Chain of Responsibility

¿Qué es el patrón Chain of Responsibility?

Chain of Responsibility (cadena de responsabilidad) es un patrón de diseño comportamental que permite pasar solicitudes a través de una cadena de manejadores. Al recibir una solicitud, cada manejador decide procesarla o pasarla al siguiente manejador de la cadena.

Es decir:

- Se establece una cadena de objetos (handlers).
- Cada handler decide si procesa una solicitud o si la pasa al siguiente en la cadena.

¿Cuándo usar Chain of Responsibility?

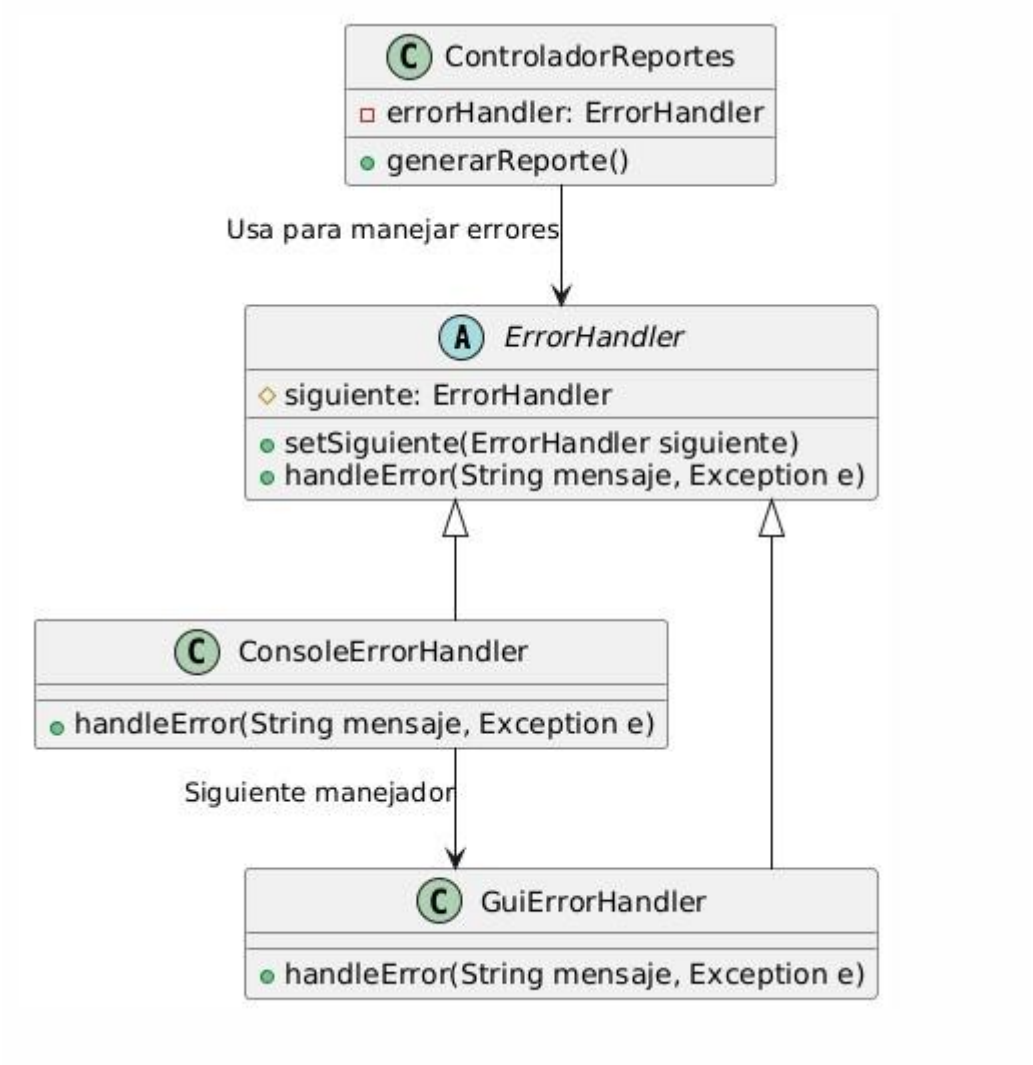
Utiliza Chain of Responsibility cuando:

- Existen varios objetos capaces de manejar una solicitud, pero el objeto específico que la manejará se determina en tiempo de ejecución.
- Se quiere evitar acoplar al remitente con el receptor específico.
- Se desea desacoplar la lógica de gestión de solicitudes, haciendo que el sistema sea flexible y fácil de modificar o extender.
- Se requiere que la solicitud se procese de forma secuencial o en etapas.

Estructura del patrón Chain of Responsibility

- La estructura básica es la siguiente:
  Handler: Define la interfaz común para todos los manejadores.
- ConcreteHandler: Implementa el manejo específico. Decide si maneja o pasa al siguiente handler.

UML PATRON RESPONSABILITY



APLICACIÓN DEL RESPONSABILITY

Clase abstracta base (ErrorHandler)

Primero definiste una clase abstracta que establece la estructura de la cadena de responsabilidad. Esto permite agregar distintos manejadores de errores posteriormente.

```
*/
package controlador.chain;

/**
 *
 * @author Betinsky
 */
public abstract class ErrorHandler {
    protected ErrorHandler siguiente;

    public void setSiguiente(ErrorHandler siguiente) {
        this.siguiente = siguiente;
    }

    // Método abstracto para procesar el error o validación
    public abstract void handleError(String mensaje, Exception e);
}
```

Clases concretas (ConsoleErrorHandler y GuiErrorHandler)

Después se crearon dos manejadores concretos que procesan el error de diferentes formas:

ConsoleErrorHandler.java (Imprime error en la consola):

GuiErrorHandler.java (Muestra error en un JOptionPane):

```java
import javax.swing.JOptionPane;

/**
 *
 * @author Betinsky
 */
public class GuiErrorHandler extends ErrorHandler {
    @Override
    public void handleError(String mensaje, Exception e) {
        JOptionPane.showMessageDialog(null, "[GUI] " + mensaje + (e != null ? " - " + e.toString() : ""));
        if (siguiente != null) {
            siguiente.handleError(mensaje, e);
        }
    }
}
```

```java
 */
package controlador.chain;

/**
 *
 * @author Betinsky
 */
public class ConsoleErrorHandler extends ErrorHandler {
    @Override
    public void handleError(String mensaje, Exception e) {
        System.out.println("[CONSOLE] " + mensaje + (e != null ? " - " + e.toString() :
        if (siguiente != null) {
            siguiente.handleError(mensaje, e);
        }
    }
}
```

Cada manejador decide cómo mostrar o registrar el error, y luego llama al siguiente manejador (si existe).

Uso práctico en métodos del controlador

En tus métodos del controlador, en lugar de llamar directamente a métodos de error específicos (como JOptionPane o imprimir directo en consola), utilizas la cadena de responsabilidad así:

```java
        double totalIVA = Double.parseDouble(formato.format(totalFactura * valorIVA));
        IVA.setText(String.valueOf(totalIVA));
        total.setText(String.valueOf(totalFactura));

        mediator.notificar(this, "ProductosFacturaEncontrados");

    } catch (NumberFormatException e) {
        errorHandler.handleError("Número de factura inválido.", e);
    } catch (Exception e) {
        errorHandler.handleError("Error al mostrar los productos", e);
    } finally {
        objetoConexion.cerrarConexion();
    }
}
```