INGENIERÍA DEL SOFTWARE

DISEÑO AVANZADO – PRINCIPIOS SOLID

AGUSTÍN BELTRÁN DE HEREDIA

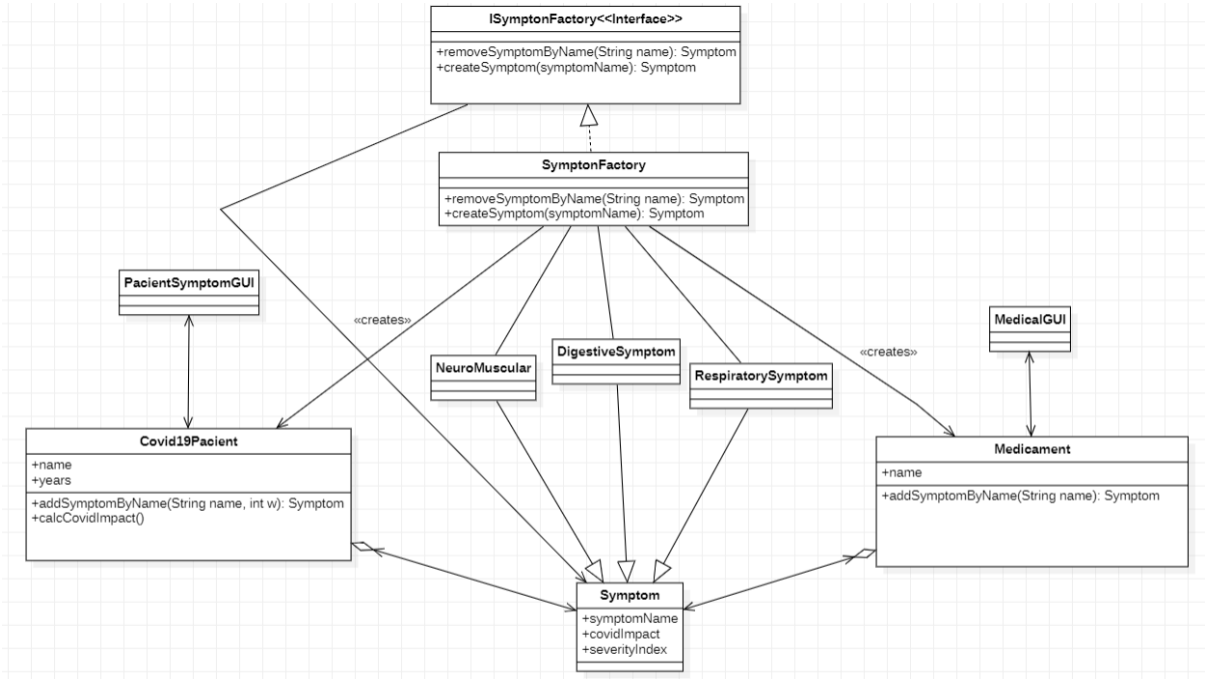
GAIZKA ACEDO

MADDI LOPEZ

09/11/2025

Simple factory

1. DIAGRAMA UML



2. Implementación de la aplicación
3. Adaptación de la clase Factory

```java
package domain;

import java.util.*;

public class SymptonFactory implements ISymptonFactory {

    private static Map<String, Symptom> cache = new HashMap<>();

    public SymptonFactory() {}

    @Override
    public Symptom createSymptom(String symptomName) {
        if (cache.containsKey(symptomName)) {
            return cache.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");
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 0.5, 1.0);

        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 s = null;

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

Hemos modificado la clase SymptonFactory para que cada síntoma del sistema se cree solo una vez. Para conseguirlo, hemos añadido un mapa que guarda los síntomas que

ya se han creado, usando su nombre como clave. Cuando se llama al método createSymptom, primero se comprueba si el síntoma ya está en el mapa. Si ya existe, se devuelve directamente. Si no existe, se crea con el código original, se guarda en el mapa y luego se devuelve. De esta forma, cada síntoma solo tiene una única instancia en todo el programa, aunque sea utilizado por distintos pacientes o medicamentos.

Patrón Observer

1. Clase CovidPacient

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

    public Covid19Pacient(String name, int years, ISymptonFactory factory) {
        this.name = name;
        this.age = years;
        this.factory=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) {
        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 = factory.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("Symptom to remove: "+s);
        if (s != null) {
            symptoms.remove(s);
            setChanged();
            notifyObservers();
        }
        return s;
    }

    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;
    }
}
```

2. PacientObserverGUI

```java
package observer;

import java.util.Iterator;
import java.util.Observable;
import java.util.Observer;

import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import domain.Symptom;

import javax.swing.JLabel;

public class PacientObserverGUI extends JFrame implements Observer{

    private JPanel contentPane;
    private final JLabel symptomLabel = new JLabel("");

    /**
     * Create the frame.
     */
    public PacientObserverGUI() {
        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);
    }
    public PacientObserverGUI(Observable obs) {
        this();
        obs.addObserver(this);
    }
     @Override
     public void update(Observable o, Object arg) {
         Covid19Pacient p = (Covid19Pacient) o;
         String s = "<html> Pacient: <b>" + p.getName() + "</b> <br>";
         s += "Covid impact: <b>" + p.covidImpact() + "</b><br><br>";
         s += " _____ <br> Symptoms: <br>";

         Iterator<Symptom> i = p.getSymptoms().iterator();
         while (i.hasNext()) {
             Symptom sym = i.next();
             s += " - " + sym.toString() + ", " + p.getWeight(sym) + "<br>";
         }
           s += "</html>";

           symptomLabel.setText(s);
        }

}
```

La clase PacientObserverGUI es una ventana que muestra la información de un paciente. Implementa Observer para recibir notificaciones de un Covid19Pacient. Se suscribe al paciente en el constructor y, cada vez que el paciente cambia, el método update actualiza el JLabel con el nombre, el impacto Covid y la lista de síntomas. También incluye un serialVersionUID por buenas prácticas.

3. PacientSymptomGUI

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

        btnNewButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                errorLabel.setText(" ");
                try {
                    int weight = Integer.parseInt(weightField.getText());
                    if (weight >= 1 && weight <= 3) {
                        Symptom selected = (Symptom) symptomComboBox.getSelectedItem();
                        if (selected != null) {
                            p.addSymptomByName(selected.getName(), weight);
                            System.out.println("Symptom added: " + selected);
                        }
                    } else {
                        errorLabel.setText("ERROR, Weight must be between [1..3]");
                    }
                } catch (NumberFormatException ex) {
                    errorLabel.setText("ERROR, Weight must be a number");
                }
            }
        });

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

    }
});
btnNewButton.setBounds(88, 202, 117, 29);
contentPane.add(btnNewButton);

btnRemoveSymptom = new JButton("Remove Symptom");
btnRemoveSymptom.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        errorLabel.setText(" ");

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

        btnRemoveSymptom.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                errorLabel.setText(" ");
                Symptom selected = (Symptom) symptomComboBox.getSelectedItem();
                if (selected != null) {
                    p.removeSymptomByName(selected.getName());
                    System.out.println("Symptom removed: " + selected);
                }
            }
        });

    }
});
btnRemoveSymptom.setBounds(255, 202, 147, 29);
```

Hemos completado los listeners de los botones de la clase PacientSymptomGUI para que interactúen con el paciente (Covid19Pacient). En el botón “Add Symptom”, hemos tomado el síntoma seleccionado y el valor de peso introducido, comprobamos que el peso esté entre 1 y 3, y llamamos a p.addSymptomByName(...), lo que añade el síntoma al paciente y notifica automáticamente a todos los observers (como la GUI que muestra los síntomas). En el botón “Remove Symptom”, hemos tomado el síntoma seleccionado y llamamos a p.removeSymptomByName(...), eliminándolo del paciente y notificando también a los observers. Con estos cambios, cualquier modificación que realicemos desde esta ventana se reflejará automáticamente en las interfaces que estén observando al paciente, cumpliendo el patrón Observer.

4. **Programa principal**

```java
package observer;

import domain.ISymptonFactory;
import domain.SymptonFactory;

public class Main {
    public static void main(String[] args) {

        ISymptonFactory factory = new SymptonFactory();

        // Primer paciente
        Covid19Pacient pacient1 = new Covid19Pacient("Aitor", 35, factory);
        new PacientObserverGUI(pacient1);   // GUI que observa al paciente 1
        new PacientSymptomGUI(pacient1);    // GUI para modificar síntomas del paciente 1

        // Segundo paciente
        Covid19Pacient pacient2 = new Covid19Pacient("Maddi", 28, factory);
        new PacientObserverGUI(pacient2);   // GUI que observa al paciente 2
        new PacientSymptomGUI(pacient2);    // GUI para modificar síntomas del paciente 2
    }
}
```

Patrón Adapter:

La implementación está en el propio código, hemos implementado los métodos que pedía en la clase Covid19PacientTableModelAdapter. También hemos fijado un tamaño mínimo en ShowPacientTableGUI para cuando ejecutase el código se viese sin tener que agrandarlo manualmente:

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

      this.pacient=pacient;

      setFonts();
      this.setSize(600, 400);
```

Patrón Iterator y Adapter

Para su realización hemos modificado la carpeta adapter, añadiendo las siguientes clases:

- Covid19PacientAdapter: Para crear el patrón adapter sobre la clase Covid19Pacient (implementando la interfaz InvertedIterator en la misma).

```java
package adapter;

import java.util.ArrayList;

public class Covid19PacientAdapter implements InvertedIterator{

    List<Symptom> symptoms;
    int position;

    public Covid19PacientAdapter(Covid19Pacient p) {
        this.symptoms = new ArrayList<>(p.getSymptoms());

        goLast();
    }

    @Override
    public Object previous() {
        Symptom s = symptoms.get(position);
        position --;
        return s;
    }

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

    @Override
    public void goLast() {
        position = symptoms.size()-1;

    }

}
```

- Main: clase Main donde realizamos todo lo que pedía (crear paciente con 5 sintomas, probar que se realice correctamente).

```java
package adapter;

import java.util.Comparator;

public class Main {

    public static void main(String[] args) {
        ISymptonFactory symptonFactory = new SymptonFactory();
        Covid19Pacient p = new Covid19Pacient("Agus", 21, symptonFactory);
        p.addSymptom(new Symptom("Sintoma1", 7, 4), 1);
        p.addSymptom(new Symptom("Sintoma2", 9, 3), 2);
        p.addSymptom(new Symptom("Sintoma3", 3, 7), 3);
        p.addSymptom(new Symptom("Sintoma4", 1, 1), 2);
        p.addSymptom(new Symptom("Sintoma5", 8, 9), 1);

        InvertedIterator pacientAdapter = new Covid19PacientAdapter(p);

        Comparator comparatorSymtomName = new SymptomNameComparator();
        Comparator comparatorSeverityIndex = new SeverityIndexComparator();

        //Ordenado por nombre
        System.out.println("Ordenado por name");
        Iterator sortedByName = Sorting.sortedIterator(pacientAdapter, comparatorSymtomName);
        while (sortedByName.hasNext()) {
            System.out.println(sortedByName.next());
        }


        //Ordenado por severity index
        System.out.println("\nOrdenado por severity");
        Iterator sortedBySeverity = Sorting.sortedIterator(pacientAdapter, comparatorSeverityIndex);
        while (sortedBySeverity.hasNext()) {
            System.out.println(sortedBySeverity.next());
        }
    }

}
```

- SeverityIndexComparator: donde se implementa el método compare para los severityIndex.

```java
package adapter;

import java.util.Comparator;

public class SeverityIndexComparator implements Comparator<Object>{

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

        return Double.compare(s1.getSeverityIndex(),s2.getSeverityIndex());
    }

}
```

- SymtomNameComparator: donde se implementa el método compare para los nombres

```java
package adapter;

import java.util.Comparator;

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());
    }

}
```

Todo esto sin modificar ni Sorting.sortedIterator ni la clase Covid19Pacient.