

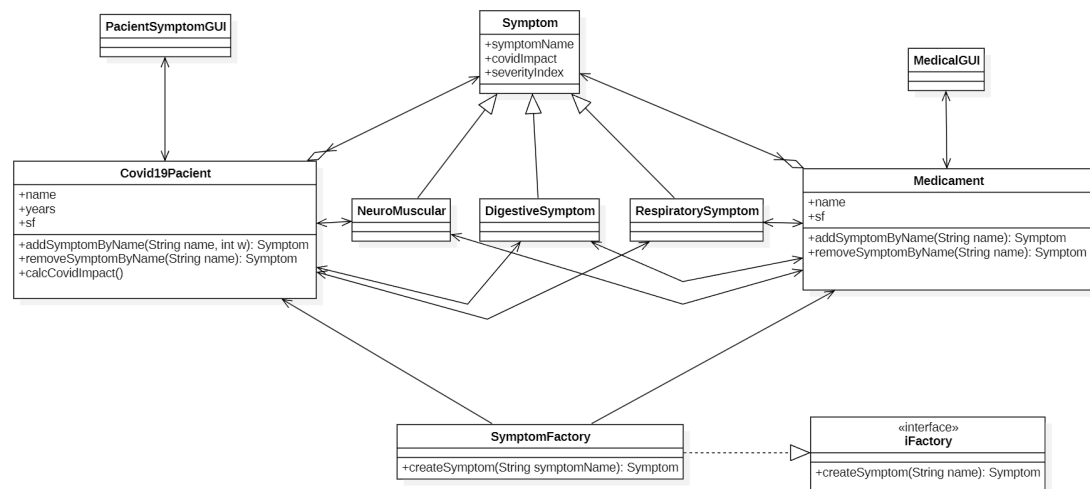
# **DISEINU PATROIAK LABORATEGIA**

Eneko Ruiz, Itxaso Urgoitia eta Oihane Pereira

Github-eko esteka: <https://github.com/ItxasoUrgoitia/labpatterns>

# Simple Factory patroia

## 1. UML diagrama Simple Factory patroia aplikatuz



SymptomFactory deituriko klasea gehitu dugu. Klase honek, iFactory interfasea implementatuko du. Klase hau Sympton berriak sortzeko erabiltzen dugu. Covid19Pacient eta Medicament klaseek SymptomFactory erabiliko dute atributu gisa (sf). Beraz ez du bakoitzak createSymptom metodoa implementatu beharko. `sf.createSymptom(symptom)`; eginez sortuko dituztelako Sympton berriak.

## 2. Aplikazioa implementatu, eta "mareos" sintoma berria gehitu 1 inpaktuarekin.

```
public class Main {
    public static void main(String[] args) {
        Covid19Pacient p1=new Covid19Pacient("aitor", 35);
        //PatientSymptomGUI psGUI1 = new PatientSymptomGUI(p1);
        new PatientSymptomGUI(p1);
        p1.addSymptomByName("mareos", 1);
        new MedicalGUI(new Medicament("Ibuprofeno"));
        //bi horiek izango dira clientak eta iFactory erabiliko dute objektuak sortzeko. sintomak sortzeko.
    }
}
```

### 3. Nola egokitu daiteke Factory klasea, Covid19Pacient eta Medicament erabiltzen dituzten Symptom objektuak bakarrak izateko?

Singleton erabili beharko genuke hori lortzeko. Negritaz dagoena da egin ditugun aldaketak singleton patroia erabiltzeko.

```
public class SymptomFactory implements iFactory {
    private static SymptomFactory instance;
    private Map<String, Symptom> symptomG;
    private SymptomFactory() {
        symptomG = new HashMap<>();
    }
    public static synchronized SymptomFactory getInstance() {
        if (instance == null) {
            instance = new SymptomFactory();
        }
        return instance;
    }
    @Override
    public Symptom createSymptom(String symptomName) {
        if (symptomG.containsKey(symptomName)) {
            return symptomG.get(symptomName);
        }
        System.out.println("Simptom name aaa: "+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"...);
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8);

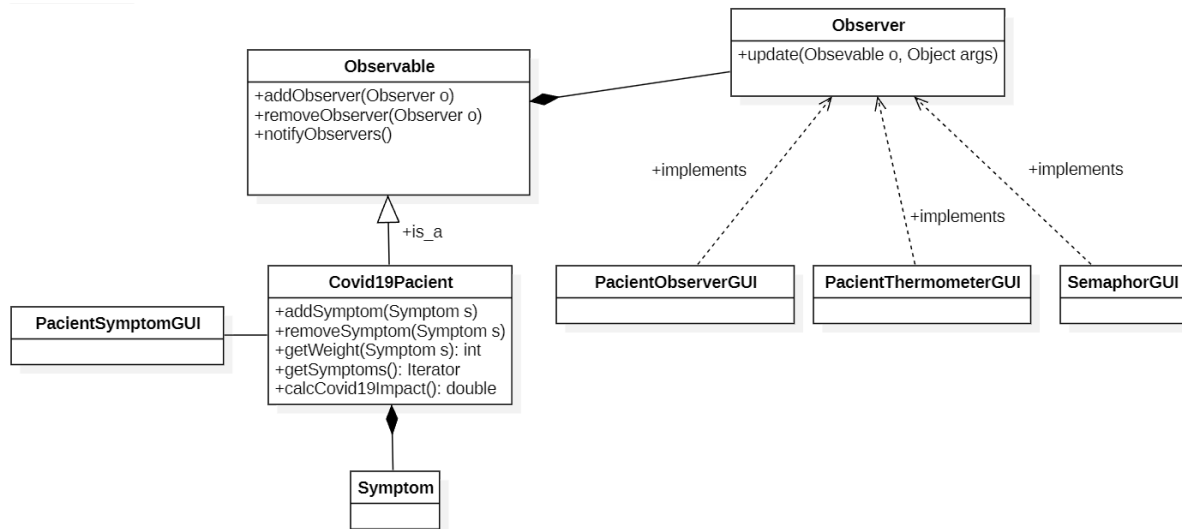
        List<String> digestiveSymptom=Arrays.asList("nauseas", "vómitos", "diarrea");
        List<String> neuroMuscularSymptom=Arrays.asList("fiebre", "astenia", "cefalea", "mialgia", "escalofrios");
        List<String> respiratorySymptom=Arrays.asList("tos seca", "expectoracion", "disnea", "dolor de garganta", ...);
        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 symptom = null;
        if (impact!=0) {
            if (digestiveSymptom.contains(symptomName)){return new DigestiveSymptom(symptomName,(int)index, impact);}
            if (neuroMuscularSymptom.contains(symptomName)) {
                System.out.println("Simptom name: "+symptomName);
                return new NeuroMuscularSymptom(symptomName,(int)index, impact);
            }
            if (respiratorySymptom.contains(symptomName)){return new RespiratorySymptom(symptomName,(int)index, impact);}
            if (symptom != null) {
                symptomG.put(symptomName, symptom);
            }
        }
        return symptom; } }
```

Covid19Pacient klasean eta Medicament klasean orain sf horrela hasieratu beharko dugu:

```
private iFactory sf = SymptomFactory.getInstance();
```

# Observer patroia

## 1. UML diagrama Observer patroia aplikatuz



## 2. Aplikazioaren implementazioa.

### Lehen prototipoa

#### 1. PAUSOA: CovidPacient Observable klasea.

Lehenengo, CovidPacient klaseari, **extends** Observable gehitu behar diogu. Sintoma bat gehitu edo ezabatzen den bakoitzean, bere subskribatzailei notifikatu behar zaie, eta horretarako, negritaz dagoena gehitu beharko dugu **Symptom** `addSymptomByName(String symptom, Integer w)` eta `removeSymptomByName(String symptomName)` metodoetan.

```
public Symptom addSymptomByName(String symptom, Integer w){
    Symptom s=null;
    s=sf.createSymptom(symptom);
    System.out.println("Simptom added 1: "+s);
    if (s!=null) {
        symptoms.put(s,w);
        System.out.println("Simptom added 2: "+s);
        setChanged();
        notifyObservers(s);
    }
    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(s);
    }
}
```

```

    }
    return s;
}

```

## 2. PAUSOA: PacientObserverGUI Observer klasea.

PacientObserverGUI klaseak Observer implementatu beharko du. Horretarako, hau gehituko diogu.

**implements** Observer.

Metodo eraikitzaileari "o" izeneko Observable parametro bat gehitu. Parametro hau subskribatu nahi dugun objektua izango da.

**public** PacientObserverGUI(Observable o) {...}

Metodoaren eraikitzailearen bukaeran, pasatutako parametroari subskribatu nahi dela adieraziko dugu.

o.addObserver(**this**);

Observer interfazea implementatzen duen update metodoa eguneratuko dugu. Metodo honetan, notifikazioak jasoko dira subskribatuta gauden objektua aldatzen denean (Observable o objektua). Lehendabiziko prototipoan args null izango da.

```

@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() + " - " + p2.getWeight(p2) + "<br>";
    }
    s = s + "</html>";
    symptomLabel.setText(s);
}

```

## 3. PAUSOA: PacientSymptomGUI klasea.

Metodo eraikitzaileari Covid19Pacient motako parametro bat gehituko diogu, aurreko "pacient" aldagaia eguneratzeko.

**public** PacientSymptomGUI(Covid19Pacient p) {...}

Bi botoien listener-etan akzioak ipiniko ditugu, pacient objektuari sintoma bat gehitu edo ezabatzeko.

```

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

```

```

:+"(Symptom)symptomComboBox.getSelectedItemId();
//addSymptomByName ...

p.addSymptomByName(((Symptom)symptomComboBox.getSelectedItemId()).getName(),
Integer.parseInt(weightField.getText()));

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

btnRemoveSymptom.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        errorLabel.setText(" ");
        System.out.println("Symptom removed
:+"(Symptom)symptomComboBox.getSelectedItemId();
//removeSymptomByName...

p.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItemId()).getName());
    }
});

```

#### 4. PAUSOA: Programa nagusia sortu.

```

public class Main {
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        Observable pacient=new Covid19Pacient("aitor", 35);
        new PatientObserverGUI (pacient);
        new PatientSymptomGUI ((Covid19Pacient) pacient);
    }
}

```

Programa nagusia aldatuko dugu, 2 Covid19Pacient sortzeko bere ondoko PatientSymptomGUI interfazearekin:

```

public class Main {
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        Observable pacient=new Covid19Pacient("aitor", 35);
        new PatientObserverGUI (pacient);
        new PatientSymptomGUI ((Covid19Pacient) pacient);

        Observable pacient2=new Covid19Pacient("eneko", 20);
        new PatientObserverGUI (pacient2);
        new PatientSymptomGUI ((Covid19Pacient) pacient2);
    }
}

```

## Bigarren prototipoa

1. PAUSOA: Observer klasea implementatu

```
public class PacientThermometerGUI extends Frame implements Observer {...}
```

2. PAUSOA: Metodo eraikitzaileari "o" izeneko Observable parametro bat gehitu. Parametro hau subskribatu nahi dugun objektua izango da. Metodoaren eraikitzailearen bukaeran, pasatutako parametroari subskribatu nahi dela adieraziko dugu.

```
public PacientThermometerGUI(Observable o) {  
    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);  
  
    o.addObserver(this);  
}
```

3. PAUSOA: Observer interfazea implementatzen duen update metodoa eguneratuko dugu.

```
@Override  
public void update(Observable o, Object args) {  
    Covid19Pacient p = (Covid19Pacient) o;  
    int fahrenheit = (int) p.covidImpact();  
    gauges.set(fahrenheit);  
    gauges.repaint();  
}
```

4. PAUSOA: Programa nagusi bat sortu hiru pazientekin bakoitzak bere 2 interfaze grafikoein.

```
public static void main(String[] args) {  
    Observable pacient=new Covid19Pacient("aitor", 35);  
    new PacientObserverGUI (pacient);  
    new PacientThermometerGUI (pacient);  
    new PacientSymptomGUI ((Covid19Pacient) pacient);  
  
    Observable pacient2=new Covid19Pacient("eneko", 20);  
    new PacientObserverGUI (pacient2);  
    new PacientThermometerGUI (pacient2);  
  
    Observable pacient3=new Covid19Pacient("itxaso", 19);  
    new PacientObserverGUI (pacient3);  
    new PacientThermometerGUI (pacient3);  
}
```

## Hirugarren prototipoa

Observer klasea implementatu behar dugu lehenengo.

```
public class SemaphorGUI extends JFrame implements Observer {...}
```

Metodo eraikitzaileari "o" izeneko Observable parametro bat gehitu. Parametro hau subskribatu nahi dugun objektua izango da. Metodoaren eraikitzailearen bukaeran, pasatutako parametroari subskribatu nahi dela adieraziko dugu.

```
public SemaphorGUI (Observable o) {  
    ...  
    o.addObserver(this);  
}
```

Observer interfazea implementatzen duen update metodoa eguneratuko dugu ere. Horrela geratuko da beraz gure SemaphorGUI klasea.

```
public class SemaphorGUI extends JFrame implements Observer {  
    /** stores the associated ConcreteSubject */  
    public SemaphorGUI (Observable o) {  
        setSize(100, 100);  
        setLocation(350,10);  
        Color c=Color.green;  
        getContentPane().setBackground(c);  
        repaint();  
        setVisible(true);  
  
        o.addObserver(this);  
    }  
    @Override  
    public void update(Observable o, Object arg) {  
        Covid19Pacient p = (Covid19Pacient) o;  
        Color c;  
        double current = p.covidImpact();  
        if (current < 5)  
            c = Color.green;  
        else if (current <= 10)  
            c = Color.yellow;  
        else  
            c = Color.red;  
        getContentPane().setBackground(c);  
        repaint();  
    }  
}
```

Amaitzeko, programa nagusia egingo dugu. Programa nagusi honek paziente bakoitzeko aurretik erakutsi ditugun interfazeaz gain, Semaphor interfazea ere erakutsiko du.

```
public static void main(String[] args) {  
    Observable pacient=new Covid19Pacient("aitor", 35);  
    new PacientObserverGUI (pacient);  
    new PacientThermometerGUI (pacient);  
    new PacientSymptomGUI ((Covid19Pacient) pacient);  
    new SemaphorGUI (pacient);  
  
    Observable pacient2=new Covid19Pacient("eneko", 20);
```

```

new    PacientObserverGUI    (pacient2);
new    PacientThermometerGUI (pacient2);
new    PacientSymptomGUI    ((Covid19Pacient) pacient2);
new    SemaphorGUI (pacient2);

Observable    pacient3=new    Covid19Pacient("itxaso",    19);
new    PacientObserverGUI    (pacient3);
new    PacientThermometerGUI (pacient3);
new    PacientSymptomGUI    ((Covid19Pacient) pacient3);
new    SemaphorGUI (pacient3);

```

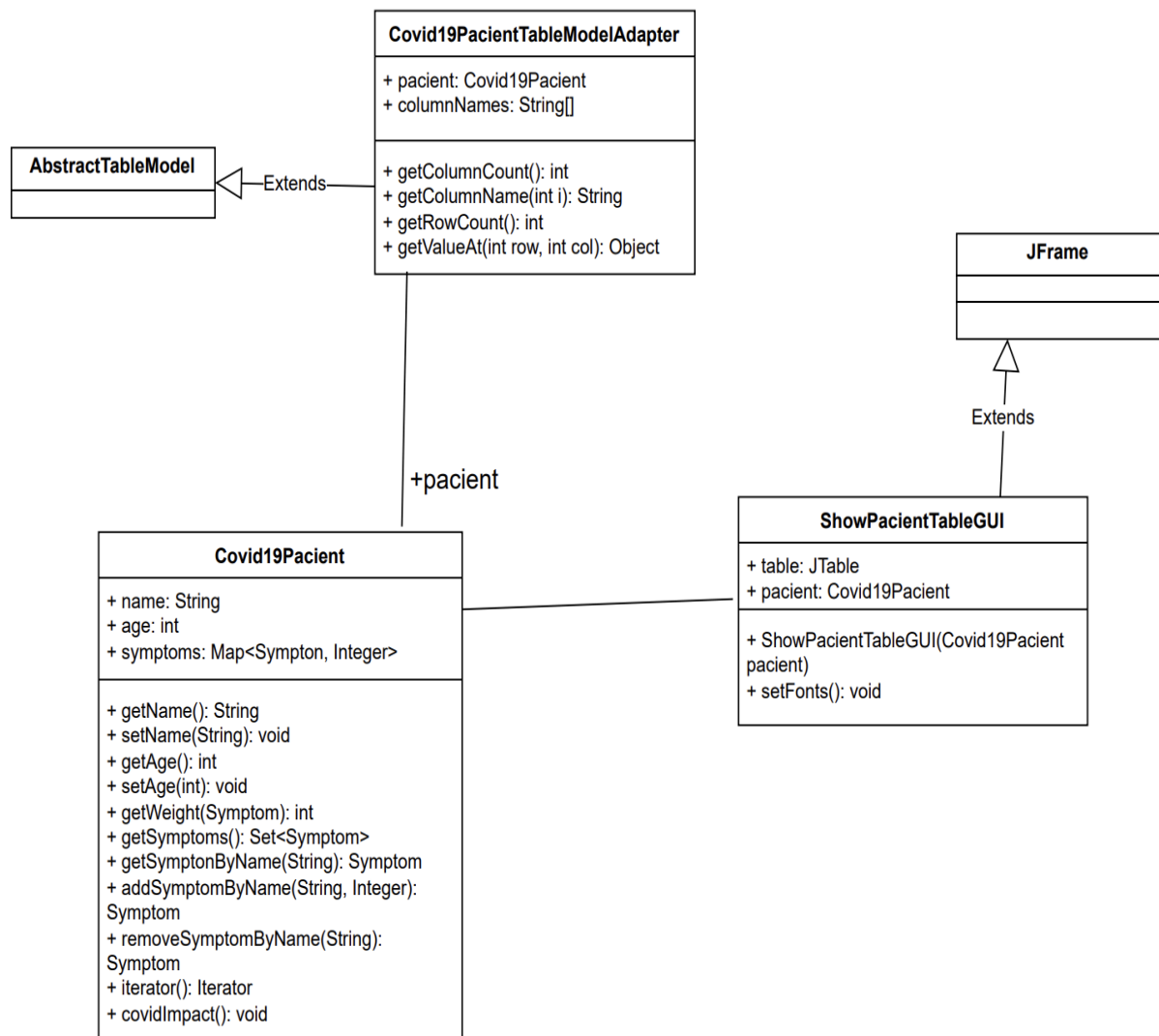
```

}

```

# Adapter patroia

## 1. UML diagrama Adapter patroia aplikatuz



## 2. Aplikazioaren implementazioa.

### 2.1. Covid19PacientTableModelAdapter klasea

```
package adapter2;
import java.util.Iterator;
import javax.swing.table.AbstractTableModel;
import domain.Covid19Pacient;
import domain.Symptom;
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() {
        // Challenge!
        return columnNames.length;
    }
    public String getColumnName(int i) {
        // Challenge!
        return columnNames[i];
    }
    public int getRowCount() {
        // Challenge!
        return pacient.getSymptoms().size();
    }
    public Object getValueAt(int row, int col) {
        // Challenge!
        if (row < 0 || row >= getRowCount() || col < 0 || col >= getColumnCount()) {
            return null;
        }

        Iterator<Symptom> iterator = pacient.getSymptoms().iterator();
        Symptom symptom = null;

        // Lortu errenkadako sintoma
        for (int i = 0; i <= row; i++) {
            if (iterator.hasNext()) {
                symptom = iterator.next();
            }
        }

        if (symptom == null) {
            return null;
        }

        // Itzuli zutabeko balioa
        switch (col) {
            case 0: // Symptom zutabea
                return symptom.getName();
            case 1: // Weight zutabea
                return pacient.getWeight(symptom);
            default:
                return null;
        }
    }
}
```

Covid19PacientTableModelAdapter klase berri honek Adapter patroia implementatzen du, Covid19Pacient objektua TableModel interfazera egokitzeko. Klase honen helburua da paziente baten sintomak taula grafiko batean bistaratu ahal izatea, JTable osagaiak espero duen formatura moldatuz

## 2.2. ShowPacientTableGUI klasea

```
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 patient;

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

        this.patient=patient;

        setFonts();

        TableModel tm=new Covid19PacientTableModelAdapter(patient);
        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);
    }
}
```

ShowPacientTableGUI klase berri honek paziente baten sintomak taula grafiko batean bistartzeko interfazea definitzen du. Klase honen eraikitzaileak paziente bat hartu eta leihoaren titulua ezartzen du pazientearen izenarekin.

## 2.3 Proba eta emaitzak (Main klasea)

```
package adapter2;
import domain.Covid19Pacient;
public class Main {
    public static void main(String[] args) {

        Covid19Pacient patient=new Covid19Pacient("aitor", 35);

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

        ShowPacientTableGUI gui=new ShowPacientTableGUI(patient);
        gui.setPreferredSize(
```

```

        new java.awt.Dimension(300, 200));
gui.setSize(600, 400);
gui.setVisible(true);

Covid19Pacient pacient2=new Covid19Pacient("itxaso", 19);

pacient2.addSymptomByName("fiebre", 1);
pacient2.addSymptomByName("nauseas", 2);

ShowPacientTableGUI gui2=new ShowPacientTableGUI(pacient2);
gui2.setPreferredSize(
    new java.awt.Dimension(300, 200));
gui2.setSize(600, 400);
gui2.setVisible(true);
    }
}

```

Main klaseak aplikazioa probatzeko kodea dauka. Bi paziente sortzen ditu sintoma desberdinekin: Aitor pazienteak disnea, cefalea eta astenia sintomak ditu, eta Itxaso pazienteak fiebre eta nauseas sintomak.

**3. (HAUTAZKOA)** Paziente bati sintoma berri bat gehitzen zaion bakoitzean bere taula leihoa eguneratzeko. Guk gure kodean ez dugu hau inplementatu baina horrela izango zen:

```

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 implements Observer {

    JTable table;
    private Covid19PacientTableModelAdapter adapter;
    //Covid19Pacient pacient;

    public ShowPacientTableGUI(Covid19Pacient patient ) {

        // 1. Adapter sortu taularako
        adapter = new Covid19PacientTableModelAdapter(patient);
        table = new JTable(adapter);

        // 2. Observer moduan erregistratu
        patient.addObserver(this); // Observer patternarekin konekatzeko
    }

    // 3. Observer metodoa
    @Override
    public void update(Observable o, Object arg) {
        // Pazientea aldatzen denean, taula aldatu adapterrekin
        table.repaint(); // edo adapter berria sortu beharrezkoa bada
    }
}

```

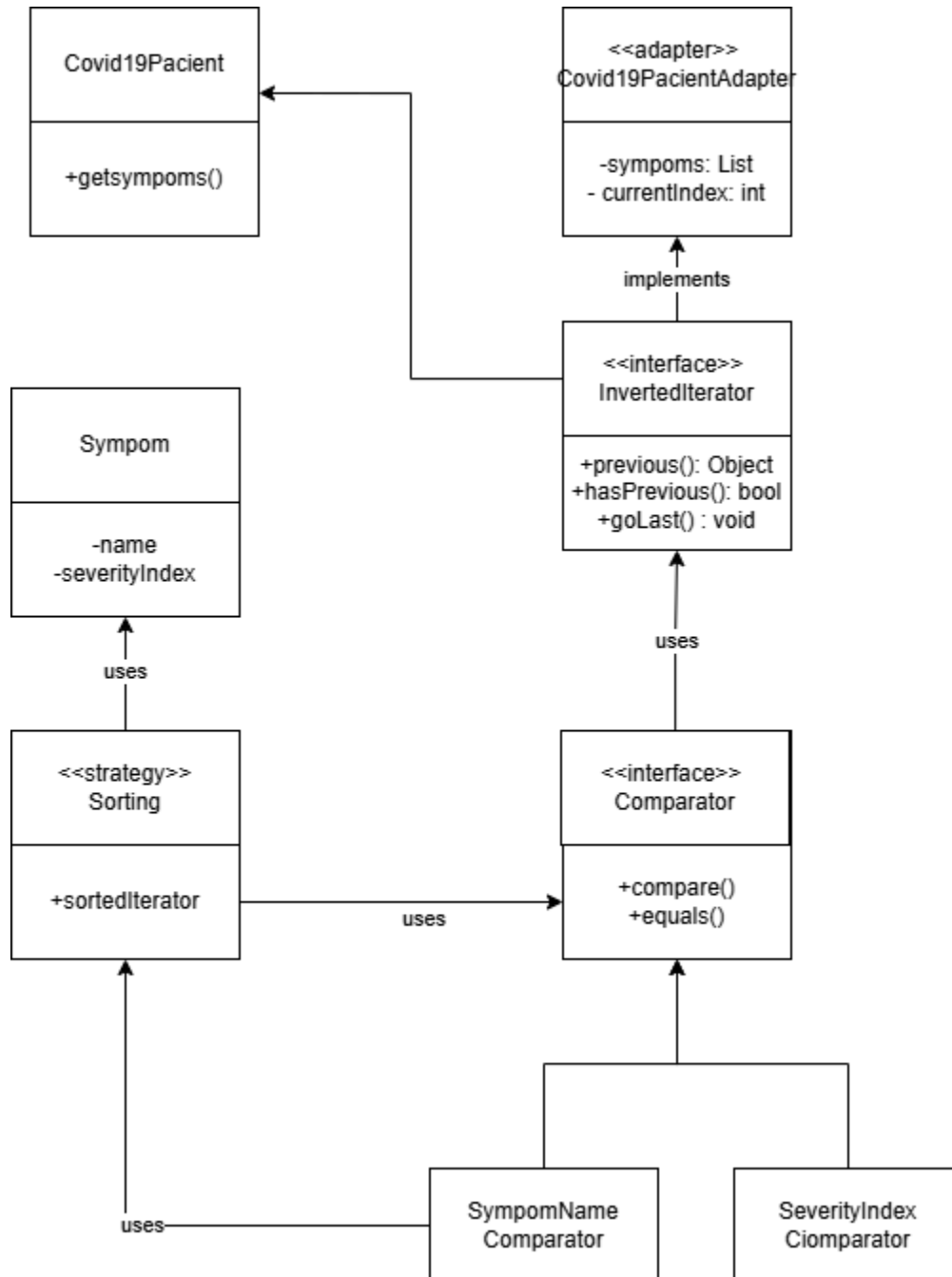
ShowPacientTableGUI klasea Observer interfazea inplementatzeko aldatu daiteke, taula automatikoki eguneratzeko pazientearen datuak aldatzen direnean. Eraikitzailean, leihoa pazientearen bezero gisa

erregistratzen da, eta update metodoak taula berriro kargatzen du pazienteak sintoma berri bat gehitzen edo kentzen duenean.

Integrazio honek ahalbidetzen du taula eguneratzea erabiltzaileak sintomak aldatzen dituenean, eskuarki beste interfaze batetik. Observer patroiarri esker, Covid19Pacient klaseak bere bezeroei jakinarazten die aldaketak, eta ShowPacientTableGUI klaseak erantzun egiten du taula berriro kargatuz.

# Adapter Iterator eta Patroiak

1. UML diagrama aplikazioaren diseinuarekin.



## 2. Aplikazioaren inplementazioa

### 2.1. Comparator interfazea inplementatzen dituzten bi klase definitu elementuak symptomName eta severityIndex ordenatzen dituztenak hurrenez hurren

```
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());
    }

    @Override
    public boolean equals(Object obj) {
        return obj instanceof SymptomNameComparator;
    }
}
```

SymptomNameComparator klase berri honek sintomen izenak alfabetikoki ordenatzeko estrategia definitzen du. Klase honen compare metodoak bi sintomen izenak konparatzen ditu String klasearen compareTo metodoa erabiliz, eta horrela sintomak A-tik Z-ra ordenatuko dira. Adibidez, "astenia" eta "cefalea" sintomen artean, "astenia" aurretik jarriko da bere A letra C baino lehenago baitago alfabetoan.

```
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());
    }

    @Override
    public boolean equals(Object obj) {
        return obj instanceof SeverityIndexComparator;
    }
}
```

SeverityIndexComparator klase berri honek sintomen severity index-a goranzko ordenan ordenatzeko estrategia definitzen du. Bere compare metodoak bi sintomen severity index-ak konparatzen ditu Integer klasearen compare metodoa erabiliz, eta horrela sintomak severity index txikienetik handienara ordenatuko dira. Adibidez, severity index 1 duen sintoma severity index 5 duen sintomaren aurretik jarriko da.

### 2.2. Covid19Pacient klasea InvertedIterator interfazera egokitzen duen klase adaptadorea sortu

```

package adapter;
import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import domain.Covid19Pacient;
import domain.Symptom;
public class Covid19PacientAdapter implements InvertedIterator {
    private List<Symptom> symptoms;
    private int currentIndex;

    public Covid19PacientAdapter(Covid19Pacient pacient) {
        this.symptoms = new ArrayList<>(pacient.getSymptoms());
        this.currentIndex = symptoms.size() - 1;
    }

    @Override
    public Object previous() {
        if (hasPrevious()) {
            return symptoms.get(currentIndex--);
        }
        return null;
    }

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

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

```

Covid19PacientAdapter klase berri honek Covid19Pacient objektua InvertedIterator interfazera egokitzen du. Klase honen eraikitzaileak Covid19Pacient objektu bat hartu eta bere sintomen Set-a List batera bihurtzen du, eta currentIndex aldagaia azken elementuan kokatzen du. Previous metodoak elementu aktuala itzultzen du eta currentIndex jaisten du atzeraka mugitzeko. HasPrevious metodoak egiaztatzen du ea korritu gabeko elementurik dagoen, eta goLast metodoak iteradorea azken elementuan kokatzeko aukera ematen du. Adapter honen bidez, Covid19Pacient objektuaren sintomak atzeraka korritu daitezke InvertedIterator interfazea erabiliz.

### 2.3. Programa nagusi batean, Covid19Pacient objektu bat sortu 5 sintomekin, eta Sorting.sort metodoari bi aldiz deitu

```

package adapter;
import domain.Covid19Pacient;
import domain.Symptom;
import java.util.Iterator;
public class Main {
    public static void main(String[] args) {

        Covid19Pacient pacient = new Covid19Pacient("Juan", 40);

        pacient.addSymptomByName("fiebre", 3);
        pacient.addSymptomByName("tos seca", 2);
        pacient.addSymptomByName("cefalea", 1);
        pacient.addSymptomByName("astenia", 4);
    }
}

```

```

    pacient.addSymptomByName("disnea", 5);

    Covid19PacientAdapter adapter = new Covid19PacientAdapter(pacient);

    SymptomNameComparator nameComparator = new SymptomNameComparator();
    SeverityIndexComparator severityComparator = new SeverityIndexComparator();

    System.out.println("SINTOMAK IZENAREN ARABERA ORDENATUTA");
    Iterator<Object> sortedByName = Sorting.sortedIterator(adapter, nameComparator);
    while (sortedByName.hasNext()) {
        Symptom s = (Symptom) sortedByName.next();
        System.out.println("Sintoma: " + s.getName() + ", Severity Index: " + s.getSeverityIndex());
    }

    System.out.println("\nSINTOMAK SEVERITY INDEX-EN ARABERA ORDENATUTA");

    Covid19PacientAdapter adapter2 = new Covid19PacientAdapter(pacient);
    Iterator<Object> sortedBySeverity = Sorting.sortedIterator(adapter2, severityComparator);
    while (sortedBySeverity.hasNext()) {
        Symptom s = (Symptom) sortedBySeverity.next();
        System.out.println("Sintoma: " + s.getName() + ", Severity Index: " + s.getSeverityIndex());
    }
}
}

```

Main klase berri honetan, Covid19Pacient objektu bat sortzen da bost sintomarekin: "fiebre", "tos seca", "cefalea", "astenia" eta "disnea". Ondoren, Covid19PacientAdapter bat sortzen da paziente horretatik, eta bi konparatzaileak sortzen dira: SymptomNameComparator eta SeverityIndexComparator. Lehenengo, Sorting.sortedIterator metodoa deitzen da adapterra eta izenaren konparatzailea pasatuz, eta sintomak izenaren arabera ordenatuta inprimatzen dira. Bigarrenik, adapter berri bat sortu behar da lehenengo adapterra agortuta dagoelako, eta Sorting.sortedIterator metodoa deitzen da berriro adapter berriarekin eta severityIndex konparatzailearekin, sintomak severityIndexaren arabera ordenatuta inprimatuz.