

## Simple factory patroia

1. Zer gertatzen da, sintoma mota berri bat agertzen bada (adb: MovilitySymptom)?

SRP printzipioa hausten du, sintoma berri bat sartzerako garaian, Covid19Pacient klasea aldatu behar delako beti.

2. Nola sortu daiteke sintoma berri bat orain arte dauden klaseak aldatu gabe (OCP printzipioa)?

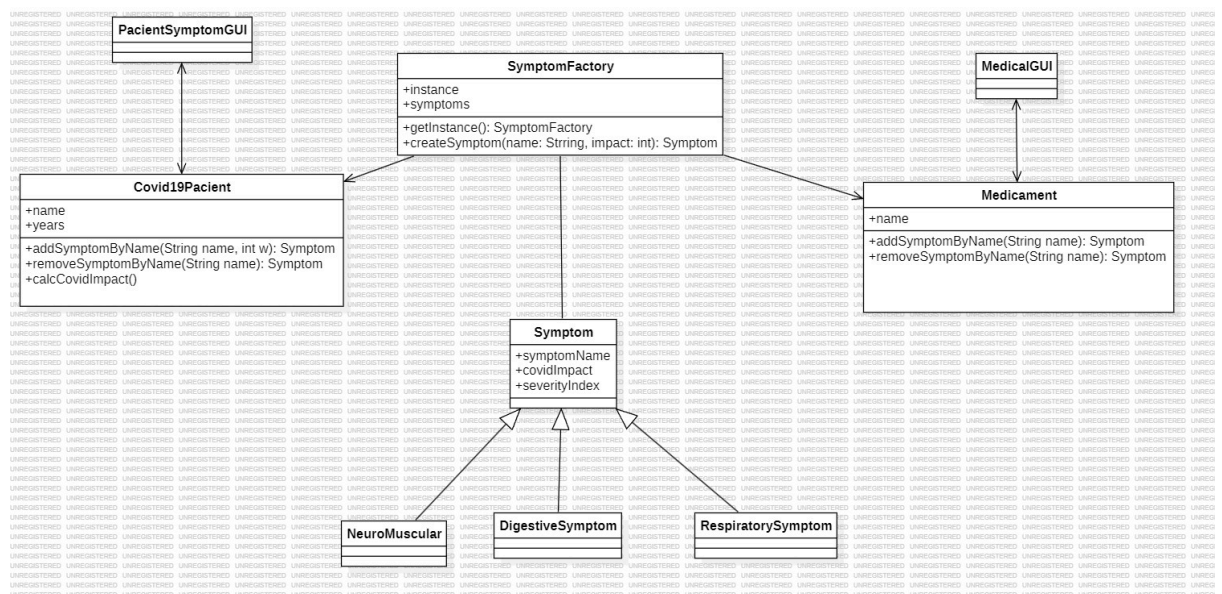
Kasu honetarako, klase berri bat sortu beharko litzateke, adibidez SymptomFactory deitzen dena, arduratuko dena sintoma berriak sortzen.

3. Zenbat erresponsabilitate dauzkate Covid19Pacient eta Medicament klaseak (SRP printzipioa)?

Covid19Pacient klaseak lau erresponsabilitate ditu eta Medicament klaseak hiru erresponsabilitate ditu. Honek esan nahi du SRP printzipioa hausten dela, gehienez klase batek erresponsabilitate bat izan behar duelako.

### Eskatzen da:

1. Aplikazioaren diseinu berri bat egin (UML diagrama) Simple Factory patroia aplikatuz, aurreko ahultasunak ezabatzeko. Jarraian deskribatu testu batean egin dituzun aldaketak.



Aplikazioaren diseinu berrirako SymptomFactory klasea sortu dut. Klase honekin, sintoma berriak sortzen ditut. Hau eginda, erresponsabilitate bat kentzen dut Symptom klaseari, eta gainera sintoma berriak sortu ditzaket klaseak aldatu gabe. Klase honetan, esan bezala, createSymptom funtzioa implementatu dut, funtzioak sortzeko.

---

## SOFTWARE INGENIARITZA II

Funtzioa Covid19Pacient eta Medicament klaseetako atzeko funtzioa da, zenbait aldaketak dituena. Honez gain, klase hauetako createSymptom funtzioak ezabatu ditut.

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

package factory;

```
import domain.Covid19Pacient;
import domain.Medicament;
import domain.Symptom;
import domain.SymptomFactory;
```

```
public class Main {
```

```
    public static void main(String[] args) {
        Covid19Pacient p1 = new Covid19Pacient("aitor", 35);
        PatientSymptomGUI psGUI1 = new PatientSymptomGUI(p1);
```

```
        SymptomFactory fac = SymptomFactory.getInstance();
        Symptom mareos = fac.createSymptom("Mareos", 1);
```

```
        fac.createSymptom("Mareos", 1);
```

```
        if (mareos != null) {
            p1.addSymptomByName(mareos.getName(), 1);
            System.out.println("Sintoma ondo inplementatu da");
        } else {
            System.out.println("Sintoma ez da ondo inplementatu");
        }
    }
}
```

3. Nola egokitu daiteke Factory klasea, Covid19Pacient eta Medicament erabiltzen dituzten Symptom objektuak bakarrak izateko? Hau da, sintoma bakoitzeko objektu bakar bat egon dadin. (x sintoma sisteman badaude, x objektu Symptom soilik)

Hau lortzeko, singleton patroia aplikatuz egingo dugu. Kasu honetan, SymptomFactory klasean instance objetua eta getInstance funtzioa sortuko dugu. Honela geldituko litzateke klase osoa singleton patroia aplikatuz:

## SOFTWARE INGENIARITZA II

```

package domain;

import java.awt.List;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
import java.util.*;

public class SymptomFactory {
    private static SymptomFactory instance;
    private Map<String, Symptom> symptoms = new HashMap<>();

    public SymptomFactory() {}

    public static SymptomFactory getInstance() {
        if (instance == null) {
            instance = new SymptomFactory();
        }
        return instance;
    }

    public Symptom createSymptom(String symptomName, int weight) {
        if (symptoms.containsKey(symptomName)) {
            return symptoms.get(symptomName);
        }

        java.util.List<String> impact5 = Arrays.asList("fiebre", "tos seca", "astenia", "expectoracion");
        java.util.List<Double> index5 = Arrays.asList(87.9, 67.7, 38.1, 33.4);
        java.util.List<String> impact3 = Arrays.asList("disnea", "dolor de garganta", "cefalea", "mialgia", "escalofrios");
        java.util.List<Double> index3 = Arrays.asList(18.6, 13.9, 13.6, 14.8, 11.4);
        java.util.List<String> impact1 = Arrays.asList("nauseas", "v@mitos", "congesti@n nasal", "diarrea", "hemoptisis", "congestion conjuntival");
        java.util.List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8);

        java.util.List<String> digestiveSymptom = Arrays.asList("nauseas", "v@mitos", "diarrea");
        java.util.List<String> neuroMuscularSymptom = Arrays.asList("fiebre", "astenia", "cefalea", "mialgia", "escalofrios");
        java.util.List<String> respiratorySymptom = Arrays.asList("tos seca", "expectoracion", "disnea", "dolor de garganta", "congesti@n nasal",

        int impact = 0;
        double index = 0;

        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 newSymptom = null;

        if (impact != 0) {
            if (digestiveSymptom.contains(symptomName)) {
                newSymptom = new DigestiveSymptom(symptomName, (int) index, impact);
            } else if (neuroMuscularSymptom.contains(symptomName)) {
                newSymptom = new NeuroMuscularSymptom(symptomName, (int) index, impact);
            } else if (respiratorySymptom.contains(symptomName)) {
                newSymptom = new RespiratorySymptom(symptomName, (int) index, impact);
            }
        }

        if (newSymptom != null) {
            symptoms.put(symptomName, newSymptom);
        }

        return newSymptom;
    }
}
  
```

# Observer Patroia

## Lehendabiziko prototipoa

1. Pausoa: CovidPacient Observable klasea.

```

package domain;
import java.util.Observable;
public class Covid19Pacient extends Observable {
    public Symptom addSymptomByName(String symptom, Integer w) {
        Symptom s = getSymptomByName(symptom);
        if (s == null) {
            s = createSymptom(symptom);
            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;
    }
}

```

2. Pausoa: PacientObserverGUI Observer klasea.

```

public class PacientObserverGUI extends JFrame implements Observer {
    private JPanel contentPane;
    private final JLabel symptomLabel = new JLabel("");
    /**
     * Create the frame.
     */
    public PacientObserverGUI(Observable obs) {
        obs.addObserver(this);
        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);
    }
}

```

## SOFTWARE INGENIARITZA II

```

@Override
public void update(Observable o, Object args) {
    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 + "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. Pausoa: PacientSymptomGUI klasea.

```

public class PacientSymptomGUI extends JFrame {
    ...
    private Covid19Pacient pacient;
    public PacientSymptomGUI(Covid19Pacient pacient) {
        this.pacient = pacient;
        ...
        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());

                    // addSymptomByName ...
                    pacient.addSymptomByName(((Symptom)
symptomComboBox.getSelectedItem()).getName(),
                    Integer.parseInt(weightField.getText()));
                } else
                    errorLabel.setText("ERROR, Weight between [1..3]");
            }
        });
        ...
        btnRemoveSymptom = new JButton("Remove Symptom");
        btnRemoveSymptom.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                errorLabel.setText("");
                System.out.println("Symptom removed : " + (Symptom)
symptomComboBox.getSelectedItem());

                // removeSymptomByName...
                pacient.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName());
            }
        });
        ...
    }
}

```

## SOFTWARE INGENIARITZA II

### 4. Pausoa: Programa nagusia sortu.

```
package observer;

import java.util.Observable;
import domain.Covid19Pacient;

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

#### Eskatzen da:

Programa nagusia aldatu 2 Covid19Pacient sortzeko bere ondoko PacientSymptomGUI interfazearekin.

```
package observer;

import java.util.Observable;
import domain.Covid19Pacient;

public class Main {
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        Observable pacient1 = new Covid19Pacient("Aitor", 35);
        new PacientSymptomGUI((Covid19Pacient) pacient1);
        new PacientObserverGUI(pacient1);
        Observable pacient2 = new Covid19Pacient("Leire", 28);
        new PacientSymptomGUI((Covid19Pacient) pacient2);
        new PacientObserverGUI(pacient2);
    }
}
```

## SOFTWARE INGENIARITZA II

### Bigarren prototipoa:

```

public class PacientThermometerGUI extends Frame implements Observer {
    private TemperatureCanvas gauges;
    /**
     * @wbp.nonvisual location=119,71
     */
    private final JLabel label = new JLabel("New label");

    public PacientThermometerGUI(Observable obs) {
        super("Temperature Gauge");
        obs.addObserver(this);
        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);
    }

    @Override
    public void update(Observable o, Object args) {
        Covid19Pacient p = (Covid19Pacient) o;
        // Obtain the current covidImpact to paint
        int fahrenheit = (int) p.covidImpact();
        // temperature gauge update
        gauges.set(fahrenheit);
        gauges.repaint();
    }
}

```

Prototipo honekin bukatzeko, **hurrengo ariketa eskatzen da:**

1. Programa nagusi bat sortu hiru pazientekin bakoitzak bere 2 interfaze grafikoekin, hurrengo irudian agertzen den bezala:

```

package observer;

import java.util.Observable;
import domain.Covid19Pacient;

public class Main {
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        Observable pacient1 = new Covid19Pacient("Aitor", 35);
        Observable pacient2 = new Covid19Pacient("Leire", 28);
        Observable pacient3 = new Covid19Pacient("Unai", 42);

        new PacientSymptomGUI((Covid19Pacient) pacient1);
        new PacientObserverGUI(pacient1);
        new PacientThermometerGUI(pacient1);

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

```

## SOFTWARE INGENIARITZA II

```

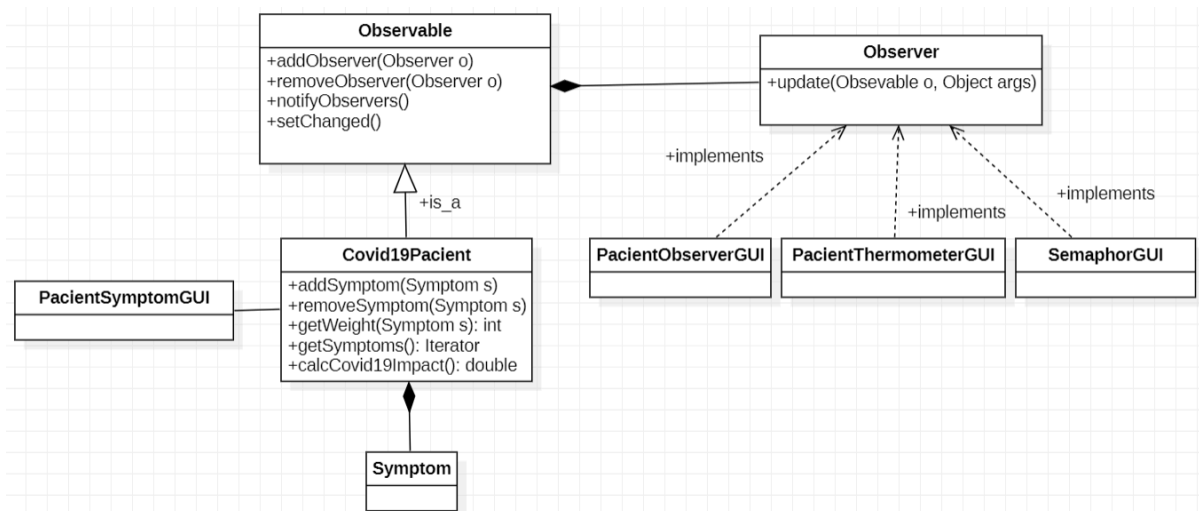
    new PatientSymptomGUI((Covid19Patient) pacient3);
    new PatientObserverGUI(pacient3);
    new PatientThermometerGUI(pacient3);
  }
}

```

Bukatzeko azken prototipo bat **garatu nahi da**. Azkeneko prototipoan semáforo pantaila bat garatu nahi da, covid19Impact balioen arabera ( $x < 5$  berdea,  $5 \leq x < 10$  oria eta  $x > 10$  gorria).

**Eskatzen da:**

1. Aplikazioaren UML diagrama hedatuta egin dituzun aldaketan aurkeztuz.



2. Aplikazioaren implementazioa.

```

package observer;

import java.awt.Color;

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

```





## SOFTWARE INGENIARITZA II

```
@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();
}
}

package observer;

import java.util.Observable;
import domain.Covid19Pacient;

public class Main {
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        Observable pacient1 = new Covid19Pacient("Aitor", 35);
        Observable pacient2 = new Covid19Pacient("Leire", 28);
        Observable pacient3 = new Covid19Pacient("Unai", 42);

        new PatientSymptomGUI((Covid19Pacient) pacient1);
        new PatientObserverGUI(pacient1);
        new PatientThermometerGUI(pacient1);
        new SemaphoreGUI(pacient1);

        new PatientSymptomGUI((Covid19Pacient) pacient2);
        new PatientObserverGUI(pacient2);
        new PatientThermometerGUI(pacient2);
        new SemaphoreGUI(pacient2);

        new PatientSymptomGUI((Covid19Pacient) pacient3);
        new PatientObserverGUI(pacient3);
        new PatientThermometerGUI(pacient3);
        new SemaphoreGUI(pacient3);
    }
}
```

## Adapter Patroia

Eskatzen da:

- Behar den kodea gehitu Covid19PacientTableModelAdapter klasean eta aplikazio exekutatu emaitza konprobatuz.

```

public class Covid19PacientTableModelAdapter extends AbstractTableModel {
    protected Covid19Pacient pacient;
    protected String[] columnNames = new String[] { "Symptom", "Weight" };

    public Covid19PacientTableModelAdapter(Covid19Pacient p) {
        this.pacient = p;
    }

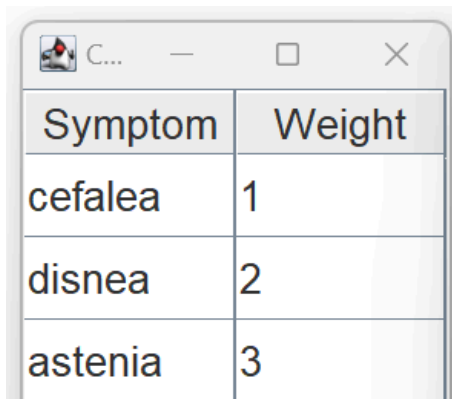
    public int getColumnCount() {
        // Challenge!
        return columnNames.length;
    }

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

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

    public Object getValueAt(int row, int col) {
        // Challenge!
        List<Symptom> symptomsList = new ArrayList<>(this.pacient.getSymptoms());
        Symptom symptom = symptomsList.get(row);
        if (col == 0) {
            return symptom.getName();
        } else if (col == 1) {
            return this.pacient.getWeight(symptom);
        } else {
            return null;
        }
    }
}

```



Symptom	Weight
cefalea	1
disnea	2
astenia	3

## SOFTWARE INGENIARITZA II

- Beste paziente bat gehitu sintoma batzuekin, eta aplikazioa exekutatu bi taulak agertzen direla konprobatuz.

```

package adapter2;

import domain.Covid19Pacient;

public class Main {

    public static void main(String[] args) {
        Covid19Pacient pacient1=new Covid19Pacient("aitor", 35);

        pacient1.addSymptomByName("disnea", 2);
        pacient1.addSymptomByName("cefalea", 1);
        pacient1.addSymptomByName("astenia", 3);

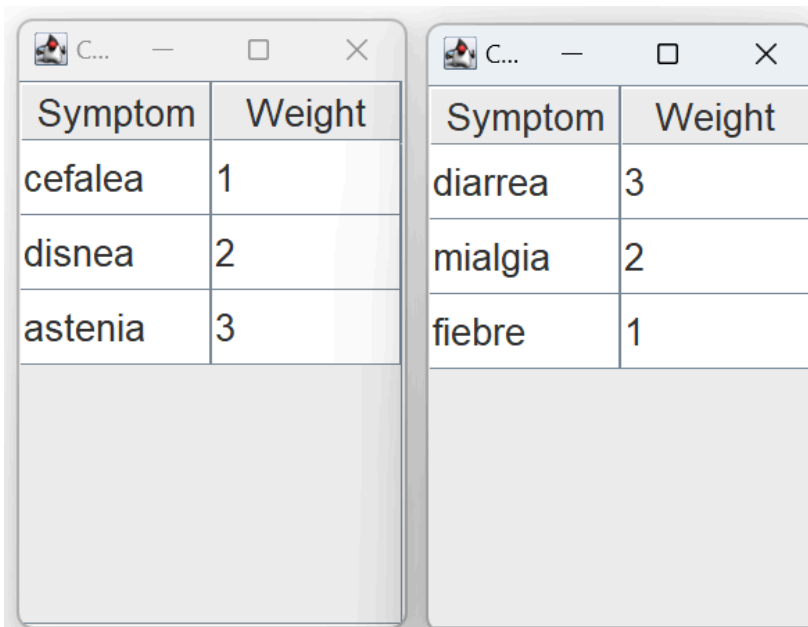
        ShowPacientTableGUI gui1=new ShowPacientTableGUI(pacient1);
        gui1.setPreferredSize(
            new java.awt.Dimension(300, 200));
        gui1.setVisible(true);

        Covid19Pacient pacient2=new Covid19Pacient("leire", 28);

        pacient2.addSymptomByName("fiebre", 1);
        pacient2.addSymptomByName("diarrea", 3);
        pacient2.addSymptomByName("mialgia", 2);

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

```



Symptom	Weight
cefalea	1
disnea	2
astenia	3

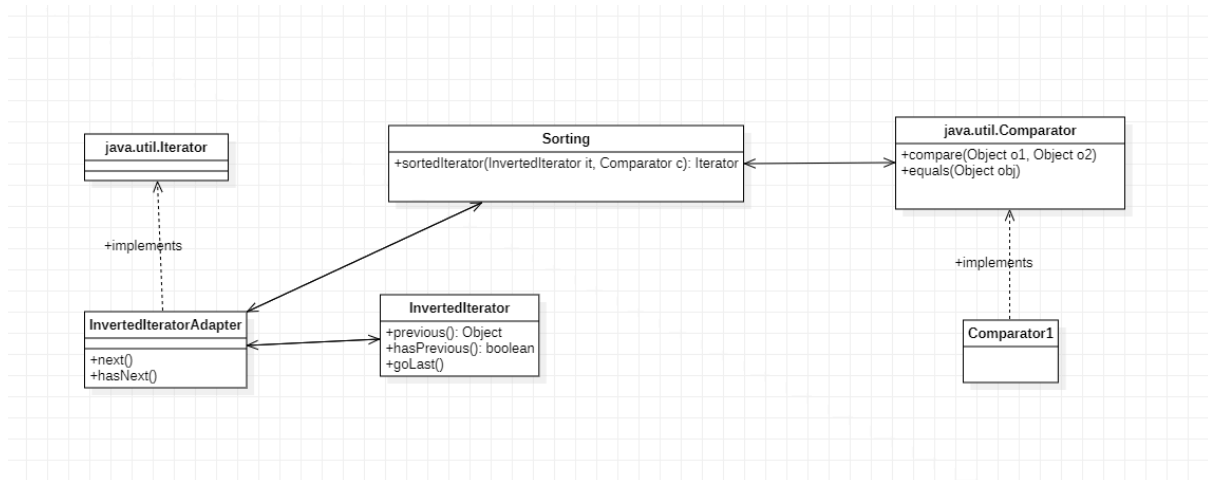
Symptom	Weight
diarrea	3
mialgia	2
fiebre	1

## Adapter Iterator eta Patroiak

### Eskatzen da:

Programa Nagusi batean 5 Symptom-a duen Covid19Pacient bat sortu, eta jarraian Sorting.sortedIterator metodoa erabiliz, bere sintomak inprimatu lehendabizi symptomName ordenatuaz, eta jarraian severityIndex ordenatuaz. Covid19Pacient eta Sorting klasetan EZIN DA EZER ALDATU. Horretarako hurrengo pausoak jarraitu:

1. UML diagrama aplikazioaren diseinuarekin.



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

```

package adapter3;

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());
    }
}
  
```

SOFTWARE INGENIARITZA II

3. Covid19Pacient klasea InvertedIterator interfazera egokitzen duen klase adaptadorea sortu. Gogoratu metodo eraikitzaile egokia sortzea pazientearen informazioa bidaltzeko.

```
package adapter3;
import java.util.Iterator;

public class Covid19PacientIterator implements Iterator<Symptom> {
    List<Symptom> symptoms = new Vector<>();
    int position = 0;

    public Covid19PacientIterator(Set<Symptom> s) {
        Iterator<Symptom> i = s.iterator();
        while (i.hasNext())
            symptoms.add(i.next());
    }

    @Override
    public boolean hasNext() {
        return position < symptoms.size();
    }

    @Override
    public Symptom next() {
        Symptom symptom = symptoms.get(position);
        position++;
        return symptom;
    }
}
```

## SOFTWARE INGENIARITZA II

4. Programa nagusi batean, Covid19Pacient objektu bat sortu 5 sintomekin, eta Sorting.sort metodoari bi aldiz deitu, sortu duzun CovidPacient klase adaptadorea eta Comparator inplementatu dituzun bi konparadorearekin. Bukatzeko emaitza inprimatu pantaiatik.

```
package iterator;

import adapter3.SymptomNameComparator;

public class Main {

    public static void main(String[] args) {
        Covid19Pacient p = new Covid19Pacient("Ane", 29);
        p.addSymptom(new Symptom("s1", 10, 10), 1);
        p.addSymptom(new Symptom("s2", 10, 10), 2);
        p.addSymptom(new Symptom("s3", 10, 10), 3);
        p.addSymptom(new Symptom("s4", 10, 10), 4);
        p.addSymptom(new Symptom("s5", 10, 10), 5);

        List<Symptom> symptomList = new ArrayList<>(p.getSymptoms());

        System.out.println("Izen arabera ordenatu:");
        Collections.sort(symptomList, new SymptomNameComparator());
        for (Symptom symptom : symptomList) {
            System.out.println(symptom);
        }

        System.out.println("Larritasun ideize arabera ordenatu");
        Collections.sort(symptomList, new SeverityIndexComparator());
        for (Symptom symptom : symptomList) {
            System.out.println(symptom);
        }
    }
}
```