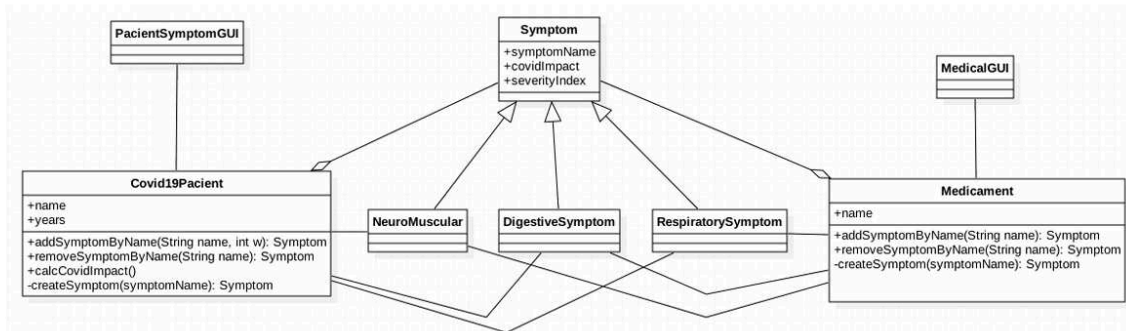


Diseinu Patroiak Laborategia

1. Simple factory.....	2
2. Observer Patroia.....	5
Observer prototipo 1.....	5
Observer prototipo 2.....	9
3. Adapter Patroia.....	11
4. Adapter,Iterator eta Patroiak.....	12
Github repositorioa.....	15

1. Simple factory



Aplikazio honek dauzkan ahultasunak ugari dira:

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

Sintoma berri bat agertuz gero, Covid19Pacient eta Medicament aldatu beharko genuke. Hori dela eta, OCP printzipio bortxatuko luke, non klaseak irekita egon behar diren zabalkuntzarako baina itxita aldaketetarako.

2. Nola sortu daiteke sintoma berri bat orain arte dauden klaseak aldatu gabe (OCP printzipioa)? **Sintoma berri bat gehitzeko simple factory patroia erabili daiteke, Covid19Pacient eta Medicament aldatu gabe.**

```
public class SymptomFactory {
    private Map<String, Symptom> symptomMap;

    public SymptomFactory() {
        symptomMap = new HashMap<>();
    }

    public Symptom createSymptom(String symptomName) {

        if (symptomName.equals("mareos")) {
            return new MovilitySymptom(symptomName, 6, 1);
        }
        //Kode gehiago
    }
}
```

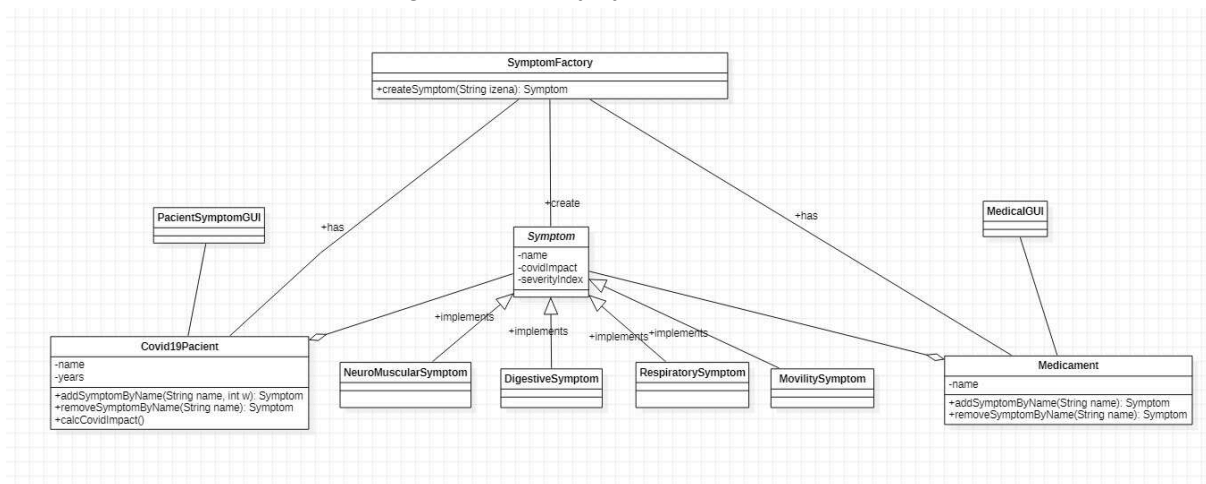
3. Zenbat erresponsabilitate dauzkate Covid19Pacient eta Medicament klaseak (SRP printzipioa)? **Bi erresponsabilitate dituzte batetik sintomen kudeaketa (gehitu,kendu kudeatu) eta bestetik sorrera, createSymtpom metodoaren bidez.Beraz SRP ere hausten dute, soilik ardura bat izan beharko lukete.**

Eskatzen da:

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

Aldaketak:

- SymptomFactory klasea sortu sintomak sortzeko, singleton bezala (sintoma bakoitza bakarra izan behar da).
- Symptom klasea interfaz bezala tratatu, horrela poliformismoa sustatuz (hedapena sustatu eta sintoma mota berria gehitzea errazten du aurrekoa aldatu gabe).
- Sintoma klase berri bat gehitu MovilitySymptom.



2. Aplikazioa implementatu, eta "mareos" sintoma berria gehitu 1 inpaktuarekin.
Aurreko orrian implementatuta.

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) **Singleton patroia erabiliz:**

```
public class SymptomFactory {
    private Map<String, Symptom> symptomMap;
    private static SymptomFactory instance;

    private SymptomFactory() {
        symptomMap = new HashMap<>();
    }

    public static SymptomFactory getInstance() {
        if (instance == null) {
            synchronized (SymptomFactory.class) {
                if (instance == null) {
                    instance = new SymptomFactory();
                }
            }
        }
        return instance;
    }
}
```

```
public class Covid19Pacient {
    private SymptomFactory sf;
    private String name;
    private int age;
    private Map<Symptom, Integer> symptoms=new HashMap<Symptom, Integer>();

    public Covid19Pacient(String name, int years) {
        this.name = name;
        this.age = years;
        this.sf = SymptomFactory.getInstance();
    }

    public String getName() {
        return name;
    }
}
```

```
public class Medicament {
    private String name;
    private List<Symptom> symptoms=new ArrayList<Symptom>();
    private SymptomFactory sf;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Medicament(String name) {
        super();
        this.name = name;
        this.sf = SymptomFactory.getInstance();
    }

    public Symptom addSymptomByName(String symptom) {
        Symptom existingSymptom = getSymptomByName(symptom);
        if (existingSymptom == null) {
            Symptom newSymptom = sf.createSymptom(symptom);
            symptoms.add(newSymptom);
            return newSymptom;
        }
        return existingSymptom;
    }
}
```

2. Observer Patroia

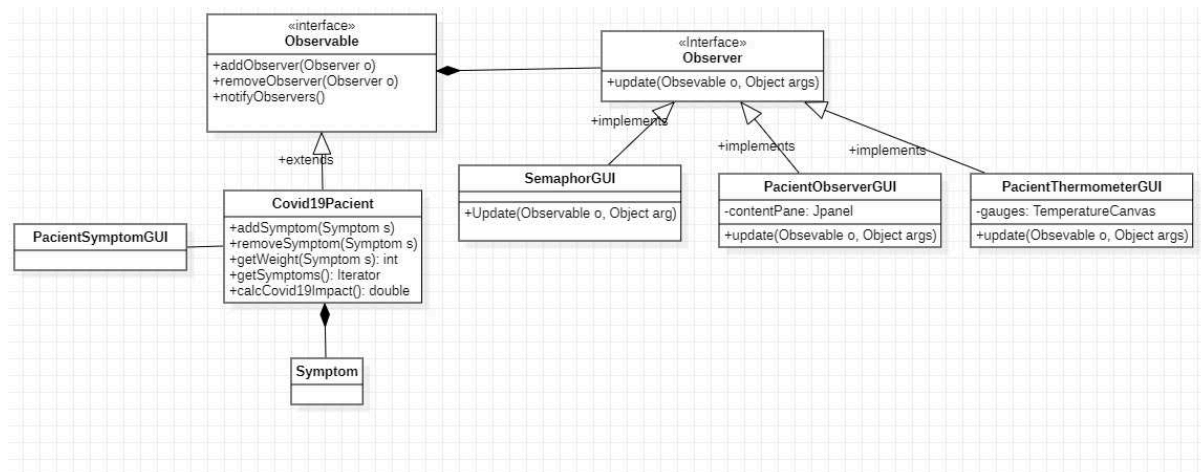
Observer prototipo 1

Eskatzen da:

1. Aplikazioaren UML diagrama hedatuta egin dituzun aldaketan aurkeztuz.

Aldaketak:

- SemaphoreGUI eta PatientThermometerGUI interfazeak sortu dira , hauek observer implementatzen dute eta aldaketak daudenean update() metodoaren bidez interfazeak automatikoki aldatzen dira.



2. Aplikazioaren implementazioa.(Marrak daude deprecated dagoelako, gaur egun ez da gomendatzen erabiltzea java 9 ondoren)

Covid19Pacient klasean:

```
public class Covid19Pacient extends Observable {
}
```

```
public void addSymptom(Symptom c, Integer w){
    symptoms.put(c,w);
    setChanged();
    notifyObservers();
}

public Symptom addSymptomByName(String symptom, Integer w){
    Symptom s=getSymptomByName(symptom);
    if (s==null) {
        s=sf.createSymptom(symptom);
        symptoms.put(s,w);
        setChanged();
        notifyObservers();
    }
    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();
    return s;
}
```

PacientObserverGUI klasean:

```
public class PacientObserverGUI extends JFrame implements Observer{

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

    /**
     * Create the frame.
     */
    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");
        obs.addObserver(this);
        this.setVisible(true);
    }

    public void update(Observable o, Object args) {
        Covid19Patient p = (Covid19Patient) 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);
    }
}
```

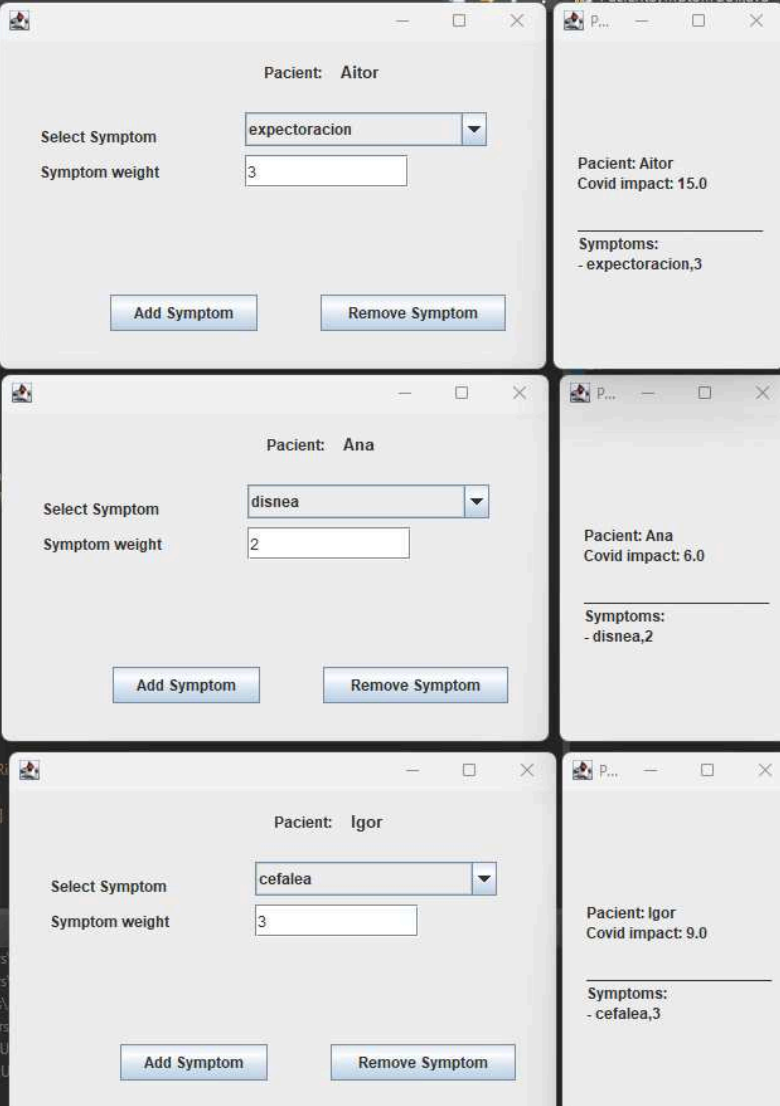

PacientSymptomGUI klasean

```
public PatientSymptomGUI(Covid19Pacient p) {  
    ...  
    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());  
                p.addSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName(),  
Integer.parseInt(weightField.getText()));  
            } 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());  
            p.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).getName());  
        }  
    });  
    ...  
}
```

Main klasean

```
public class Main {  
    public static void main(String args[]) {  
        Observable pacient = new Covid19Pacient("Aitor", 35);  
        new PatientSymptomGUI((Covid19Pacient) pacient);  
        new PatientObserverGUI(pacient);  
  
        Observable pacient2 = new Covid19Pacient("Ana", 55);  
        new PatientSymptomGUI((Covid19Pacient) pacient2);  
        new PatientObserverGUI(pacient2);  
  
        Observable pacient3 = new Covid19Pacient("Igor", 15);  
        new PatientSymptomGUI((Covid19Pacient) pacient3);  
        new PatientObserverGUI(pacient3);  
    }  
}
```

Azken Emaita



The image displays three instances of a web application for managing COVID-19 symptoms, arranged in a 3x2 grid. Each instance represents a different patient and shows the process of adding or removing symptoms.

Patient: Aitor

- Select Symptom: expectoracion
- Symptom weight: 3
- Buttons: Add Symptom, Remove Symptom
- Summary: Patient: Aitor, Covid impact: 15.0, Symptoms: - expectoracion,3

Patient: Ana

- Select Symptom: disnea
- Symptom weight: 2
- Buttons: Add Symptom, Remove Symptom
- Summary: Patient: Ana, Covid impact: 6.0, Symptoms: - disnea,2

Patient: Igor

- Select Symptom: cefalea
- Symptom weight: 3
- Buttons: Add Symptom, Remove Symptom
- Summary: Patient: Igor, Covid impact: 9.0, Symptoms: - cefalea,3

Observer prototipo 2

SemaphorGUI klasean

```
1 public class SemaphorGUI extends JFrame implements Observer {
2     /** stores the associated ConcreteSubject */
3     public SemaphorGUI (Observable obs) {
4         setSize(100, 100);
5         setLocation(350,10);
6         Color c=Color.green;
7         getContentPane().setBackground(c);
8         repaint();
9         obs.addObserver(this);
10        setVisible(true);
11    }
12    public void update(Observable o, Object arg) {
13        Covid19Pacient p = (Covid19Pacient) o;
14        Color c;
15        double current = p.covidImpact();
16        if (current < 5)
17            c = Color.green;
18        else if (current <= 10)
19            c = Color.yellow;
20        else
21            c = Color.red;
22        getContentPane().setBackground(c);
23        repaint();
24    }
25 }
26
```

PacientThermometerGUI klasean

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

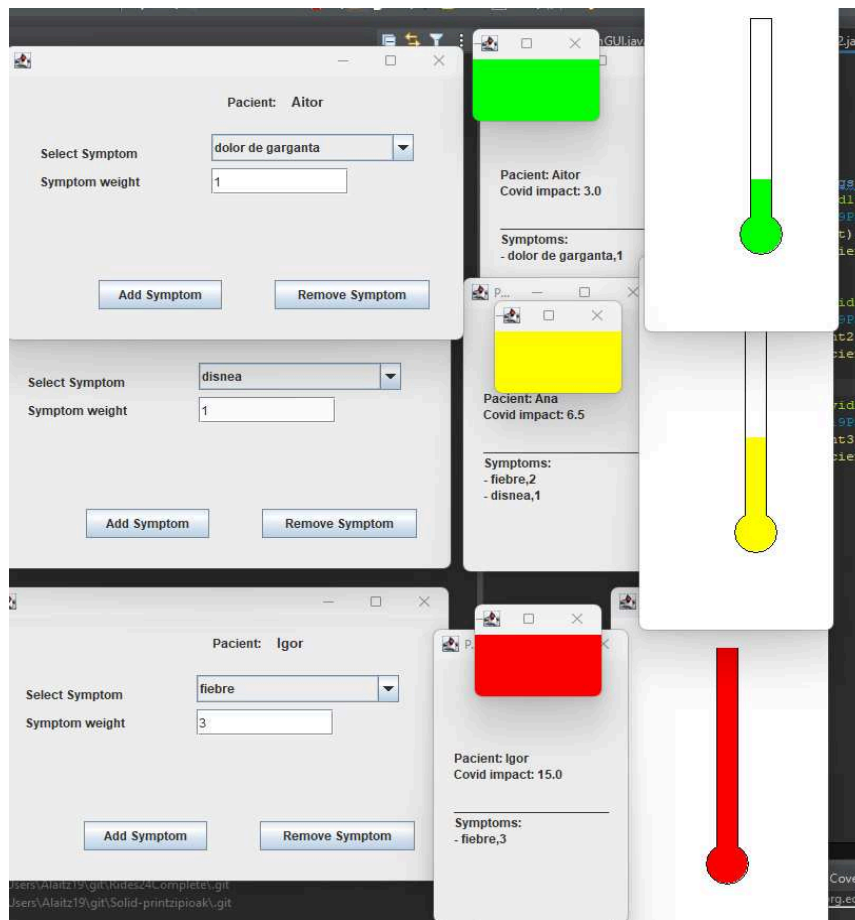
    public PacientThermometerGUI (Observable obs){
        super("Temperature Gauge");
        Panel Top = new Panel();
        add("North", Top);
        gauges = new TemperatureCanvas(0,15);
        gauges.setSize(500,280);
        add("Center", gauges);
        setSize(200, 380);
        setLocation(0,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

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

Main klasean

```
public class Main2 {  
    public static void main(String args[]) {  
        Observable pacient = new Covid19Pacient("Aitor", 35);  
        new PatientSymptomGUI((Covid19Pacient) pacient);  
        new PatientObserverGUI(pacient);  
        new PatientThermometerGUI(pacient);  
        new SemaphorGUI(pacient);  
  
        Observable pacient2 = new Covid19Pacient("Ana", 55);  
        new PatientSymptomGUI((Covid19Pacient) pacient2);  
        new PatientObserverGUI(pacient2);  
        new PatientThermometerGUI(pacient2);  
        new SemaphorGUI(pacient2);  
  
        Observable pacient3 = new Covid19Pacient("Igor", 15);  
        new PatientSymptomGUI((Covid19Pacient) pacient3);  
        new PatientObserverGUI(pacient3);  
        new PatientThermometerGUI(pacient3);  
        new SemaphorGUI(pacient3);  
    }  
}
```

Azken emaitza



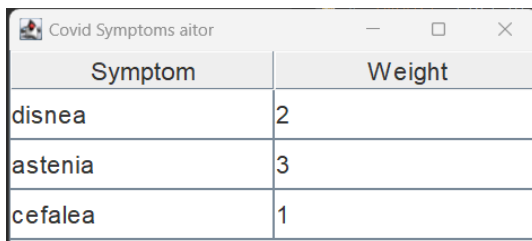
3.Adapter Patroia

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

Covid19PacientTableModelAdapter klasean:

```
public class Covid19PacientTableModelAdapter extends AbstractTableModel implements TableModel {  
    protected Covid19Pacient pacient;  
    protected String[] columnNames = new String[] { "Symptom", "Weight" };  
  
    public Covid19PacientTableModelAdapter(Covid19Pacient p) {  
        this.pacient = p;  
    }  
  
    public int getColumnCount() {  
        return columnNames.length;  
    }  
  
    public String getColumnName(int i) {  
        return columnNames[i];  
    }  
  
    public int getRowCount() {  
        return pacient.getSymptoms().size();  
    }  
  
    public Object getValueAt(int row, int col) {  
        Object Symptom = pacient.getSymptoms().toArray()[row];  
        if (col == 0) {  
            Symptom s = (Symptom) Symptom;  
            return (Object) s.getName();  
        } else {  
            return pacient.getWeight((Symptom) Symptom);  
        }  
    }  
}
```

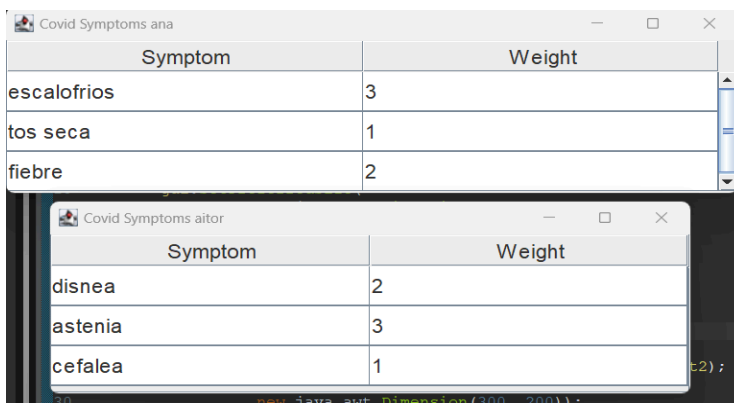
Emaitza:



Symptom	Weight
disnea	2
astenia	3
cefalea	1

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

Emaitza:



Symptom	Weight
escalofrios	3
tos seca	1
fiebre	2

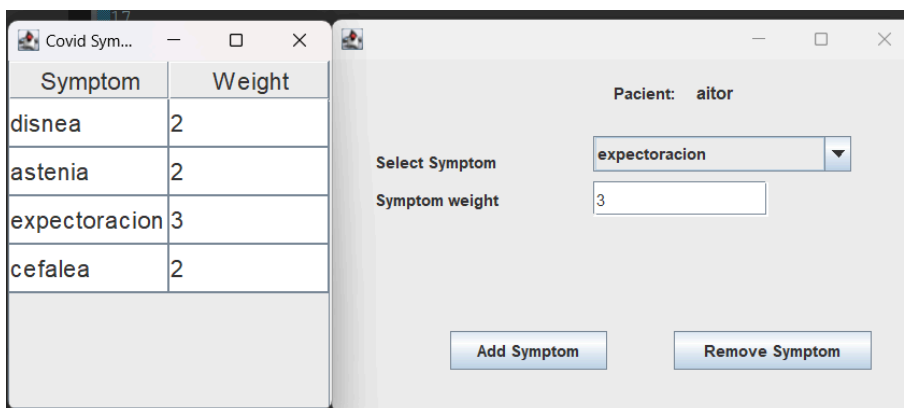
Symptom	Weight
disnea	2
astenia	3
cefalea	1

- (hautazkoa). Nola gehituko zenuke taula lehio bat (ShowPacientTableGUI) aurreko observer ariketan, paziente bateri sintoma berri bat gehitzen zaion bakoitzean bere taula leihoa eguneratzeko?

ShowPacientTableGUI klasean:

```
public class ShowPacientTableGUI extends JFrame implements Observer {  
  
    JTable table;  
    Covid19Pacient pacient;  
  
    public ShowPacientTableGUI(Observable pacient) {  
        this.pacient = (Covid19Pacient) pacient;  
  
        this.setTitle("Covid Symptoms " + this.pacient.getName());  
  
        setFonts();  
  
        TableModel tm = new Covid19PacientTableModelAdapter(this.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);  
        pacient.addObserver(this);  
    }  
  
    public void update(Observable o, Object args) {  
        Covid19Pacient p = (Covid19Pacient) o;  
        table.setModel(new Covid19PacientTableModelAdapter(p));  
    }  
  
    private static void setFonts() {  
        Font font = new Font("Dialog", Font.PLAIN, 18);  
        UIManager.put("Table.font", font);  
        UIManager.put("TableHeader.font", font);  
    }  
}
```

Emailtza:

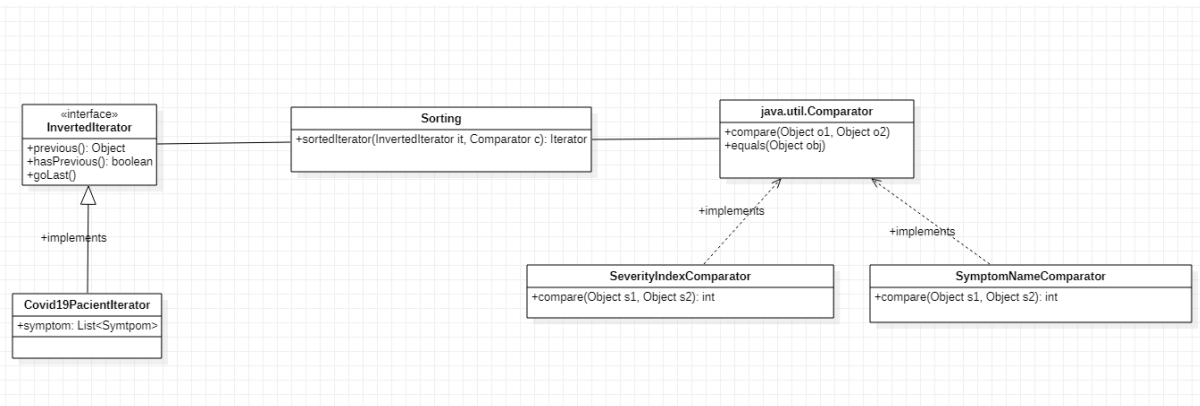


4. Adapter, Iterator eta Patroiak

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.



Aldaketak:

- Bi comparator implementatu bata, sintoma izenak konparatuz ordenatuko ditu sintomak eta bestea severityindex bidez
- Covid19PacientIterator sortu, ahalbidetuz iterazioa atzeruntz(InvertedIterator bidez) edo aurreruntz.

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

```
import java.util.Comparator;

public class SymptomNameComparator implements Comparator<Object> {
    @Override
    public int compare(Object o1, Object o2) {
        if (o1 instanceof Symptom && o2 instanceof Symptom) {
            Symptom s1 = (Symptom) o1;
            Symptom s2 = (Symptom) o2;
            return s1.getName().compareTo(s2.getName());
        }
        throw new IllegalArgumentException("Expected Symptom objects");
    }
}

public class SeverityIndexComparator implements Comparator<Object> {
    @Override
    public int compare(Object o1, Object o2) {
        if (o1 instanceof Symptom && o2 instanceof Symptom) {
            Symptom s1 = (Symptom) o1;
            Symptom s2 = (Symptom) o2;
            return Integer.compare(s1.getSeverityIndex(), s2.getSeverityIndex());
        }
        throw new IllegalArgumentException("Expected Symptom objects");
    }
}
```

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


```
16 public class Covid19PacientIterator implements Iterator, InvertedIterator {
17     List<Symptom> symptoms = new Vector<Symptom>();
18     ListIterator<Symptom> iterator;
19
20     int position = 0;
21     public Covid19PacientIterator(Covid19Pacient pacient) {
22
23         this.symptoms = new ArrayList<>(pacient.getSymptoms());
24         this.goLast();
25     }
26
27     public Covid19PacientIterator(Set<Symptom> s) {
28         Iterator<Symptom> i = s.iterator();
29         while (i.hasNext())
30             symptoms.add(i.next());
31     }
32
33     @Override
34     public boolean hasNext() {
35         return position < symptoms.size();
36     }
37
38     @Override
39     public Object next() {
40         Symptom symptom = symptoms.get(position);
41         position++;
42         return symptom;
43     }
44
45     @Override
46     public Object previous() {
47         return iterator.previous();
48     }
49 }
```

```
    @Override
    public boolean hasPrevious() {
        return iterator.hasPrevious();
    }

    @Override
    public void goLast() {
        this.iterator = symptoms.listIterator(symptoms.size());
    }
}
```


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.

```
public class Main {  
    public static void main(String[] args) {  
  
        Covid19Pacient pacient = new Covid19Pacient("Ane", 29);  
        SymptomFactory factory = SymptomFactory.getInstance();  
  
        pacient.addSymptom(factory.createSymptom("fiebre"), 1);  
        pacient.addSymptom(factory.createSymptom("tos seca"), 2);  
        pacient.addSymptom(factory.createSymptom("astenia"), 3);  
        pacient.addSymptom(factory.createSymptom("cefalea"), 4);  
        pacient.addSymptom(factory.createSymptom("mialgia"), 5);  
  
        Covid19PacientIterator pacientAdapter = new Covid19PacientIterator(pacient);  
  
        System.out.println("Ordenado por symptomName:");  
        Iterator<Object> nameSortedIterator = Sorting.sortedIterator(pacientAdapter, new SymptomNameComparator());  
        while (nameSortedIterator.hasNext()) {  
            System.out.println(nameSortedIterator.next());  
        }  
  
        System.out.println("\nOrdenado por severityIndex:");  
        Iterator<Object> severitySortedIterator = Sorting.sortedIterator(pacientAdapter, new SeverityIndexComparator());  
        while (severitySortedIterator.hasNext()) {  
            System.out.println(severitySortedIterator.next());  
        }  
    }  
}
```

Emaitzan:

```
Ordenado por symptomName:  
astenia  
cefalea  
fiebre  
mialgia  
tos seca  
  
Ordenado por severityIndex:  
cefalea  
mialgia  
tos seca  
astenia  
fiebre
```

Ondorioak

Laborategi honetan, software-sistemen egitura hobetzeko eta eskalagarriago egiteko hainbat diseinu-patroi aztertu dira, horrela, mantentze-lanak eta aldaketak errazagoak izan daitezten:

Simple Factory: Patroi honek objektuak modu zentralizatuan sortzea ahalbidetzen du, sortuko diren objektuen motak zehazki adierazi gabe. Horrela, sistema hedagarriago bihurtzen da, eta sintoma mota berriak gehitzeko aukera dago dagoen sisteman aldaketarik egin gabe, Irekita/Itxita Printzipioa (OCP) betez. Dokumentuan, sintomak kudeatzeko SymptomFactory izeneko klase bat sortu da. Gainera, Singleton patroia erabili da sintoma bakoitzaren instantzia bakarra dagoela bermatzeko.

Observer: Patroi honen bidez, objektu batzuk beste objektu batzuen aldaketen berri automatikoki jasotzeko konfigura daitezke. Dokumentuan, SemaphoreGUI eta PatientThermometerGUI izeneko bi interfaze sortu dira Observer gisa, eta Covid19Patient klaseko objektu baten egoeran aldaketak gertatzen direnean automatikoki eguneratzen dira. Horrez gain, botoiak eta ekintzak gehitu dira sintomak gehitu edo kentzeko.

Adapter: Patroi honek beste interfaze batera egokitzea ahalbidetzen du, sistemak espero duen formatua erabil ahal izateko. Kasu honetan, Covid19PatientTableModelAdapter izeneko klasea sortu da Covid19Patient datuak taula formatuan bistaratzea ahalbidetzeko. Honek pazienteen informazioa taula batean automatikoki erakutsi eta eguneratzeko aukera ematen du.

Iterator eta besteak: Iterator eta Comparator patroiak erabiliz, sintomak hainbat irizpidetan oinarrituta ordenatu eta arakatu daitezke, adibidez, izenaren edo larritasun-indizearen arabera. Dokumentuan, bi konparatzaile eta iteradore bat sortu dira sintomak aurrera edo atzera arakatzeko. Honek sistema malguago bihurtzen du, eta sintomak modu erraz batean ordenatzeko aukera ematen du.

Github repositorioa

<https://github.com/Alaitz19/labpatterns>