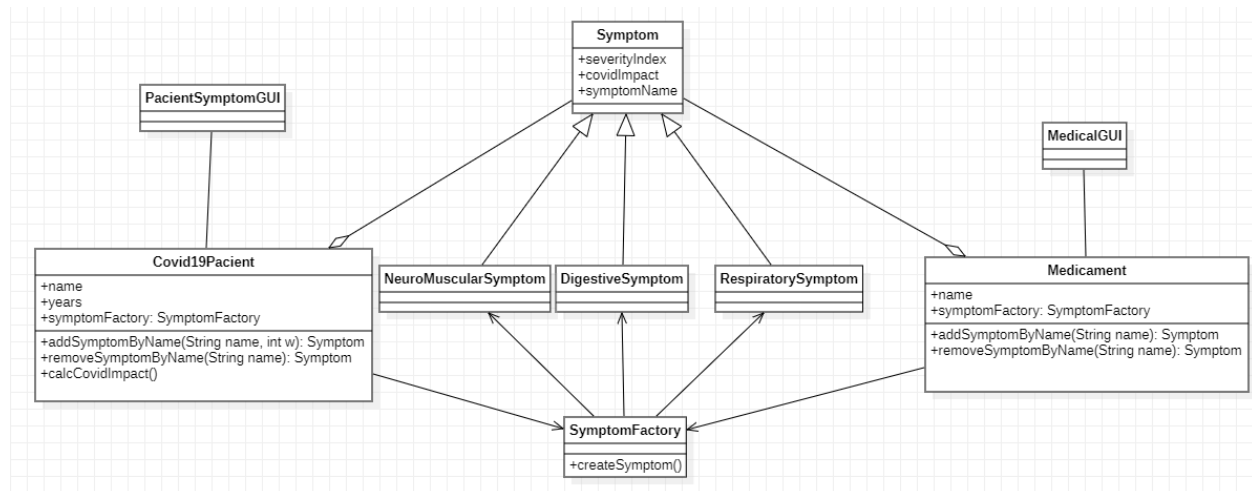


LABORATORIO PATRONES DE DISEÑO

Repositorio GitHub: <https://github.com/JonAnderIturrioz/labpatterns.git>

Simple Factory.

1. Realiza un nuevo diseño de la aplicación (diagrama UML) aplicando el patrón Simple Factory para eliminar vulnerabilidades anteriores y mejorar el diseño en general. Describe con claridad los cambios realizados.



Hemos creado una clase `SymptomFactory` que se va a encargar de crear los síntomas necesarios. Dicho de otra forma, hemos extraído la responsabilidad de crear los síntomas de `Covid19Pacient` y `Medicament` a la nueva clase.

El diagrama UML refleja los cambios mencionados: Covid19Pacient y Medicament ya no se relacionan con las diferentes clases hijas de Symptom para crearlas, sino que utilizan SymptomFactory. El método createSymptom se ha trasladado a la nueva clase, como ya hemos mencionado.

2. Implementa la aplicación y agrega el nuevo síntoma "mareos" asociado a un tipo de impacto 1.

Para la implementación solicitada, se han modificado las clases Covid19Pacient y Medicament, y creado SymptomFactory. En cuanto al síntoma adicional, se ha añadido la opción a SymptomFactory (impacto 1, peso 6, NeuroMuscularSymptom).

- Covid19Pacient:
<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/domain/Covid19Pacient.java>
- Medicament:
<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/domain/Medicament.java>
- SymptomFactory:
<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/factory/SymptomFactory.java>

3. Cómo se puede adaptar la clase Factory, para que los objetos Symptom que utilicen las clases Covid19Pacient y Medicament sean únicos. Es decir, para cada síntoma sólo exista un objeto. (Si hay x síntomas en el sistema, haya únicamente x objetos Symptom)

Para conseguir este efecto, vamos a hacer que el método createSymptom sea static. A su vez, vamos a crear un mapa hash para guardar los síntomas ya creados.

Cada vez que se le llame al método, vamos a comprobar que el síntoma no esté creado ya. Si lo está, devolveremos el síntoma, y si no, se procederá con la creación del objeto.

Patrón Observer.

1. Cambia el programa principal para crear 2 pacientes Covid19Pacient con sus interfaces PatientSymptomGUI y PatientObserverGUI.

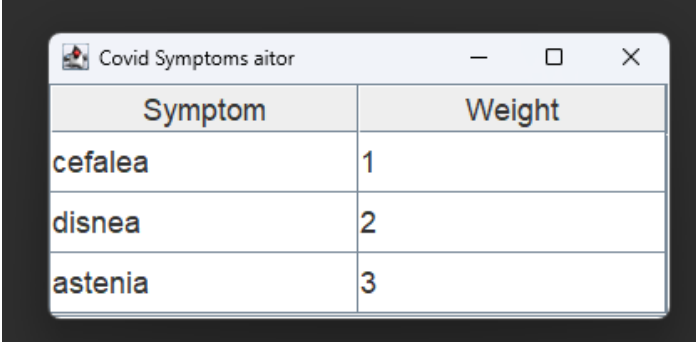
<https://github.com/JonAnderIturrioz/labpatterns/tree/master/src/observer>

Patrón Adapter.

1. Añade el código necesario en la clase Covid19PacientTableModelAdapter y ejecuta la aplicación para comprobar que funciona correctamente.

Covid19PacientTableModelAdapter:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter2/Covid19PacientTableModelAdapter.java>



Symptom	Weight
cefalea	1
disnea	2
astenia	3

2. Añade otro paciente con otros síntomas, y ejecuta la aplicación para que aparezcan los 2 pacientes con sus síntomas.

Main:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter2/Main.java>

```
package adapter2;

import domain.Covid19Pacient;

public class Main {

    public static void main(String[] args) {
        Covid19Pacient pacient=new Covid19Pacient("aitor", 35);

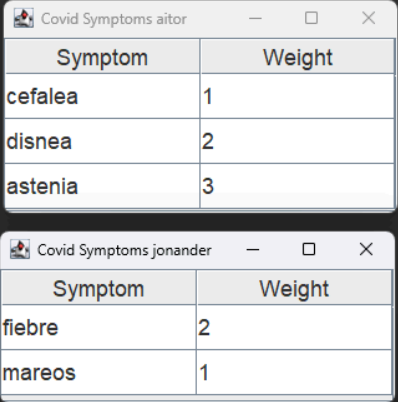
        pacient.addSymptomByName("disnea", 2);
        pacient.addSymptomByName("cefalea", 1);
        pacient.addSymptomByName("astenia", 3);

        ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient);
        gui.setPreferredSize(
            new java.awt.Dimension(300, 200));
        gui.setVisible(true);

        Covid19Pacient pacient2=new Covid19Pacient("jonander", 22);

        pacient2.addSymptomByName("fiebre", 2);
        pacient2.addSymptomByName("mareos", 1);

        ShowPacientTableGUI gui2=new ShowPacientTableGUI(pacient2);
        gui2.setPreferredSize(
            new java.awt.Dimension(300, 200));
        gui2.setVisible(true);
    }
}
```



Symptom	Weight
cefalea	1
disnea	2
astenia	3

Symptom	Weight
fiebre	2
mareos	1

3. (opcional). Cómo podrías añadir esta nueva pantalla al ejercicio anterior del observer, de forma que cada vez que se añada un nuevo síntoma a un paciente, se actualice la tabla.

-

Patrón Iterator y Adapter.

1. Crea un programa principal utilizando el método `Sorting.sortedIterator` que imprima los 5 síntomas que debe tener un paciente `Covid19Pacient`. Se imprimirá primero ordenando por `symptomName` y luego por `severityIndex`.
 - Crea un paciente `Covid19Pacient` con cinco síntomas. La clase `Covid19Pacient` NO PUEDE CAMBIARSE NADA.
 - Implementa las interfaces `Comparator`: una para la ordenación por `symptomName`, y otra para la ordenación según `severityIndex`.
 - Crea el patrón adapter sobre la clase `Covid19Pacient`, implementando la interfaz `InvertedIterator`. Recuerda crear una constructora adecuada para enviarle la información del paciente.
 - TAMPOCO SE PUEDE MODIFICAR `Sorting.sortedIterator`.

Programa principal:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter/Main.java>

`Covid19PacientInvertedIteratorAdapter`:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter/Covid19PacientInvertedIteratorAdapter.java>

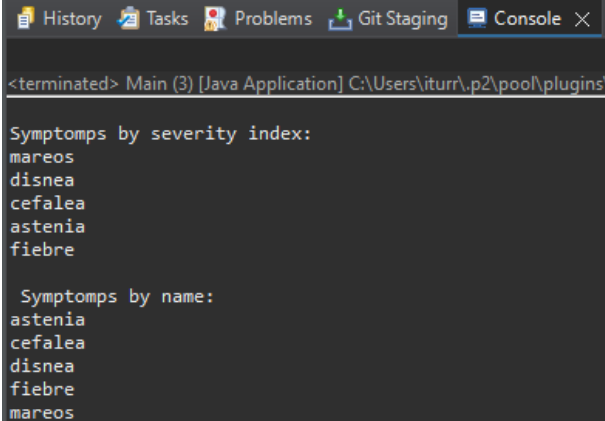
`SeverityIndexComparator`:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter/SeverityIndexComparator.java>

`SymptomNameComparator`:

<https://github.com/JonAnderIturrioz/labpatterns/blob/master/src/adapter/SymptomNameComparator.java>

Ejecución del programa principal:



```
<terminated> Main (3) [Java Application] C:\Users\iturr\p2\pool\plugins\

Symptoms by severity index:
mareos
disnea
cefalea
astenia
fiebre

Symptoms by name:
astenia
cefalea
disnea
fiebre
mareos
```