|  |  |
| --- | --- |
| Una imagen de una carretera de curvas con árboles  PATRONES DE DISEÑO  Implementación de los patrones de diseño. | Descripción breve  Este documento contiene las especificaciones sobre la implementación de los patrones de diseño solicitados para la elaboración del proyecto integrador para la asignatura de diseño de software.   * Eusebio Ajax Santos. * Emmanuel Chablé Collí. * Nicolás Canul Ibarra. |

Contenido

[Patrón de diseño: Decorator. 3](#_Toc42531457)

[Patrón de diseño: Iterator. 4](#_Toc42531458)

[Patrón de diseño: Factory. 5](#_Toc42531459)

[Patrón de diseño: Singleton. 6](#_Toc42531460)

[Patrón de diseño: State. 7](#_Toc42531461)

[Patrón de diseño: Observer. 8](#_Toc42531462)

# Patrón de diseño: Decorator.

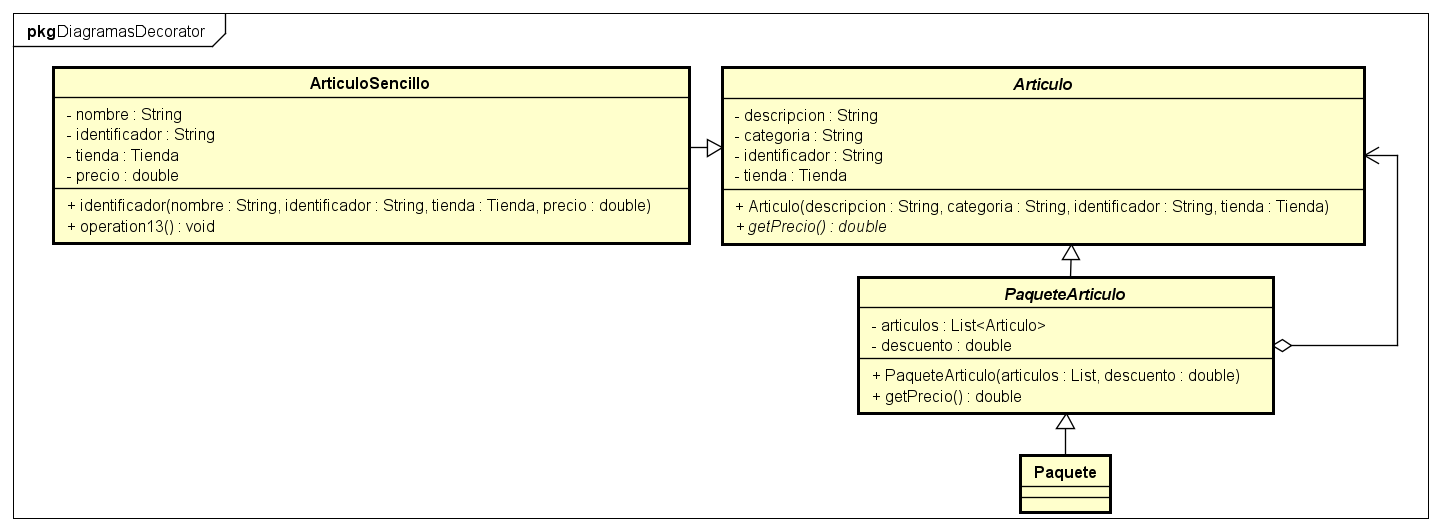
Diagrama de clases:

Ilustración 1. Diagrama de clases del patrón de diseño **Decorator**.

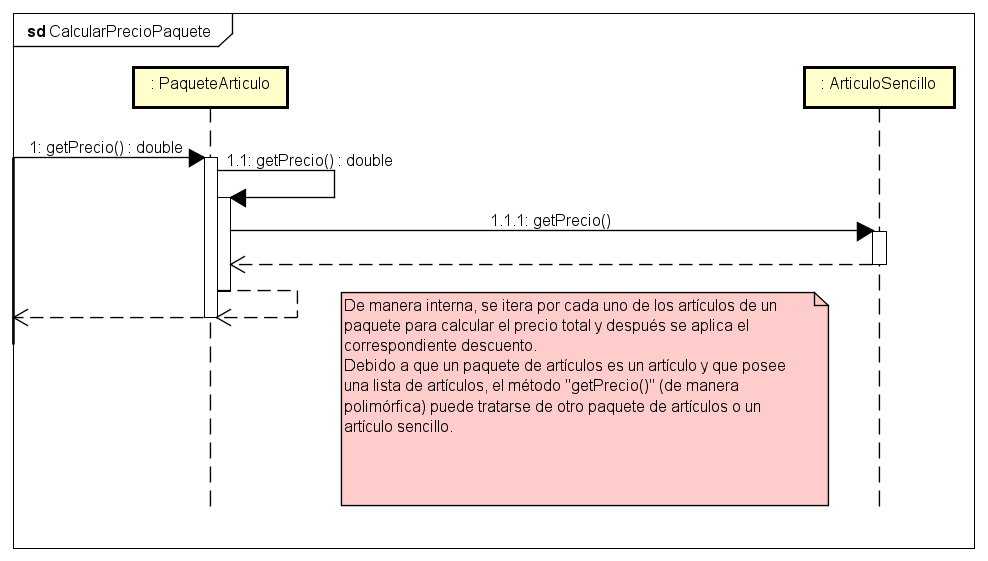
Diagrama de secuencia:

Ilustración 2. Diagrama de secuencia para el cálculo del precio de un paquete de artículos.

Observaciones:

Para esta situación, el uso del patrón **Decorator** nos es muy útil para crear paquetes de artículos e incluso paquetes de paquetes, ya que cada paquete es un artículo. De igual forma, el precio para un paquete se puede calcular de una manera muy sencilla siguiendo el diseño que este patrón nos ofrece.

# Patrón de diseño: Iterator.

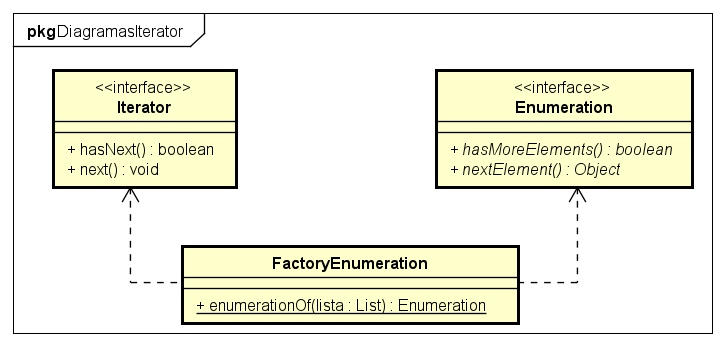
Diagrama de clases:

Ilustración 3. Diagrama de clases para el patrón de diseño **Iterator**.

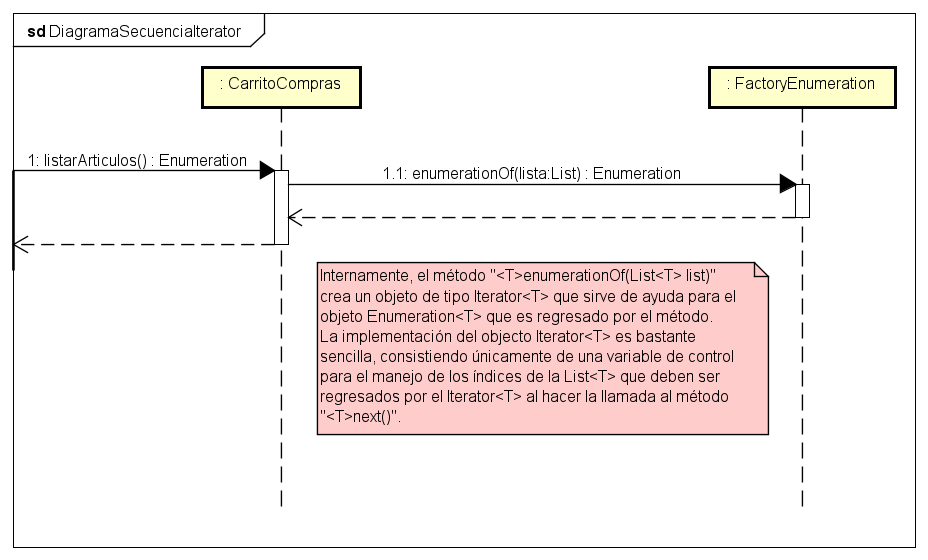
Diagrama de secuencia:

Ilustración 4. Diagrama de secuencia para la creación de objeto de tipo **Enumeration**<T> a partir de una List<T>.

Explicación:

Los objectos **Enumeration**<T> son utilizados para recorrer una colección de elementos en algún orden no especificado. Para nuestro caso, estos objectos se usan para iterar sobre algún conjunto de datos (clientes o artículos) y posteriormente ir mostrándolos en pantalla (tablas o listas).

# Patrón de diseño: Factory.

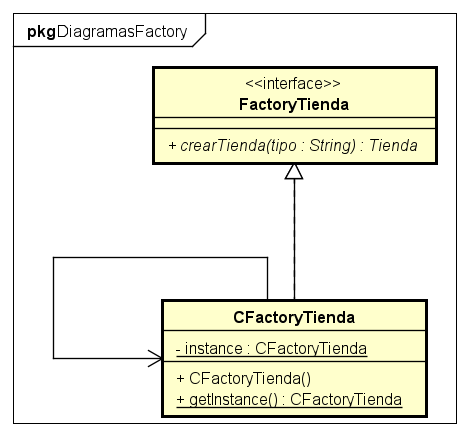
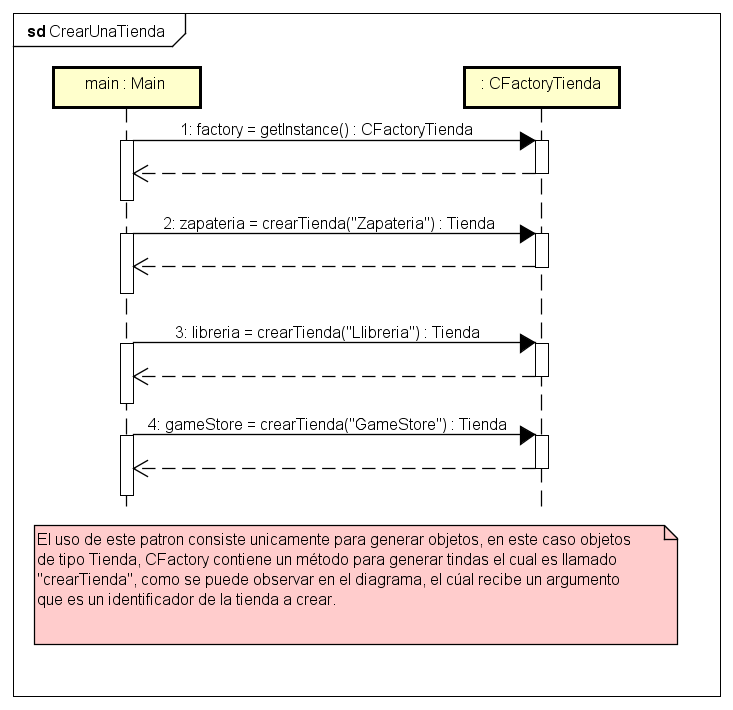
Diagrama de clases:

Diagrama de secuencia:



Explicación:

Por razones de practicidad este patrón es utilizado en el método main.

La clase CFactoryTienda nos permite generar tiendas sin necesidad de conocer cosas innecesarias, solo importa pasarle un identificador de alguna tienda, y este se encargará de crearla.

Esté patrón solo es utilizado en esos momentos, aunque en un futuro si se requiriera agregar nuevas tiendas, podría agregarse muy fácilmente.

# Patrón de diseño: Singleton.

Diagrama de clases:

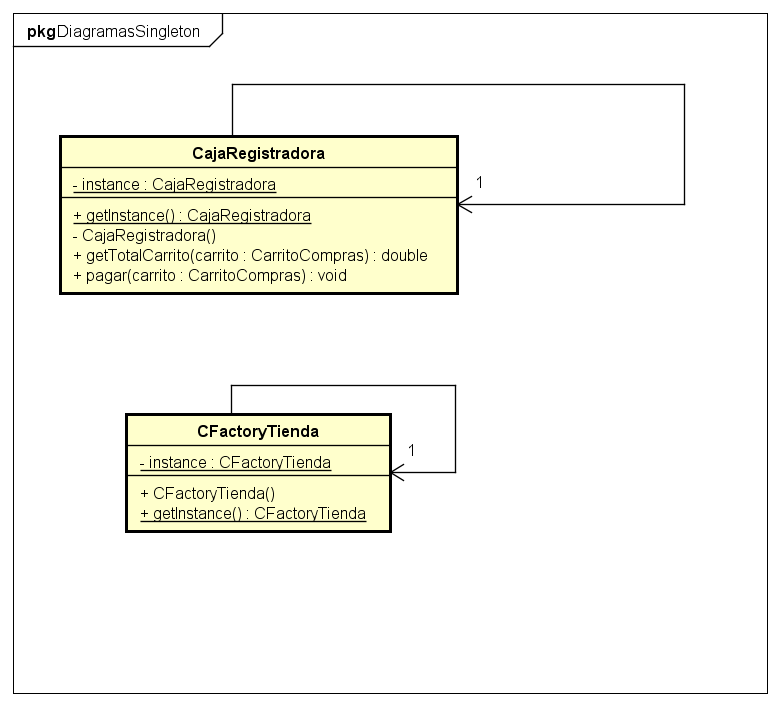
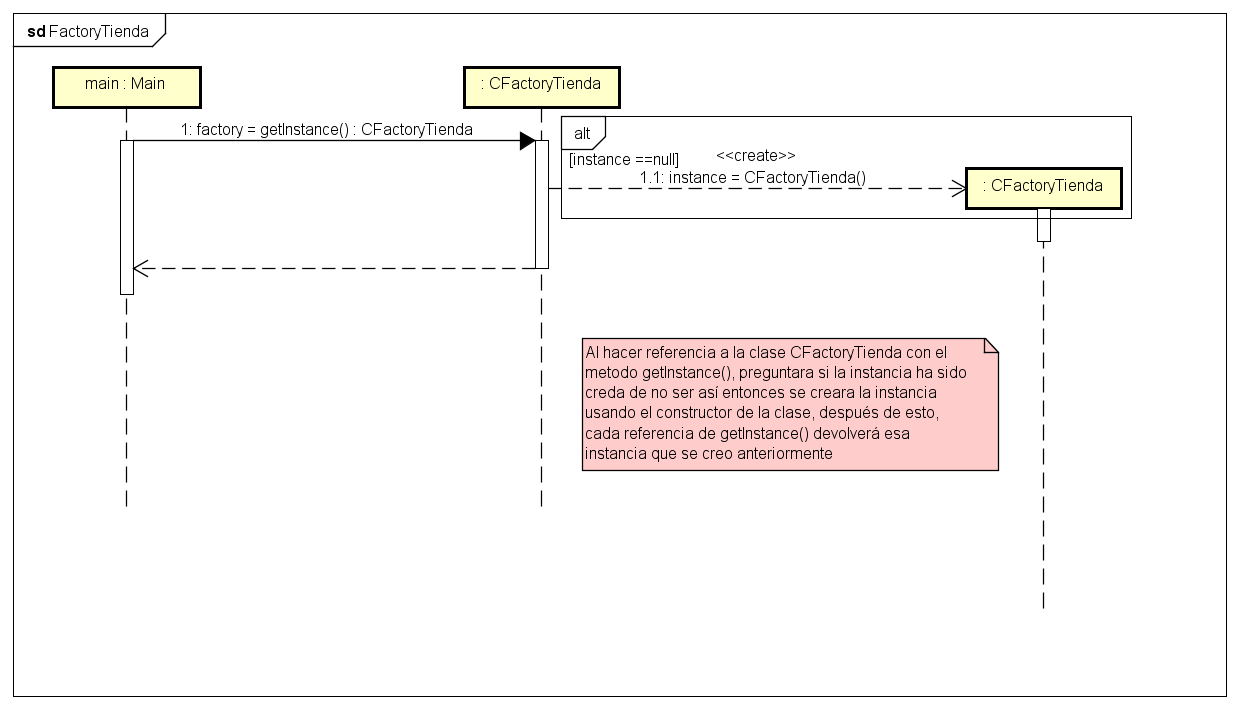


Diagrama de secuencia:



Explicación:

Este patrón es utilizado para generar una única instancia de las clases CFactory y CajaRegistradora de esta manera no se crearán más objetos de estas clases, ya que estas clases no necesitan ser modificadas porque brindan métodos que no necesitan cambios internos de su clase.

# Patrón de diseño: State.

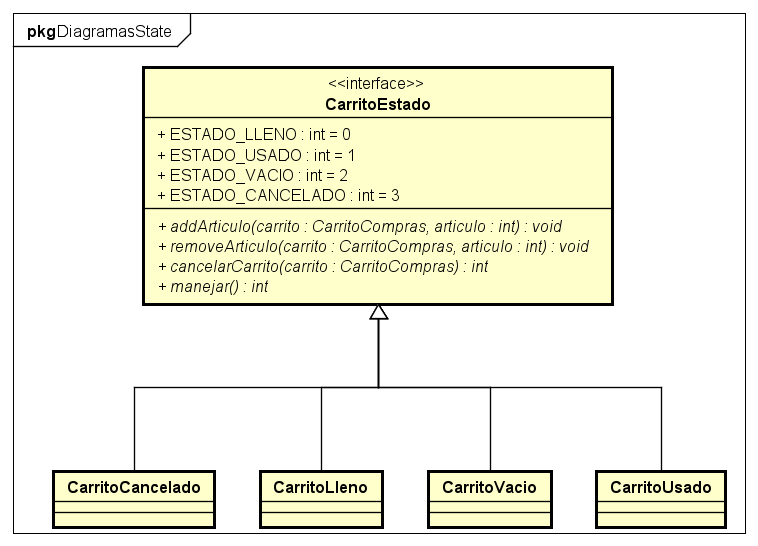
Diagrama de clases:

Diagrama de secuencia:

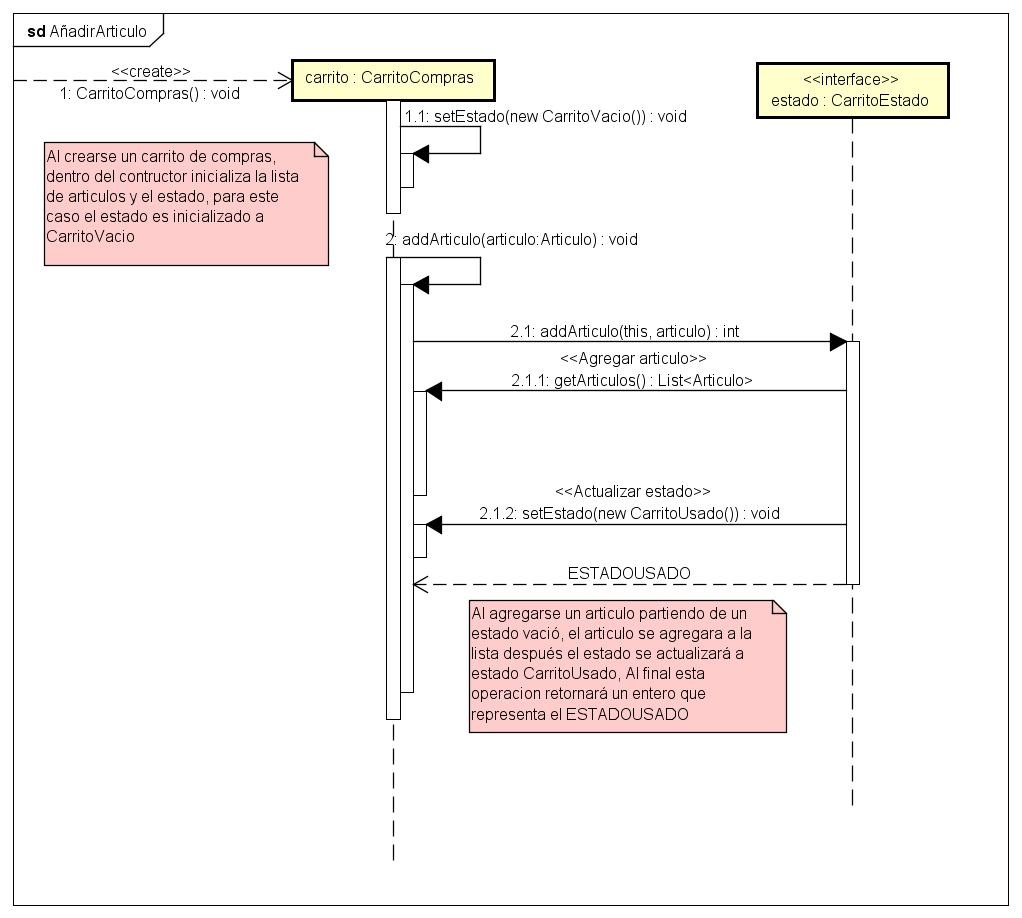
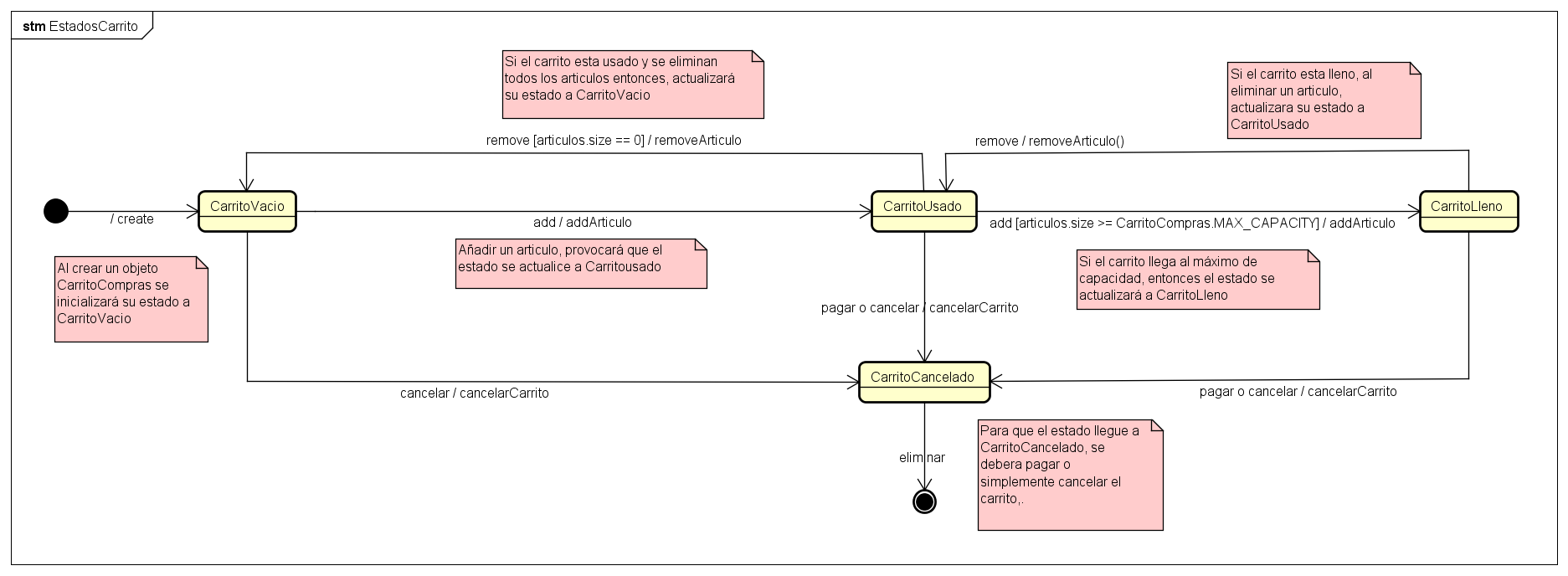


Diagrama de estados:



Explicación:

Este patrón es utilizado para la clase CarritoCompras el cuál nos ayuda a procesar los estados que el carrito tiene cuando se ejecuta una acción. Este patrón también es utilizado para la parte gráfica uniéndolo con el patrón Observer nos permite actualizar elementos visuales que representan al carrito.

# Patrón de diseño: Observer.

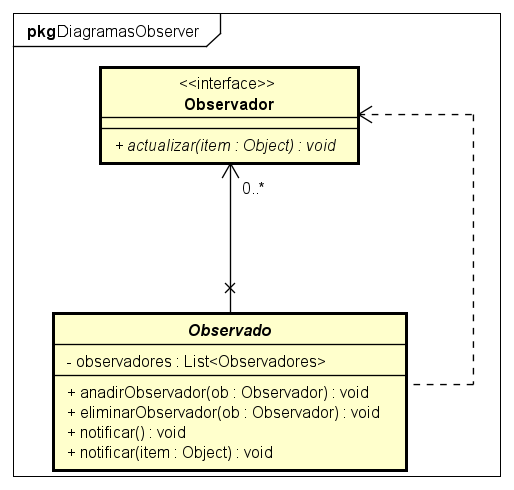
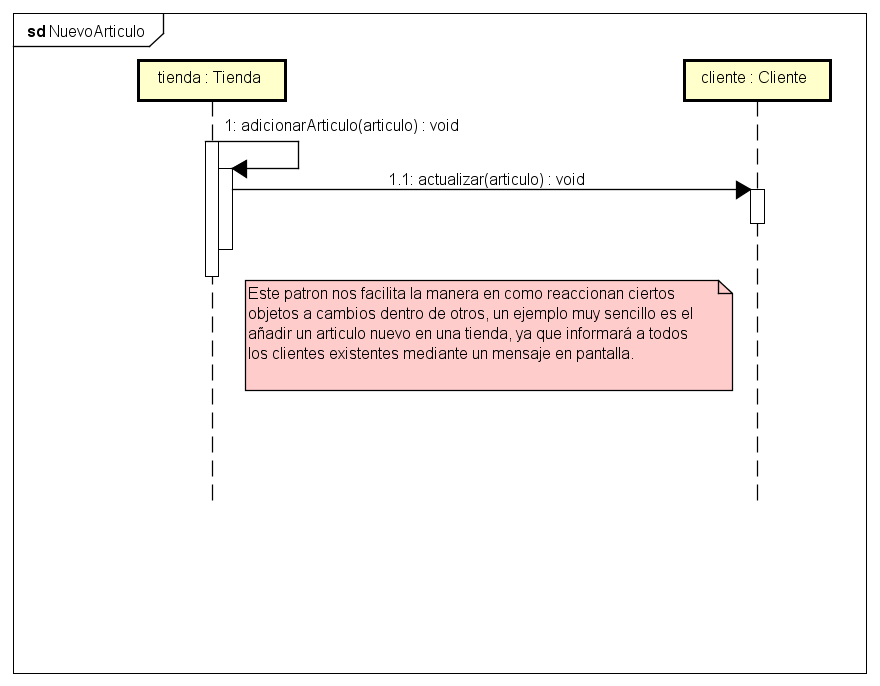
Diagrama de clases:

Diagrama de secuencia:



Explicación:

La manera de implementar este patrón es muy sencilla, debemos tener dos clases que estarán mutuamente relacionadas Observador y Observado

Observador es una interfaz que definirá métodos, los cuales se enfocarán en mostrar como una clase reaccionará ante las acciones de un Observado.

Observado es una clase abstracta que contiene objetos Observadores dentro de él, ya que mediante el un método notificara a los Observadores de algún cambio dentro de él.