LAB5 PATRONES

https://github.com/Wakte98/labpatterns

1.- Simple Factory

Hecho por Iñaki Inda

Hay muchas vulnerabilidades en esta aplicación:

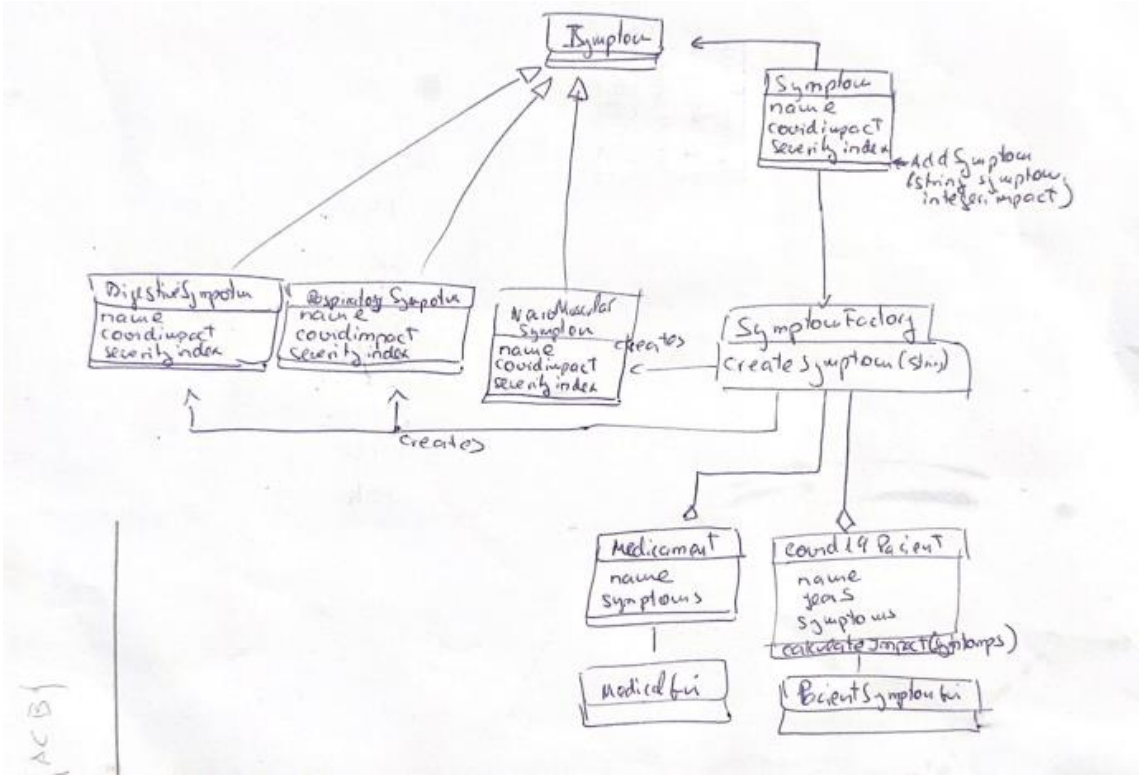
A. ¿Qué sucede si aparece un nuevo síntoma (por ejemplo, mareos)?

   En el diseño actual, si aparece un nuevo síntoma, no hay un mecanismo claro para manejarlo sin modificar las clases existentes (no cumple OCP). Para agregar un nuevo síntoma, necesitarías modificar el método createSymptom en la clase Covid19Pacient y cualquier otra clase que necesite crear síntomas. Esto no es escalable y podría generar problemas de mantenibilidad.

B. ¿Cómo se puede crear un nuevo síntoma sin cambiar las clases existentes (principio OCP)?
   Para cumplir con el principio OCP, hay que implementar el patron "Factory Method". Primero crear la interfaz SymptomFactory, que define un método para crear síntomas. Luego, cada clase que necesite crear síntomas implementa la una versión específica .

C. ¿Cuántas responsabilidades tienen las clases de Covid19Pacient y Medicament (principio SRP)?
   Mas de 2, responsables de administrar síntomas, crear síntomas y realizar otras operaciones relacionadas con los pacientes o medicamentos

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.



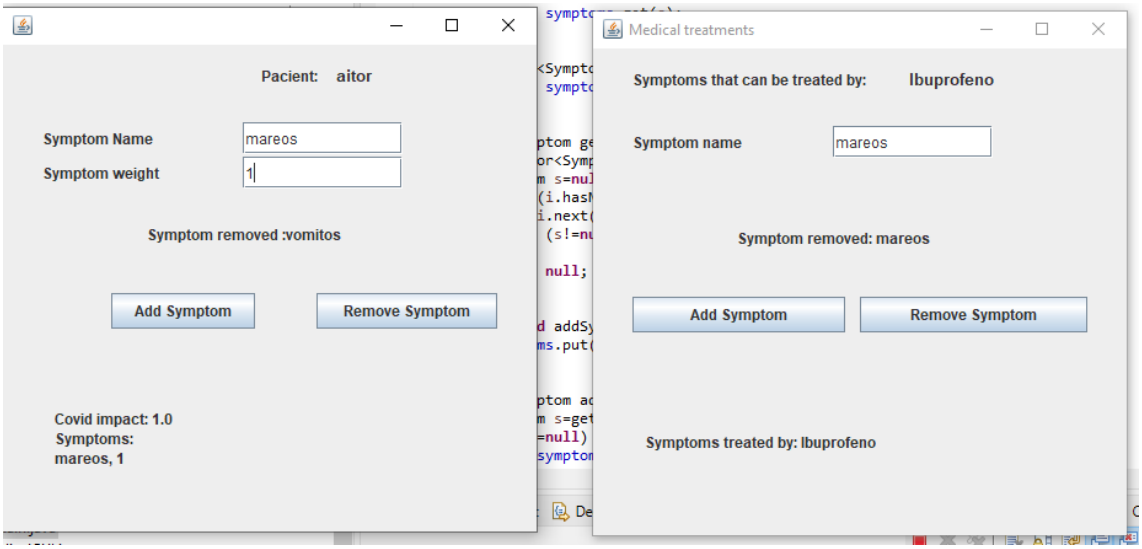
- He creado la interfaz, SymptomFactory, que define un método addSymptom() para crear objetos de tipo Symptom.

- Se agregaron tres clases nuevas, NeuroMuscularSymptomFactory, DigestiveSymptomFactory y RespiratorySymptomFactory, que implementan la interfaz SymptomFactory. Estas clases crean objetos de tipos específicos de síntomas.

- Se modificó la clase Symptom para eliminar el método createSymptom(). Este método ahora está implementado en las clases NeuroMuscularSymptomFactory, DigestiveSymptomFactory y RespiratorySymptomFactory.

- •Se modificó la clase Covid19Pacient para eliminar el método addSymptom(). Este método ahora está implementado en la clase SymptomManager.

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



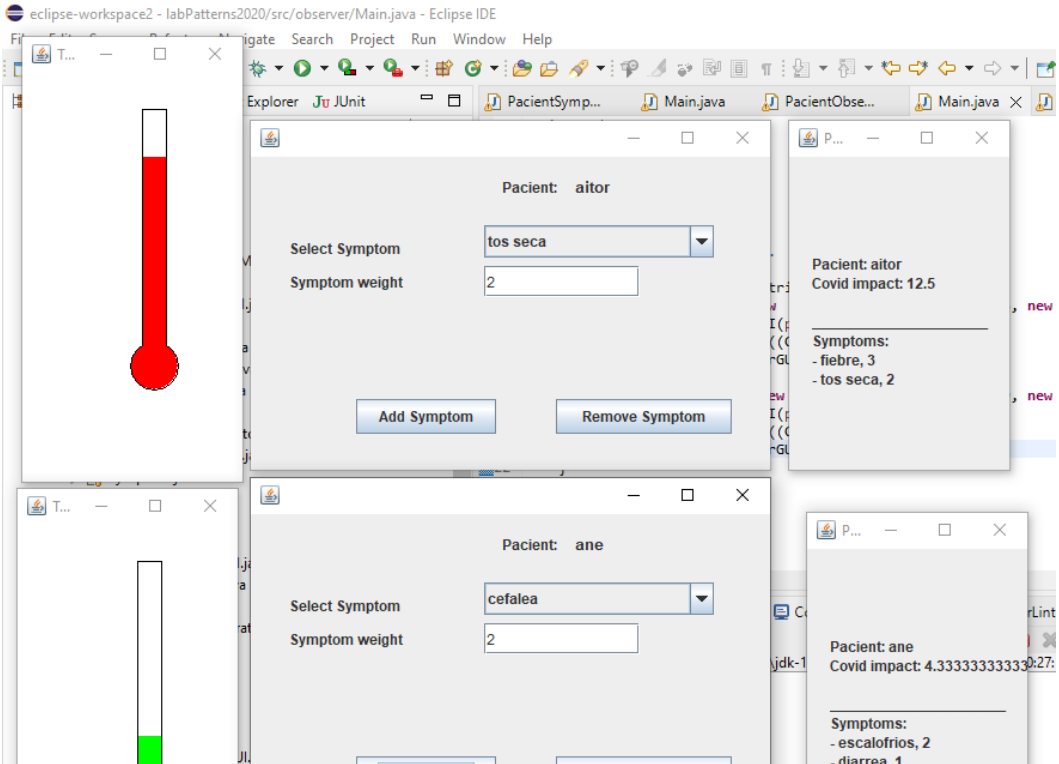
Realizado commit: LAB TAREAS IÑAKI INDA

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)

   Mediante un patron Singleton. Esto garantiza que una clase tenga una única instancia. En la clase SymptomFactor habría que crear una instancia y un vector/map de síntomas y cada vez que se crea un nuevo síntoma recorrer ese vector para devolverlo en el caso de que ya exista o crearlo en el caso de que no.

2. Patrón Observer

Hecho por Iñaki Inda



2.- Patrón Observer

3.- Patrón Adapter

Hecho por Josu Sáez

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

   Se ha añadido una nueva variable a la clase, un vector de síntomas obtenido del keyset que devuelve getSimptoms de la clase Covid19Pacient.

```
public class Covid19PacientTableModelAdapter extends AbstractTableModel {

      protected Covid19Pacient pacient;

      protected String[] columnNames =
        new String[] {"Symptom", "Weight" };

      protected List<Symptom> symptoms = new Vector<Symptom>();

      public Covid19PacientTableModelAdapter(Covid19Pacient p) {
        this.pacient=p;
        Iterator<Symptom> i= pacient.getSymptoms().iterator();
          while(i.hasNext()) {
              this.symptoms.add(i.next());
          }
      }
```

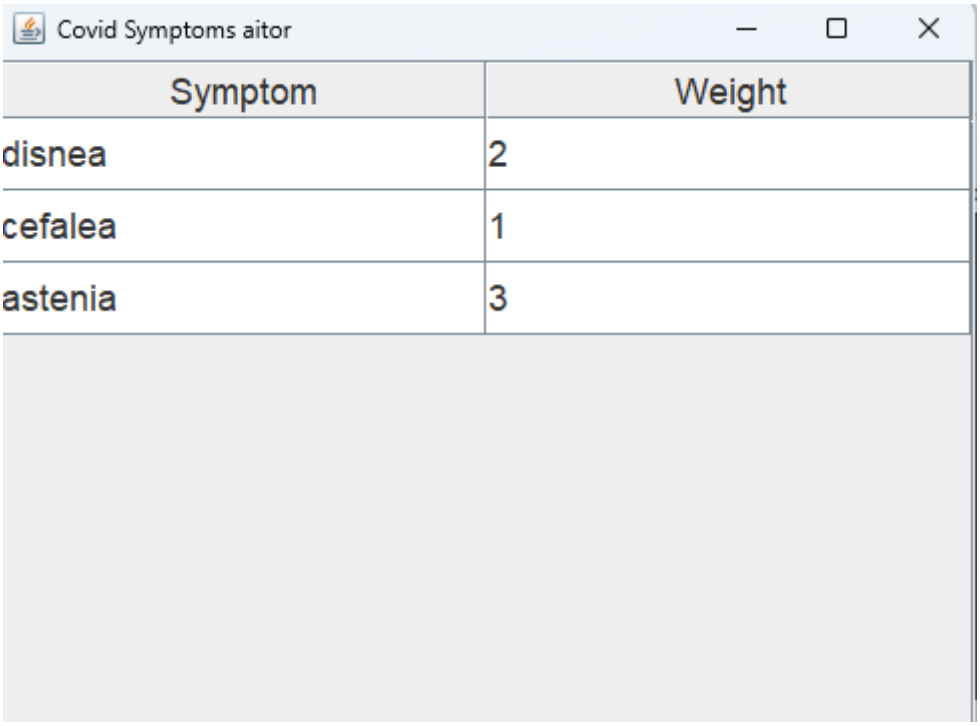
   Métodos getColumnCount y getColumnName:

```
      public int getColumnCount() {
        // Challenge!
         return columnNames.length;
      }

      public String getColumnName(int i) {
          return columnNames[i];
      }
```

Métodos getRowCount y getValueAt:

```java
public int getRowCount() {
    return pacient.getSymptoms().size();
}

public Object getValueAt(int row, int col) {
    Symptom s = symptoms.get(row);
    if(col==0) {
        return s.getName();
    }
    else return pacient.getWeight(s);
}
```

Resultado de la ejecución:



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

He añadido un nuevo paciente y he llamado a ShowPacientTableGUI pasándole como parámetro una lista que contiene a estos dos pacientes:

```
public class Main {

    public static void main(String[] args) {
        Covid19Pacient pacient=new Covid19Pacient("Aitor", 35, new SymptomFactory());
        Covid19Pacient pacient2=new Covid19Pacient("Ainara", 26, new SymptomFactory());

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

        pacient2.addSymptomByName("diarrea", 1);
        pacient2.addSymptomByName("astenia", 3);
        pacient2.addSymptomByName("nauseas", 1);

        List<Covid19Pacient> pacients = new Vector<Covid19Pacient>();
        pacients.add(pacient);
        pacients.add(pacient2);

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


    }

}
```

Para ello he tenido que modificar la clase ShowPacientTableGUI:

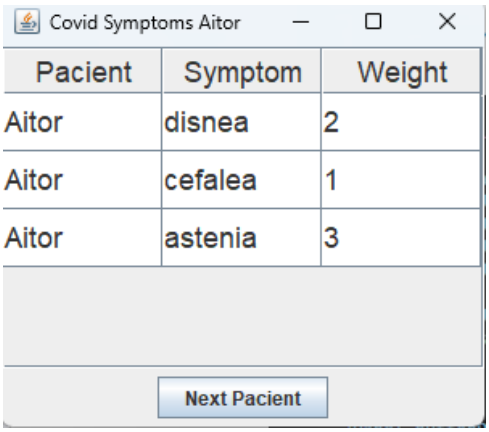
```
public class ShowPacientTableGUI extends JFrame{

JTable table;
Covid19Pacient pacient;
List<Covid19Pacient> pacients;
int index;

public ShowPacientTableGUI(Covid19Pacient pacient) {
      this.setTitle("Covid Symptoms "+pacient.getName());

      this.pacient=pacient;

      setFonts();

      TableModel tm=new Covid19PacientTableModelAdapter(pacient);
      table = new JTable(tm);
      table.setRowHeight(36);
      JScrollPane pane = new JScrollPane(table);
      pane.setPreferredSize(
        new java.awt.Dimension(300, 200));
      this.getContentPane().add(pane);

      addNextButton();
}

public ShowPacientTableGUI(List<Covid19Pacient> pacients, int index) {
    this(pacients.get(index));
    this.index=index;
    this.pacients=pacients;

    addNextButton();
}

public ShowPacientTableGUI(List<Covid19Pacient> pacients) {
    this(pacients.get(0));
    this.index = 0;
    this.pacients=pacients;

    addNextButton();
}
```

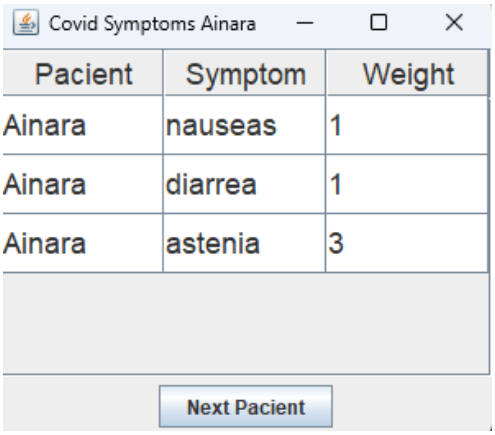
He creado dos nuevas constructoras que reciben una lista de pacientes, y estas llaman a la constructora original;

He creado además un botón debajo de la tabla para poder cambiar entre pacientes:

```java
private void addNextButton() {
      JButton nextButton = new JButton("Next Pacient");
      nextButton.addActionListener(new ActionListener() {
          @Override
          public void actionPerformed(ActionEvent e) {
              index = (index + 1) % pacients.size();
              ShowPacientTableGUI nextGUI = new ShowPacientTableGUI(pacients, index);
              nextGUI.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
              nextGUI.pack();
              nextGUI.setLocationRelativeTo(null);
              nextGUI.setPreferredSize(
                      new java.awt.Dimension(300, 200));
              nextGUI.setVisible(true);
              dispose();
          }
      });

      JPanel buttonPanel = new JPanel();
      buttonPanel.add(nextButton);
      this.getContentPane().add(buttonPanel, BorderLayout.SOUTH);
  }
```

El resultado de la ejecución es el siguiente:





4.- Patrón Iterator y Adapter

Hecho por Josu Sáez

Clase principal Main del paquete adapter:

```java
public class Main {
    public static void main(String[] args) {
        Covid19Pacient p=new Covid19Pacient("Amagoia", 23, new SymptomFactory());
        p.addSymptom(new Symptom("nauseas", 10, 10), 1);
        p.addSymptom(new Symptom("astenia", 10, 12), 2);
        p.addSymptom(new Symptom("expectoracion", 10, 11), 5);
        p.addSymptom(new Symptom("vomitos", 10, 10), 3);
        p.addSymptom(new Symptom("mialgia", 10, 15), 4);


        //Instancias de los comparadores
        SymptomNameComparator nameComparator = new SymptomNameComparator();
        SeverityIndexComparator severityComparator = new SeverityIndexComparator();

        //Instancio el adapter
        InvertedIterator adapter = new Covid19PacientInvertedIteratorAdapter(p);

        //Imprimimos por orden de nombre
        System.out.println("Symptoms sorted by name: ");
        Iterator i=Sorting.sortedIterator(adapter, nameComparator);
        while(i.hasNext()) {
            System.out.println(i.next());
        }

        //Ordenamos por orden de severidad
        System.out.println("\nSymptoms sorted by severity index: ");
        i=Sorting.sortedIterator(adapter, severityComparator);
        while(i.hasNext())
            System.out.println(i.next());
    }

}
```

Clases del paquete comparator:

- Clase SymptomNameComparator:

```java
public class SymptomNameComparator implements Comparator<Object>{
    public int compare(Object o1, Object o2){
        Symptom s1=(Symptom) o1;
        Symptom s2=(Symptom) o2;
        return s1.getName().compareTo(s2.getName());
    }
}
```

- Clase SeverityIndexComparator:

```java
public class SeverityIndexComparator implements Comparator<Object>{
    public int compare (Object o1, Object o2) {
        Symptom s1=(Symptom) o1;
        Symptom s2=(Symptom) o2;
        return Integer.compare(s1.getSeverityIndex(), s2.getSeverityIndex());
    }
}
```

Adaptador de InvertedIterator soble la clase Covid19Pacient:

```java
public class Covid19PacientInvertedIteratorAdapter implements InvertedIterator{

    protected Covid19Pacient pacient;
    protected Symptom[] symptoms;
    protected int lastIndex;
    protected int position;

    public Covid19PacientInvertedIteratorAdapter(Covid19Pacient pacient) {
        this.pacient = pacient;
        this.symptoms = pacient.getSymptoms().toArray(new Symptom[pacient.getSymptoms().size()]);
        this.lastIndex = symptoms.length;
    }

    @Override
    public Object previous() {
        if(position>0) {
            this.position-=1;
            return symptoms[position];
        }else return null;
    }

    @Override
    public boolean hasPrevious() {
        return position>0;
    }

    @Override
    public void goLast() {
        this.position = lastIndex;

    }

}
```

Resultado de la ejecución:

```
Console × SonarLint Report SonarLint Issue Locations
<terminated> Main (6) [Java Application] C:\Program Files\Java\j
Symptoms sorted by name:
astenia
expectoracion
mialgia
nauseas
vomitos

Symptoms sorted by severity index:
vomitos
nauseas
expectoracion
astenia
mialgia
```

Los resultados son los esperados para ambos comparadores.