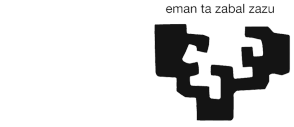
Grado en Ingeniería Informática
Facultad de Informática
UPV/EHU

LABORATORIO 3.2
DISEÑO AVANZADO - PATRONES DE DISEÑO



Mikel Pallin
Iker López

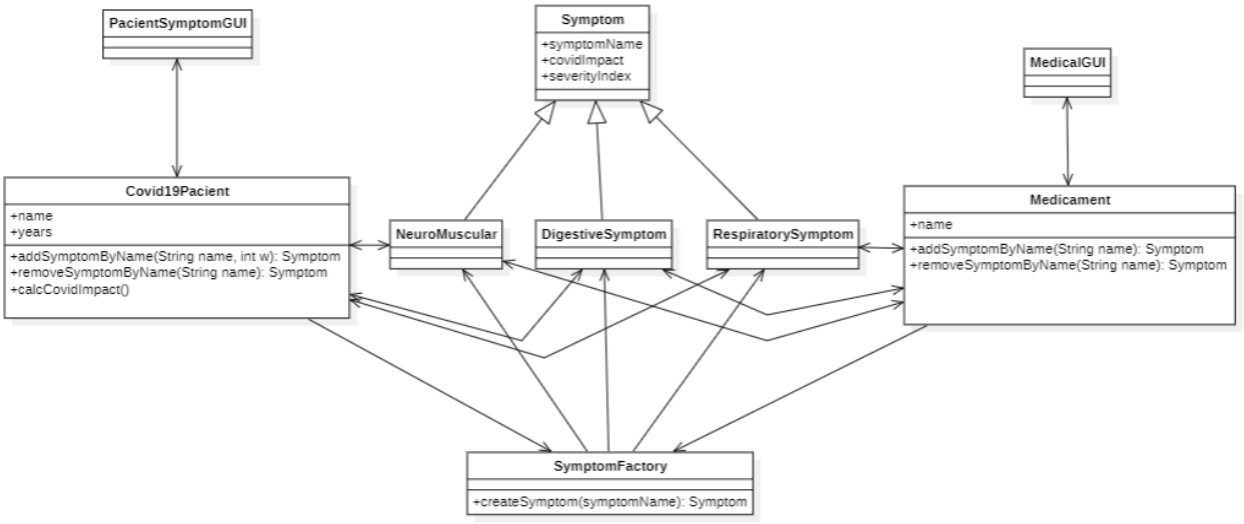
Índice

1. Simple Factory (SF)

1.1. SF Tarea 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.

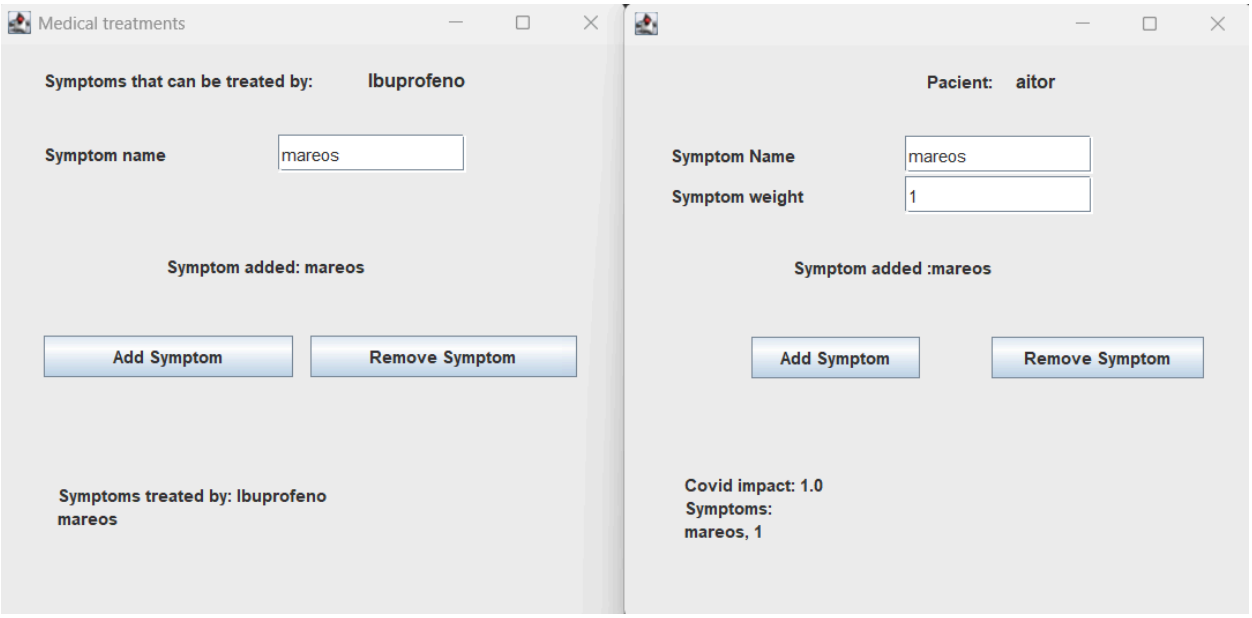
Para poder aplicar el patrón Simple Factory se ha creando la clase SymptomFactory, la cual es la encargada de crear nuevos síntomas. Después se ha eliminado el método createSymptom de Covid19Pacient y Medicament, añadiendolo a la clase destinada a crear los síntomas.



1.2. SF Tarea 2

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

```
public class SymptomFactory {

    public 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("mareos", "nauseas", "vómitos", "congestión nasal","diarrea","hemoptisis","congestion conjuntival");
        List<Double> index1 = Arrays.asList(99.99, 5.0, 4.8, 3.7, 0.9, 0.8);

        List<String> digestiveSymptom=Arrays.asList("nauseas", "vómitos","diarrea");
        List<String> neuroMuscularSymptom=Arrays.asList("mareos", "fiebre", "astenia", "cefalea", "mialgia","escalofrios");
        List<String> respiratorySymptom=Arrays.asList("tos seca","expectoracion","disnea","dolor de garganta", "congestión nasal","hemoptisis","c


        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);
        }
        return null;

    }
```



1.3. SF Tarea 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).

Esto se puede conseguir adaptando la clase SymptomFactory, añadiendo por ejemplo un hashMap que funcione como un registro de síntomas, poniendo los nombres de estos como keys. Antes de crear un nuevo Symptom, la clase consulta el hashMap y si el síntoma ya existe, devuelve el síntoma ya creado anteriormente, garantizando que solo haya un único objeto para cada nombre de síntoma.

2. Patrón Observer (PO)

2.1. PO Tarea 1

cambia el programa principal para crear 2 pacientes Covid19Pacient con sus interfaces PacientSymptomGUI y PacientObserverGUI.

```
public class Main {

    /**
     * Launch the application.
     */
    public static void main(String[] args) {

        Observable pacient = new Covid19Pacient("aitor", 35);
        new PacientObserverGUI (pacient);
        new PacientSymptomGUI ((Covid19Pacient)pacient);
        new PacientThermometerGUI ((Covid19Pacient)pacient);

        Observable pacient2 = new Covid19Pacient("ane", 30);
        new PacientObserverGUI (pacient2);
        new PacientSymptomGUI ((Covid19Pacient)pacient2);
        new PacientThermometerGUI ((Covid19Pacient)pacient2);
    }


}
```

Simplemente se ha añadido un paciente2 al programa principal, añadiendo sus respectivos GUI. Para el termómetro, se han seguido los mismos pasos del laboratorio para que éste funcionase con cada paciente por separado, quedando el código de la siguiente manera.

```
public class PacientThermometerGUI extends Frame implements Observer{
    private TemperatureCanvas gauges;
    /**
     * @wbp.nonvisual location=119,71
     */
    private final JLabel label = new JLabel("New label");

    public PacientThermometerGUI(observer.Covid19Pacient obs){
        super("Temperature Gauge");
        Panel Top = new Panel();
        add("North", Top);
        gauges = new TemperatureCanvas(0,15);
        gauges.setSize(500,280);
        add("Center", gauges);
        setSize(200, 380);
        setLocation(0, 100);
        setVisible(true);
        obs.addObserver(this);
    }
```

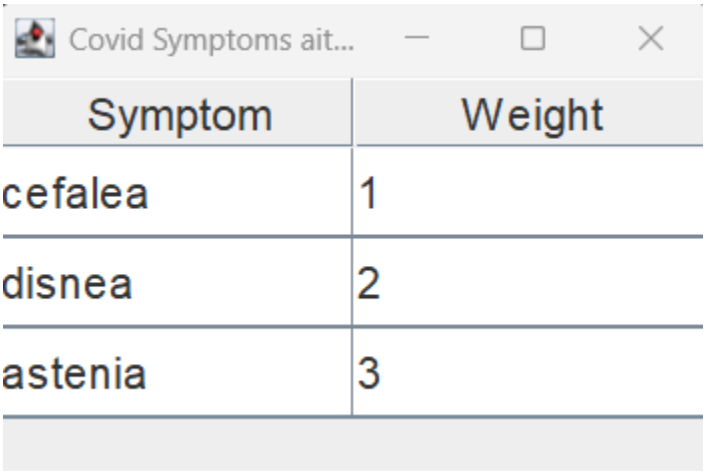
```
@Override
public void update(Observable o, Object arg) {
    Covid19Pacient p = (Covid19Pacient) o;
    // Obtain the current covidImpact to paint
    int farenheit = (int) p.covidImpact();
    // temperature gauge update
    gauges.set(farenheit);
    gauges.repaint();

}
```

3. Patrón Adapter (PA)

3.1. PA Tarea 1

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

```
public class Covid19PacientTableModelAdapter extends AbstractTableModel {
      protected Covid19Pacient pacient;
      protected String[] columnNames =
        new String[] {"Symptom", "Weight" };

      public Covid19PacientTableModelAdapter(Covid19Pacient p) {
        this.pacient=p;
      }

      public int getColumnCount() {
        // Challenge!
         return 2;
      }

      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!
          Set<Symptom> ss = pacient.getSymptoms();
          Object[] sa =  ss.toArray();
          Symptom s = (Symptom) sa[row];

          if (col == 0) return s.getName();
          else return pacient.getWeight(s);
```



3.2. PA Tarea 2

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

```
public class Main {

    public static void main(String[] args) {
        Covid19Pacient pacient=new Covid19Pacient("aitor", 35);
        Covid19Pacient pacient2=new Covid19Pacient("ana", 35);

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

        pacient2.addSymptomByName("mareos", 1);
        pacient2.addSymptomByName("nauseas", 2);
        pacient2.addSymptomByName("vómitos", 3);

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

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

Covid Symptoms ana

Symptom	Weight
vómitos	3
nauseas	2
mareos	1

Covid Symptoms ait...

Symptom	Weight
cefalea	1
disnea	2
astenia	3

4. Patrón Iterator y Adapter (PIA)

4.1. PIA Tarea 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.**

```
Covid19Pacient pacient = new Covid19Pacient("ana", 35);

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

- **Implementa las interfaces Comparator: una para la ordenación por symptomName, y otra para la ordenación según severityIndex.**

```
public class SymptomNameComparator implements Comparator<Object> {

    @Override
    public int compare(Object o1, Object o2) {
        Symptom s1 = (Symptom) o1;
        Symptom s2 = (Symptom) o2;

        return s1.getName().compareTo(s2.getName());
    }
```

```
public class SymptomSeverityComparator implements Comparator<Object> {

    @Override
    public int compare(Object o1, Object o2) {
        Symptom s1 = (Symptom) o1;
        Symptom s2 = (Symptom) o2;

        return Integer.compare(s1.getSeverityIndex(), s2.getSeverityIndex());
    }
```

- **Crea el patrón adapter sobre la clase Covid19Pacient, implementando la interfaz InvertedIterator. Recuerda crear una constructora adecuada para enviarle la información del paciente.**

```
public class Covid19PacientAdapter implements InvertedIterator {

    private List<Symptom> symptomsList;
    private int index;

    public Covid19PacientAdapter(Covid19Pacient pacient) {
        this.symptomsList = new ArrayList<>(pacient.getSymptoms());
        this.index = symptomsList.size();
    }

    public void goLast() {
        index = symptomsList.size() - 1;
    }

    public boolean hasPrevious() {
        return index >= 0;
    }

    public Object previous() {
        if (!hasPrevious()) {
            return null;
        }
        return symptomsList.get(index--);
    }
```

- **Clase main final:**

```java
public class Main {

    public static void main(String[] args) {

        Covid19Pacient pacient = new Covid19Pacient("ana", 35);

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

        InvertedIterator adapter = new Covid19PacientAdapter(pacient);

        System.out.println("Ordenados por nombre:");
        Iterator<Object> itName = Sorting.sortedIterator(adapter, new SymptomNameComparator());
        while (itName.hasNext()) {
            Symptom s = (Symptom) itName.next();
            System.out.println(s.getName());
        }


        System.out.println("\nOrdenados por severidad:");
        Iterator<Object> itSeverity = Sorting.sortedIterator(adapter, new SymptomSeverityComparator());
        while (itSeverity.hasNext()) {
            Symptom s = (Symptom) itSeverity.next();
            System.out.println(s.getName() + " - severidad: " + s.getSeverityIndex());
        }
    }
```