**ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

**FACULTAD DE INGENIERÍA EN ELECTRICIDAD Y COMPUTACIÓN**

**DISEÑO DE SOFTWARE**

TALLER DE PATRONES DE DISEÑO y UML

**EQUIPO 6**

MIGUEL ANGEL MISE CHICA

JHORDANA LISSETTE MIRANDA VIVANCO

GEORGE SEBASTIAN YAGUANA MENDOZA

**CONTENIDO**

[**Sección A**](#_mtoegrx9hfj) **3**

[Identificación de Patrones](#_74b2a1prm96n) 3

[**Sección B**](#_4erevrm7niqw) **5**

[Diagramas de clases y secuencias](#_k5xr8ab1xpe) 5

[Iterator Diagrama de Clases.](#_9ixhcc8e6zr9) 5

[Iterator Diagrama de Secuencia.](#_hrguj040uyw9) 5

[Strategy Diagrama de Clases.](#_38rr29gazhwc) 6

[Strategy Diagrama de Secuencia.](#_356sy1vd490s) 6

[Chain Responsibility Diagrama de Clases.](#_j2z73an31l) 7

[Chain Responsibility de Secuencia.](#_y5cit6ccmqta) 7

[**Sección C**](#_v3oy0xinqpt8) **8**

[Implementación](#_qou4dlbqg3g8) 8

# Sección A

## Identificación de Patrones

**Párrafo 1:** Considere una empresa, con alcance a nivel nacional, de venta de artículos de tecnología - de computación, impresión, televisión, video, celulares, tabletas, entre otros. Todos estos productos se almacenan en las bodegas de las tiendas a nivel nacional. El software debe permitir que diversos actores puedan conocer el inventario de un producto dependiendo de su necesidad específica. Por ejemplo, a un comprador Web le interesa navegar a través de productos disponibles; al vendedor de una tienda, navegar entre productos disponibles en su tienda; a un proveedor, navegar entre productos que han presentado fallas dentro del periodo de garantía; al asistente de mantenimiento, navegar entre productos en reparación, etc.

**Patrón identificado a utilizar**: Iterator e Iterator

**Motivación para usar este patrón:** En el problema se necesita recorrer una lista de productos de distintas formas dependiendo de los privilegios de acceso del usuario puede revisar los productos en un orden específico, el patrón que permite realizar esto es el Iterator ya que permite recorrer elementos de una colección d una forma especifica indicada en cada ejecución.

**Beneficios de usar este patrón**: Utilizando el patrón Iterator podremos reducir la duplicación en el código de recorrido de la lista de artículos tecnológicos a lo largo de la ejecución, y lo podemos hacer transparente para el usuario, además si en el futuro se agregan nuevos tipos de colecciones seguirá funcionando ya que depende de interfaces. Este patrón de diseño permite aplicar el principio abierto/cerrado y el de responsabilidad única.

**Párrafo 2:** El departamento de logística de la empresa distribuye los artículos entre sus tiendas a nivel nacional.Actualmente, transporta sus artículos únicamente por vía automotriz; sin embargo, en un futuro cercano podría empezar su distribución también por vía ciclista, férrea, aérea y fluvial. Las estrategias de distribución son intercambiables y seleccionadas en tiempo de ejecución.

**Patrón identificado a utilizar**: Strategy

**Motivación para usar este patrón:** Dado que se plantea una distribución de productos por diferentes transportes ciclista, férrea, aérea y fluvial se escoge este patrón de diseño porque permitirá cambiar su comportamiento en tiempo de ejecución por estrategias concretas.

**Beneficios de usar este patrón**: Al utilizar este patrón ayuda a implementar distintos medios de transporte que distribuyen productos. Dicho patrón nos permite aislar detalles de implementación del código que lo utiliza, además se reemplaza la herencia por composición. Y si se quisiera agregar nuevas estrategias no se tendría que cambiar el contexto ya que está relacionado con el principio abierto/cerrado.

**Párrafo 3:** En caso de inconformidad con algún artículo por parte de un comprador, este se puede acercar a la tienda a solicitar un cambio. Primero un asistente del departamento de atención al usuario debe verificar que el producto se encuentra en periodo de garantía; si es así, un miembro del departamento técnico constatar que el tipo de fallo se encuentre dentro de lo cubierto por la garantía; a continuación, el jefe de inventario debe certificar la existencia del producto en bodega para reposición. Si el precio de venta del producto

supera los 1000 USD, se requiere además la aprobación del gerente de la tienda. Los responsables de las actividades podrían cambiar en cualquier momento y el software debería ser lo suficientemente flexible para lidiar con ello.

**Patrón identificado a utilizar**: Chain Responsibility

**Motivación para usar este patrón:** Dado que es un conjunto de pasos que se dan en forma de cadena en tiempo de ejecución el patrón antes mencionado cumple con la función de hacer escalable y mantenible al programa.

**Beneficios de usar este patrón**:

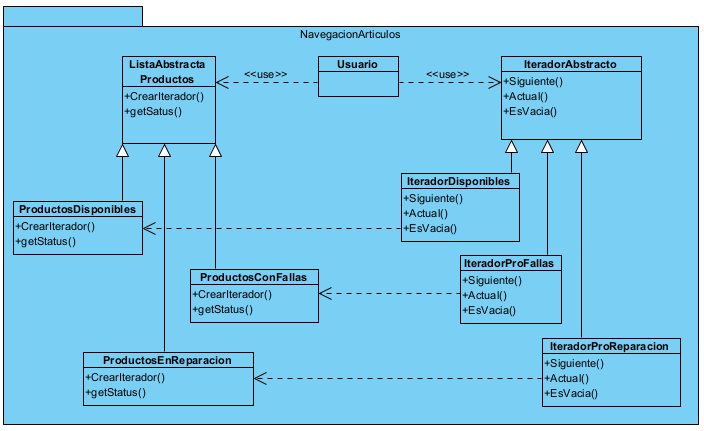
Single Responsability es un punto fundamental dado que el nivel de acoplamiento y cohesión es óptimo al dividir las responsabilidades en una clase además de que dependen enteramente de una interfaz de esta manera cumpliendo el principio Inverse dependency , en conclusión se logra crear un software mantenible y escalable con un trade off entre cohesión y acoplamiento además que el usuario instancia la interfaz que a su vez se implementa en clases concretas para su uso futuro ( también se pueden cambiar la interacción entre manejadores).

# 

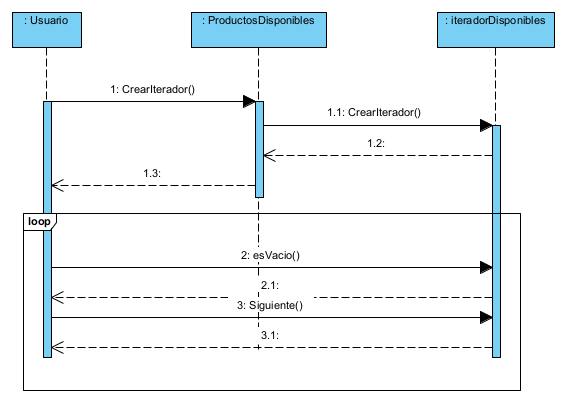
# Sección B

## Diagramas de clases y secuencias

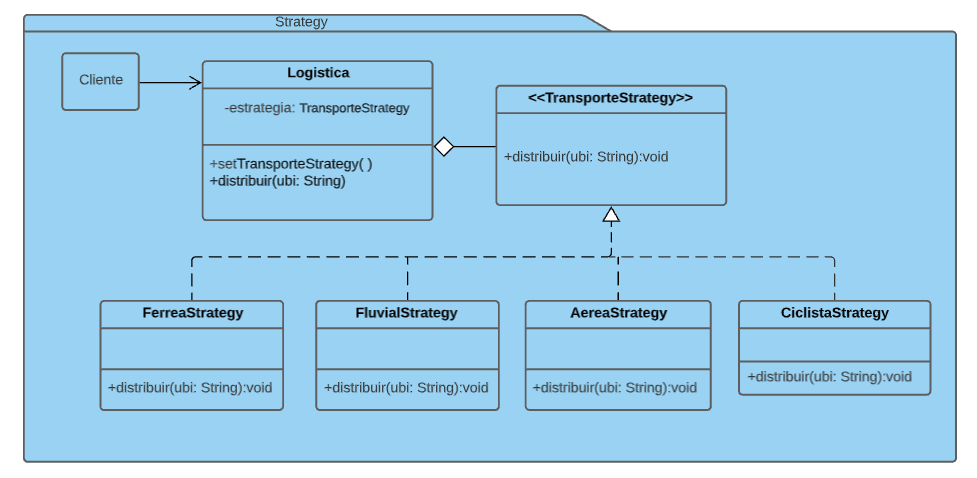
### Iterator Diagrama de Clases.



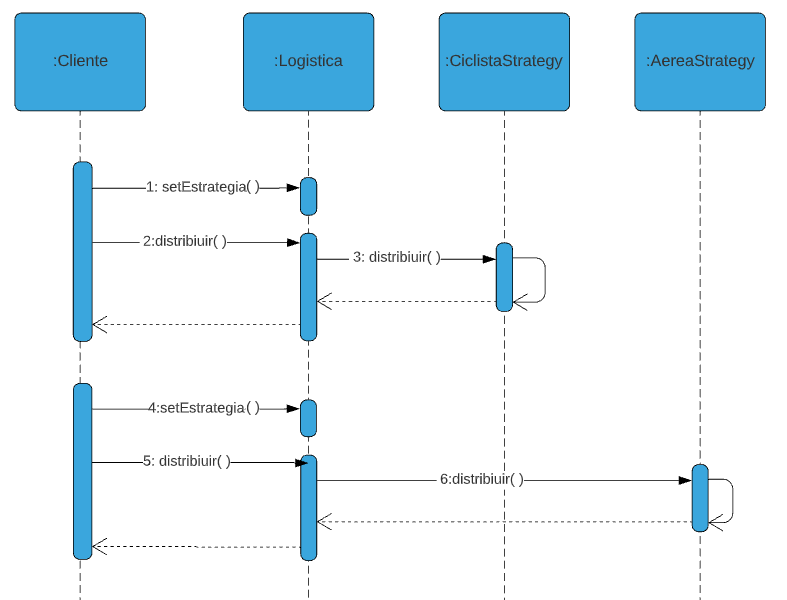
### Iterator Diagrama de Secuencia.



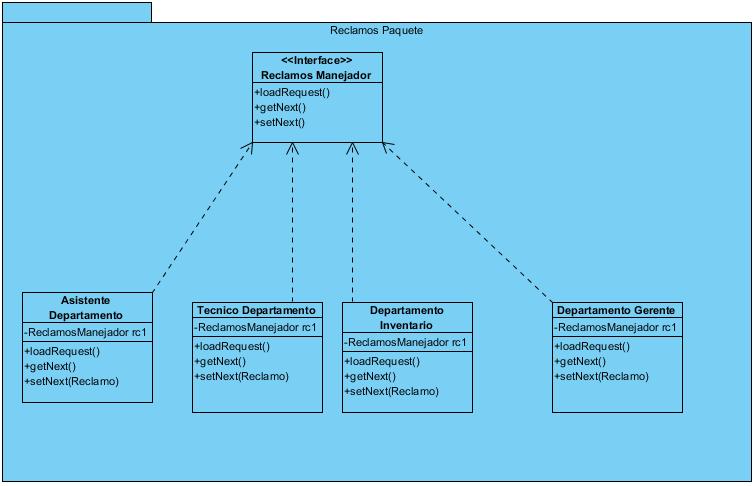
### Strategy Diagrama de Clases.



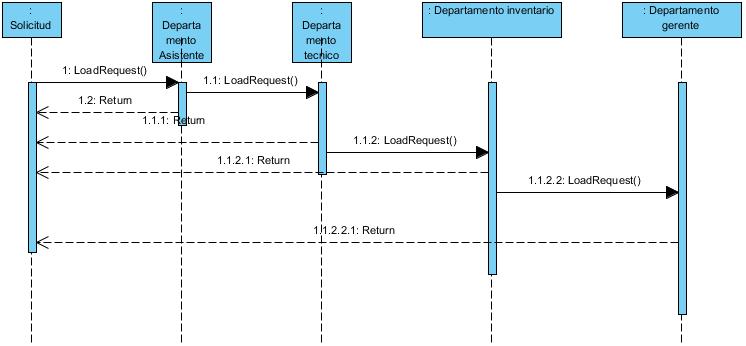
### Strategy Diagrama de Secuencia.



### Chain Responsibility Diagrama de Clases.



### Chain Responsibility de Secuencia.



# 

# Sección C

## Implementación **Iterator**

package NavegacionArticulos.NavegacionArticulos;

public class IteradorAbstracto {

public void Siguiente() {

// TODO - implement Iterador.Siguiente

}

public void Actual() {

// TODO - implement Iterador.Actual

}

public void EsVacia() {

// TODO - implement Iterador.EsVacia

}

public class ListaAbstractaProductos {

public void CrearIterador() {

// TODO - implement ListaProductos.CrearIterador

}

public void getSatus() {

// TODO - implement ListaProductos.getSatus

}

}

public class ProductosDisponibles extends ListaAbstractaProductos {

public void CrearIterador() {

// TODO - implement ProductosDisponibles.CrearIterador

}

public void getStatus() {

// TODO - implement ProductosDisponibles.getStatus

}

}

public class IteradorDisponibles extends IteradorAbstracto {

public void Siguiente() {

// TODO - implement IteradorDisponibles.Siguiente

}

public void Actual() {

// TODO - implement IteradorDisponibles.Actual

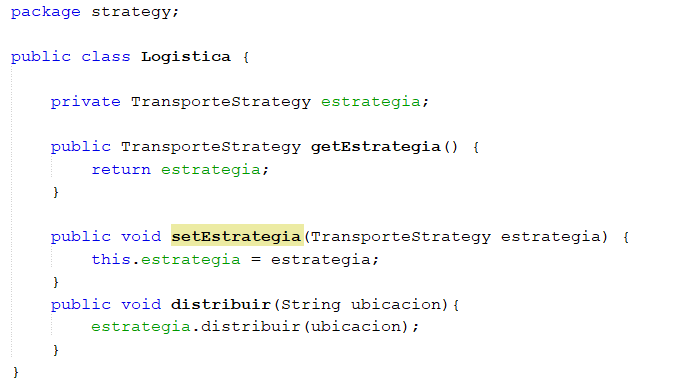
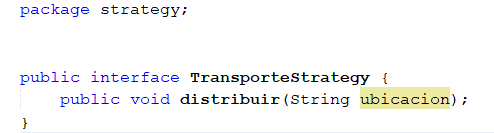
}

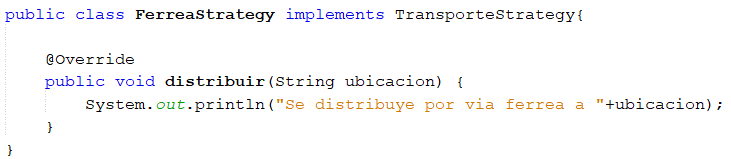
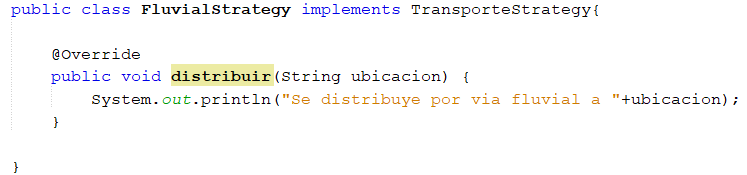
public void EsVacia() {

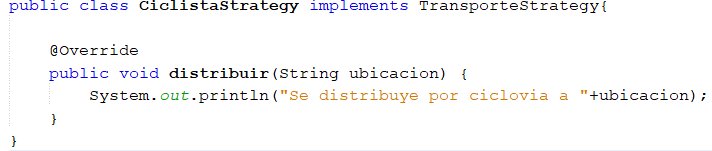
// TODO - implement IteradorDisponibles.EsVacia

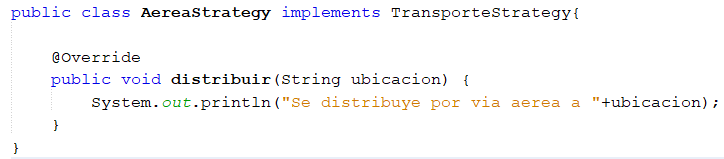
}}

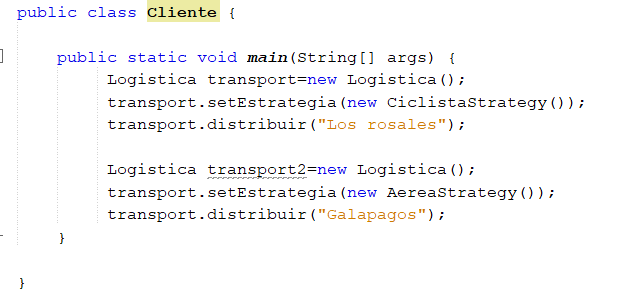
Implementación **Strategy**











Implementación **Chain Responsibility**

package Reclamos\_Paquete;

public interface Reclamos\_Manejador {

public void loadRequest();

public void getNext();

public void setNext();

}

package Reclamos\_Paquete;

public class Tecnico\_Departamento implements Reclamos\_Manejador{

private Object \_reclamosManejador\_rc1;

public void loadRequest() {

throw new UnsupportedOperationException();

}

public void getNext() {

throw new UnsupportedOperationException();

}

public void setNext(Object aReclamo) {

throw new UnsupportedOperationException();

}

}

public class Departamento\_Gerente implements Reclamos\_Manejador {

private Object \_reclamosManejador\_rc1;

public void loadRequest() {

throw new UnsupportedOperationException();

}

public void getNext() {

throw new UnsupportedOperationException();

}

public void setNext(Object aReclamo) {

throw new UnsupportedOperationException();

}

}

public class Departamento\_Inventario implements Reclamos\_Manejador{

private Object \_reclamosManejador\_rc1;

public void loadRequest() {

throw new UnsupportedOperationException();

}

public void getNext() {

throw new UnsupportedOperationException();

}

public void setNext(Object aReclamo) {

throw new UnsupportedOperationException();

}

}

public class Asistente\_Departamento implements Reclamos\_Manejador{

private Object \_reclamosManejador\_rc1;

public void loadRequest() {

throw new UnsupportedOperationException();

}

public void getNext() {

throw new UnsupportedOperationException();

}

public void setNext(Object aReclamo) {

throw new UnsupportedOperationException();

}

}

Herramientas que se utilizó para diagramas: Visual Paradigm y Lucidchart

**Repositorio Github**: https://github.com/miguelmise/PatronesDisenoUMLTaller.git