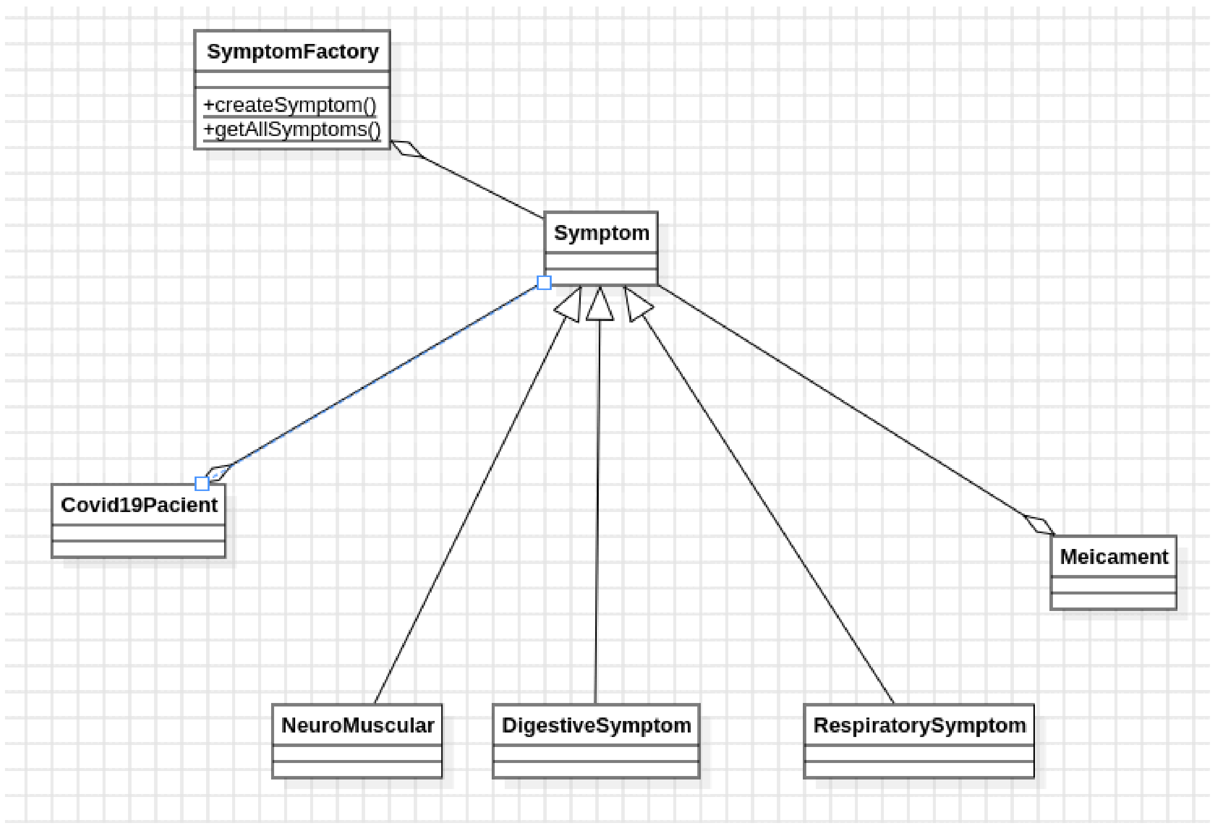
PATROIAK

Oier, Aretz eta Manex.

SIMPLE FACTORY

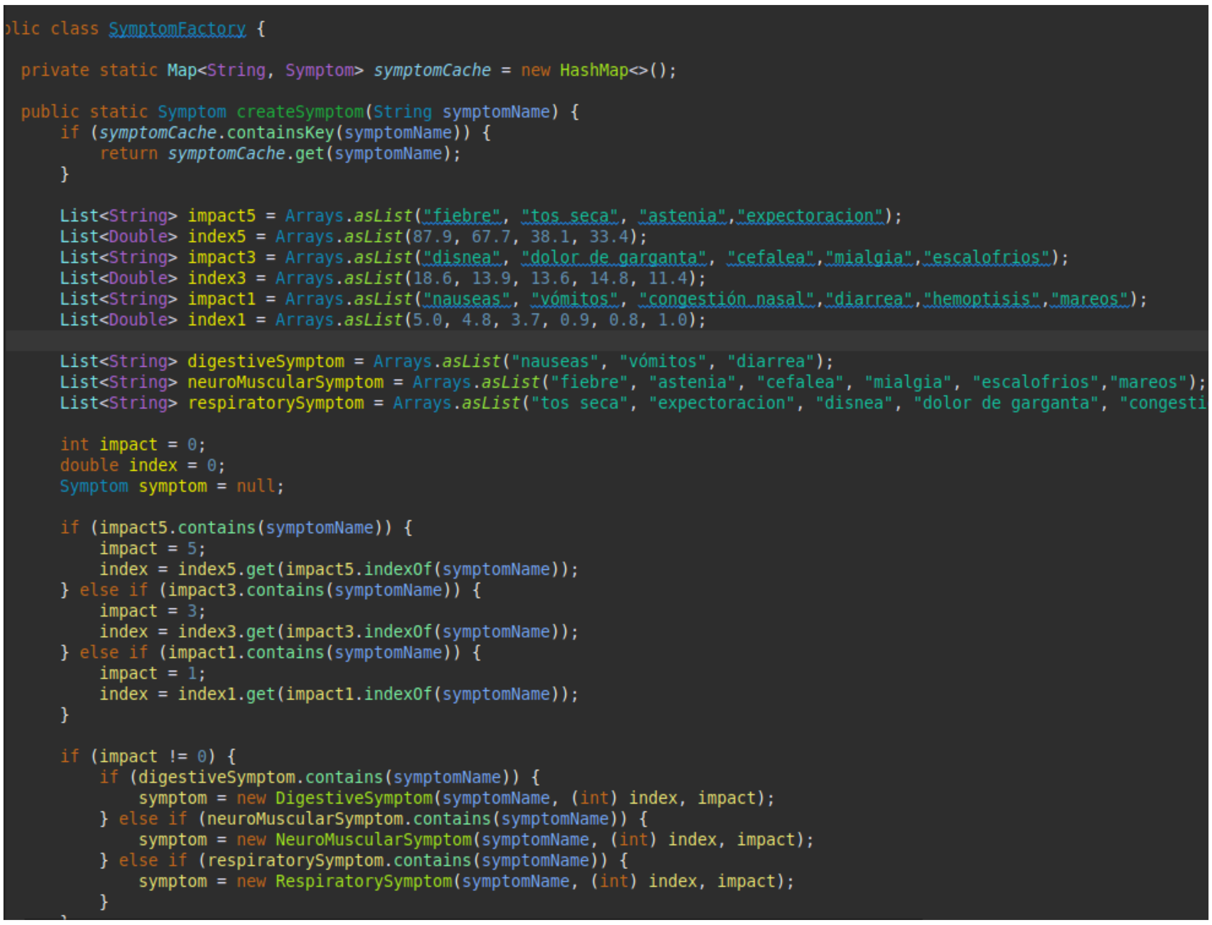
UML:



IMPLEMENTAZIOA:

Hau lortzeko Symptom objektuen sorketa zentralizatzen duen SymptomFactory klasea sortu dut. Honek konpontzen du Covid19Pacient eta Medicament klaseetan logika bikoiztuaren arazoa, biak Symptom-instantiak lortzeko fabrika hau erabiltzen dutelarik.
HashMap bat erabiliz cache bat implementatu da, sortutako sintoma instantziak gordetzeko. Horrela, sintoma bera behin eta berriro eskatzen bada, Symptom objektuaren instantzia bera itzuliko da, eta hori aseguru egiten du ezin izango direla sintoma mota bereko objektu anitz sortu.
Baita ere getAllSymptoms metodoa sortu dugu, sintoma guztiak bueltatzen dituena, hau erabilgarria izango delako beranduago bakoitzaren sintoma guztiak lortzeko.

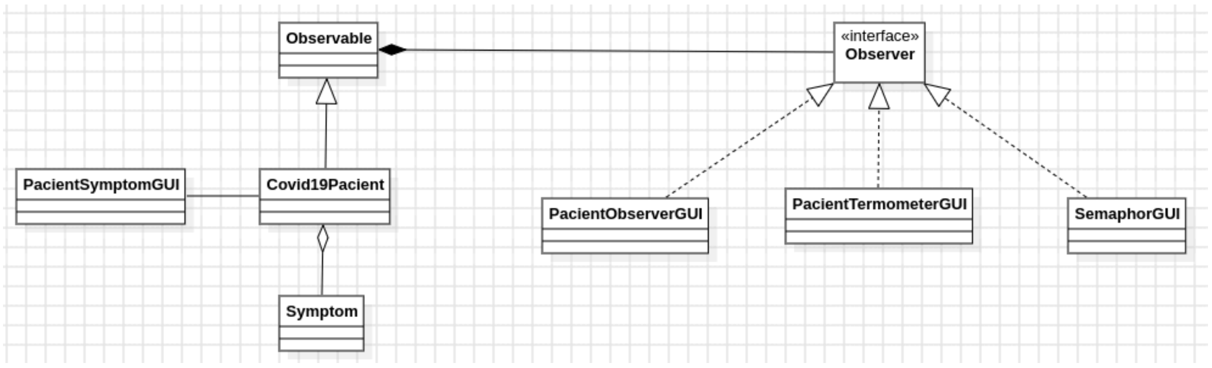
KODE AZPIMAGARRIA



```
lic class SymptomFactory {

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

    public static Symptom createSymptom(String symptomName) {
        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","mareos");
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 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", "congesti

        int impact = 0;
        double index = 0;
        Symptom symptom = null;

        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)) {
                symptom = new DigestiveSymptom(symptomName, (int) index, impact);
            } else if (neuroMuscularSymptom.contains(symptomName)) {
                symptom = new NeuroMuscularSymptom(symptomName, (int) index, impact);
            } else if (respiratorySymptom.contains(symptomName)) {
                symptom = new RespiratorySymptom(symptomName, (int) index, impact);
            }
```

OBSERVER PATROIA

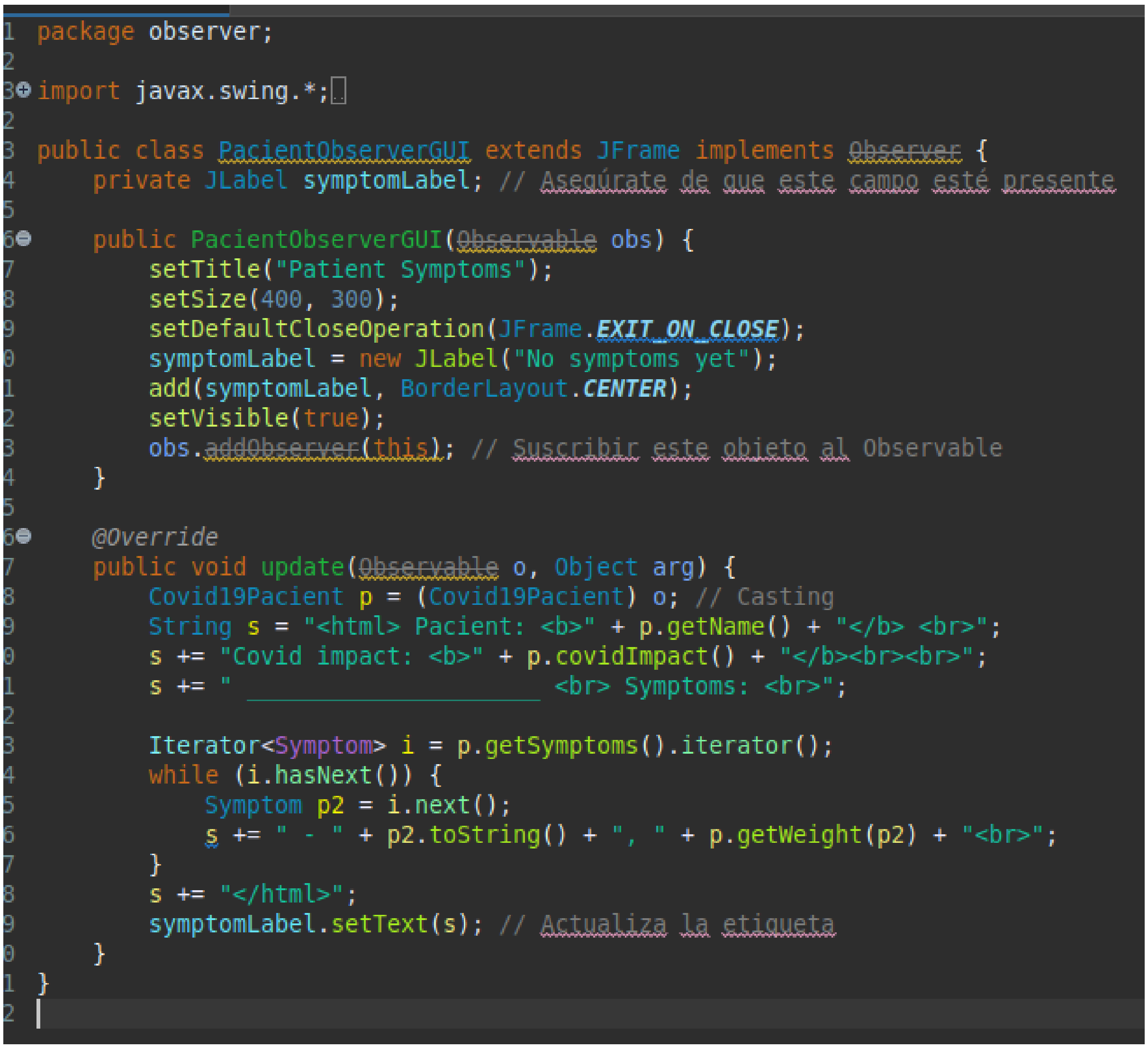
UML:



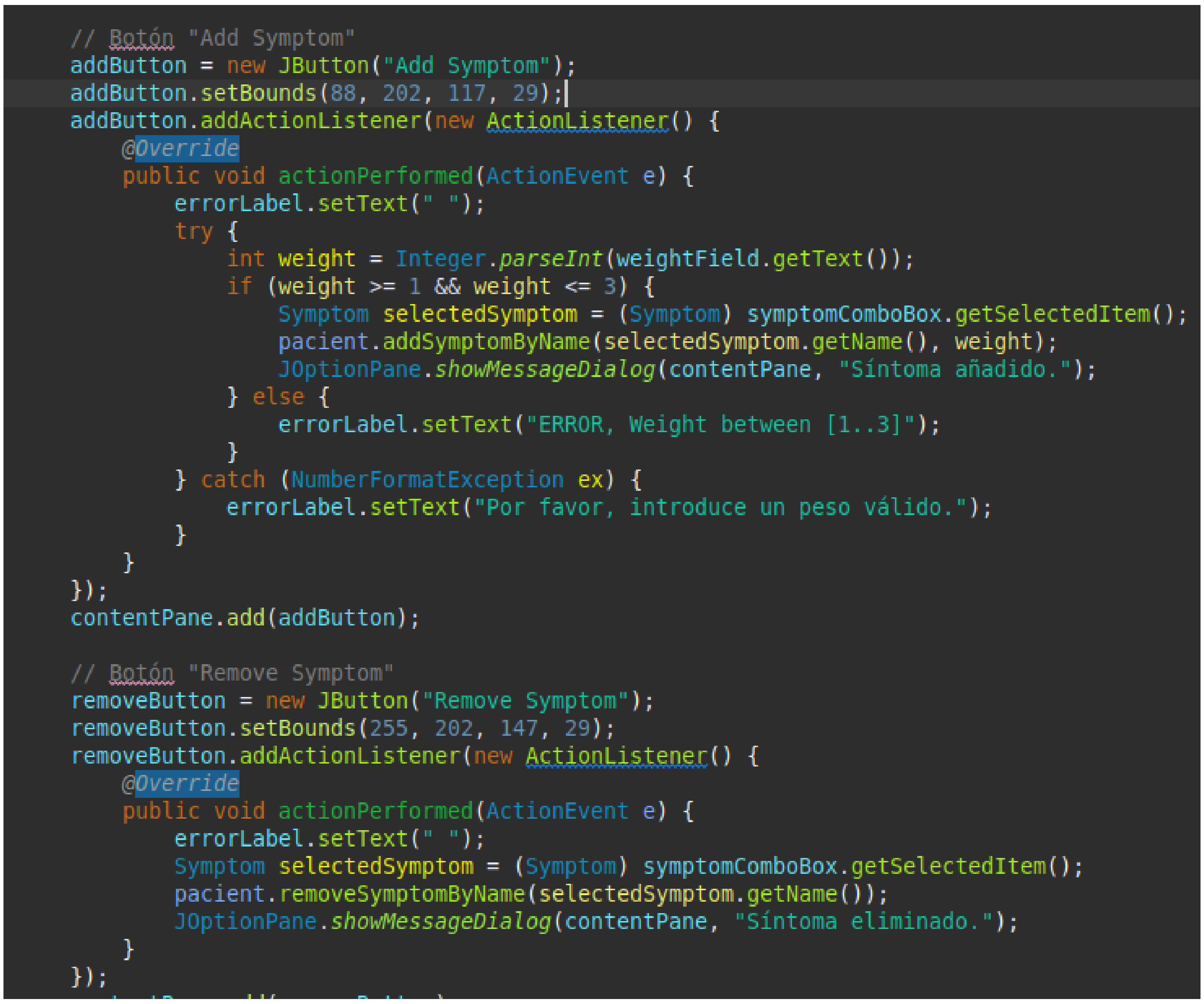
IMPLEMENTAZIOA

Hemen 3 prototipo eskatzen ziren eta hiruretan antzekoa egin behar zen, lehendabizikoan soilik eskatzen zuen bi GUI-ek Observer klasea implementatzea, eraikitzailean aldaketak egiten eta update metodoa implementatzen. Bestean, pacientSymptomGUI klasean aldiz bi botoi jarri behar dira istener akzioekin, horrela klik egiten dugun bitartean update egingo dute.
Bigarren prototipoan, gauza ez da asko aldatzen. GUI berri bat daukagu termometro batekin. Beraz honetan lehengo berdina egin behar da, eraikitzailea eta update jartzea. Updatean kontutan izan behar dugu impaktua, izan ere honen arabera kolorea aldatzen joango da.
Hirugarren prototipoa berdina da baina termometro bat izan ordez, semaforo bat.

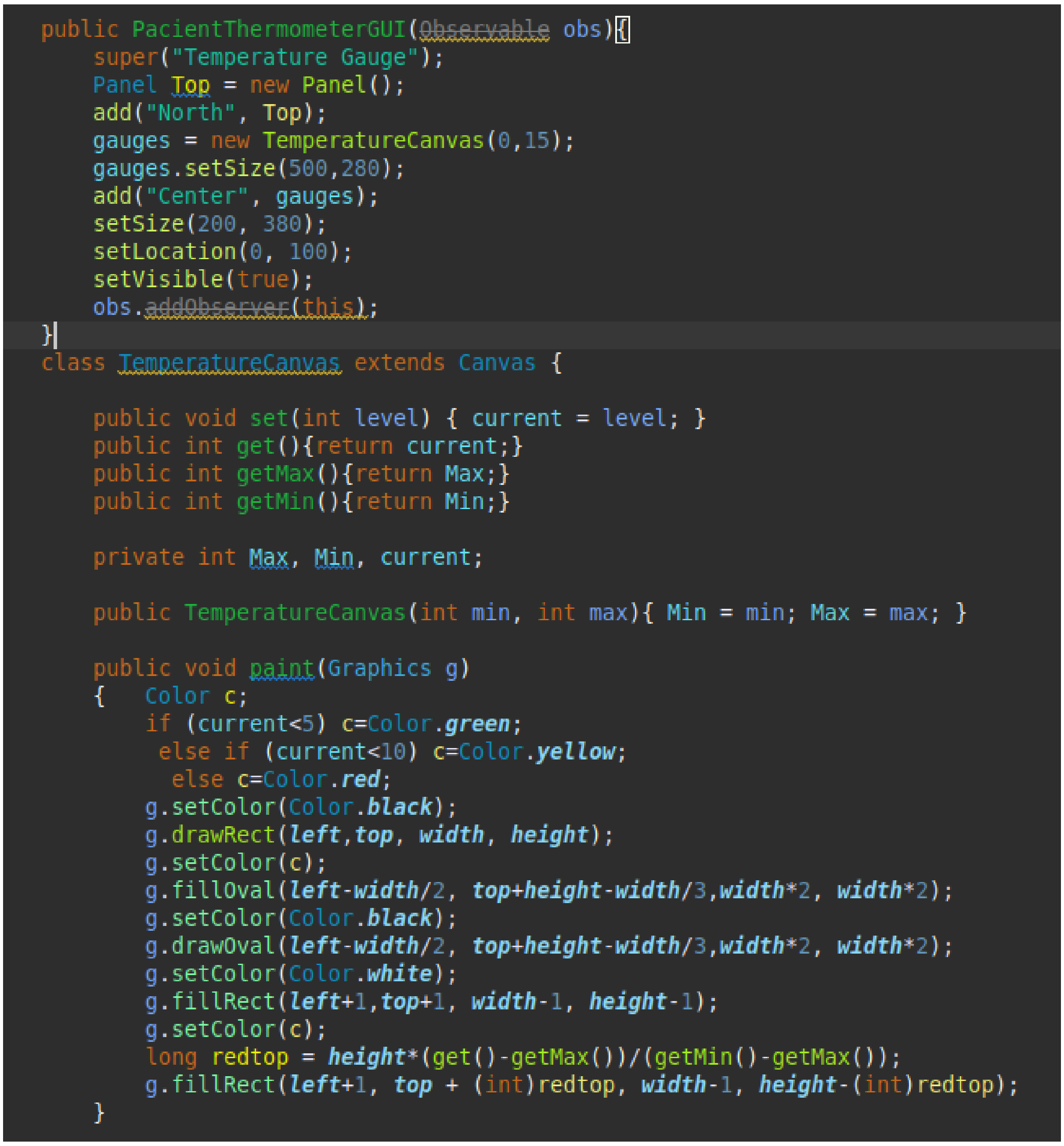
KODE AZPIMAGARRIA



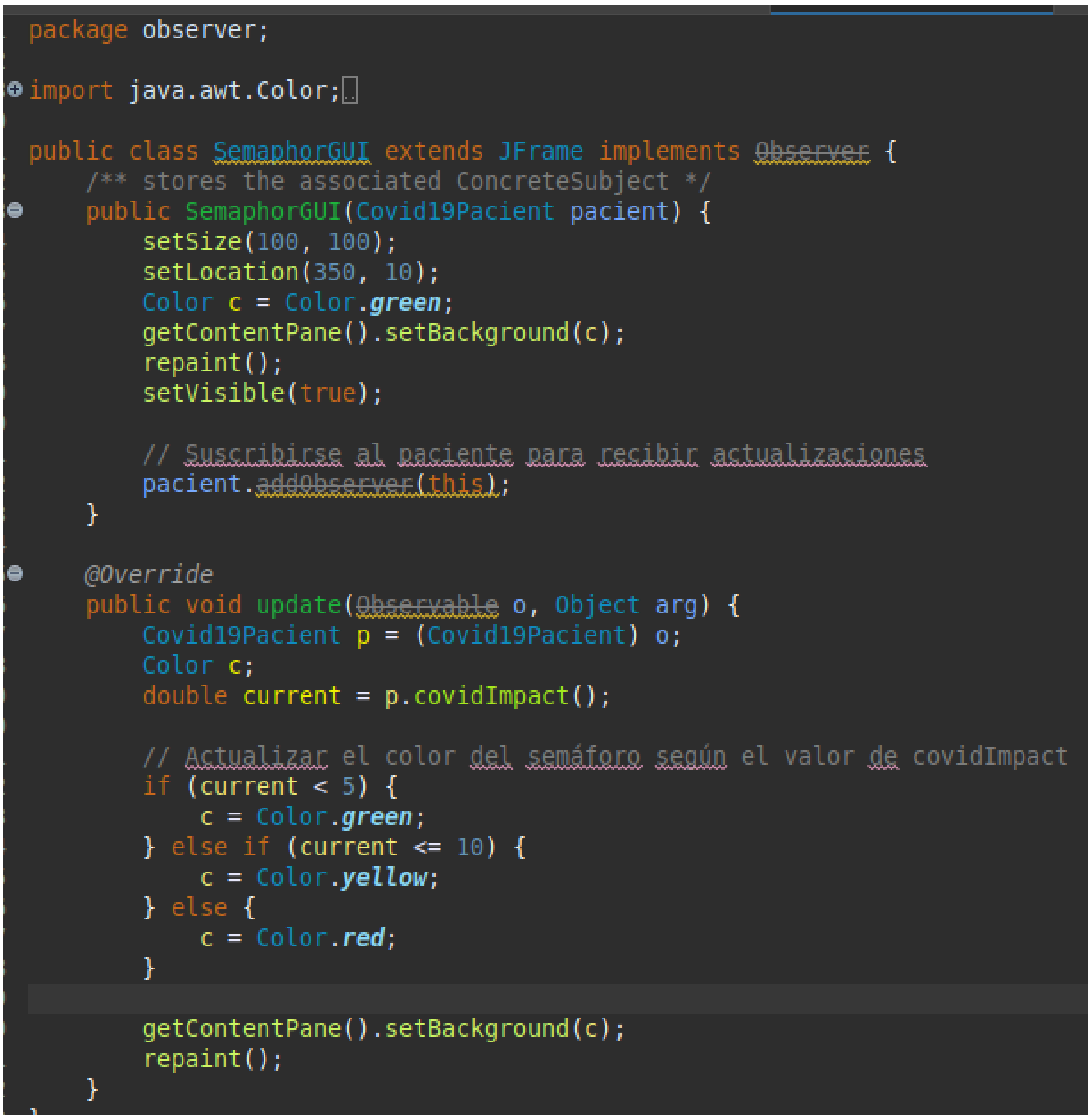
```java
package observer;

import javax.swing.*;

public class PacientObserverGUI extends JFrame implements Observer {
    private JLabel symptomLabel; // Asegúrate de que este campo esté presente

    public PacientObserverGUI(Observable obs) {
        setTitle("Patient Symptoms");
        setSize(400, 300);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        symptomLabel = new JLabel("No symptoms yet");
        add(symptomLabel, BorderLayout.CENTER);
        setVisible(true);
        obs.addObserver(this); // Suscribir este objeto al Observable
    }

    @Override
    public void update(Observable o, Object arg) {
        Covid19Pacient p = (Covid19Pacient) o; // Casting
        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 p2 = i.next();
            s += " - " + p2.toString() + ", " + p.getWeight(p2) + "<br>";
        }
        s += "</html>";
        symptomLabel.setText(s); // Actualiza la etiqueta
    }
}
```



```java
// Botón "Add Symptom"
addButton = new JButton("Add Symptom");
addButton.setBounds(88, 202, 117, 29);
addButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        errorLabel.setText(" ");
        try {
            int weight = Integer.parseInt(weightField.getText());
            if (weight >= 1 && weight <= 3) {
                Symptom selectedSymptom = (Symptom) symptomComboBox.getSelectedItem();
                pacient.addSymptomByName(selectedSymptom.getName(), weight);
                JOptionPane.showMessageDialog(contentPane, "Síntoma añadido.");
            } else {
                errorLabel.setText("ERROR, Weight between [1..3]");
            }
        } catch (NumberFormatException ex) {
            errorLabel.setText("Por favor, introduce un peso válido.");
        }
    }
});
contentPane.add(addButton);

// Botón "Remove Symptom"
removeButton = new JButton("Remove Symptom");
removeButton.setBounds(255, 202, 147, 29);
removeButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        errorLabel.setText(" ");
        Symptom selectedSymptom = (Symptom) symptomComboBox.getSelectedItem();
        pacient.removeSymptomByName(selectedSymptom.getName());
        JOptionPane.showMessageDialog(contentPane, "Síntoma eliminado.");
    }
});
```



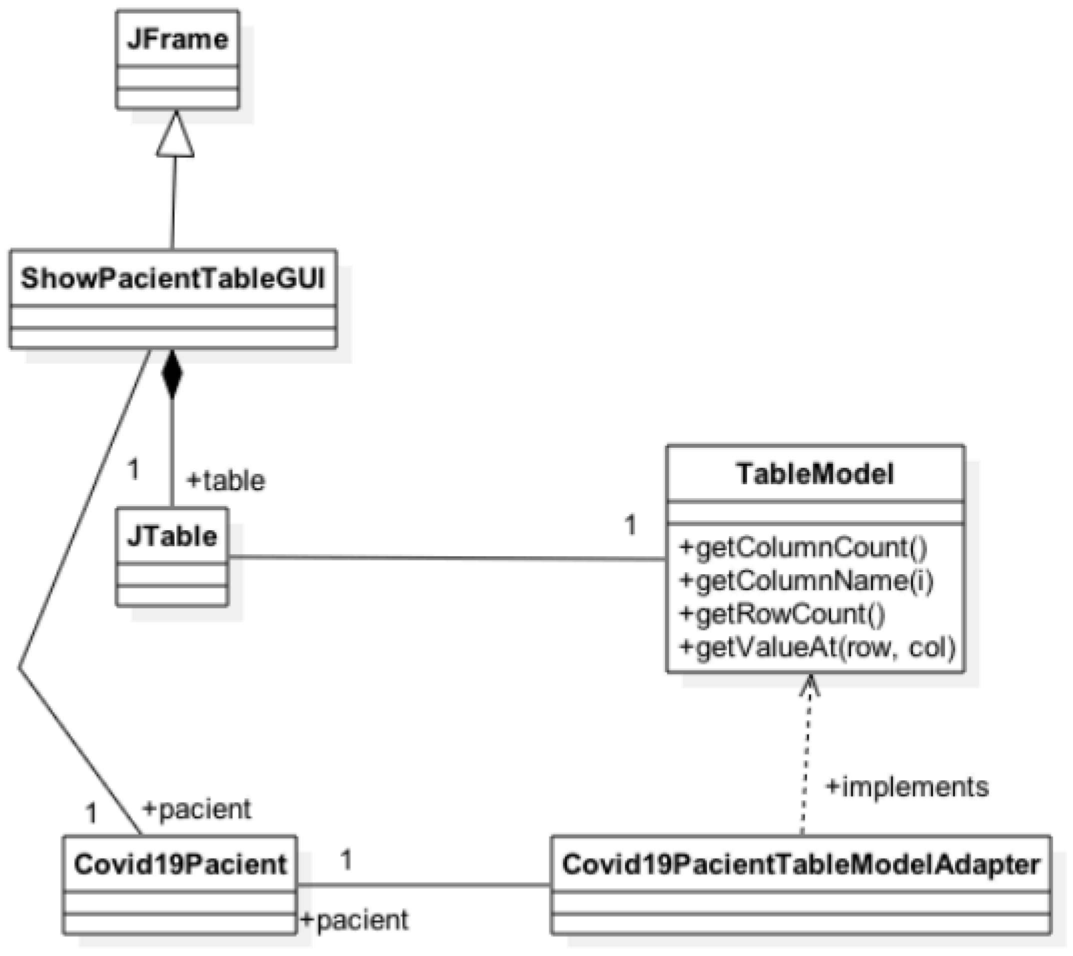
```
public PacientThermometerGUI(Observable obs){
    super("Temperature Gauge");
    Panel Top = new Panel();
    add("North", Top);
    gauges = new TemperatureCanvas(0,15);
    gauges.setSize(500,280);
    add("Center", gauges);
    setSize(200, 380);
    setLocation(0, 100);
    setVisible(true);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

    public void set(int level) { current = level; }
    public int get(){return current;}
    public int getMax(){return Max;}
    public int getMin(){return Min;}

    private int Max, Min, current;

    public TemperatureCanvas(int min, int max){ Min = min; Max = max; }

    public void paint(Graphics g)
    {   Color c;
        if (current<5) c=Color.green;
         else if (current<10) c=Color.yellow;
          else c=Color.red;
        g.setColor(Color.black);
        g.drawRect(left,top, width, height);
        g.setColor(c);
        g.fillOval(left-width/2, top+height-width/3,width*2, width*2);
        g.setColor(Color.black);
        g.drawOval(left-width/2, top+height-width/3,width*2, width*2);
        g.setColor(Color.white);
        g.fillRect(left+1,top+1, width-1, height-1);
        g.setColor(c);
        long redtop = height*(get()-getMax())/(getMin()-getMax());
        g.fillRect(left+1, top + (int)redtop, width-1, height-(int)redtop);
    }
```



```
package observer;

import java.awt.Color;

public class SemaphorGUI extends JFrame implements Observer {
    /** stores the associated ConcreteSubject */
    public SemaphorGUI(Covid19Pacient pacient) {
        setSize(100, 100);
        setLocation(350, 10);
        Color c = Color.green;
        getContentPane().setBackground(c);
        repaint();
        setVisible(true);

        // Suscribirse al paciente para recibir actualizaciones
        pacient.addObserver(this);
    }

    @Override
    public void update(Observable o, Object arg) {
        Covid19Pacient p = (Covid19Pacient) o;
        Color c;
        double current = p.covidImpact();

        // Actualizar el color del semáforo según el valor de covidImpact
        if (current < 5) {
            c = Color.green;
        } else if (current <= 10) {
            c = Color.yellow;
        } else {
            c = Color.red;
        }

        getContentPane().setBackground(c);
        repaint();
    }
```

ADAPTER PATROIA

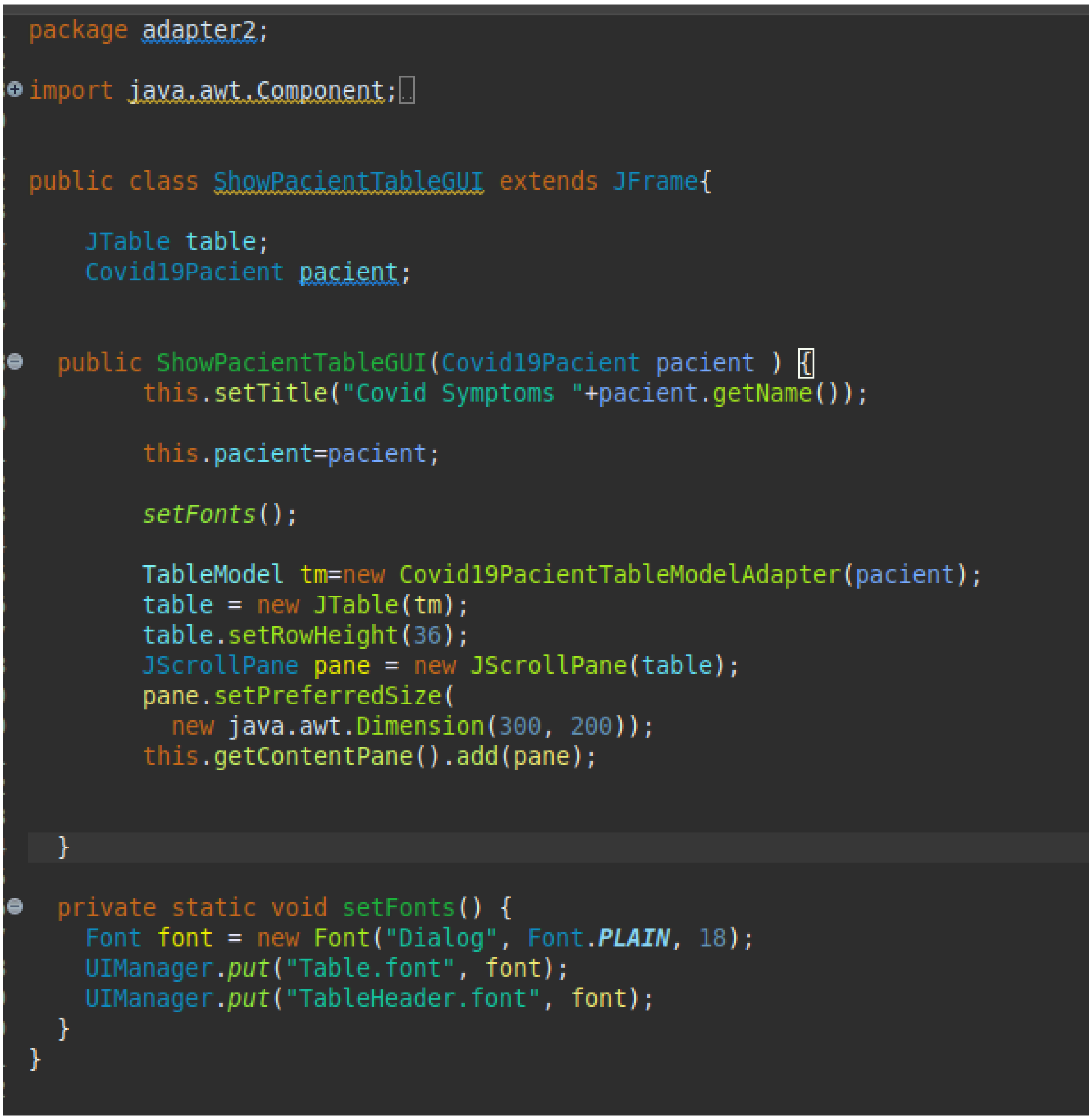
UML:



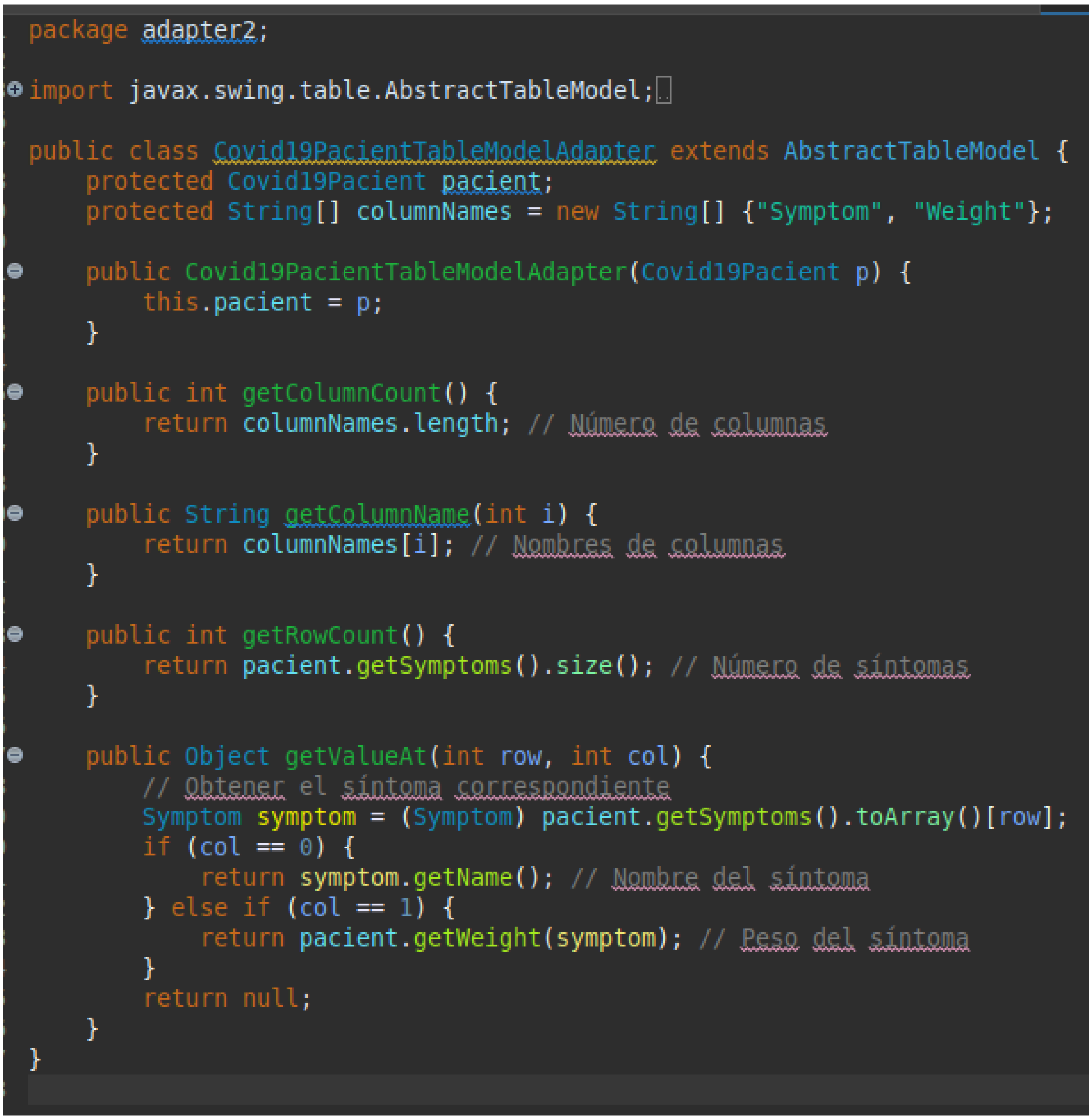
IMPLEMENTAZIOA:

Adaptadorea: Covid19PacientTableModelAdapter implementatu dut, pazientearen sintomak taularekin (JTable) konektatzeko.
Interfazea: ShowPacientTableGUI sortu dut, sintomak dituzten taula erakusteko.
Optimizazioa: toArray() aldatu dut sintomak zerrendara zuzenean sartzeko.

KODE AZPIMAGARRIA



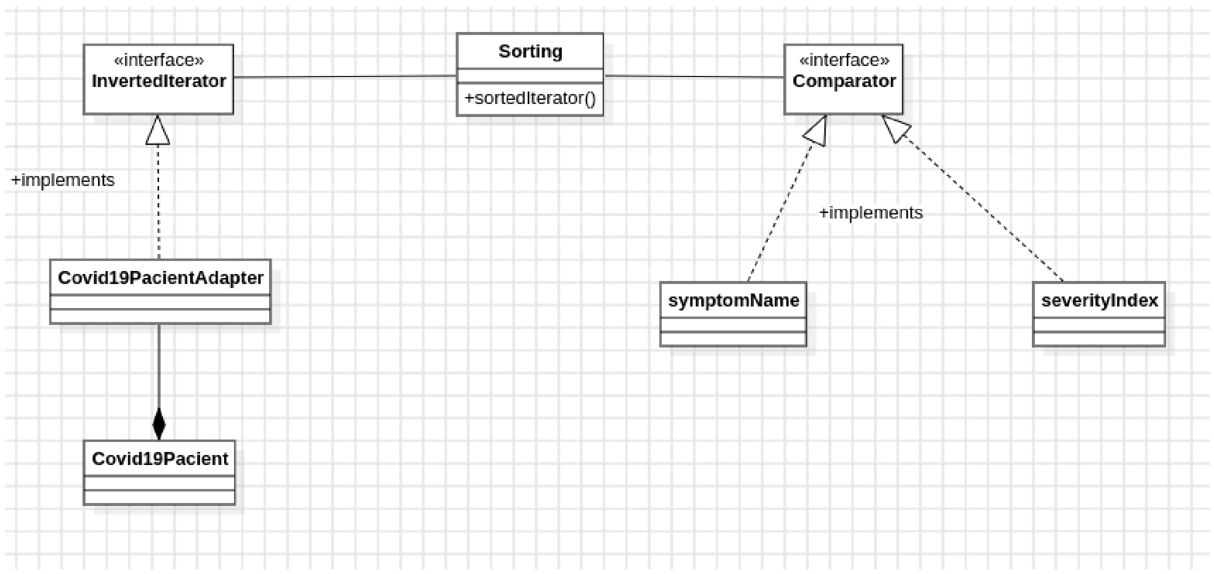
```java
package adapter2;

import java.awt.Component;


public class ShowPacientTableGUI extends JFrame{

    JTable table;
    Covid19Pacient pacient;


  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);


  }

  private static void setFonts() {
    Font font = new Font("Dialog", Font.PLAIN, 18);
    UIManager.put("Table.font", font);
    UIManager.put("TableHeader.font", font);
  }
}
```



```java
package adapter2;

import javax.swing.table.AbstractTableModel;

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() {
        return columnNames.length; // Número de columnas
    }

    public String getColumnName(int i) {
        return columnNames[i]; // Nombres de columnas
    }

    public int getRowCount() {
        return pacient.getSymptoms().size(); // Número de síntomas
    }

    public Object getValueAt(int row, int col) {
        // Obtener el síntoma correspondiente
        Symptom symptom = (Symptom) pacient.getSymptoms().toArray()[row];
        if (col == 0) {
            return symptom.getName(); // Nombre del síntoma
        } else if (col == 1) {
            return pacient.getWeight(symptom); // Peso del síntoma
        }
        return null;
    }
}
```

ITERATOR ETA ADAPTER PATROIAK

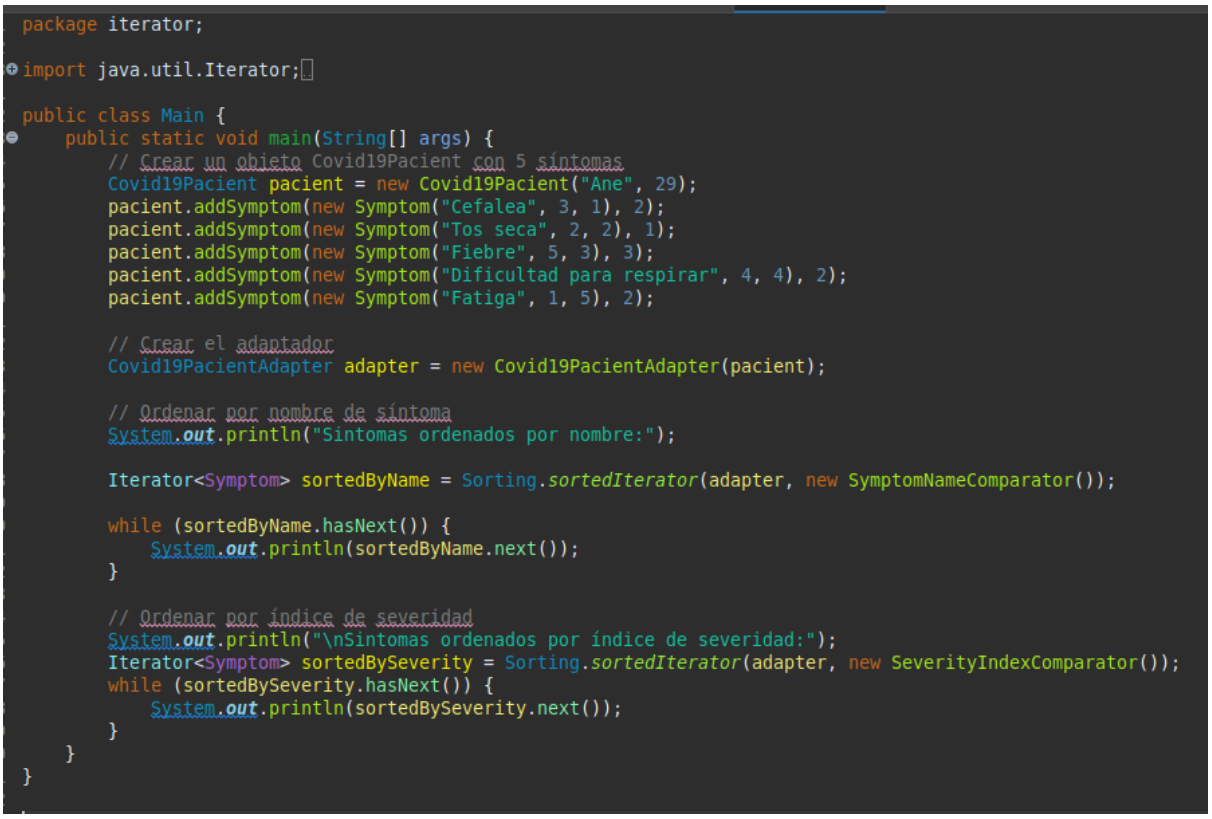
UML:



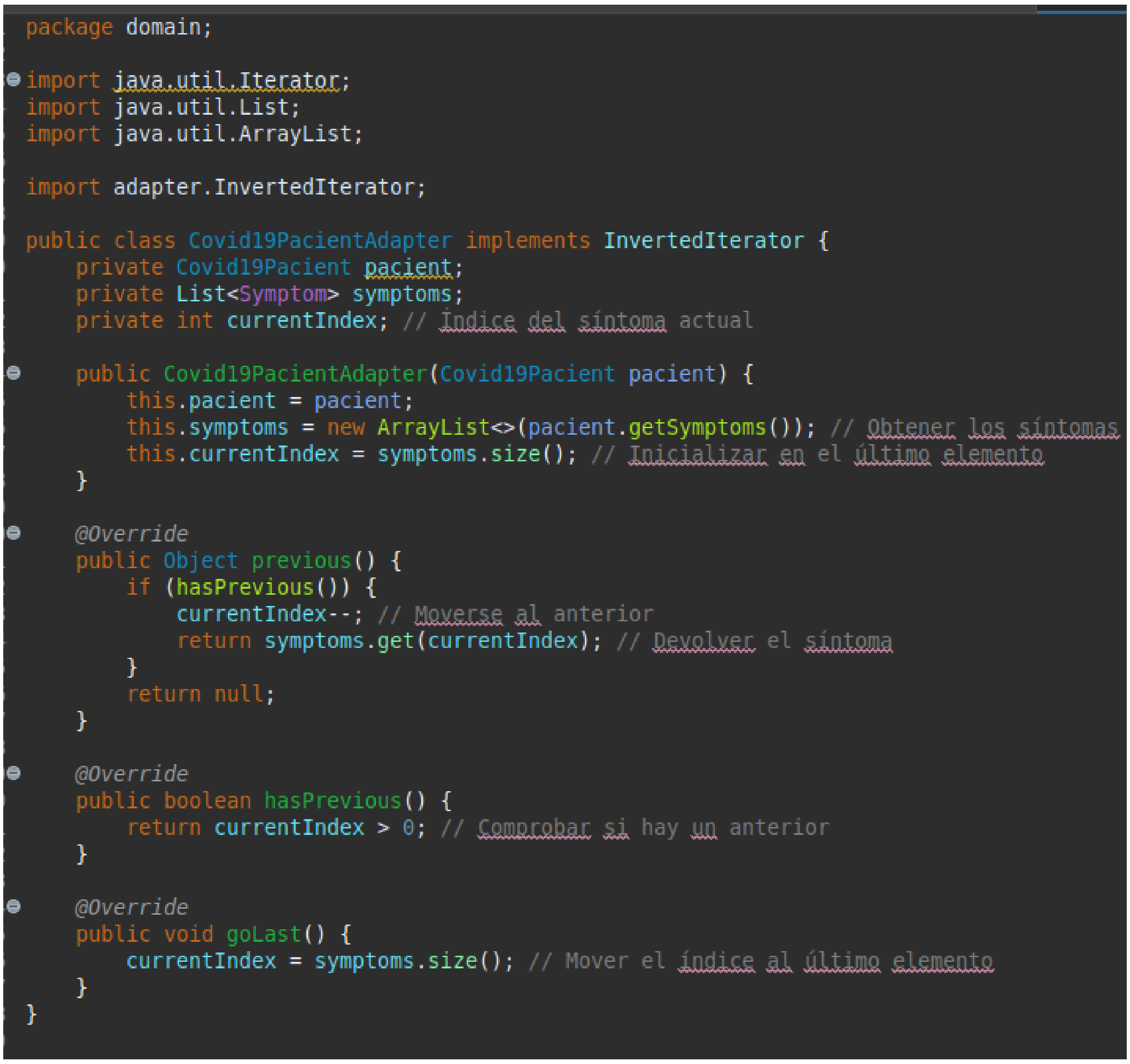
IMPLEMENTAZIOA:

InvertedIterator: Interfaz bat sortu dut, alderantzikoz ibiltzeko, previous(), hasPrevious(), eta goLast() metodoez.
Sorting: sortedIterator metodoa sortu dut, InvertedIterator eta Comparator erabiliz sintomak ordenatzeko.
Covid19PacientAdapter: Adaptadorea sortu dut, Covid19Pacient bat InvertedIterator bihurtzen duena, sintomak alderantzikoz iteratzeko.
Comparadoreak: Bi comparadore sortu ditut, bat symptomName arabera eta bestea severityIndex arabera.
Main: Covid19Pacient bat sortu da 5 sintomarekin, eta Sorting.sortedIterator erabiltzen da sintomak izenaren eta larritasunaren arabera ordenatzeko

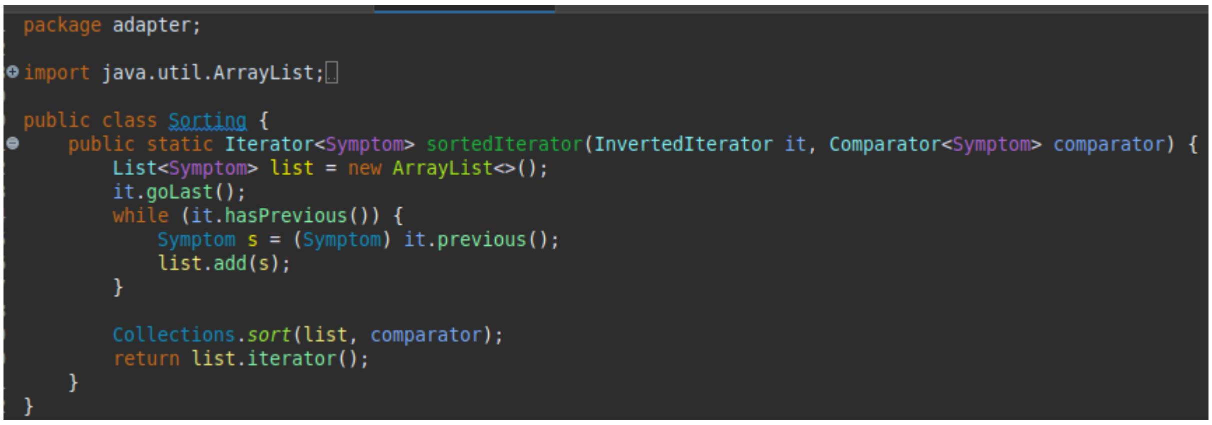
KODE AZPIMAGARRIA:



```
package iterator;

import java.util.Iterator;

public class Main {
    public static void main(String[] args) {
        // Crear un objeto Covid19Pacient con 5 síntomas
        Covid19Pacient pacient = new Covid19Pacient("Ane", 29);
        pacient.addSymptom(new Symptom("Cefalea", 3, 1), 2);
        pacient.addSymptom(new Symptom("Tos seca", 2, 2), 1);
        pacient.addSymptom(new Symptom("Fiebre", 5, 3), 3);
        pacient.addSymptom(new Symptom("Dificultad para respirar", 4, 4), 2);
        pacient.addSymptom(new Symptom("Fatiga", 1, 5), 2);

        // Crear el adaptador
        Covid19PacientAdapter adapter = new Covid19PacientAdapter(pacient);

        // Ordenar por nombre de síntoma
        System.out.println("Sintomas ordenados por nombre:");

        Iterator<Symptom> sortedByName = Sorting.sortedIterator(adapter, new SymptomNameComparator());

        while (sortedByName.hasNext()) {
            System.out.println(sortedByName.next());
        }

        // Ordenar por índice de severidad
        System.out.println("\nSintomas ordenados por índice de severidad:");
        Iterator<Symptom> sortedBySeverity = Sorting.sortedIterator(adapter, new SeverityIndexComparator());
        while (sortedBySeverity.hasNext()) {
            System.out.println(sortedBySeverity.next());
        }
    }
}
```



```java
package domain;

import java.util.Iterator;
import java.util.List;
import java.util.ArrayList;

import adapter.InvertedIterator;

public class Covid19PacientAdapter implements InvertedIterator {
    private Covid19Pacient pacient;
    private List<Symptom> symptoms;
    private int currentIndex; // Índice del síntoma actual

    public Covid19PacientAdapter(Covid19Pacient pacient) {
        this.pacient = pacient;
        this.symptoms = new ArrayList<>(pacient.getSymptoms()); // Obtener los síntomas
        this.currentIndex = symptoms.size(); // Inicializar en el último elemento
    }

    @Override
    public Object previous() {
        if (hasPrevious()) {
            currentIndex--; // Moverse al anterior
            return symptoms.get(currentIndex); // Devolver el síntoma
        }
        return null;
    }

    @Override
    public boolean hasPrevious() {
        return currentIndex > 0; // Comprobar si hay un anterior
    }

    @Override
    public void goLast() {
        currentIndex = symptoms.size(); // Mover el índice al último elemento
    }
}
```



```java
package adapter;

import java.util.ArrayList;

public class Sorting {
    public static Iterator<Symptom> sortedIterator(InvertedIterator it, Comparator<Symptom> comparator) {
        List<Symptom> list = new ArrayList<>();
        it.goLast();
        while (it.hasPrevious()) {
            Symptom s = (Symptom) it.previous();
            list.add(s);
        }

        Collections.sort(list, comparator);
        return list.iterator();
    }
}
```