
# GRUPO #2

1. Kevin Alejandro Castro Ortiz
2. Steven Ariel Chóez Villacis
3. Erick Stalyn Lino Reyes
4. Juan Francisco Quimí Granados

## TALLER 6 – PATRONES DE DISEÑO Y UML

**Paralelo. -** 2

**Profesor. -** Jurado Mosquera David Alonso

**Fecha De Presentación. -**

22/7/2021

**Guayaquil – Ecuador**

**Contenido**



**Herramienta utilizada: LucidChart**
**URL del trabajo en LucidChart: https://lucid.app/lucidchart/invitations/accept/inv_60d2e6a5-8872-4a6e-bbca-333f240e7817**

**Sección A: Identificar los patrones de diseño aplicables al sistema de software**

**Patrón: Facade**

**Motivación.- Minimizar las comunicaciones y dependencias entre sistemas.**

**Consecuencias.- Reduce la cantidad de objetos con los que interactúa el cliente, pero no evita que estos usen las clases del subsistema.**

**Relación con SOLID.- Principio de segregación de interfaces, ya que hacemos interfaces específicas para cada situación que se nos presenta, con sus requerimientos y accesos; mas no una general.**

**Patrón: Factory Method**

**Motivación.- Nos permite optimizar el proceso de instanciación de objetos al provocar que las subclases especifiquen los objetos que crea la clase principal.**

**Consecuencias.- Facilita la escalabilidad del sistema. Permite el ingreso sencillo de nuevas vías de distribución.**

**Relación con SOLID.- Al asignar la única responsabilidad de crear nuevas instancias de vías de distribución al creator se aplica el Single Responsability Principle. De igual forma, debido a que las dependencias entre componentes se encontrarán invertidas se cumple con el Dependency Inversion Principle y no es necesario decidir qué subtipo de objeto constructor instanciar.**

**Patrón: Cadena de responsabilidad**

**Motivación.- Nos permite llevar un control, contamos con manejadores que se hacen cargo de comprobar si se procesa o no la solicitud.**

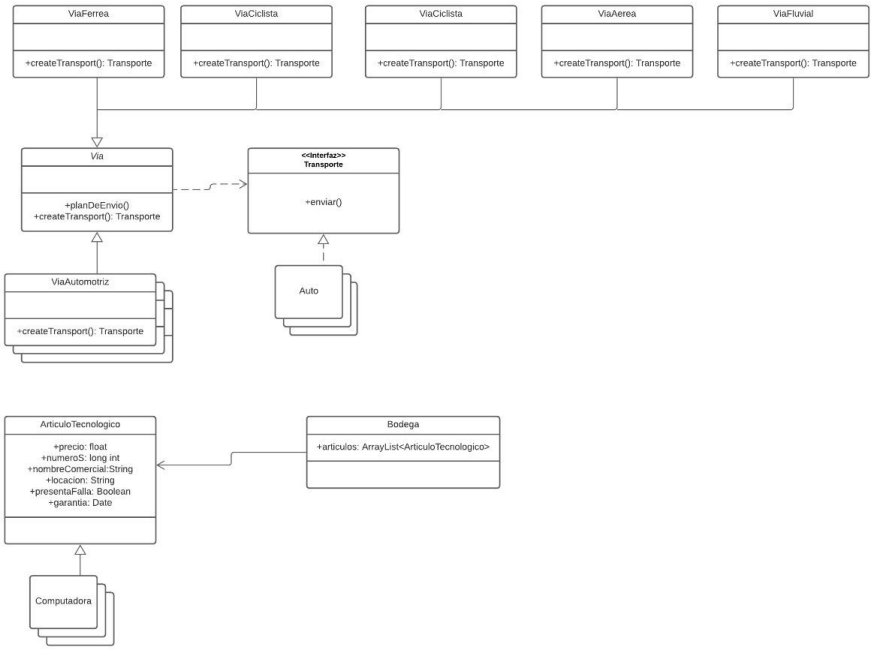
**Consecuencias.- Algunas solicitudes no llegan a ser gestionadas.**

**Relación con SOLID.- Cada manejador posee un único método con el que realiza la comprobación de la solicitud (Single Responsability Pinciple). Se pueden añadir más manejadores sin descomponer el codigo (Open/Closed Principle)**

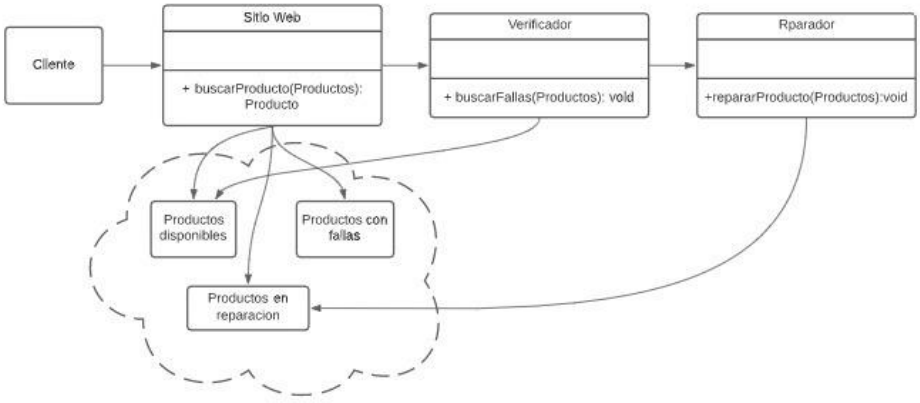
## Sección B: Diagramas

### Diagrama de clases

Factory Method




Facade



Chain of responsability

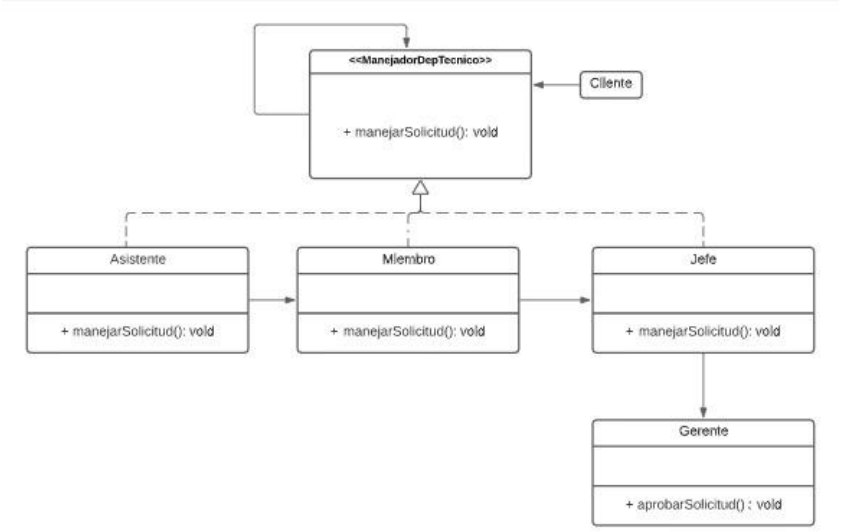
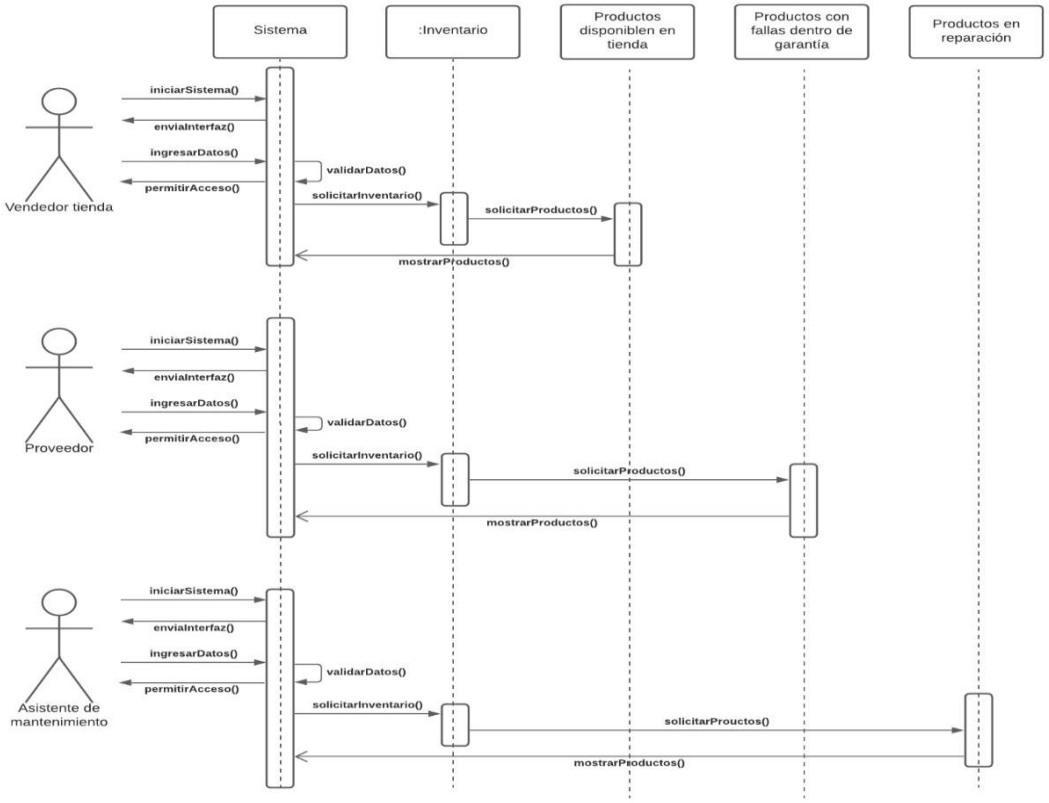
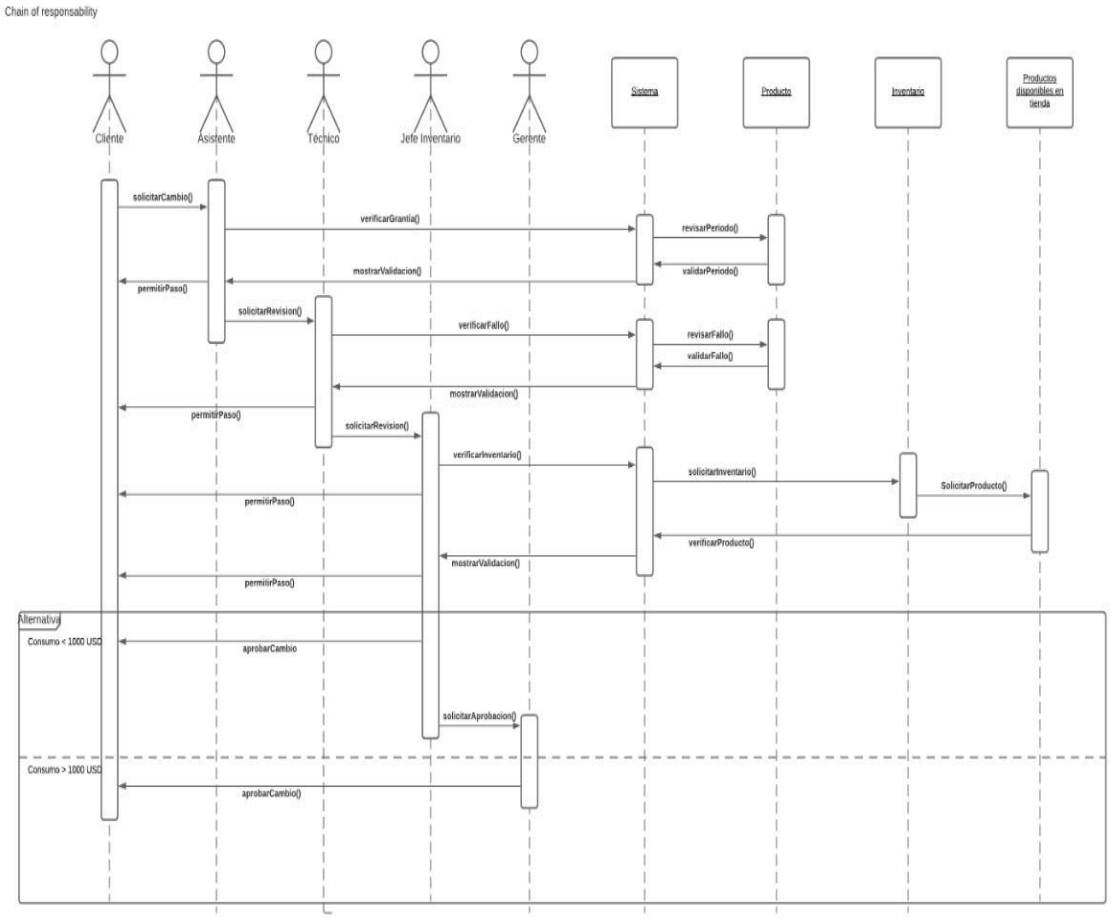



**Diagrama de secuencia**

Facade

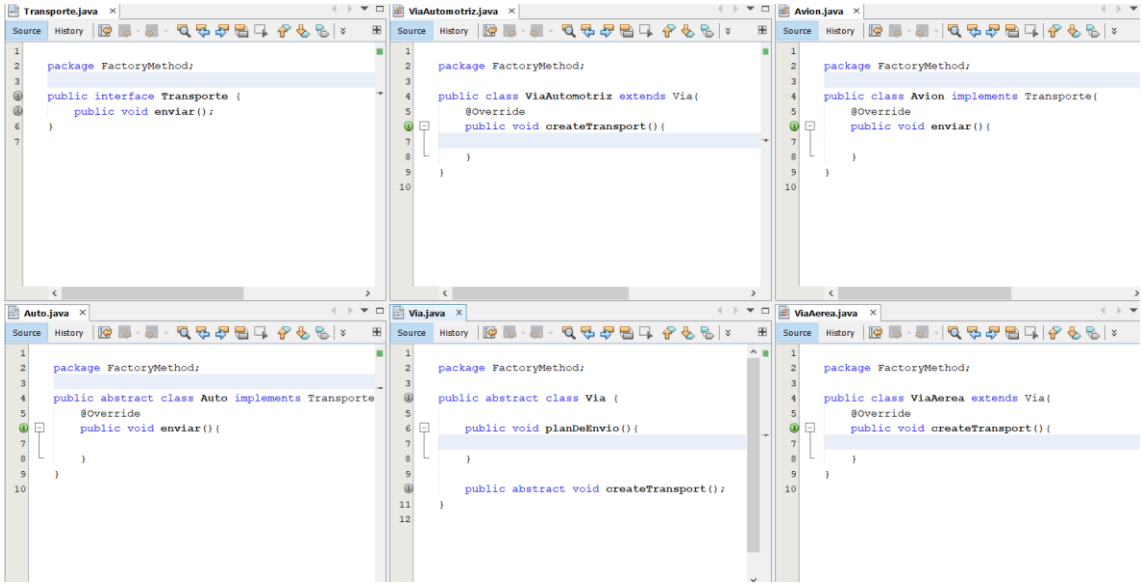


Chain of responsability



**Sección C: Java**

Factory Method



Transporte.java

```
package FactoryMethod;

public interface Transporte {
    public void enviar();
}
```

ViaAutomotriz.java

```
package FactoryMethod;

public class ViaAutomotriz extends Via{
    @Override
    public void createTransport(){

    }
}
```

Avion.java

```
package FactoryMethod;

public class Avion implements Transporte{
    @Override
    public void enviar(){

    }
}
```

Auto.java

```
package FactoryMethod;

public abstract class Auto implements Transporte
    @Override
    public void enviar(){

    }
}
```

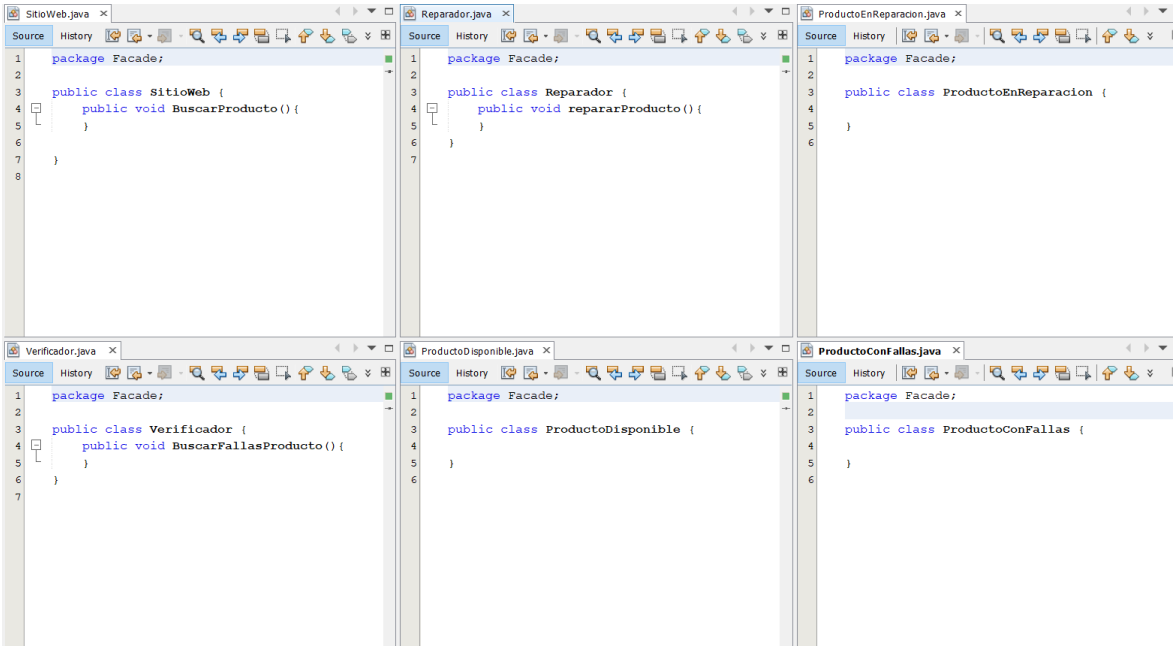
Via.java

```
package FactoryMethod;

public abstract class Via {

    public void planDeEnvio(){

    }

    public abstract void createTransport();
}
```

ViaAerea.java

```
package FactoryMethod;

public class ViaAerea extends Via{
    @Override
    public void createTransport(){

    }
}
```

Facade



```
SitioWeb.java
package Facade;

public class SitioWeb {
    public void BuscarProducto(){
    }

}
```

```
Reparador.java
package Facade;

public class Reparador {
    public void repararProducto(){
    }
}
```

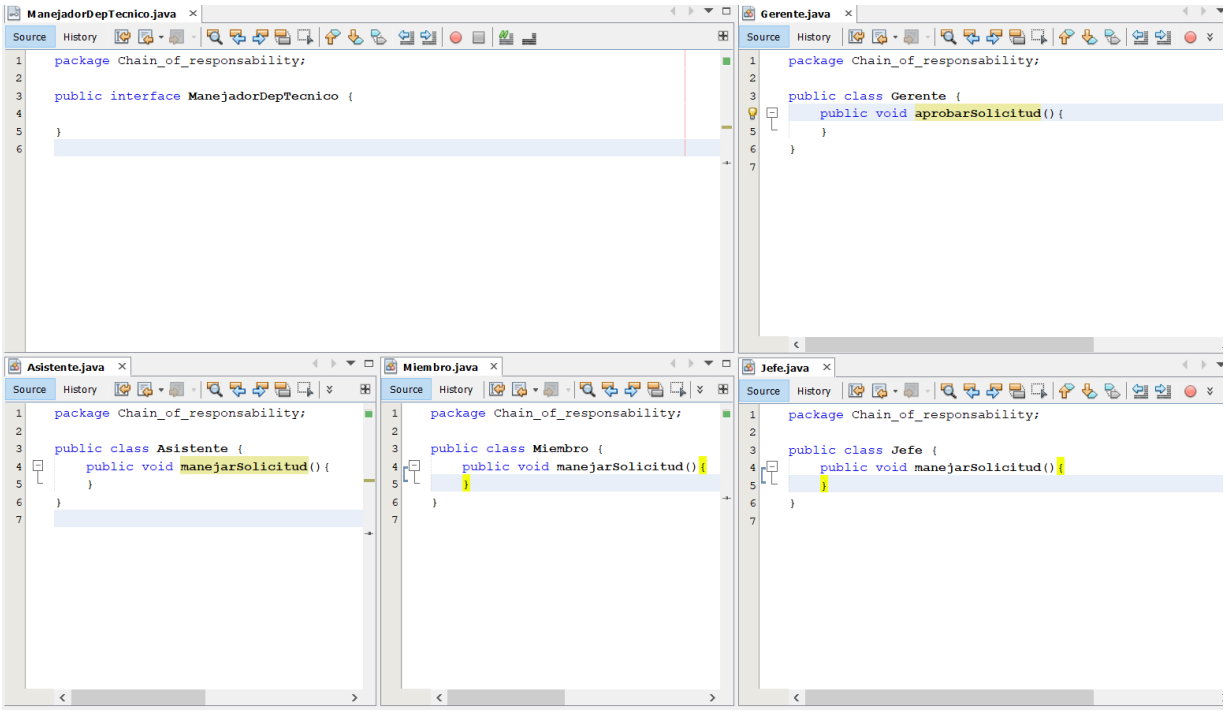
```
ProductoEnReparacion.java
package Facade;

public class ProductoEnReparacion {

}
```

```
Verificador.java
package Facade;

public class Verificador {
    public void BuscarFallasProducto(){
    }
}
```

```
ProductoDisponible.java
package Facade;

public class ProductoDisponible {

}
```

```
ProductoConFallas.java
package Facade;

public class ProductoConFallas {

}
```

## Chain of responsability



```
ManejadorDepTecnico.java
package Chain_of_responsability;

public interface ManejadorDepTecnico {

}
```

```
Gerente.java
package Chain_of_responsability;

public class Gerente {
    public void aprobarSolicitud(){
    }
}
```

```
Asistente.java
package Chain_of_responsability;

public class Asistente {
    public void manejarSolicitud(){
    }
}
```

```
Miembro.java
package Chain_of_responsability;

public class Miembro {
    public void manejarSolicitud(){
    }
}
```

```
Jefe.java
package Chain_of_responsability;

public class Jefe {
    public void manejarSolicitud(){
    }
}
```