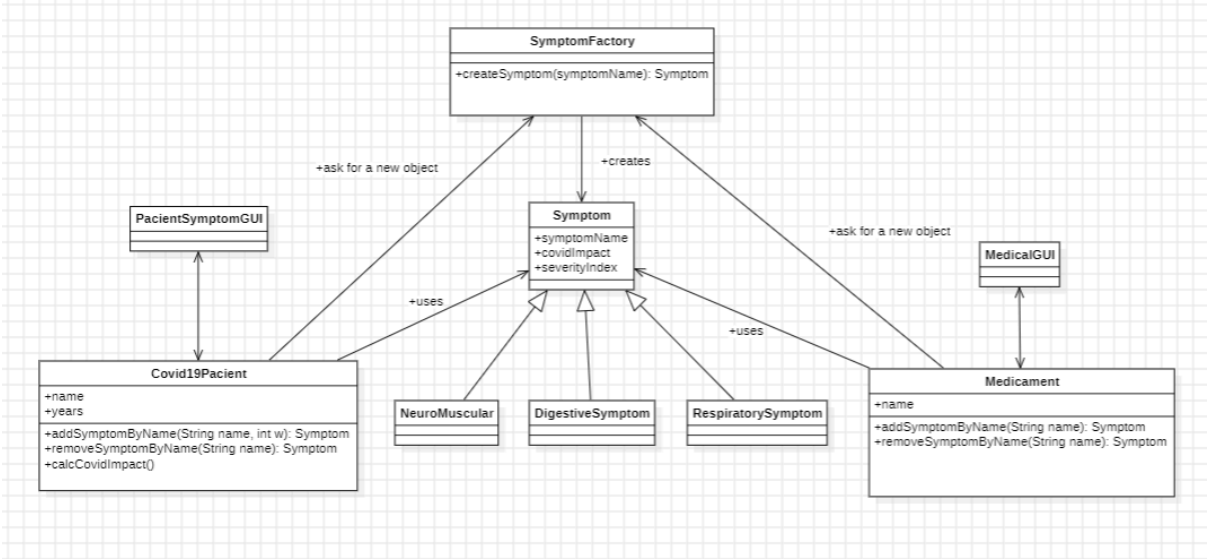
SIMPLE FACTORY

1.



Pauso honetan “Simple Factory” patroia aplikatu diogu gure programaria. Honekin lortu dugu alde batetik Covid19Pacient eta Medicament klaseei Sintomen sorkunten errespontsabilitatea kentzea SRP printzioa jarraituz; eta gainera, sintoma mota berria bat sortzean kodea ez aldatu beharra (OCP) izan ere, klase hauek Factory-ari objetua sortzeko eskatuko dio eta honek egingo du sorkuntza prozesua eta mota berri bat badago honi gehituko diogu.

2. Aplikazioaren implementazioa eta “mareos” sintoma gehitzen

```
public Symptom addSymptomByName(String symptom, Integer w)
    Symptom s = factory.createSymptom(symptom);
    if (s!=null)
        symptoms.put(s,w);
    return s;
}
```

```java
package factory;

import java.util.*;

public class SymptomFactory {

    private static final List<String> impact5 = Arrays.asList("fiebre", "tos seca", "astenia","expectoracion");
    private static final List<Double> index5 = Arrays.asList(87.9, 67.7, 38.1, 33.4);

    private static final List<String> impact3 = Arrays.asList("disnea", "dolor de garganta", "cefalea","mialgia","escalofrios");
    private static final List<Double> index3 = Arrays.asList(18.6, 13.9, 13.6, 14.8, 11.4);

    private static final List<String> impact1 = Arrays.asList("nauseas", "vómitos", "congestión nasal","diarrea","hemoptisis","congestion conjuntival", "mareos");
    private static final List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 1.2, 3.5);

    private static final List<String> digestiveSymptom = Arrays.asList("nauseas", "vómitos","diarrea");
    private static final List<String> neuroMuscularSymptom = Arrays.asList("fiebre", "astenia", "cefalea", "mialgia","escalofrios", "mareos");
    private static final List<String> respiratorySymptom = Arrays.asList("tos seca","expectoracion","disnea","dolor de garganta", "congestión nasal","hemoptisis","congestion conjuntival");

    public Symptom createSymptom(String symptomName) {
        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));
        }

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

        return null;
    }
```

3. Sintoma instantzia bakarra egoteko, Factory klasean “cache” edo mapa moduko bat gordeko dugu sortutako sintomekin; batenbat errepikatzen bada, zuzenean aurretik sortutako sintoma bueltatuko dio beste bat sortu beharrean. Ez bada existitzen, noski, sortu egingo dugu.

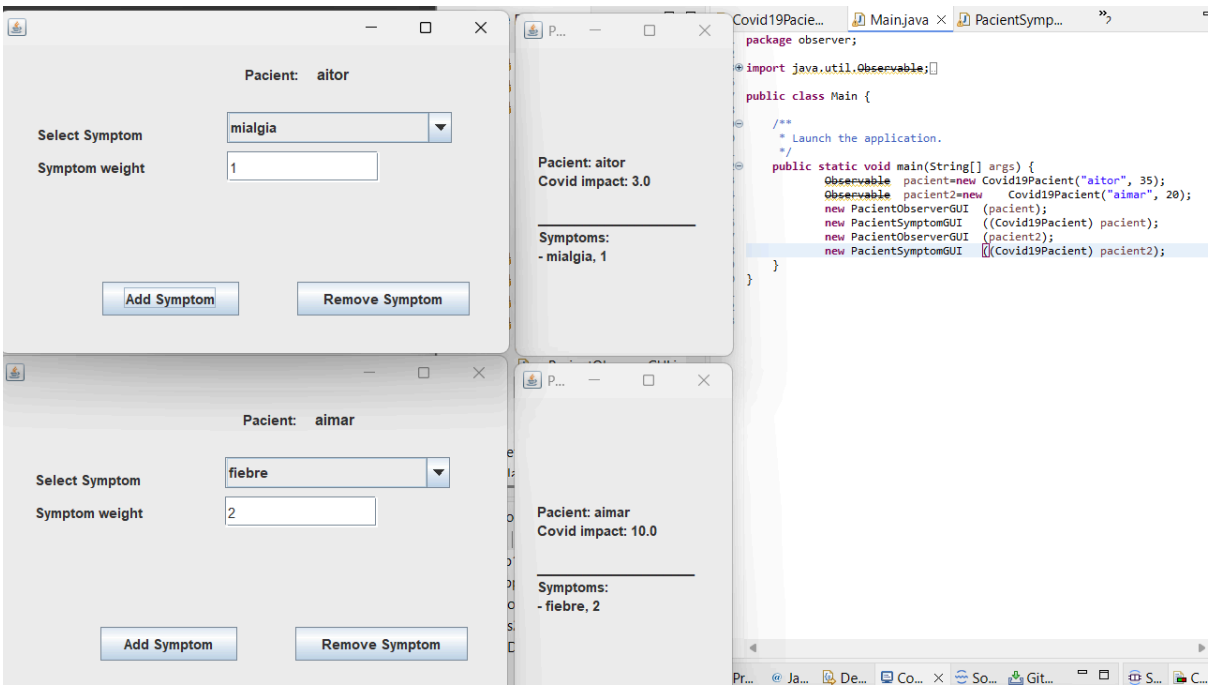
OBSERVER

1. LEHEN PROTOTIPOA:

```
@SuppressWarnings("deprecation")
public class Covid19Pacient extends Observable
```

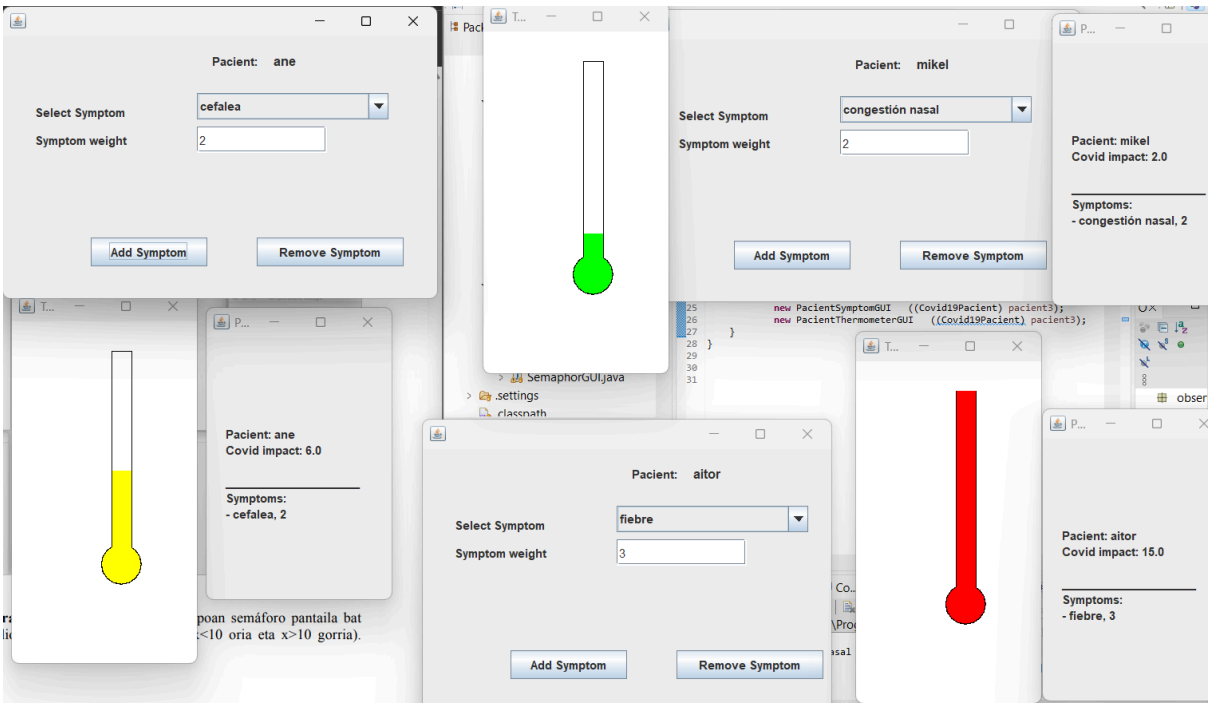
```java
public Symptom addSymptomByName(String symptom, Integer w){
    Symptom s = factory.createSymptom(symptom);
    if (s!=null)
        symptoms.put(s,w);
    setChanged();
    notifyObservers();
    return s;
}

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;
}
```

```java
package observer;

import java.util.Iterator;

public class PacientObserverGUI extends JFrame implements Observer{

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

    /**
     * Create the frame.
     */
    @SuppressWarnings("deprecation")
    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);
        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);
    }
}
    private Covid19Pacient pacient;

    /**
     * Create the frame.
     */
    public PacientSymptomGUI(Covid19Pacient p) {
        this.pacient = p;
```

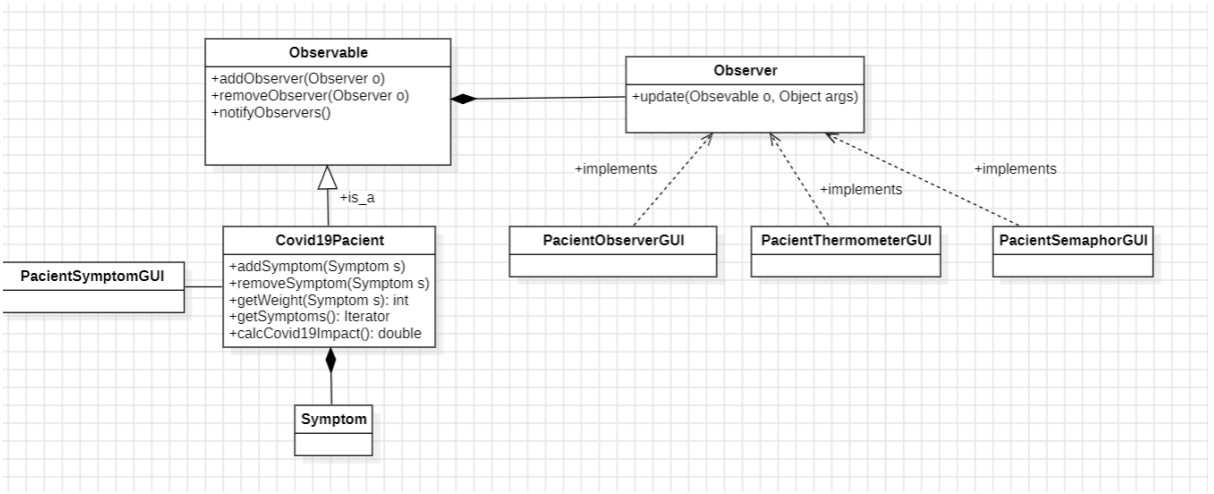


2. BIGARREN PROTOTIPOA

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

    @SuppressWarnings("deprecation")
    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);
    }

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

    }

    public static void main(String[] args) {
            Observable  pacient=new Covid19Pacient("ane",   40);
            Observable  pacient2=new    Covid19Pacient("mikel", 20);
            Observable  pacient3=new    Covid19Pacient("aitor", 35);
            new PacientObserverGUI  (pacient);
            new PacientSymptomGUI   ((Covid19Pacient) pacient);
            new PacientThermometerGUI   ((Covid19Pacient) pacient);

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

            new PacientObserverGUI  (pacient3);
            new PacientSymptomGUI   ((Covid19Pacient) pacient3);
            new PacientThermometerGUI   ((Covid19Pacient) pacient3);
    }
}
```



3. HIRUGARREN PROTOTIPOA



```java
package observer;

import java.awt.Color;


@SuppressWarnings("serial")
public class SemaphorGUI extends JFrame implements Observer {
    private Observable obs;

    /** stores the associated ConcreteSubject */
    public SemaphorGUI (Observable obs) {
        this.obs = obs;
        setSize(100, 100);
        setLocation(350,10);
        Color c=Color.green;
        getContentPane().setBackground(c);
        repaint();
        setVisible(true);
        obs.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();
    }
}
    public static void main(String[] args) {
            Observable  pacient=new Covid19Pacient("ane",   40);
            Observable  pacient2=new    Covid19Pacient("mikel", 20);
            Observable  pacient3=new    Covid19Pacient("aitor", 35);
            new PacientObserverGUI  (pacient);
            new PacientSymptomGUI   ((Covid19Pacient) pacient);
            new PacientThermometerGUI   ((Covid19Pacient) pacient);
            new SemaphorGUI ((Covid19Pacient) pacient);

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

            new PacientObserverGUI  (pacient3);
            new PacientSymptomGUI   ((Covid19Pacient) pacient3);
            new PacientThermometerGUI   ((Covid19Pacient) pacient3);
            new SemaphorGUI ((Covid19Pacient) pacient3);
    }
```

ADAPTER

1.-

Covid19PacientTableModelAdapter klasean hainbat funtzio bete genuen taulara adaptazioa egiteko, hauetatik komplexuena getValueAt da, non zutabe eta errenkada sartuta bere balioa itzultzen duen.

```java
public class Covid19PacientTableModelAdapter extends AbstractTableModel {

    private static final long serialVersionUID = 1L;
    protected Covid19Pacient pacient;
    protected String[] columnNames = {"Symptom", "Weight"};
    protected List<Symptom> symptomList;

    public Covid19PacientTableModelAdapter(Covid19Pacient p) {
        this.pacient = p;
        // Guardamos los síntomas en una lista para acceder por fila
        this.symptomList = new ArrayList<>(p.getSymptoms());
    }

    @Override
    public int getColumnCount() {
        return columnNames.length; // 2 columnas: Symptom y Weight
    }

    @Override
    public String getColumnName(int i) {
        return columnNames[i];
    }

    @Override
    public int getRowCount() {
        return symptomList.size(); // una fila por síntoma
    }

    @Override
    public Object getValueAt(int row, int col) {
        Symptom s = symptomList.get(row);
        switch (col) {
            case 0: // columna 1: nombre del síntoma
                return s.getName();
            case 1: // columna 2: peso del síntoma
                return pacient.getWeight(s);
            default:
                return null;
        }
    }
```

2.-
Main klasean bi paziente sortu ditugu, eta bakoitzarentzat gui bat ireki dugu.

```
public class Main {

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

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

        Covid19Pacient pacient2 = new Covid19Pacient("Jon", 40);
        pacient2.addSymptomByName("tos seca", 3);
        pacient2.addSymptomByName("fiebre", 2);

        ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient);

        ShowPacientTableGUI gui2 = new ShowPacientTableGUI(pacient2);
        gui2.pack();
        gui2.setLocation(400, 100); // posición diferente
        gui2.setVisible(true);

        gui.setPreferredSize(
                  new java.awt.Dimension(400, 200));
        gui.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);
        gui.pack();
        gui.setLocationRelativeTo(null);
        gui.setVisible(true);


    }

}
```

ADAPTER ITERATOR ETA PATROIAK

1.-

2-

Symptom name klasea definitu dugu, non izenak alderatzen dituen, string claseko compareTo erabiliz.

```
public class SymptomName implements Comparator{

        public int compare(Object  s1, Object s2) {
            String sn1 = ((Symptom) s1).getName();
            String sn2 = ((Symptom) s2).getName();
            return sn1.compareTo(sn2);
        }

}
```

Baita ere severityIndex klasea definitu dugu, severity gatik ordenatzeko, Integer klaseko compare funtzioa erabiliz

```
public class SeverityIndex implements Comparator {

    public int compare(Object  s1, Object s2) {
        int sn1 = ((Symptom) s1).getSeverityIndex();
        int sn2 = ((Symptom) s2).getSeverityIndex();
        return Integer.compare(sn1, sn2);
    }

}
```

3.-
Covid19PacientInvertedIteratorAdapter clasea sortu dugu, InvertedIterator interfazea implementatzen duena. Clase honek paziente bat jasotzen du eta honen sintomen lista bat gordetzen du, horrela hiru funtzioak erabili ahal izateko, horrela lista iterator moduan erabili ahal izateko

```
public class Covid19PacientInvertedIteratorAdapter implements InvertedIterator {

    private List<Symptom> symptomsList;
    private int currentIndex;

    public Covid19PacientInvertedIteratorAdapter(Covid19Pacient pacient) {
        Set<Symptom> symptoms = pacient.getSymptoms();
        this.symptomsList = new ArrayList<>(symptoms);
    }

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

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

    @Override
    public Object previous() {
        return symptomsList.get(currentIndex--);
    }
}
```

4.-
programa nagusia bi zatitan banatu dugu, bat izenaren arabera ordenatzeko eta bestea severity arabera, sortedIterator funtzioa erabili.

```
    public static void main(String[] args) {
        Covid19Pacient p=new Covid19Pacient("Ane", 29);
        p.addSymptom(new Symptom("s1", 7, 7), 1);
        p.addSymptom(new Symptom("s2", 100, 100), 2);
        p.addSymptom(new Symptom("s3", 1, 1), 3);
        p.addSymptom(new Symptom("s4", 10, 10), 4);
        p.addSymptom(new Symptom("s5", 9, 9), 5);

        Covid19PacientInvertedIteratorAdapter adapter = new Covid19PacientInvertedIteratorAdapter(p);
        // Comparator 1: name
        Iterator<Symptom> it1 = Sorting.sortedIterator(adapter, new SymptomName());
        System.out.println("Ordenado por nombre:");
        while (it1.hasNext()) {
            System.out.println(it1.next());
        }
        // Comparator 2: severity
        Iterator<Symptom> it2 = Sorting.sortedIterator(adapter, new SeverityIndex());
        System.out.println("\nOrdenado por severidad:");
        while (it2.hasNext()) {
            System.out.println(it2.next());
        }
    }
}
```