Principios de diseño: Laboratorio

Autores:

- Rubén Gallego
- Mikel Berasategui

Asignatura: Ingeniería de Software II

Enlaces del proyecto:

-Github: https://github.com/RubenGGBC/labpatterns.git

Índice

1.Simple Factory

Vulnerabilidades de la aplicación:

A. ¿Qué sucede si aparece un nuevo síntoma (por ejemplo, mareos)?

-Tanto la clase Covid19Pacient como la clase Medicament tienen el método "createSymptom", el cual es igual en ambas clases. En caso de que aparezca un nuevo síntoma, tendríamos que modificar ambos métodos para añadir a mano el nuevo síntoma en ambos. ****

B. ¿Cómo se puede crear un nuevo síntoma sin cambiar las clases existentes (principio OCP)?

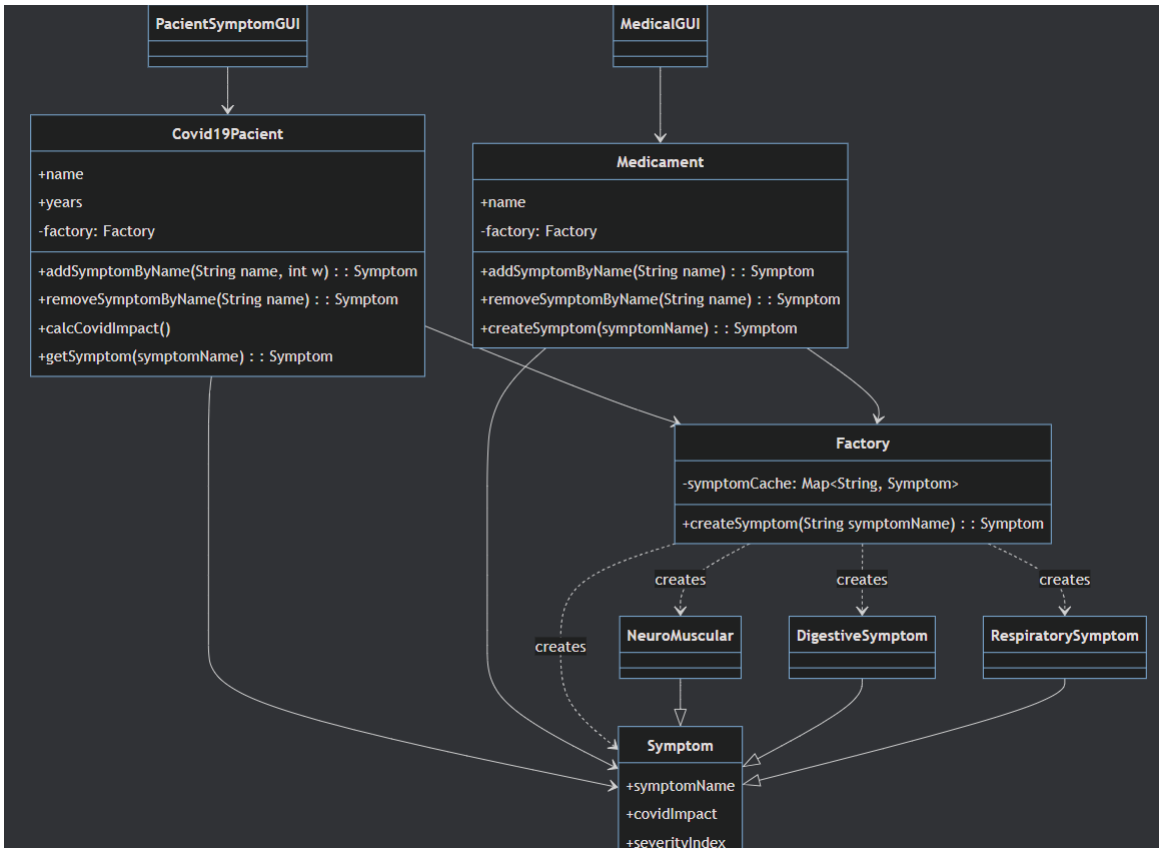
-El principio OCP dice que las entidades software tienen que estar abiertas a extensiones pero cerradas a modificaciones, en este caso se incumple claramente, ya que si queremos modificar el método "createSymptom", tenemos que hacerlo en 2 clases diferentes. ****

C. ¿Cuántas responsabilidades tienen las clases de Covid19Pacient y Medicament (principio SRP)?

-El principio SRP dice que una clase debe tener una única razón para cambiar. En este caso, las clases Covid19Pacient y Medicament tienen muchas responsabilidades y, en caso de querer modificar/añadir una de esas, tenemos que cambiar de nuevo la clase (tiene muchas razones para cambiar). ****

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.



-Ahora, Medicament y Covid19Pacient no se relacionan las 2 directamente con los 3 tipos de síntomas, sino que es Factory quien los crea. Esta vez, en caso de crear un nuevo síntoma, lo hacemos desde Factory solamente y no desde Medicament y Covid19Pacient.**

-(El uml se ha añadido como factoryUML.md en lenguaje mermaid dentro de la carpeta Factory. Para poder visualizarlo dentro de algún IDE, al menos en IntelliJ, se necesita el plugin de mermaid. Si no existe ese plugin en otro IDE, se puede visualizar el diagrama copiando el contenido del archivo factoryUML.md, obviando la primera y última línea que indica que es lenguaje mermaid y se pega en la siguiente página: https://mermaid.live/)

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

-Nueva clase *Factory*:

Factory.java

```java
public class Factory {  9 usages   mikel +1 *


    public Symptom createSymptom(String symptomName) {  mikel +1 *


        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", "mareos"); //se añade mareos (2.)
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 0.7, 2.0); //se han añadido 2 index nuevos (faltaba uno en la respuesta original + el nuevo de mareos)

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

        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));}

        Symptom newSymptom = null;

        if (impact!=0)  {
            if (digestiveSymptom.contains(symptomName)) newSymptom =  new DigestiveSymptom(symptomName,(int)index, impact);
            if (neuroMuscularSymptom.contains(symptomName)) newSymptom = new NeuroMuscularSymptom(symptomName,(int)index, impact);
            if (respiratorySymptom.contains(symptomName)) newSymptom = new RespiratorySymptom(symptomName,(int)index, impact);
        }

        return newSymptom;
    }
}
```

-Clase *Covid19Pacient* modificada:

Covid19Pacient.java

```java
public class Covid19Pacient {  jon +3
    private String  name;   3 usages
    private int age;  3 usages
    private Map<Symptom,Integer> symptoms=new HashMap<>();  9 usages
    private Factory symptomFactory;  2 usages

    public Covid19Pacient(String name, int years) {  jon +1
        this.name = name;
        this.age = years;
        this.symptomFactory = new Factory();
    }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public int getAge() { return age; }
    public void setAge(int age) { this.age = age; }
    public int getWeight(Symptom s) { return symptoms.get(s); }

    public Set<Symptom> getSymptoms() { return symptoms.keySet(); }

    public Symptom getSymptomByName(String symptomName) {  3 usages  jon
        Iterator<Symptom> i= getSymptoms().iterator();
        Symptom s=null;
        while (i.hasNext()) {
            s=i.next();
            if (s!=null && s.getName().equals(symptomName)) return s;
        }
        return null;
    }

    public void addSymptom(Symptom c, Integer w) { symptoms.put(c,w); }

    public Symptom addSymptomByName(String symptom, Integer w){  jon +2
        Symptom s=null;
        s=symptomFactory.createSymptom(symptom);
        if (s!=null)
            symptoms.put(s,w);
        return s;
    }

    public Symptom removeSymptomByName(String symptomName) {  jon
        Symptom s=getSymptomByName(symptomName);
        System.out.println("Simptom to remove: "+s);
        if (s!=null) symptoms.remove(s);
        return s;
    }
    public Iterator iterator() { return new Covid19PacientIterator(this.symptoms.keySet()); }

    public double covidImpact() {  jon
        double afection=0;
        double increment=0;
        double impact=0;

        //calculate afection
        for (Symptom c: symptoms.keySet()) {
            if (c!=null )
                afection=afection+c.getSeverityIndex()*symptoms.get(c);
        }
        afection=afection/symptoms.size();

        //calculate increment
        if (getAge()>65) increment=afection*0.5;

        //calculate impact
        impact=afection+increment;
        return impact;
    }

}
```

-Clase *Medicament* modificada:

Medicament.java

```java
public class Medicament {  jon +3
    private String name;  3 usages
    private List<Symptom> symptoms=new ArrayList<~>();  4 usages
    private Factory symptomFactory;  2 usages


    public String getName() { return name; }

    public void setName(String name) { this.name = name; }

    public Medicament(String name) {  jon +1
        super();
        this.name = name;
        this.symptomFactory = new Factory();
    }

    public Symptom addSymptomByName(String symptom){  iturrioz +2
        Symptom s=null;
        s=symptomFactory.createSymptom(symptom);
        if (s!=null) {
            symptoms.add(s);
        }
        return s;

    }

    public void removeSymptom(Symptom s){  1 usage  jon
        symptoms.remove(s);

    }
    public Iterator<Symptom> getSymptoms() { return symptoms.iterator(); }
    public Symptom getSymptomByName(String symptomName) {  3 usages  jon
        Iterator<Symptom> i= symptoms.iterator();
        Symptom s=null;
        while (i.hasNext()) {
            s=i.next();
            if (s!=null && s.getName().compareTo(symptomName)==0) return s;
        }
        return null;
    }
    public Symptom removeSymptomByName(String symptomName) {  jon
        Symptom s=getSymptomByName(symptomName);
        if (s!=null)removeSymptom(s);
        return s;
    }
}
```

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)

   -Se puede cachear la lista de síntomas y hacer que cuando aparezca uno nuevo, este se añada a la lista, pero si se quiere añadir uno que ya existe en la caché, el método devuelva dicho síntoma sin crear nada nuevo ni añadir nada nuevo a la caché. ****

   -Clase Factory modificada:

Factory.java

```
public class Factory {  9 usages   mikel +1 *

    private Map<String, Symptom> symptomCache = new HashMap<>(); //Se cachea los síntomas creados (3.)  3 usages

    public Symptom createSymptom(String symptomName) {   mikel +1 *

        if (symptomCache.containsKey(symptomName)) {
            return symptomCache.get(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", "mareos"); //se añade mareos (2.)
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 0.7, 2.0); //se han añadido 2 index nuevos (faltaba uno en la respuesta original + el nuevo de mareos)

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

        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));}

        Symptom newSymptom = null;

        if (impact!=0)  {
            if (digestiveSymptom.contains(symptomName)) newSymptom =  new DigestiveSymptom(symptomName,(int)index, impact);
            if (neuroMuscularSymptom.contains(symptomName)) newSymptom = new NeuroMuscularSymptom(symptomName,(int)index, impact);
            if (respiratorySymptom.contains(symptomName)) newSymptom = new RespiratorySymptom(symptomName,(int)index, impact);
        }

        if (newSymptom != null) {
            symptomCache.put(symptomName, newSymptom);
        }

        return newSymptom;
    }
}
```

2. Patrón Observer

2.1-Primer prototipo:

Paso 1. Clase CovidPacient (extends Observable).

Copia la clase CovidPacient en el package observer, para no modificar la original de domain. Deberás cambiar las referencias a esa clase para que no tome la definición de domain. Primero tienes que extender CovidPacient con la clase Observable (es decir, añadir en la definición de clase extends Observable). Cada vez que se añade o elimina un síntoma, se debe informar a sus subscriptores, por lo que en los métodos addSymptomByName y removeSymptomBuName deberán añadirse las siguientes instrucciones: setChanged(); notifyObservers();

-Clase *Covid19Pacient* modificada:

Covid19Pacient.java

```
public class Covid19Pacient extends Observable {
```

Covid19Pacient.java

```
public Symptom addSymptomByName(String symptom, Integer w){
    Symptom s=null;
    s=symptomFactory.createSymptom(symptom);
    if (s!=null)
        symptoms.put(s,w);
    setChanged();
    notifyObservers();
    return s;
}

public Symptom removeSymptomByName(String symptomName) {
    Symptom s=getSymptomByName(symptomName);
    System.out.println("Simptom to remove: "+s);
    if (s!=null) symptoms.remove(s);
    setChanged();
    notifyObservers();
    return s;
}
```

Paso 2. PacientObserverGUI (implements Observer).

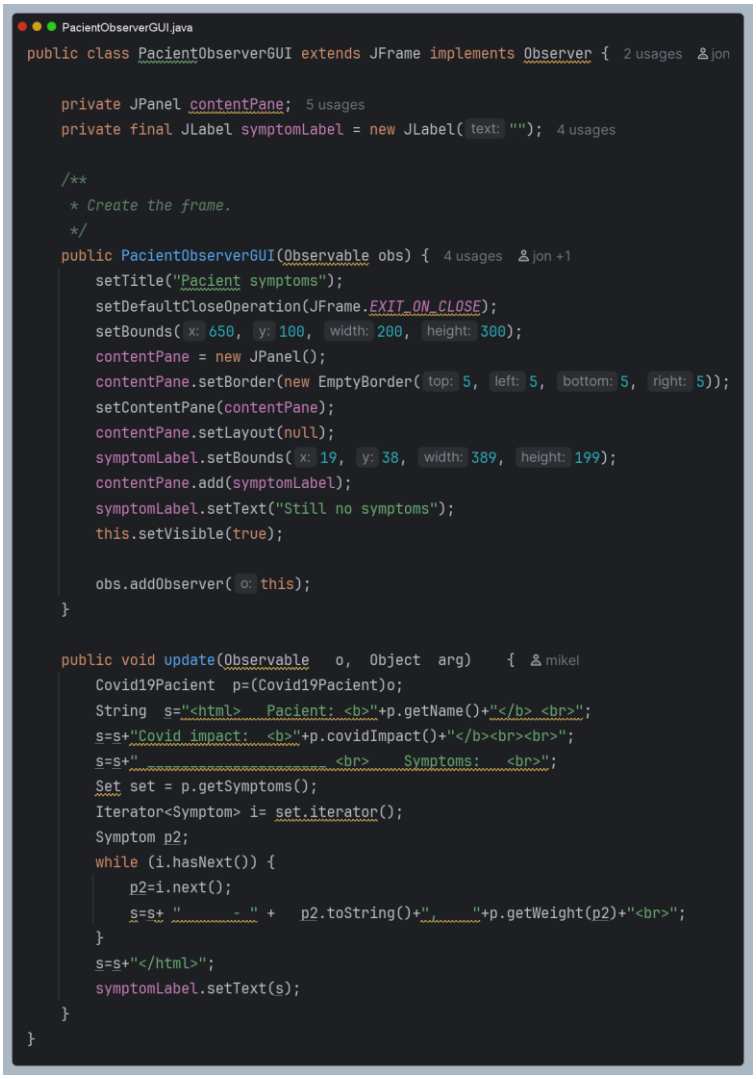
Se deben realizar tres modificaciones:

A) Implementa la clase Observer, es decir, añadir a la definición de clase implements Observer.

B) Añade a la constructora un parámetro Observable (obs, por ejemplo). Este parámetro será el objeto que queremos subscribir. Por ello, añadiremos en la constructora la sentencia obs.addObserver(this);

C) La clase debe añadir el método update (más abajo) que implementa la interfaz Observer. Este método se ejecutará cuando se modifique el objeto al que estamos subscritos (Observable). Para nosotros el valor del prototipo args es null. El método crea un texto HTML en una variable String a través de la información del paciente observable (o). El texto contiene el nombre del paciente, su impacto Covid y sus síntomas (obtiene los síntomas del paciente). Finalmente, actualiza la etiqueta symptomLabel de la ventana.

-Clase *PacientObserverGui* modificada:



```
PacientObserverGUI.java
public class PacientObserverGUI extends JFrame implements Observer {  2 usages  jon

    private JPanel contentPane;  5 usages
    private final JLabel symptomLabel = new JLabel( text: "");  4 usages

    /**
     * Create the frame.
     */
    public PacientObserverGUI(Observable obs) {  4 usages  jon +1
        setTitle("Pacient symptoms");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setBounds( x: 650,  y: 100,  width: 200,  height: 300);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder( top: 5,  left: 5,  bottom: 5,  right: 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);
        symptomLabel.setBounds( x: 19,  y: 38,  width: 389,  height: 199);
        contentPane.add(symptomLabel);
        symptomLabel.setText("Still no symptoms");
        this.setVisible(true);

        obs.addObserver( o: this);
    }

    public void update(Observable   o,  Object  arg)    {  mikel
        Covid19Pacient  p=(Covid19Pacient)o;
        String  s="<html>   Pacient: <b>"+p.getName()+"</b> <br>";
        s=s+"Covid impact:  <b>"+p.covidImpact()+"</b><br><br>";
        s=s+"                    <br>    Symptoms:   <br>";
        Set set = p.getSymptoms();
        Iterator<Symptom> i= set.iterator();
        Symptom p2;
        while (i.hasNext()) {
            p2=i.next();
            s=s+ "          - " +   p2.toString()+"      "+p.getWeight(p2)+"<br>";
        }
        s=s+"</html>";
        symptomLabel.setText(s);
    }
}
```

Paso 3. Clase PacientSymptomGUI

Esta clase recoge a través de la pantalla ya mostrada, los síntomas de un paciente Covid19Pacient y lo actualiza. Los cambios a realizar son poner acciones en los listeners de ambos botones para añadir o eliminar un síntoma al objeto paciente, es decir:
p.addSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName(), Integer.parseInt(weightField.getText()));
p.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName());

-Clase *PacientSymptomGUI* modificada (solo los listeners):

PacientSymptomGUI.java

```
btnNewButton.addActionListener(new ActionListener() {  jon +1
    public void actionPerformed(ActionEvent e) {  jon +1
        errorLabel.setText(" ");
        if (Integer.parseInt(weightField.getText())<=3) {
        System.out.println("Symptom added :"+(Symptom)symptomComboBox.getSelectedItem());

        p.addSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName(), Integer.parseInt(weightField.getText()));

    } else errorLabel.setText("ERROR, Weight between [1..3]");

    }
});
```

PacientSymptomGUI.java

```
btnRemoveSymptom = new JButton( text: "Remove Symptom");
btnRemoveSymptom.addActionListener(new ActionListener() {  jon +1
    public void actionPerformed(ActionEvent e) {  jon +1
        errorLabel.setText(" ");

        System.out.println("Symptom removed :"+(Symptom)symptomComboBox.getSelectedItem());

        p.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName());
    }
});
```

Paso 4. Actualiza el programa principal

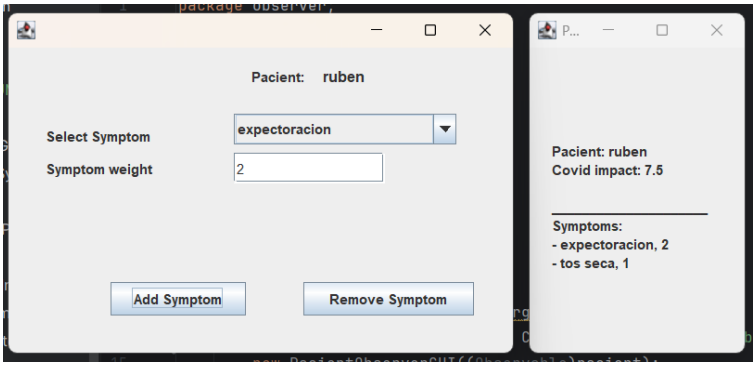
Para terminar, actualizaremos el programa principal para conectar todos los objetos. Ya está creado un objeto Covid19Pacient (model). A continuación, crearemos un objeto de la clase PacientObserverGUI pasándole como modelo a través de la constructora el objeto anterior, y para terminar crearemos la ventana PacientSymptomGUI para actualizar los síntomas del paciente:

```
public class Main {

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

                new PacientObserverGUI (pacient);

                new PacientSymptomGUI ((Covid19Pacient))pacient);

        }

}
```

Ejecuta el programa principal y comprueba que la aplicación funciona correctamente.

-Nuevo main:

-Resultado de la ejecución:



Se pide:

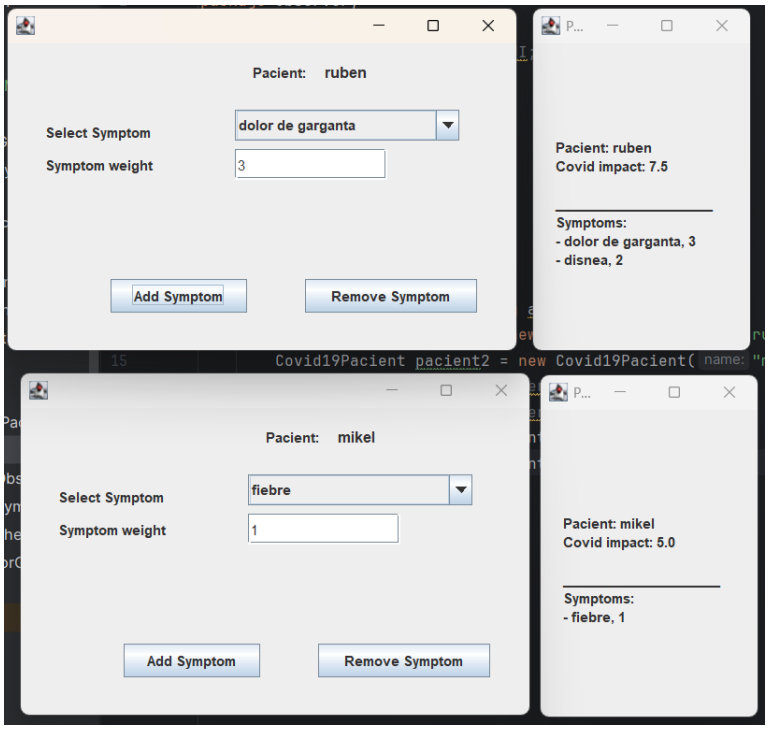
cambia el programa principal para crear 2 pacientes Covid19Pacient con sus interfaces PacientSymptomGUI y PacientObserverGUI.

-Clase *main* modificada:

Main.java

```
public class Main {  mikel +1 *

    /**
     * Launch the application.
     */
    public static void main(String args[]){  mikel
        Covid19Pacient pacient = new Covid19Pacient( name: "ruben",  years: 19);
        Covid19Pacient pacient2 = new Covid19Pacient( name: "mikel",  years: 20);
        new PacientObserverGUI((Observable)pacient);
        new PacientObserverGUI((Observable)pacient2);
        new PacientSymptomGUI(pacient);
        new PacientSymptomGUI(pacient2);
    }
}
```

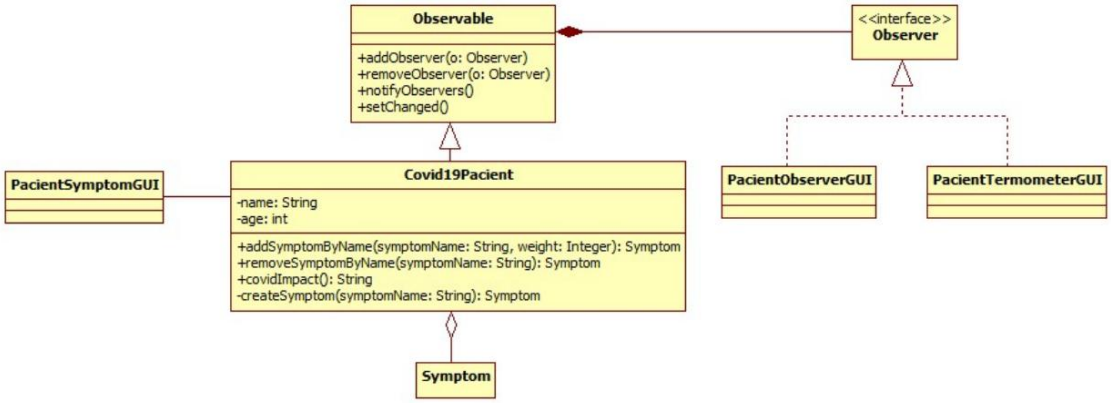
-Resultado de su ejecución:



2.2-Segundo prototipo:

En este prototipo queremos lograr que cuando se agregue un síntoma, se actualice la interfaz anterior, pero también el gráfico del termómetro con color según el valor del covidImpact (si covidImpact <= covidImapct <=10 y rojo si covidImpact > 10)

Para completar esta aplicación crearemos una nueva clase en la estructura del patrón Observer. Es de destacar que el patrón Observer nos permite extender la aplicación sin cambiar ninguna clase desarrollada previamente. Este es el nuevo diseño:



Solo hay que actualizar la clase PacientThermometerGUI. Con los pasos semejantes a los realizados para PacientObserverGUI y la implementación del método update es:

```
public void update(Observable o, Object args) {
        Covid19Pacient p=(Covid19Pacient) o;
        // Obtain the current covidImpact to paint int farenheit = (int)
        p.calcCovid19Impact();
        // temperature gauge update
        gauges.set(farenheit);
        gauges.repaint();
}
```

-Clase *PacientThermometerGUI* modificada:

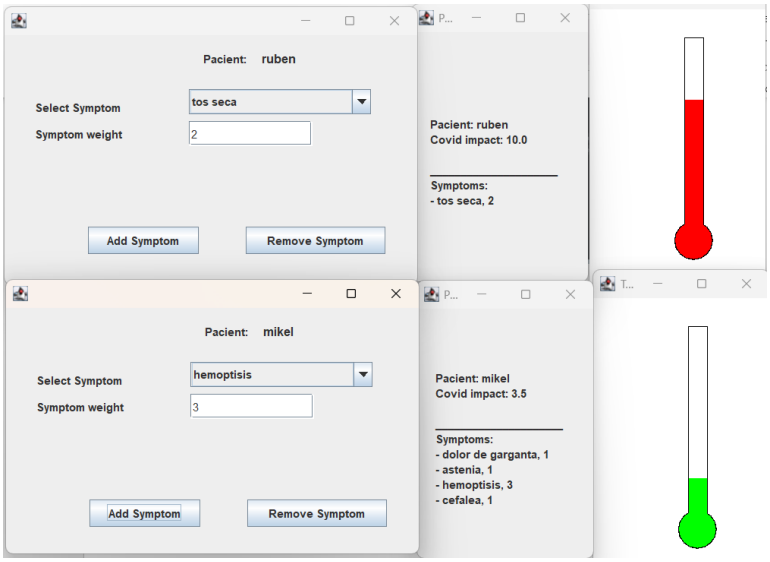
```java
PacientThermometerGUI.java
public class PacientThermometerGUI extends Frame implements Observer {
    private TemperatureCanvas gauges;  5 usages
    /**
     * @wbp.nonvisual location=119,71
     */
    private final JLabel label = new JLabel( text: "New label");  no usag

    public PacientThermometerGUI(Observable obs){  4 usages  jon +1
        super( title: "Temperature Gauge");
        Panel Top = new Panel();
        add( name: "North", Top);
        gauges = new TemperatureCanvas(0,15);
        gauges.setSize( width: 500, height: 280);
        add( name: "Center", gauges);
        setSize( width: 200,  height: 380);
        setLocation( x: 0,  y: 100);
        setVisible(true);

        obs.addObserver( o: this);
    }
```

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

-Clase *main* modificada:

```java
Main.java
public class Main {  mikel +1 *

    /**
     * Launch the application.
     */
    public static void main(String args[]){  mikel
        Covid19Pacient pacient = new Covid19Pacient( name: "ruben",  years: 19);
        Covid19Pacient pacient2 = new Covid19Pacient( name: "mikel",  years: 20);
        new PacientObserverGUI((Observable)pacient);
        new PacientObserverGUI((Observable)pacient2);
        new PacientSymptomGUI(pacient);
        new PacientSymptomGUI(pacient2);
        new PacientThermometerGUI((Observable)pacient);
        new PacientThermometerGUI((Observable)pacient2);
    }
}
```

-Resultado de su ejecución:



3. Patrón Adapter

Se pide:

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

-Clase *Covid19PacientTableModelAdapter* modificada:

Covid19PacientTableModelAdapter.java

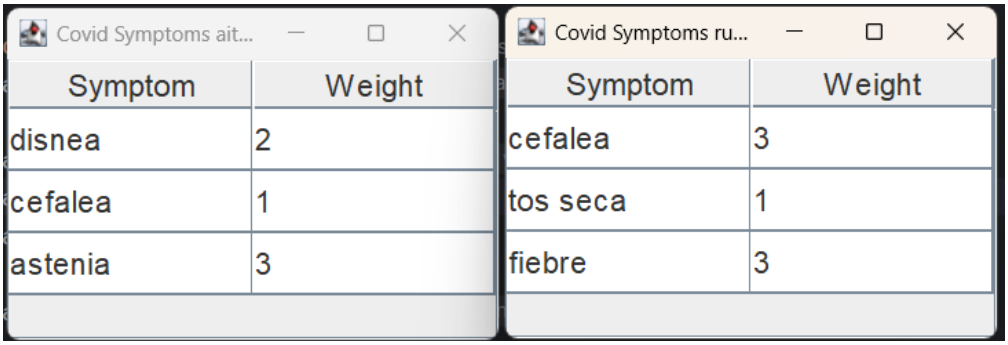
```
public class Covid19PacientTableModelAdapter extends AbstractTableModel {  4 usages
      protected Covid19Pacient pacient;  4 usages
      protected String[] columnNames =  1 usage
        new String[] {"Symptom", "Weight" };

      public Covid19PacientTableModelAdapter(Covid19Pacient p) {  2 usages  iturrioz +
          this.pacient=p;
      }

      public int getColumnCount() {  iturrioz +1
        // Challenge!
         return 2;
      }

      public String getColumnName(int i) {  iturrioz +1
        // Challenge!
          return columnNames[i];
      }

      public int getRowCount() {  iturrioz +1
        // Challenge!
          return pacient.getSymptoms().size();
      }

      public Object getValueAt(int row, int col) {  mBerasategui-ehu +1
        // Challenge!
          Set<Symptom> symptoms = pacient.getSymptoms();
          Symptom[] symptom_array = symptoms.toArray(new Symptom[symptoms.size()]);
          Symptom s = symptom_array[row];
          if (col == 0) {
              return s.getName();
          } else if (col == 1) {
              return pacient.getWeight(s);
          }
          else return null;
      }
```

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

-Clase *Main* Modificada:

```java
Main.java
public class Main {  jon +1

    public static void main(String[] args) {  mBerasategui-ehu +1
        Covid19Pacient pacient=new Covid19Pacient( name: "aitor",  years: 35);

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

        Covid19Pacient pacient2=new Covid19Pacient( name: "ruben",  years: 19);

        pacient2.addSymptomByName( symptom: "tos seca",  w: 1);
        pacient2.addSymptomByName( symptom: "fiebre",  w: 3);
        pacient2.addSymptomByName( symptom: "cefalea",  w: 3);

        ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient2);
        gui.setPreferredSize(
                  new java.awt.Dimension( width: 300,  height: 200));
        gui.setVisible(true);

        ShowPacientTableGUI gui2=new ShowPacientTableGUI(pacient);
        gui2.setPreferredSize(
                new java.awt.Dimension( width: 300,  height: 200));
        gui2.setVisible(true);


    }

}
```

-Resultado de su ejecución:



• (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.

-Clase *Covid19PacientTableModelAdapter* modificada:

Covid19PacientTableModelAdapter.java

```
import observer.Covid19Pacient;
import domain.Symptom;

import java.util.Observable;
import java.util.Observer;
import java.util.Set;

public class Covid19PacientTableModelAdapter extends AbstractTableModel implements Observer {
      protected Covid19Pacient pacient;  4 usages
      protected String[] columnNames =  1 usage
        new String[] {"Symptom", "Weight" };

      public Covid19PacientTableModelAdapter(Covid19Pacient p) {  2 usages  iturrioz +2
          this.pacient=p;
          ((Observable)p).addObserver( o: this);
      }
```

Covid19PacientTableModelAdapter.java

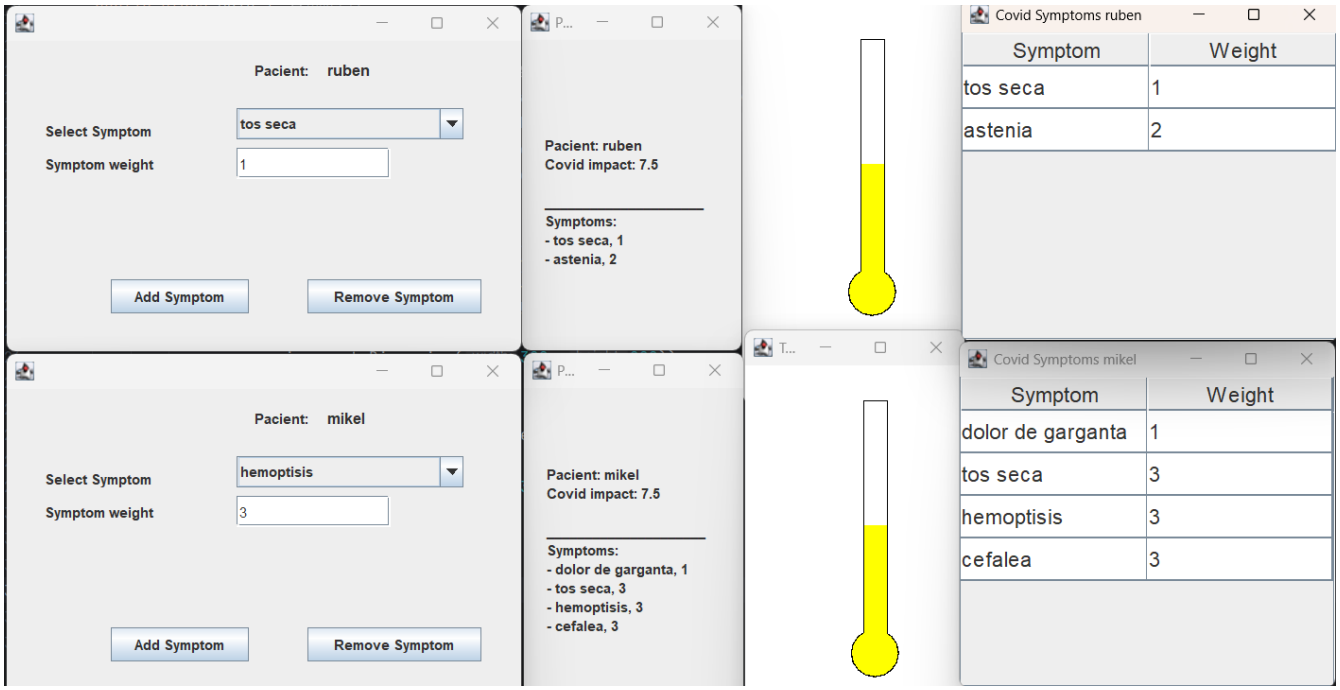
```
public void update(Observable o, Object arg) {
    fireTableDataChanged();
}
```

-Clase *Main (del package Observer)* modificada:

Main.java

```
public class Main {  mikel +2

    /**
     * Launch the application.
     */
    public static void main(String args[]){  mikel +1
        Covid19Pacient pacient = new Covid19Pacient( name: "ruben",  years: 19);
        Covid19Pacient pacient2 = new Covid19Pacient( name: "mikel",  years: 20);
        new PacientObserverGUI((Observable)pacient);
        new PacientObserverGUI((Observable)pacient2);
        new PacientSymptomGUI(pacient);
        new PacientSymptomGUI(pacient2);
        new PacientThermometerGUI((Observable)pacient);
        new PacientThermometerGUI((Observable)pacient2);
        ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient);
        gui.setPreferredSize(
                new java.awt.Dimension( width: 300,  height: 200));
        gui.setVisible(true);

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


}
```

-Resultado de su ejecución:



4. Patrón Observer

Se pide:

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.

-Clase *Main* modificada:

Main.java

```
public static void main(String[] args) {  jon +1
    Covid19Pacient p=new Covid19Pacient( name: "Ane",  years: 29);
    p.addSymptom(new Symptom( name: "s1",  covidImpact: 10,  severityIndex: 10),  w: 1);
    p.addSymptom(new Symptom( name: "s2",  covidImpact: 10,  severityIndex: 10),  w: 2);
    p.addSymptom(new Symptom( name: "s3",  covidImpact: 10,  severityIndex: 10),  w: 3);
    p.addSymptom(new Symptom( name: "s4",  covidImpact: 10,  severityIndex: 10),  w: 4);
    p.addSymptom(new Symptom( name: "s5",  covidImpact: 10,  severityIndex: 10),  w: 5);
```

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

-Clase *SeverityIndexComparator* modificada:

SeverityIndexComparator.java

```
public class SeverityIndexComparator implements Comparator<Symptom> {

    @Override  mBerasategui-ehu
    public int compare(Symptom s1, Symptom s2) {
        return s1.getSeverityIndex() - s2.getSeverityIndex();
    }
}
```

-Clase SymptomNameComparator modificada:

SymptomNameComparator.java

```
public class SymptomNameComparator implements Comparator<Symptom> {  1 usage  mBerasategui-ehu

    @Override  mBerasategui-ehu
    public int compare(Symptom s1, Symptom s2) { return s1.getName().compareTo(s2.getName()); }
}
```

• 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.

-**Clase *Covid19PacientInvertedIteratorAdapter* modificada:**

Covid19PacientInvertedIteratorAdapter.java

```
public class Covid19PacientInvertedIteratorAdapter implements InvertedIterator{

    private List<Symptom> symptoms;  4 usages
    private int position;  5 usages

    public Covid19PacientInvertedIteratorAdapter(Covid19Pacient pacient) {  1 us
        this.symptoms = new ArrayList<>(pacient.getSymptoms());
        this.position = symptoms.size();
    }

    @Override  2 usages  mBerasategui-ehu
    public Object previous() {
        if (!hasPrevious()) {
            return null;
        }
        position--;
        return symptoms.get(position);
    }

    @Override  3 usages  mBerasategui-ehu
    public boolean hasPrevious() { return position>0; }

    @Override  1 usage  mBerasategui-ehu
    public void goLast() { position = symptoms.size(); }
}
```

-Clase *Main* final:

Main.java

```java
public class Main {  mBerasategui-ehu +1

    public static void main(String[] args) {  mBerasategui-ehu +1
        Covid19Pacient p=new Covid19Pacient( name: "Ane",  years: 29);
        p.addSymptom(new Symptom( name: "s1",  covidImpact: 10,  severityIndex: 10),  w: 1);
        p.addSymptom(new Symptom( name: "s2",  covidImpact: 11,  severityIndex: 9),  w: 2);
        p.addSymptom(new Symptom( name: "s3",  covidImpact: 12,  severityIndex: 8),  w: 3);
        p.addSymptom(new Symptom( name: "s4",  covidImpact: 13,  severityIndex: 7),  w: 4);
        p.addSymptom(new Symptom( name: "s5",  covidImpact: 14,  severityIndex: 6),  w: 5);

        Iterator i=p.iterator();
        while(i.hasNext())
            System.out.println(i.next());
        System.out.println();


        System.out.println("Pruebas de SymptomNameComparator:");
        ArrayList<Symptom> symptomsList = new ArrayList<>(p.getSymptoms());

        Collections.sort(symptomsList, new SymptomNameComparator());
        for (Symptom s : symptomsList) {
            System.out.println(s.getName() + ", " + s.getCovidImpact() + ", " +s.getSeverityIndex());
        }
        System.out.println();


        System.out.println("Pruebas de SeverityIndexComparator:");
        ArrayList<Symptom> symptomsList2 = new ArrayList<>(p.getSymptoms());

        Collections.sort(symptomsList2, new SeverityIndexComparator());
        for (Symptom s : symptomsList2) {
            System.out.println(s.getName() + ", " + s.getCovidImpact() + ", " +s.getSeverityIndex());
        }
        System.out.println();


        System.out.println("Pruebas de Inverted Iterator:");
        InvertedIterator iterator = new Covid19PacientInvertedIteratorAdapter(p);

        while (iterator.hasPrevious()) {
            Symptom s = (Symptom) iterator.previous();
            System.out.println(s.getName() + ", " + s.getCovidImpact() + ", " +s.getSeverityIndex());
        }
        System.out.println();

    }

}
```

-Resultado de su ejecución (en pantalla):