

eman ta zabal zazu



Universidad
del País Vasco Euskal Herriko
Unibertsitatea

Diseinu patroien laborategia



Egileak: Ander Mena, Ekhi Ugarte, Haritz Eizagirre

Data: 2025/11/09

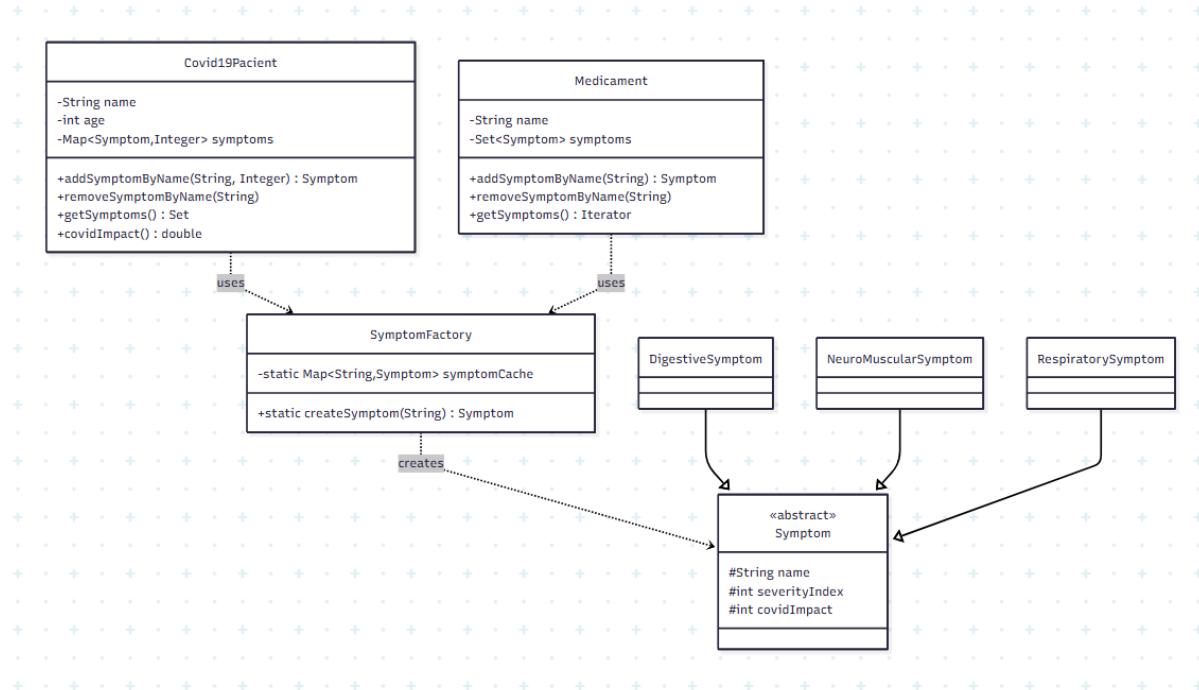
Irakaslea: Jon Iturrioz

Irakasgaia: Software Ingeniaritza II

Github errepositorioa: <https://github.com/andermena23/labpatterns>

Simple Factory

1. UML diagrama:



Egindako aldaketak:

- SymptomFactory klase berria sortu da:
 - `createSymptom(String symptomName)` metodo estatiko bat dauka sintomak sortzeko
 - Sintoma sortzeko logika guztsia klase honetan zentralizatu da
 - `symptomCache` Map estatiko bat gehitu da sintoma instantzia bakarrak gordetzeko
- Covid19Patient klasean:
 - `createSymptom()` metodo pribatua kendu da
 - `addSymptomByName()` metodoa aldatu da `SymptomFactory.createSymptom()` erabiltzeko:

```
public Symptom addSymptomByName(String symptomName, Integer weight) {  
    Symptom s = SymptomFactory.createSymptom(symptomName);  
    if (s != null) {  
        symptoms.put(s, weight);  
    }  
    return s;
```

```
}
```

c. Medicament klasean:

- createSymptom() metodoa pribatua kendu da
- addSymptomByName() metodoa aldatu da
SymptomFactory.createSymptom() erabiltzeko:

```
public Symptom addSymptomByName(String symptomName) {  
    Symptom s = SymptomFactory.createSymptom(symptomName);  
    if (s != null) {  
        symptoms.add(s);  
    }  
    return s;  
}
```

2. "mareos" sintoma berria implementatu

SymptomFactory klasean "mareos" sintoma gehitu da impact 1 duelarik:

```
public static Symptom createSymptom(String symptomName) {  
  
    // Cache konprobaketa  
  
    if (symptomCache.containsKey(symptomName)) {  
  
        return symptomCache.get(symptomName);  
    }  
  
    // ... bestelako kodea ...  
  
    // "mareos" gehitu impact 1 duten sintomen zerrendan  
  
    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, 0.4, 0.4);  
}
```

```

// "mareos" neuroMuscular kategoriako sintoma da

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

// ... sintoma sortzeko logika ...

}

```

3. Factory klasea egokitu sintoma objektu bakarrak izateko

Singleton patroia aplikatu da SymptomFactory klasean, sintoma bakoitzeko objektu bakarra sortzeko:

```

public class SymptomFactory {

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

    public static Symptom createSymptom(String symptomName) {
        if (symptomCache.containsKey(symptomName)) {
            return symptomCache.get(symptomName); // Dagoeneko sortuta badago, bera
        itzuli
        }

        Symptom symptom = null;
        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);
            }
        }
    }
}

```

```

    }

    if (symptom != null) {

        symptomCache.put(symptomName, symptom);

    }

}

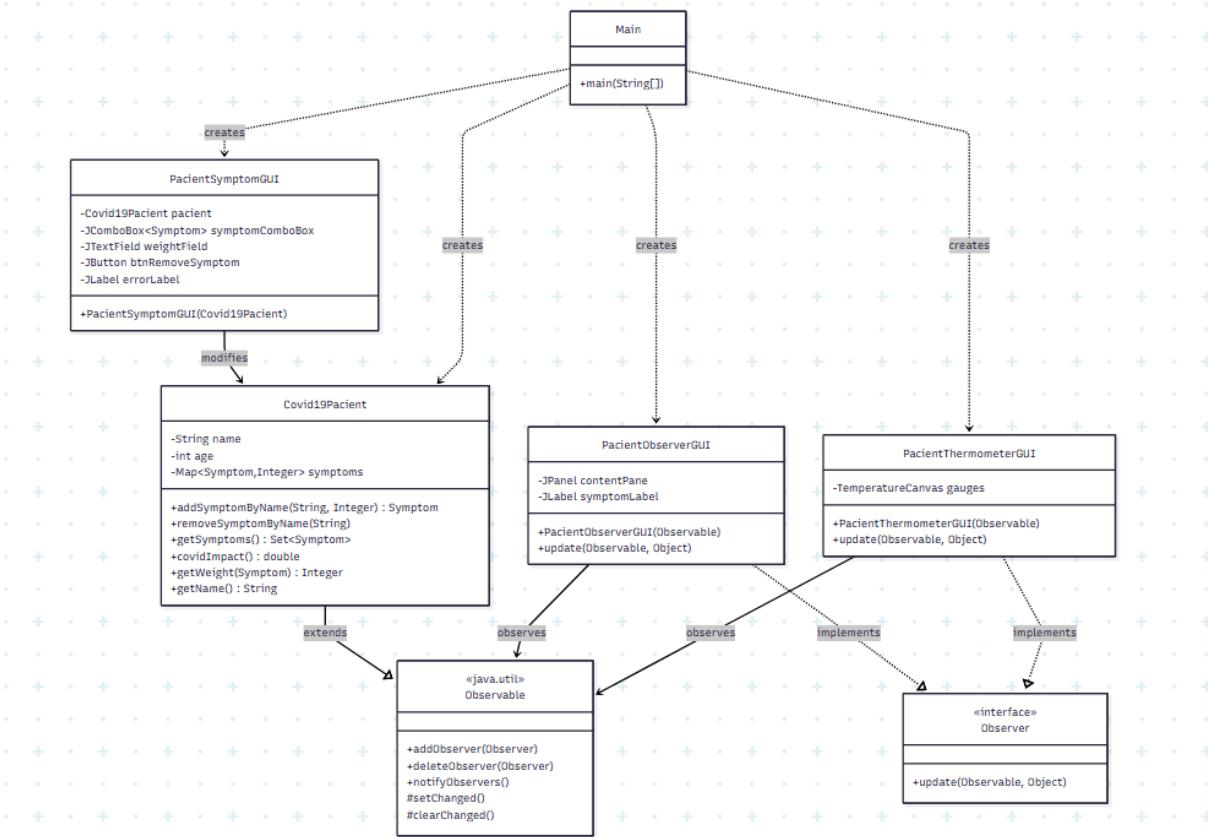
return symptom;

}
}

```

Observer Patroia

1. UML diagrama



2. Aplikazioaren implementazioa

Covid19Pacient klasea (Observable)

```
public class Covid19Pacient extends Observable {  
    private String name;  
    private int age;  
    private Map<Symptom, Integer> symptoms;  
  
    public Symptom addSymptomByName(String symptomName, Integer weight) {  
        Symptom s = SymptomFactory.createSymptom(symptomName);  
        if (s != null) {  
            symptoms.put(s, weight);  
            // Notifikatu Observer-ei aldaketa gertatu dela  
            setChanged();  
            notifyObservers();  
        }  
        return s;  
    }  
  
    public void removeSymptomByName(String symptomName) {  
        Symptom s = getSymptomByName(symptomName);  
        if (s != null) {  
            symptoms.remove(s);  
            // Notifikatu Observer-ei aldaketa gertatu dela  
            setChanged();  
            notifyObservers();  
        }  
    }  
  
    // Beste metodoak: getName(), getSymptoms(), covidImpact(), getWeight()...  
}
```

PacientObserverGUI klasea (Observer)

```
public class PacientObserverGUI extends JFrame implements Observer {  
    private JPanel contentPane;  
    private final JLabel symptomLabel = new JLabel("");  
  
    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);

        // Observable-ari subscribe egin
        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 = 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);
    }
}

```

PacientThermometerGUI klasea (Observer)

```

public class PacientThermometerGUI extends Frame implements Observer {
    private TemperatureCanvas gauges;

    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);

        // Observable-ari subscribe egin
        obs.addObserver(this);
    }

    @Override
    public void update(Observable o, Object args) {
        Covid19Pacient p = (Covid19Pacient) o;
        // covidImpact lortu termometroa margotzeko
    }
}

```

```

int farenheit = (int) p.covidImpact();
// Termometroa eguneratu
gauges.set(farenheit);
gauges.repaint();
}

class TemperatureCanvas extends Canvas {
    private int Max, Min, current;

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

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

    public void paint(Graphics g) {
        Color c;
        // Kolorea covidImpact-aren arabera
        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);
    }

    private static final int width = 20;
    private static final int top = 20;
    private static final int left = 100;
    private static final int right = 250;
    private static final int height = 200;
}
}

```

PacientSymptomGUI klasea

```
public class PacientSymptomGUI extends JFrame {
```

```

private JPanel contentPane;
private JTextField weightField;
JComboBox<Symptom> symptomComboBox;
private JButton btnRemoveSymptom;
private JLabel errorLabel;
private Covid19Pacient pacient;

public PacientSymptomGUI(Covid19Pacient p) {
    this.pacient = p;
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setBounds(200, 100, 450, 300);
    // ... GUI osagaiak hasieratu ...

    // Add Symptom botoia
    JButton btnNewButton = new JButton("Add Symptom");
    btnNewButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            errorLabel.setText(" ");
            if (new Integer(weightField.getText()) <= 3) {
                pacient.addSymptomByName(
                    ((Symptom)symptomComboBox.getSelectedItem()).getName(),
                    Integer.parseInt(weightField.getText())
                );
            } else {
                errorLabel.setText("ERROR, Weight between [1..3]");
            }
        }
    });
}

// Remove Symptom botoia
btnRemoveSymptom = new JButton("Remove Symptom");
btnRemoveSymptom.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        errorLabel.setText(" ");
        pacient.removeSymptomByName(
            ((Symptom)symptomComboBox.getSelectedItem()).getName()
        );
    }
});
}
}

```

Main programa

```

public class Main {
    public static void main(String[] args) {
        // Lehenengo pazientea
        Observable pacient1 = new Covid19Pacient("aitor", 35);
        PacientObserverGUI observer1 = new PacientObserverGUI(pacient1);
        observer1.setLocation(650, 100);
        PacientThermometerGUI thermo1 = new PacientThermometerGUI(pacient1);
    }
}

```

```

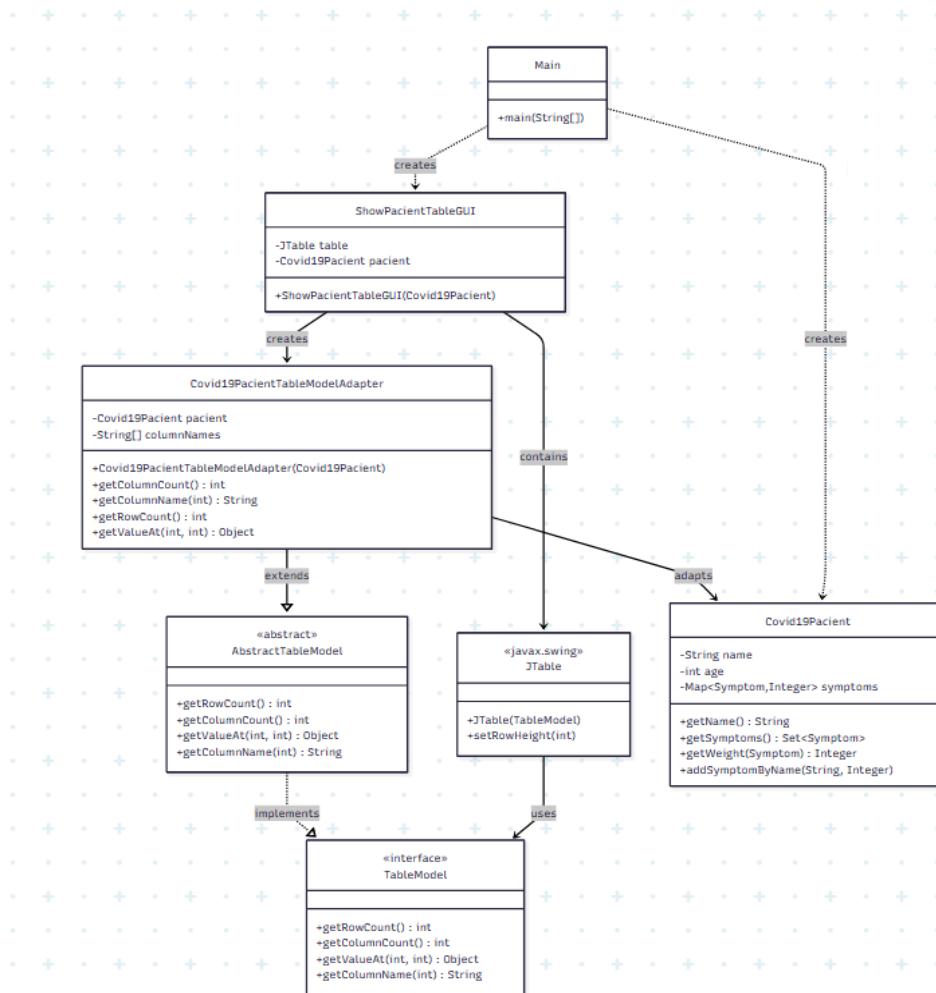
thermo1.setLocation(0, 100);
PacientSymptomGUI symptom1 = new PacientSymptomGUI((Covid19Patient)
pacient1);
symptom1.setLocation(200, 100);

// Bigarren pazientea
Observable pacient2 = new Covid19Patient("maria", 50);
PacientObserverGUI observer2 = new PacientObserverGUI(pacient2);
observer2.setLocation(650, 420);
PacientThermometerGUI thermo2 = new PacientThermometerGUI(pacient2);
thermo2.setLocation(0, 500);
PacientSymptomGUI symptom2 = new PacientSymptomGUI((Covid19Patient)
pacient2);
symptom2.setLocation(200, 420);
}
}
}

```

Adapter patroia

1. UML diagrama



2. Covid19PacientTableModelAdapter klasea

```
package adapter2;

import javax.swing.table.AbstractTableModel;
import domain.Covid19Pacient;

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; // 2 zutabe
    }

    public String getColumnName(int i) {
        return columnNames[i]; // "Symptom" edo "Weight"
    }

    public int getRowCount() {
        return pacient.getSymptoms().size(); // Sintoma kopurua
    }

    public Object getValueAt(int row, int col) {
        // Sintomak array batera bihurtu
        domain.Symptom[] symptoms = pacient.getSymptoms().toArray(new
domain.Symptom[0]);
        domain.Symptom symptom = symptoms[row];

        // Zutabearen arabera, sintomaren izena edo pisua itzuli
        if (col == 0) {
            return symptom.getName(); // Lehenengo zutabea: sintomaren izena
        } else {
            return pacient.getWeight(symptom); // Bigarren zutabea: pisua
        }
    }
}
```

3. Main programa 2 pazienterekin

```
package adapter2;

import domain.Covid19Pacient;

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

```

// Lehenengo pazientea: aitor
Covid19Pacient pacient = new Covid19Pacient("aitor", 35);
pacient.addSymptomByName("disnea", 2);
pacient.addSymptomByName("cefalea", 1);
pacient.addSymptomByName("astenia", 3);

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

// Bigarren pazientea: maria
Covid19Pacient paciente2 = new Covid19Pacient("maria", 28);
paciente2.addSymptomByName("fiebre", 3);
paciente2.addSymptomByName("tos seca", 2);
paciente2.addSymptomByName("disnea", 1);
paciente2.addSymptomByName("cefalea", 2);

ShowPacientTableGUI gui2 = new ShowPacientTableGUI(paciente2);
gui2.setPreferredSize(new java.awt.Dimension(300, 200));
gui2.setLocation(350, 0);
gui2.pack();
gui2.setVisible(true);
}
}

```

4. ShowPacientTableGUI klasea pazienteen taula eguneratzeko

```

package adapter2;

import java.awt.Component;
import java.awt.Font;
import javax.swing.*;
import javax.swing.table.TableModel;
import domain.Covid19Pacient;

public class ShowPacientTableGUI extends JFrame {
    JTable table;
    Covid19Pacient paciente;

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

        setFonts();

        // Adapter-a sortu eta JTable-ari pasa
    }
}

```

```

TableModel tm = new Covid19PatientTableModelAdapter(pacient);
table = new JTable(tm);
table.setRowHeight(36);

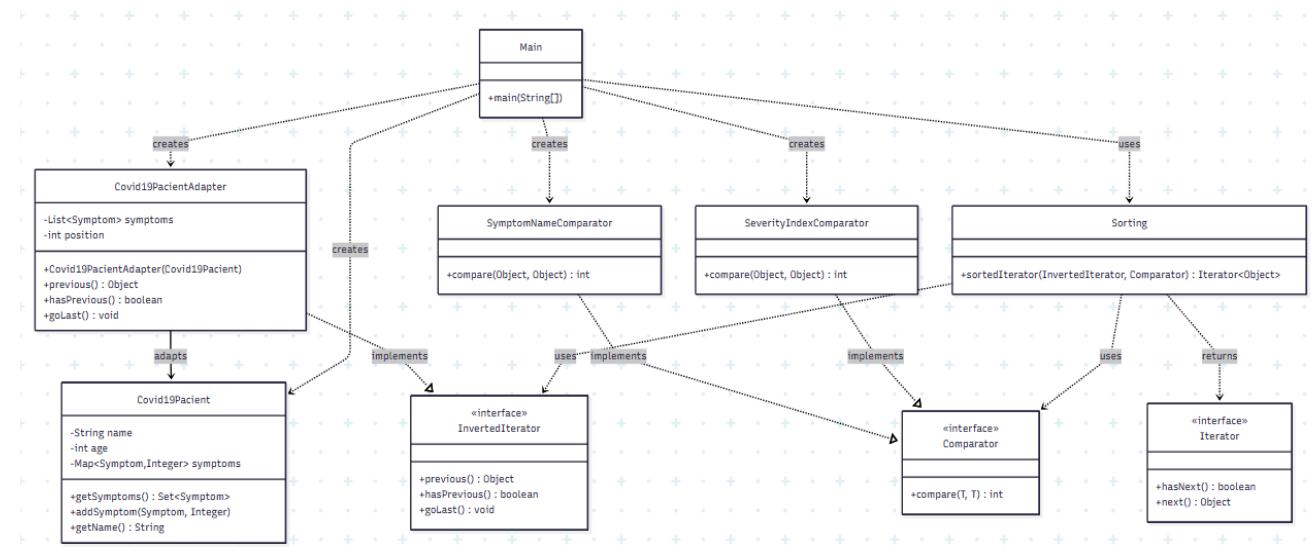
JScrollPane pane = new JScrollPane(table);
pane.setPreferredSize(new java.awt.Dimension(300, 200));
this.getContentPane().add(pane);
}

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

```

Adapter Iterator eta Patroiak

1. UML Diagrama



2. Comparator implementazioak

SymptomNameComparator klasea

```

package adapter;

import java.util.Comparator;
import domain.Symptom;

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());  
}  
}
```

SeverityIndexComparator klasea

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

3. Covid19PacientAdapter klasea (Adapter patroia)

```
package adapter;  
  
import domain.Covid19Pacient;  
import domain.Symptom;  
import java.util.ArrayList;  
import java.util.List;  
  
public class Covid19PacientAdapter implements InvertedIterator {  
  
    private List<Symptom> symptoms;  
    private int position;  
  
    public Covid19PacientAdapter(Covid19Pacient pacient) {  
        this.symptoms = new ArrayList<>(pacient.getSymptoms());  
        this.position = 0;  
    }  
  
    @Override
```

```

public Object previous() {
    if (hasPrevious()) {
        Object symptom = symptoms.get(position);
        position--;
        return symptom;
    }
    return null;
}

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

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

```

InvertedIterator interfazea

```

package adapter;

public interface InvertedIterator {
    // return the actual element and go to the previous
    public Object previous();

    // true if there is a previous element
    public boolean hasPrevious();

    // It is placed in the last element
    public void goLast();
}

```

4. 5 sintomako pazientea programa nagusia (Sorting klasea erabiliz)

Sorting klasea

```

package adapter;

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

```

```

import domain.Symptom;

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

```

Programa nagusia

```

package adapter;

import domain.Covid19Patient;
import domain.Symptom;
import java.util.Iterator;

public class Main {

    public static void main(String[] args) {
        // Covid19Patient objektu bat sortu 5 sintomekin
        Covid19Patient patient = new Covid19Patient("Ander", 25);

        // 5 sintomak gehitu
        patient.addSymptom(new Symptom("Cough", 3, 2), 5);
        patient.addSymptom(new Symptom("Fever", 4, 4), 7);
        patient.addSymptom(new Symptom("Headache", 2, 1), 3);
        patient.addSymptom(new Symptom("Fatigue", 3, 3), 6);
        patient.addSymptom(new Symptom("Breathlessness", 5, 5), 8);

        System.out.println("Pazientea: " + patient.getName());
        System.out.println("Sintoma kopurua: " + patient.getSymptoms().size());
        System.out.println();

        // Covid19PatientAdapter sortu
        Covid19PatientAdapter adapter = new Covid19PatientAdapter(patient);

        // Comparator-eak sortu
        SymptomNameComparator nameComparator = new SymptomNameComparator();
        SeverityIndexComparator severityComparator = new SeverityIndexComparator();

        // Lehendabizi symptomName ordenatua inprimatu
        System.out.println("==== Sintomak symptomName arabera ordenatuta ===");
    }
}

```

```
Iterator<Object> sortedByName = Sorting.sortedIterator(adapter, nameComparator);
while (sortedByName.hasNext()) {
    Symptom s = (Symptom) sortedByName.next();
    System.out.println(" - " + s.getName() + " (severityIndex: " + s.getSeverityIndex() +
    ")");
}
System.out.println();

// Adapter berri bat sortu bigarren ordenaziorako
adapter = new Covid19PacientAdapter(pacient);

// Jarraian severityIndex ordenatuaz inprimatu
System.out.println("== Sintomak severityIndex arabera ordenatuta ==");
Iterator<Object> sortedBySeverity = Sorting.sortedIterator(adapter,
severityComparator);
while (sortedBySeverity.hasNext()) {
    Symptom s = (Symptom) sortedBySeverity.next();
    System.out.println(" - " + s.getName() + " (severityIndex: " + s.getSeverityIndex() +
    ")");
}
}
```