paso 1.3 A la variable cliente le asignamos la función *next* que devuelve el siguiente cliente en el iterador, como se pueda notar en el **paso 1.3.1**

Paso 1.4 En dicho cliente se utiliza la función *notificarCliente*, para hacerle llegar la notificación del producto nuevo al cliente.

Observer

La funcionalidad de este patrón fue que las tiendas fuesen observables y podamos notificar a los clientes sobre los nuevos productos, sin tener que entrar a la tienda.



Paso 1: Se le agrega un Observador a la clase Tienda para que este pueda ser notificado después. Se llama a la función *agregarObservador*..

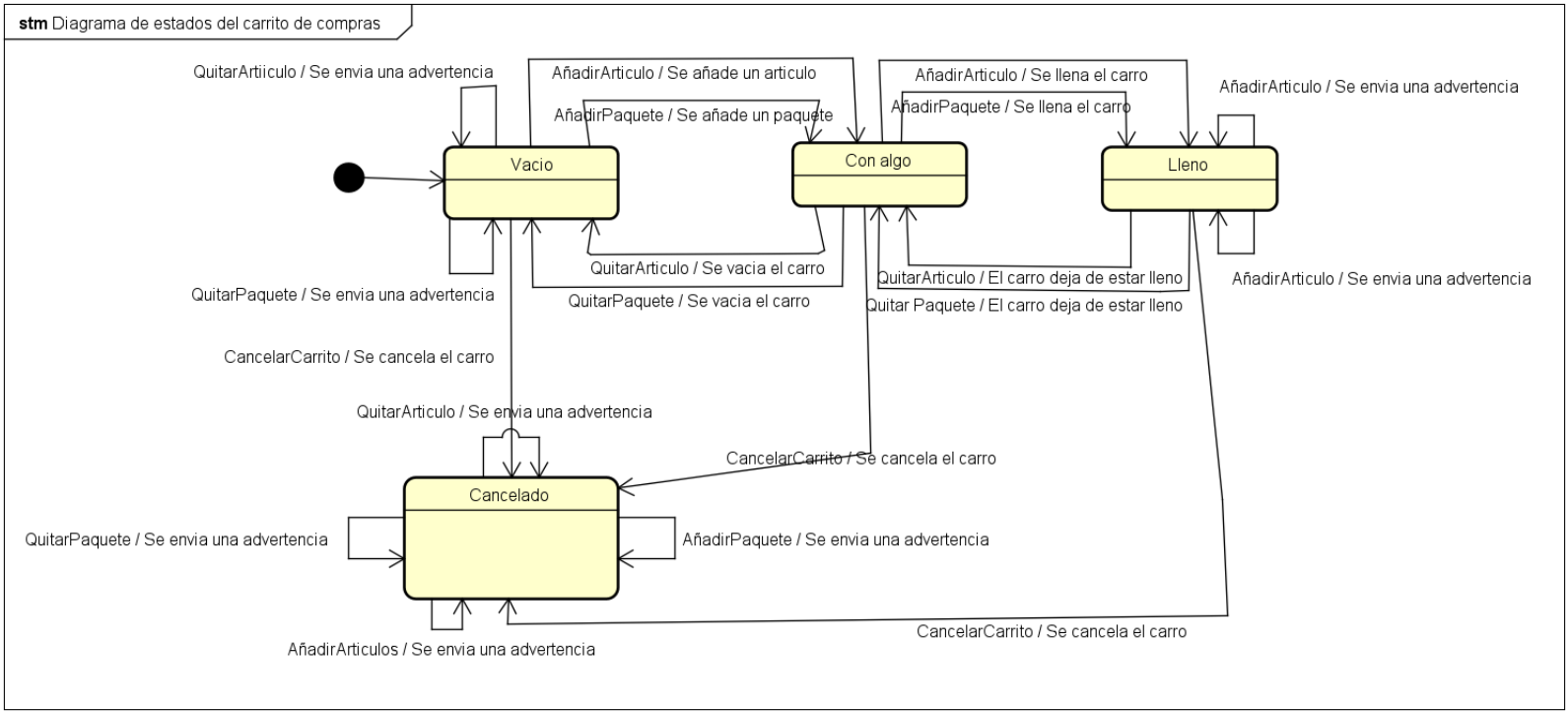
Paso 1.1: Esta función llama a la función *agregarObservador* del atributo Observado de la clase Tienda para ya finalmente agregar al observador a la lista de observadores.

Paso 2: Para notar dicho funcionamiento, se agrega un artículo a la tienda con la función *addArticulo*.

Paso 2.1: Aparte de agregar el artículo, esta función llama a la función notificar del atributo Observado para que este realice las notificaciones de articulo nuevo.

Paso 2.1.1: Esto lo hace llamando a la función *update* de cada Observador que posee.

Diagrama de Estado del Carrito de Compras



Como se puede observar, el carrito puede estar en 4 estados: **Vacío, Con algo, Lleno** o **cancelado**.

El estado que es el nexo entre el estado vacío y el lleno es el estado "con algo" ya que se estarían tomando casos donde el carro no está vacío, pero tampoco *lleno*; para pasar del estado vacío a con algo se debe tener al menos 1 producto, para pasar del estado lleno a con algo se debe remover o eliminar un producto. con respecto al estado cancelar se puede acceder a él desde cualquier estado, sin embargo, este representa un punto muerto ya que en caso de querer realizar alguna acción se emitirá una advertencia, existen otras restricciones, por ejemplo, estar en el estado lleno y querer agregar un nuevo producto ocasionaría que se emita una advertencia, este caso es similar a al estado vacío ya que si se intenta eliminar algo se emitirá una advertencia.