LABORATORIO PATRONES DE DISEÑO

IS2

2023/2024

Proyecto Bets

Autores:
Unai Artano
Asier Contreras
Martin Ian Horsfield

ENLACE AL PROYECTO

https://github.com/UnaiAD22/labpatterns

1.- Simple Factory

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

- No va a dejar añadirlo ya que sólo deja añadir los síntomas que existan en la lista de síntomas del método createSymptom

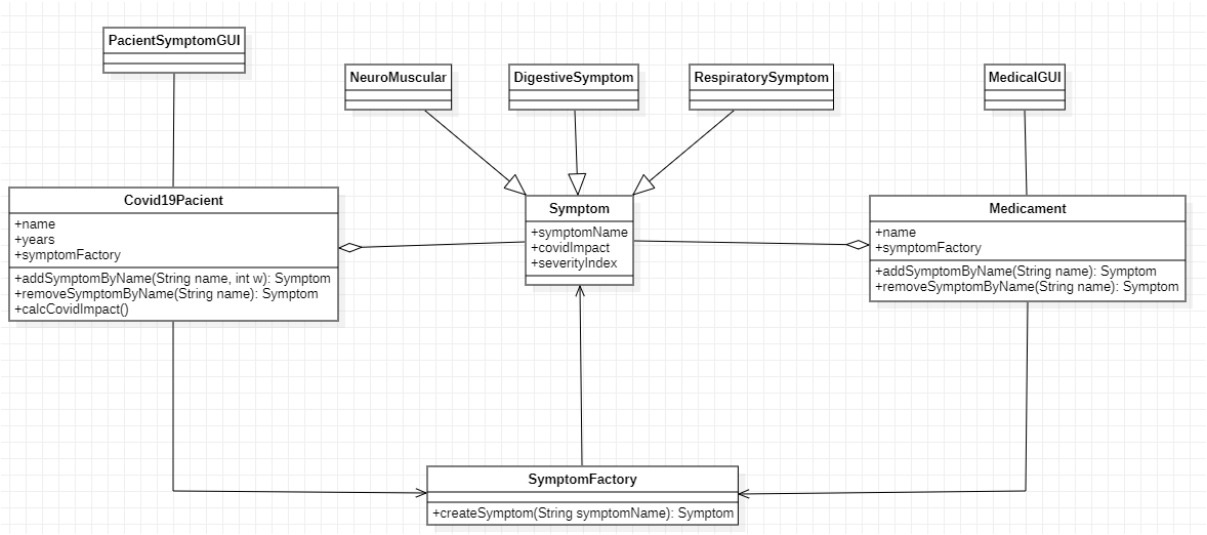
B. ¿Cómo se puede crear un nuevo síntoma sin cambiar las clases existentes (principio OCP)?

- Se puede crear un síntoma nuevo con la clase Symptom pero no se actualizará en las listas de createSymptom de Medicament.

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

- Además de las constructoras y getters, se encargan de añadir síntomas, eliminar síntomas y crear los síntomas. La clase Covid19Pacient además se encarga también de calcular el impactoCovid que tendrá esa persona según los síntomas que tenga añadidos. Por lo que no cumplen el principio de SRP (Single Responsibility Principle).

1.- UML



2.- Añadir un nuevo síntoma: mareos asociado a un impacto de 1.

```
List<String> impact1 = Arrays.asList("nauseas", "vomitos", "congestion nasal","diarrea","hemoptisis","congestion conjuntival", "mareos");
List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 3.0, 1.0);
```

```
List<String> neuroMuscularSymptom=Arrays.asList("fiebre", "astenia", "cefalea", "mialgia","escalofrios", "mareos");
```

3. Adaptar la clase Factory para que los objetos Symptom sean únicos.

Hemos creado un HashMap que comprobará si el síntoma ya ha sido instanciado anteriormente, y si lo ha sido devolverá la instancia del síntoma. En caso de que sea la primera vez que se instancia, lo añadirá al HashMap.

```
HashMap<String, Symptom> hashMapSymptom = new HashMap<>();

public Symptom createSymptom(String symptomName) {

    if(hashMapSymptom.containsKey(symptomName)) {
        return hashMapSymptom.get(symptomName);
    }
```

```
if (impact!=0)  {
    if (digestiveSymptom.contains(symptomName)) {
        Symptom ds = new DigestiveSymptom(symptomName,(int)index, impact);
        hashMapSymptom.put(symptomName, ds);
        return ds;
    }
    if (neuroMuscularSymptom.contains(symptomName)) {
        Symptom nms = new NeuroMuscularSymptom(symptomName,(int)index, impact);
        hashMapSymptom.put(symptomName, nms);
        return nms;
    }
    if (respiratorySymptom.contains(symptomName)) {
        Symptom rs = new RespiratorySymptom(symptomName,(int)index, impact);
        hashMapSymptom.put(symptomName, rs);
        return rs;
    }
}
```

2.- Patrón Observer

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

```
public class Covid19Pacient extends Observable{
    private String  name;
    private int age;
    private Map<Symptom,Integer> symptoms=new HashMap<Symptom,Integer>();
    private SymptomFactory symptomFactory;

    public Covid19Pacient(String name, int years, SymptomFactory sf) {
        this.name = name;
        this.age = years;
        symptomFactory=sf;
    }

    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) {
        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){
    Symptom s=getSymptomByName(symptom);
    if (s==null) {
        s=symptomFactory.createSymptom(symptom);
        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;
}
public Iterator iterator() {
    return new Covid19PacientIterator(this.symptoms.keySet());
}

public double covidImpact() {
    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;
    }
}
```

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);

```
public PacientObserverGUI(Observable obs) {
    setTitle("Pacient symptoms");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(650, 100, 200, 300);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(null);
    symptomLabel.setBounds(19, 38, 389, 199);
    contentPane.add(symptomLabel);
    symptomLabel.setText("Still no symptoms");
    this.setVisible(true);

    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.

```
@Override
public void update(Observable o, Object arg) {
    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>";
    Iterator<Symptom> i = p.getSymptoms().iterator();
    Symptom p2;
    while (i.hasNext()) {
        p2 = i.next();
        s = s + "          - " + p2.toString() + ".      " + p.getWeight(p2) + "<br>";
    }
    s = s + "</html>";
    symptomLabel.setText(s);
}
```

3.- Patrón Adapter

- 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" };
      Vector<Symptom> symptoms = new Vector<Symptom>();

      public Covid19PacientTableModelAdapter(Covid19Pacient p) {
        this.pacient=p;
        Iterator<Symptom> iP = pacient.getSymptoms().iterator(); //Hemos serializado los sintomas del cliente
        while(iP.hasNext()) {
            symptoms.add(iP.next());
        }
      }

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

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

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

      public Object getValueAt(int row, int col) {
        // Challenge!
          //return "value";
          Symptom s = symptoms.get(row);
          if(col==0) return s.getName(); //Devolver sintoma
          if(col==1) return pacient.getWeight(s); //Devolver el peso del sintoma del paciente
          return null;
      }
    }
```

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

```
Covid19Pacient pacient2=new Covid19Pacient("ane", 29, new SymptomFactory());
pacient2.addSymptomByName("fiebre", 3);
pacient2.addSymptomByName("dolor de garganta", 2);
pacient2.addSymptomByName("expectoracion", 1);

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

4.- Patrón Iterator y Adapter

Para este ejercicio se han creado 2 clases nuevas que implementan la interfaz Comparator. Esas clases son SeverityIndexComparator, que ordenará según el índice de severidad y SymtomNameComparator, que ordenará por orden alfabético. Además se ha creado la clase Covid19Adapter, que implementará la interfaz InvertedIterator.

SeverityIndexComparator:

```
package adapter;

import java.util.Comparator;

public class SeverityIndexComparator implements Comparator<Symptom> {
    @Override
    public int compare(Symptom s1, Symptom s2) {
        return Integer.compare(s1.getSeverityIndex(), s2.getSeverityIndex());
    }

}
```

SymtomNameComparator:

```
package adapter;
import java.util.Comparator;

public class SymptomNameComparator implements Comparator<Symptom>{
    @Override
    public int compare(Symptom s1, Symptom s2) {
        return s1.getName().compareTo(s2.getName());
    }
}
```

Covid19Adapter:

```
package adapter;

import java.util.Set;

public class Covid19Adapter implements InvertedIterator {
    private Set<Symptom> symptoms;
    private Symptom[] symptomArray;
    private int currentIndex;

    public Covid19Adapter(Covid19Pacient p) {
        symptoms = p.getSymptoms();
        symptomArray = symptoms.toArray(new Symptom[0]);
        currentIndex = symptomArray.length - 1;
    }

    @Override
    public Object previous() {
        if (currentIndex >= 0) {
            return symptomArray[currentIndex--];
        }
        return null;
    }

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

    @Override
    public void goLast() {
        currentIndex = symptomArray.length - 1;
    }

}
```

Finalmente se ha hecho una clase main donde se ha ejecutado y comprobado el funcionamiento obteniendo el siguiente resultado en la consola:

```
Ordenado por el nombre del sintoma:
Sintoma: Astenia, Severidad: 6
Sintoma: Dolor abdominal, Severidad: 2
Sintoma: Fiebre, Severidad: 4
Sintoma: Tos, Severidad: 3
Sintoma: Vomito, Severidad: 3

Ordenado de menor a mayor por el indice de severidad:
Sintoma: Dolor abdominal, Severidad: 2
Sintoma: Tos, Severidad: 3
Sintoma: Vomito, Severidad: 3
Sintoma: Fiebre, Severidad: 4
Sintoma: Astenia, Severidad: 6
```