

# PATRONES DE DISEÑO

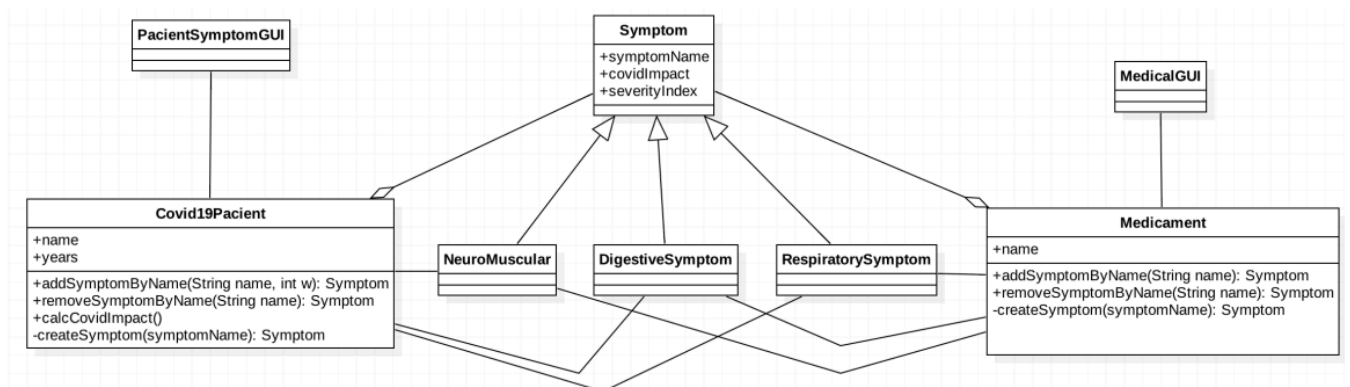
Paula Recaj, Irati Cruz y Paula Recaj

## 1. Simple factory

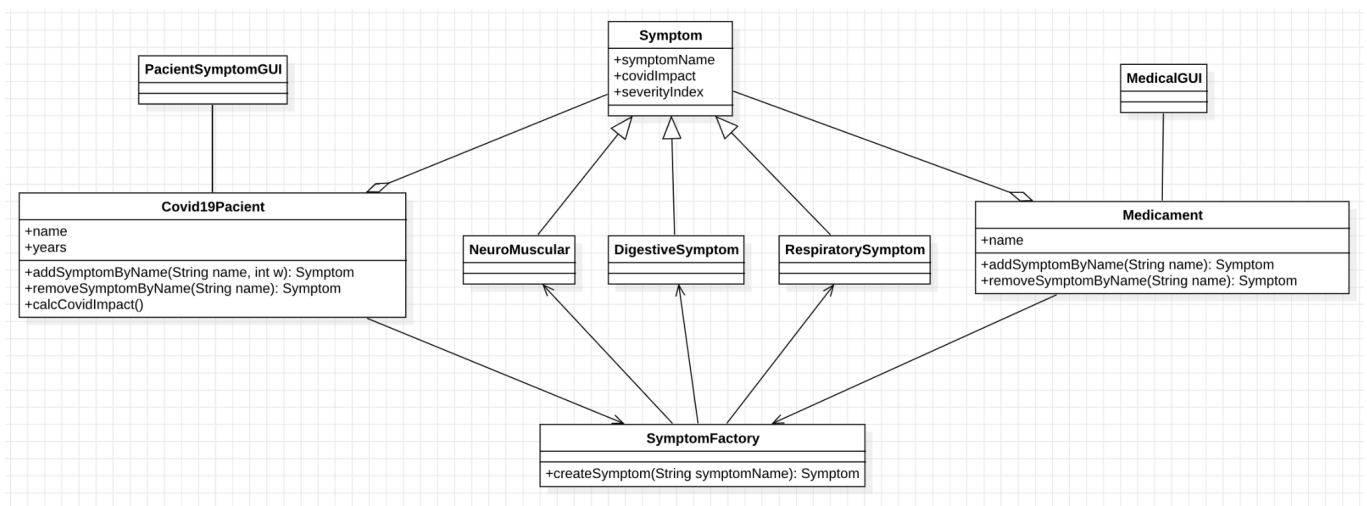
Se pide:

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.

UML previo:



UML final:



- Hemos creado una clase `SymptomFactory` para que se encargue de la creación de los síntomas en vez de que `Covid19Pacient` y `Medicament` los creen cada uno individualmente, ya que son los mismos síntomas que utilizan ambas y el mismo código por lo cual se genera código duplicado y resulta ser redundante.
- Hemos movido el método `createSymptom()` que es el encargado de crear los síntomas a `SymptomFactory` y lo hemos convertido estático para que así utilicen la misma instancia las clases que hagan uso del método, en nuestro caso, `Medicament` y `Covid19Pacient`.

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

```
public class SymptomFactory {

    public static Symptom createSymptom(String symptomName) {
        List<String> impact5 = Arrays.asList("fiebre", "tos seca", "astenia", "expectoracion");
        List<Double> index5 = Arrays.asList(87.9, 67.7, 38.1, 33.4);
        List<String> impact3 = Arrays.asList("disnea", "dolor de garganta",
"cefalea", "mialgia", "escalofrios");
        List<Double> index3 = Arrays.asList(18.6, 13.9, 13.6, 14.8, 11.4);
        List<String> impact1 = Arrays.asList("nauseas", "vómitos", "congestión
nasal", "diarrea", "hemoptisis", "congestion conjuntival", "vertigo", "visión borrosa", "desequilibrio", "sudoración
excesiva", "visión borrosa", "aturdimiento");
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 1.2, 3.4, 0.7, 0.5, 2.3, 1.4, 3.3 );

        List<String> digestiveSymptom=Arrays.asList("nauseas", "vómitos", "diarrea");
        List<String> neuroMuscularSymptom=Arrays.asList("fiebre", "astenia", "cefalea",
"mialgia", "escalofrios");
        List<String> respiratorySymptom=Arrays.asList("tos seca", "expectoracion", "disnea", "dolor de
garganta", "congestión nasal", "hemoptisis", "congestion conjuntival");
        List<String> mareosSymptom = Arrays.asList("nauseas", "vertigo", "visión borrosa",
"desequilibrio", "sudoración excesiva", "visión borrosa", "aturdimiento");
        int impact=0;
        double index=0;

        if (impact5.contains(symptomName)) {impact=5; index=
index5.get(impact5.indexOf(symptomName));}
        else if (impact3.contains(symptomName)) {impact=3; index=
index3.get(impact3.indexOf(symptomName));}
        else if (impact1.contains(symptomName)) {impact=1; index=
index1.get(impact1.indexOf(symptomName));}

        if (impact!=0) {
            if (digestiveSymptom.contains(symptomName)) return new
DigestiveSymptom(symptomName,(int)index, impact);
            if (neuroMuscularSymptom.contains(symptomName)) return new
NeuroMuscularSymptom(symptomName,(int)index, impact);
            if (respiratorySymptom.contains(symptomName)) return new
RespiratorySymptom(symptomName,(int)index, impact);
            if (mareosSymptom.contains(symptomName)) return new MareosSymptom(symptomName,
(int)index, impact);
        }
        return null;
    }
}
```

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`)
  - Haciendo el método estático.

## 2. Patrón Observer

**Se pide:** cambia el programa principal para crear 2 pacientes `Covid19Pacient` con sus interfaces `PacientSymptomGUI` y `PacientObserverGUI`.

- He realizado los cambios necesarios para aplicar el patrón Observer, pero este al estar “deprecated” no deja que el main funcione, por lo que no se puede ejecutar.

```
public static void main(String[] args) {  
    Observable pacient = new Covid19Pacient("aitor", 35);  
    new PacientObserverGUI (pacient);  
    new PacientSymptomGUI ((Covid19Pacient))pacient);  
  
    Observable pacient2 = new Covid19Pacient("alex", 22);  
    new PacientObserverGUI (pacient2);  
    new PacientSymptomGUI ((Covid19Pacient))pacient2);  
}
```

## 3. Patrón Adapter

**Se pide:**

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

```
public int getColumnCount() {  
    // Challenge!  
    return columnNames.length;  
}  
public String getColumnName(int i) {  
    // Challenge!  
    return columnNames[i];  
}  
public int getRowCount() {  
    // Challenge!  
    return pacient.getSymptoms().size();  
}
```

```

public Object getValueAt(int row, int col) {
    // Challenge!

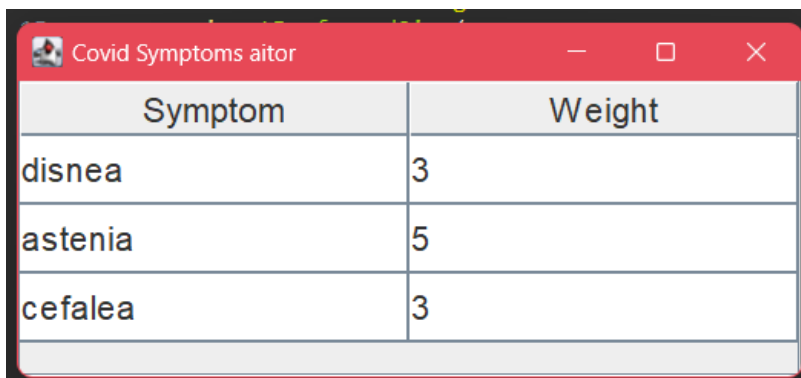
    Symptom s = (Symptom) pacient.getSymptoms().toArray()[row];

    switch(col) {

    case 0:
        return s.getName();
    case 1:
        return s.getSeverityIndex();

    default:
        return null;
    }
}

```



Symptom	Weight
disnea	3
astenia	5
cefalea	3

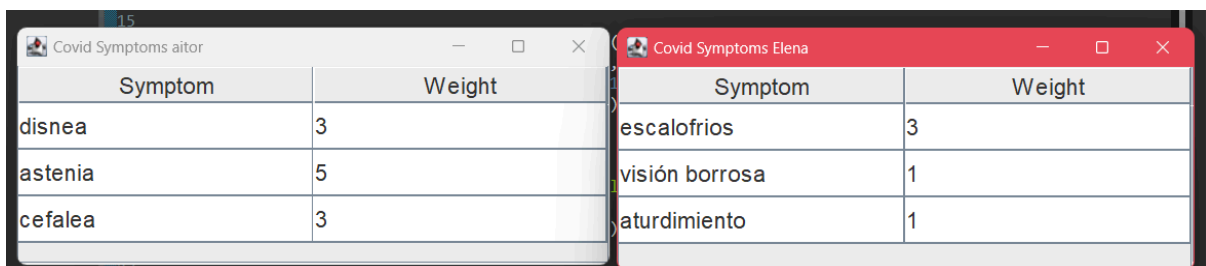
- Añade otro paciente con otros síntomas, y ejecuta la aplicación para que aparezcan los 2 pacientes con sus síntomas.

```

Covid19Pacient pacient2=new Covid19Pacient("Elena", 50);
    pacient2.addSymptomByName("visión borrosa", 1);
    pacient2.addSymptomByName("aturdimiento", 1);
    pacient2.addSymptomByName("escalofrios", 3);

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

```



Symptom	Weight
disnea	3
astenia	5
cefalea	3

Symptom	Weight
escalofrios	3
visión borrosa	1
aturdimiento	1

## 4. Patrón Iterator y Adapter

### Se pide:

Crear un programa principal utilizando el método `Sorting.sortedIterator` que imprima los 5 síntomas que debe tener un paciente `Covid19Pacient`. Se imprimirá primero ordenando `symptomName` y luego por `severityIndex`.

- Crea un paciente `Covid19Pacient` con cinco síntomas. La clase `Covid19Pacient` NO PUEDE CAMBIARSE NADA
- Implementa las interfaces `Comparator`: uno 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`.

Se han implementado 3 clases nuevas:

- En el package **adapter**: `Covid19PacientInvertedIterator` (implementa `InvertedIterator`)
- En el package **iterator**: `SymptomNameComparator` y `SeverityIndexComparator` (implementan `Comparator`)