

FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN

DISEÑO DE SOFTWARE

TALLER 6 PATRONES

HERRAMIENTAS: LUCIDCHART

INTEGRANTES

NATALIA RAMIREZ YEPEZ

LIZBETH PEÑA

KATHERINE MORALES TIERRA

PARALELO 102°

PAO-2021

Tabla de Contenido

[**Sección A**](#_tck4nd9clix8) **3**

[ITERATOR](#_uw95w2hh9i6u) 3

[CHAIN OF RESPONSIBILITY](#_ux20nbs1ja39) 3

[BRIDGE](#_6qyc8zx555ch) 4

[STRATEGY](#_ypqraq8x2abp) 4

[**Sección B**](#_fk75yjbrfvoj) **5**

[DIAGRAMA DE CLASES](#_h7tvkv3vji2w) 5

[ITERATOR](#_cgshw2nt8jfh) 6

[CLASES](#_u5l3qxw8jeao) 6

[SECUENCIA](#_cq3bqujpkmul) 6

[CHAIN OF RESPONSIBILITY](#_67b2xxg0joh7) 7

[CLASES](#_lumwjtphfbvr) 7

[SECUENCIA](#_3ygju2cospmn) 7

[BRIDGE](#_81cl5ugfu9gv) 8

[CLASES](#_upq8l3ymcmn) 8

[SECUENCIA](#_fckdnstmsla6) 8

[STRATEGY](#_ez9nrgh3eeqg) 9

[CLASES](#_gxqwjadvpjnq) 9

[SECUENCIA](#_fxlx68fxk747) 9

[**Sección C**](#_yae0lr6okch3) **10**

# Sección A

**Elabore un reporte en el que identifique los patrones de diseño aplicables para el sistema de software descrito. Para cada patrón que considere pertinente, debe indicar la motivación de su uso y las consecuencias de su decisión (ej.: pros, contras, relación con principios SOLID). Indique cualquier asunción que realice.**

## ITERATOR

Iterator es un patrón de diseño que permite recorrer elementos de una colección sin exponer su representación subyacente. Esto puede ser aplicado cuando los usuarios deben manejar entre los productos disponibles dependiendo de la necesidad.

El objetivo por el cual se aplica este patrón de diseño es debido que en la empresa todos los actores tienen una lista específica de artículos, cada vez que un comprador quiera ver los artículos disponibles, se deberá iterar entre los ítems actuales sin repetir o perder alguno en el camino, así mismo el proveedor con sus productos respectivos, etc.

Se recomienda este patrón de diseño porque permite mostrar un orden específico, para evitar que el usuario pierda su tiempo viendo contenido que no es de su interés.

## CHAIN OF RESPONSIBILITY

Chain of responsability es un patrón de diseño de comportamiento que permite pasar solicitudes a lo largo de una cadena de controladores; cada vez que se recibe una solicitud, cada uno de los controladores tiene la potestad de decidir si procesar dicha solicitud o pasarla al siguiente controlador de la cadena. Esto puede ser aplicado en el caso de los reembolsos o solicitud de cambio.

El objetivo por el cual se aplica este patrón de diseño es que cuando el comprador quiere solicitar un cambio debe acercarse a la tienda, donde hay una cadena de autoridades por la cual dicha solicitud debe pasar antes de que sea aprobada. Cada vez que se haga el cambio, se irá recorriendo la cadena para solicitar la aprobación necesaria por parte del controlador.

Debe usarse este patrón porque permite que varios objetos gestionen una solicitud sin la necesidad de acoplar la clase emisora a las clases concretas de los receptores. Lo mejor es que este patrón es escalable, por lo tanto cualquier problema en alguna parte del proceso, esto no afectará el comportamiento del resto.

## BRIDGE

Bridge es un patrón de diseño estructural que te permite dividir una clase grande, o un grupo de clases estrechamente relacionadas, en dos jerarquías separadas (abstracción e implementación) que pueden desarrollarse independientemente la una de la otra. Esto puede ser aplicado en la forma de distribución de artículos porque podemos extender la jerarquía de tipo de vehículo a el medio por el cual se moviliza.

El objetivo por el cual se recomienda aplicar este patrón es que el departamento de logística puede dividir sus opciones de transporte, los cuales tienen la misma funcionalidad, por lo tanto, usando este patrón se podrá dividir la clase monolítica en varias jerarquías de clase, de esta manera se podría simplificar el mantenimiento de código y se minimiza el riesgo de errores en el código actual.

Dentro de los beneficios obtenidos en el uso de este patrón es que se pueden crear clases independientes de la plataforma, además de que cumple con el principio de abierto/cerrado y se pueden introducir nuevas abstracciones e implementaciones independientes entre sí

## STRATEGY

Strategy o Estrategia es un patrón de diseño de comportamiento el cual consiste de familias de algoritmos las cuales pueden ser intercambiables en cualquier momento según la necesidad. Esto puede ser aplicado para la parte del departamento de logística la cual se encarga del transporte de los productos. Se especifica que habrán diferentes estrategias en un futuro y que estas deberían ser intercambiables además de ser escogidas en tiempo de ejecución, lo cual cumple la descripción de este patrón.

El usarlo ayuda a eliminar sentencias condicionales, factorizar funcionalidad común de los algoritmos.

Es una alternativa a las subclases por su uso del encapsulamiento de los algoritmos en clases separadas. También aumenta el número de objetos.

Al haber mucha diversidad de estrategias para un tipo de comportamiento el cliente debe comprender muy bien cómo funcionan cada una de ellas de antemano. Esto en ciertos casos puede ser una desventaja.

# Sección B

**Elabore los diagramas de clases y de secuencia para el sistema de software descrito en el que se reflejen los patrones de diseño escogidos y la escalabilidad de su solución. Presente el diagrama de clases de cada patrón en un paquete separado. Identifique herencias, multiplicidades, visibilidad de atributos y métodos relevantes para el patrón. Por medio de notación UML, muestre si las entidades corresponden a interfaces, clases abstractas o clases concretas. Además, muestre el uso de mensajes síncronos, asíncronos y de retorno que sean pertinentes. Indique cualquier asunción que realice.**

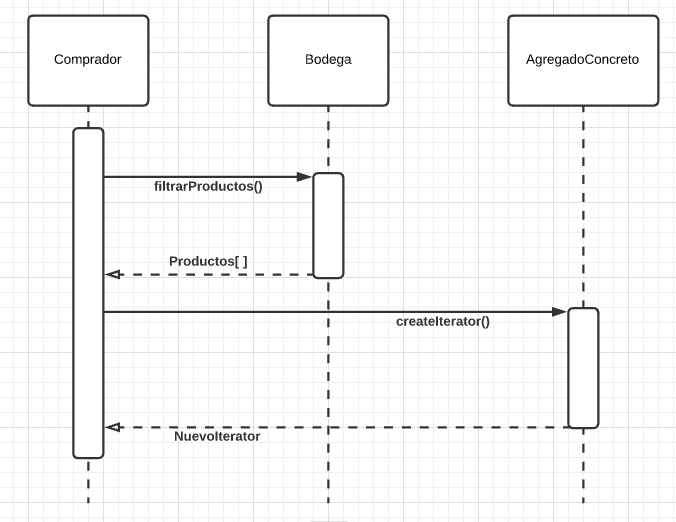
## DIAGRAMA DE CLASES

## ITERATOR

### CLASES

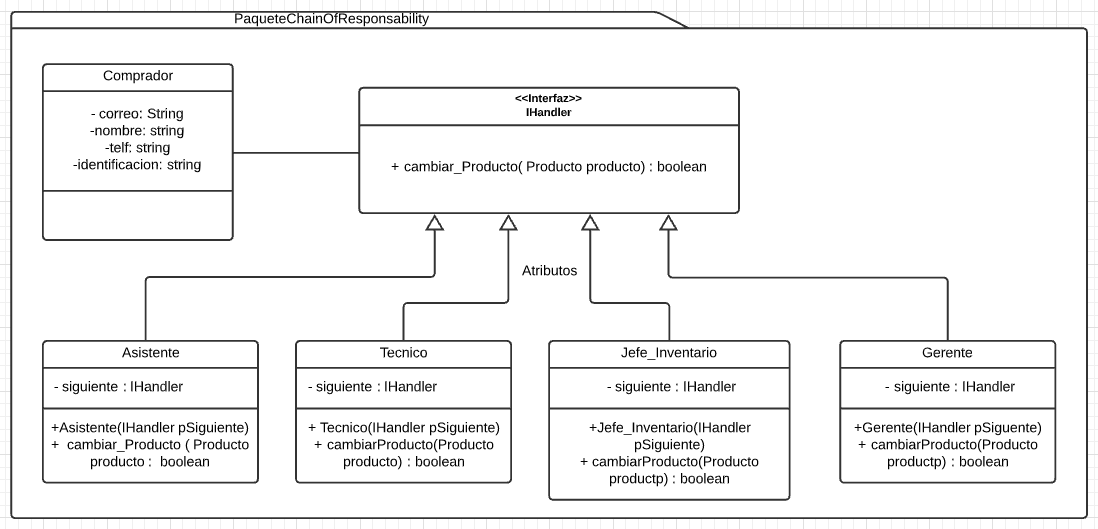
### 

### SECUENCIA

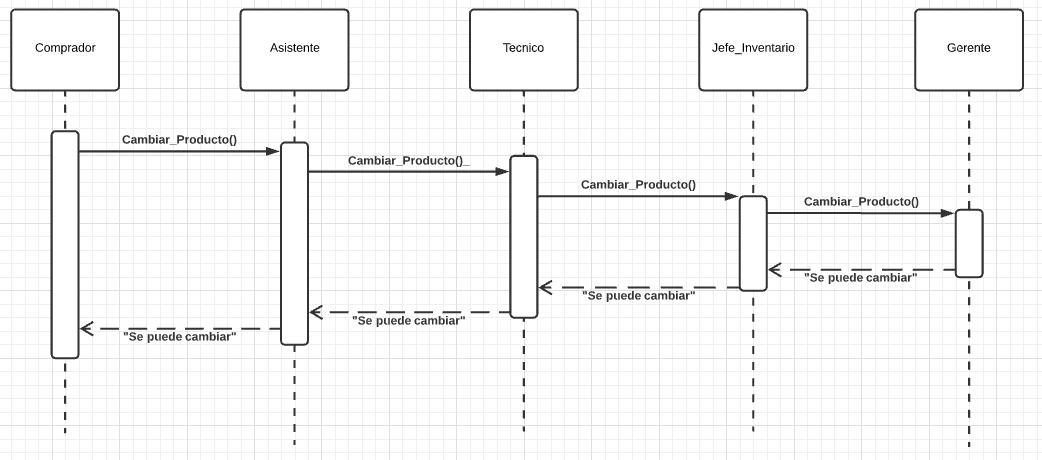


## CHAIN OF RESPONSIBILITY

### CLASES

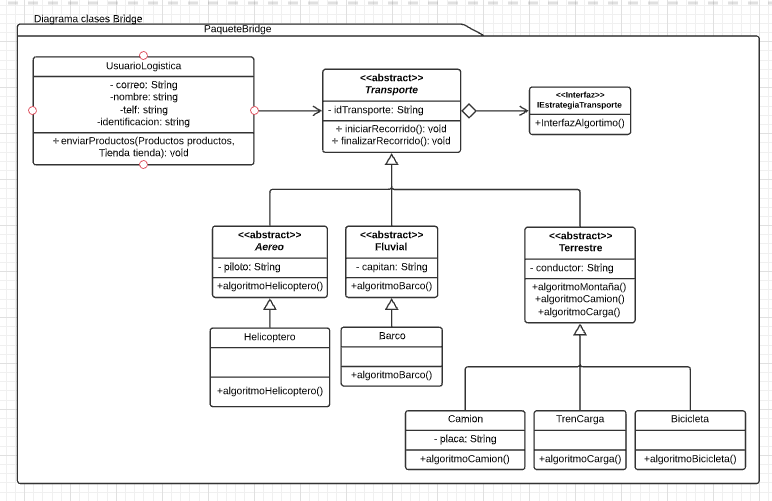


### SECUENCIA

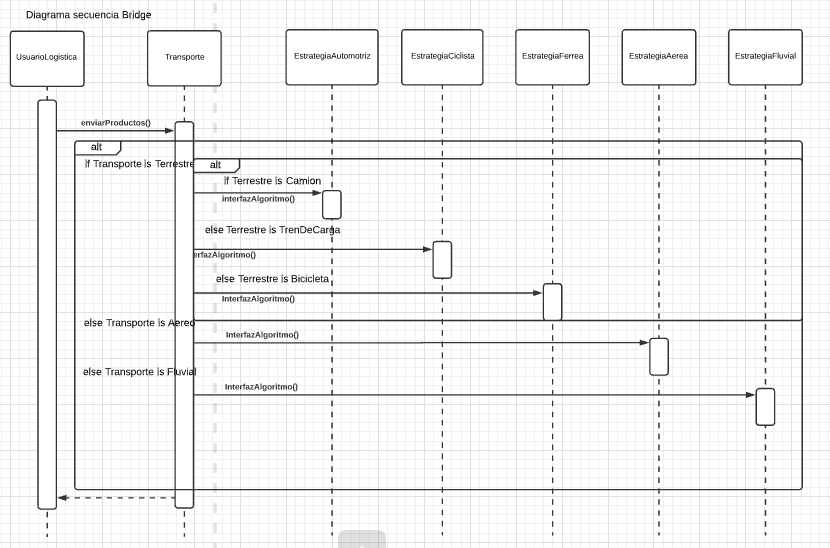


## BRIDGE

### CLASES

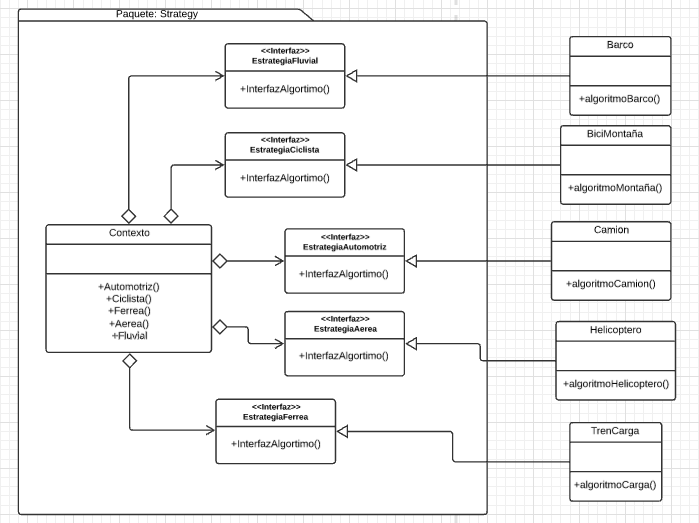


### SECUENCIA

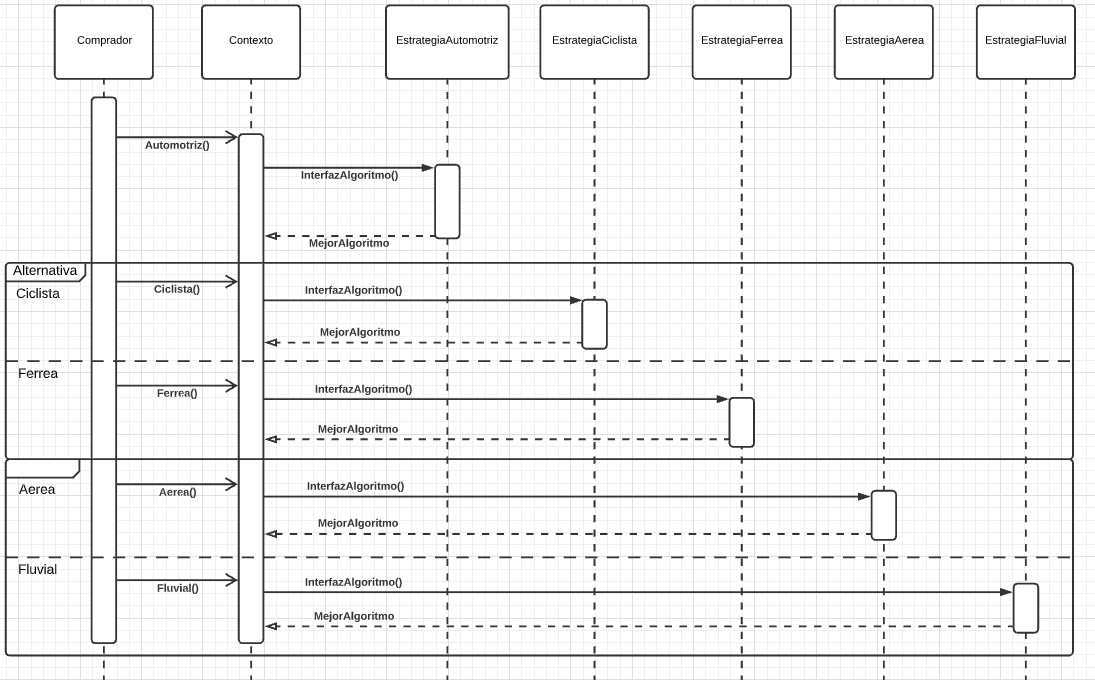


## STRATEGY

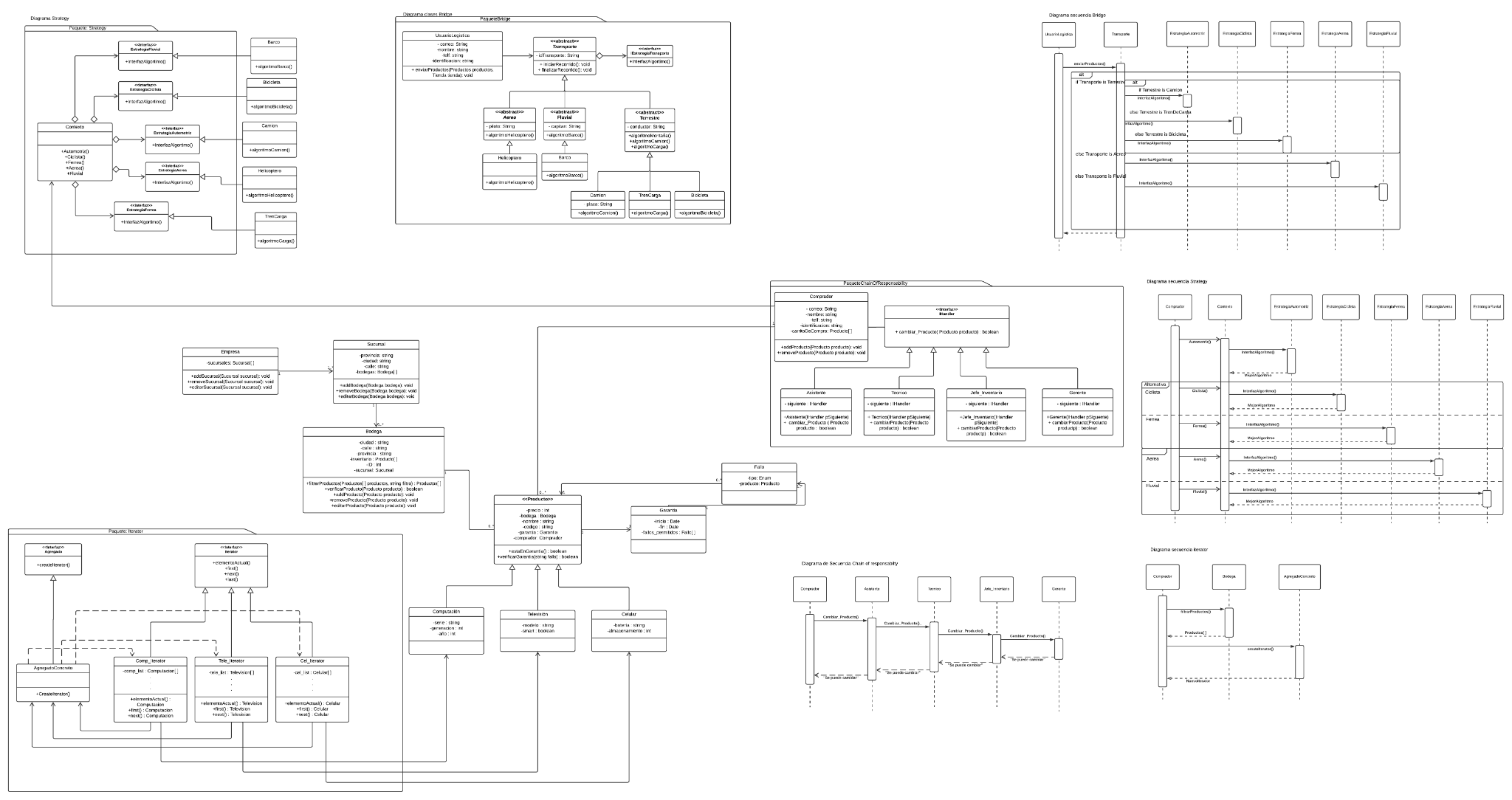
### CLASES



### SECUENCIA



**DIAGRAMA GENERAL**



# Sección C

**Implemente en Java las clases relevantes que ilustren la aplicación de patrones de su diseño en su solución.**

Link del repositorio en github: <https://github.com/Lizz25/TallerPatrones.git>