DISEINU PATROIEN LABORATEGIA

Egileak: Jon Olea, Iker Galarraga eta Ibai Oñatibia

		Helburua		
		Sorketa	Egitura	Portaera
Context	Klasea	Factory Method	Adapter	Interperter
	Objektua	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Facade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Momento Observer State Strategy Vistor

Simple Factory

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

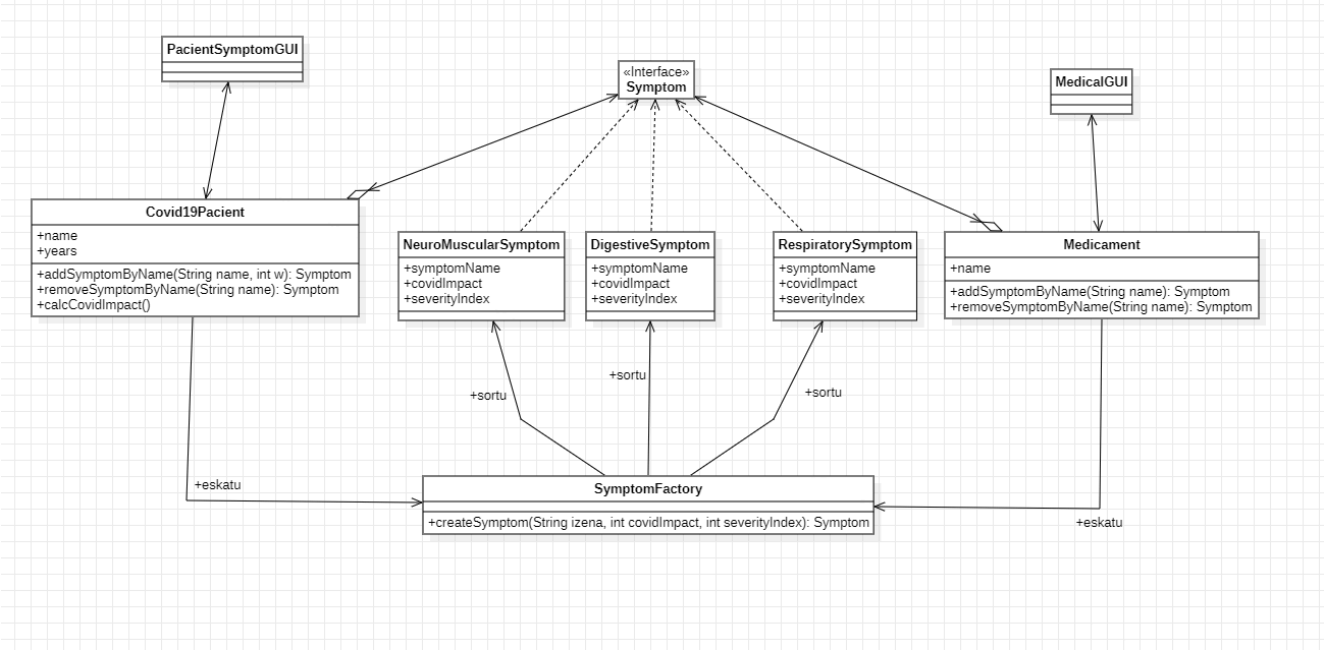
   Sintoma berri bat sortu nahi dugunean Symptom goi klasearen ume bat sortu beharra dago, bainan Covid19Pacient eta Medicament klaseetako createSymptom metodoa eskuz aldatu beharko genuke, sintoma berria txertatuz, ondorioz OCP printzipio hausten da.

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

   Covid19Pacient eta Medicament klaseei sintoma sortzeko erresponsabilitatea kendu behar zaie, eta sintomak sortzeko metodoa klase bakar batean eduki beharra dauka.

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

1. Aplikazioaren diseinu berri bat egin (UML diagrama) Simple Factory patroia aplikatuz, aurreko ahultasunak ezabatzeko



SymtomFactory klase berri bat sortu dugu Sintomak sortzeko, eta klase berri honi lotuta daude Covid19Pacient eta Medicament klaseak horrela errespontsabilitateak kentzen dizkiegu eta ez ditugu OCP eta SRP printzipioak hausten.

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

```
public class SymptomFactory {

    public Symptom createSymptom(String 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","hemoptisis","congestion conjuntival", "mareos");
        List<Double> index1 = Arrays.asList(5.0, 4.8, 3.7, 0.9, 0.8, 1.0, 1.1);

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


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

```java
    private Map<String, Symptom> symptomMap;

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

    public Symptom createSymptom(String symptomName) {

        if (symptomMap.containsKey(symptomName)) {
            return symptomMap.get(symptomName);
        }

        List<String> impact5 = Arrays.asList("fiebre",
```

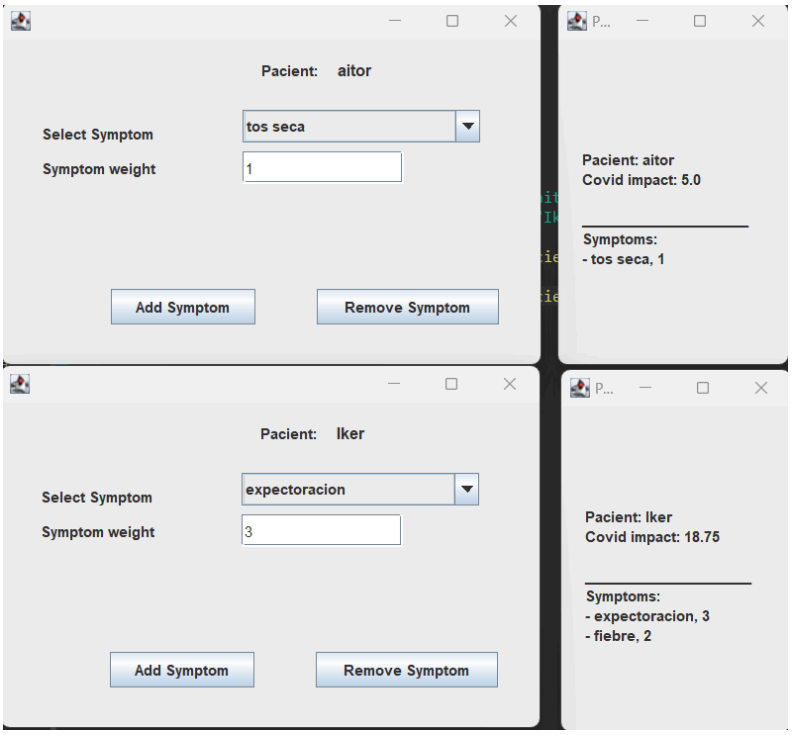
```java
Symptom symptom = null;
if (impact!=0)  {
    if (digestiveSymptom.contains(symptomName)) symptom = new DigestiveSymptom(symptomName,(int)index, impact);
    if (neuroMuscularSymptom.contains(symptomName)) symptom = new NeuroMuscularSymptom(symptomName,(int)index, impact);
    if (respiratorySymptom.contains(symptomName)) symptom = new RespiratorySymptom(symptomName,(int)index, impact);
}

if (symptom != null) {
    symptomMap.put(symptomName, symptom);
}

return symptom;
```

Symptom objetuak bakarrak izateko factory klasean hashMap bat sortu dugu, String-Symptom erlazioarekin. Symptom bat sortu aurretik, ea Map-ean dagoen begiratzen da, hori bueltatzeko, ez badago, sortu egiten da eta Map barruan sartzen da.

Observer Patroia

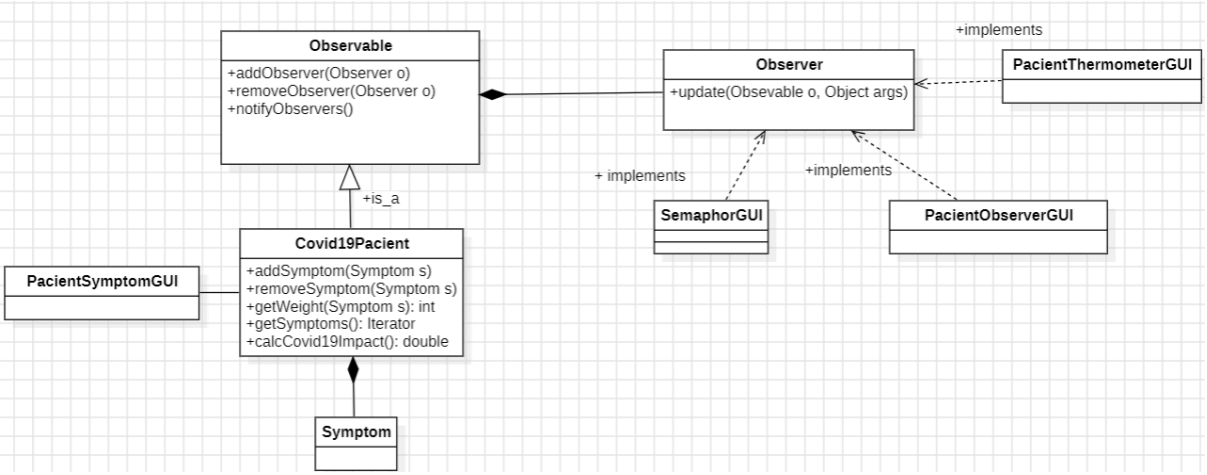
1. Programa nagusia aldatu 2 Covid19Pacient sortzeko bere ondoko PacientSymptomGUI interfazearekin.

   Covid19Pacient bakoitzarentzat, Observable objetu bat sortu dugu, ondoren, bakoitzarentzat bi GUI sortuz, bat sintomak sortzeko (parametro bezala Covid19Pacient pasaz), eta beste bat sintomak ikusteko (parametro bezala Covid19Pacient-aren observer-a)




```java
public class Main {

    /**
     * Launch the application.
     */
    public static void main(String args[]) {
        Observable pacient = new Covid19Pacient("aitor", 35);
        Observable pacient2 = new Covid19Pacient("Iker", 70);
        new PacientObserverGUI(pacient);
        new PacientSymptomGUI((Covid19Pacient) pacient);
        new PacientObserverGUI(pacient2);
        new PacientSymptomGUI((Covid19Pacient) pacient2);
        //new PacientThermometerGUI(pacient);
        //new SemaphorGUI(pacient);
    }

}
```

2. Aplikazioaren UML diagrama hedatuta egin dituzun aldaketan aurkeztuz.



Egin dugun aldaketa bakarra SemaphorGUI klasea gehitzea izan da, ere Observer interfazea inplementatzen du eta, kolorea aldatzeko Covid19Pacient-aren impact-aren arabera.

3. Aplikazioaren inplementazioa

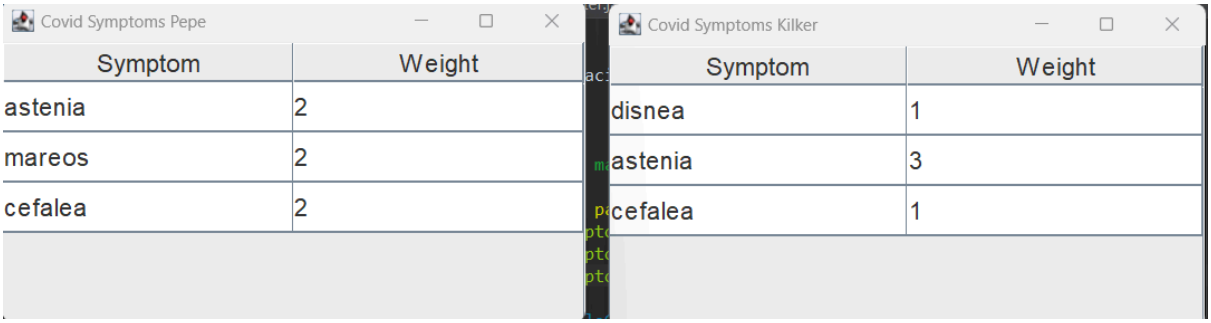
```
public class Main {

    /**
     * Launch the application.
     */
    public static void main(String args[]) {
        Observable pacient = new Covid19Pacient("aitor", 35);
        Observable pacient2 = new Covid19Pacient("Iker", 70);
        Observable pacient3 = new Covid19Pacient("Pepe", 33);

        new PacientSymptomGUI((Covid19Pacient) pacient);
        new PacientThermometerGUI(pacient);
        new SemaphorGUI(pacient);

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

        new PacientSymptomGUI((Covid19Pacient) pacient3);
        new PacientThermometerGUI(pacient3);
        new SemaphorGUI(pacient3);

    }

}
```

Adapter Patroia

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

```
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() {
        // 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!
          Object Symptom = pacient.getSymptoms().toArray()[row];
          if(col == 0) {
              Symptom s = (Symptom) Symptom;
              return (Object) s.getName();
          }
          else {
              return pacient.getWeight((Symptom) Symptom);
          }
      }
    }
```

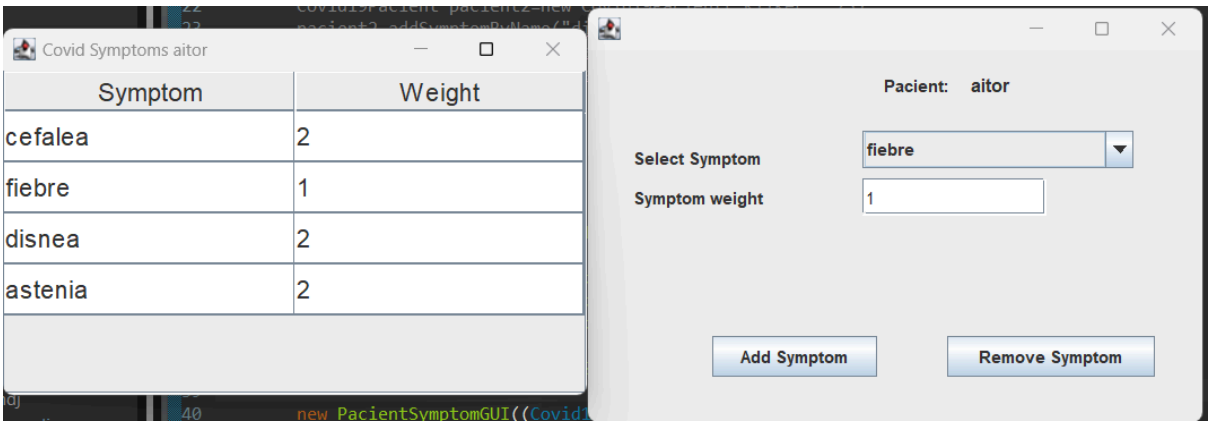
2. Beste paziente bat gehitu sintoma batzuekin, eta aplikazioa exekutatu bi taulak agertzen direla konprobatuz.



```java
public class Main {

    public static void main(String[] args) {

        Covid19Pacient pacient=new Covid19Pacient("Pepe", 33);
        pacient.addSymptomByName("mareos", 2);
        pacient.addSymptomByName("cefalea", 2);
        pacient.addSymptomByName("astenia", 2);

        ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient);
        gui.setVisible(true);

        Covid19Pacient pacient2=new Covid19Pacient("Kilker", 73);
        pacient2.addSymptomByName("disnea", 1);
        pacient2.addSymptomByName("cefalea", 1);
        pacient2.addSymptomByName("astenia", 3);

        ShowPacientTableGUI gui2=new ShowPacientTableGUI(pacient2);
        gui2.setVisible(true);
```

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

```java
Observable pacient=new Covid19Pacient("aitor", 35);

((Covid19Pacient)pacient).addSymptomByName("disnea", 2);
((Covid19Pacient)pacient).addSymptomByName("cefalea", 2);
((Covid19Pacient)pacient).addSymptomByName("astenia", 2);

ShowPacientTableGUI gui=new ShowPacientTableGUI(pacient);

new PacientSymptomGUI((Covid19Pacient) pacient);

gui.setVisible(true);
```



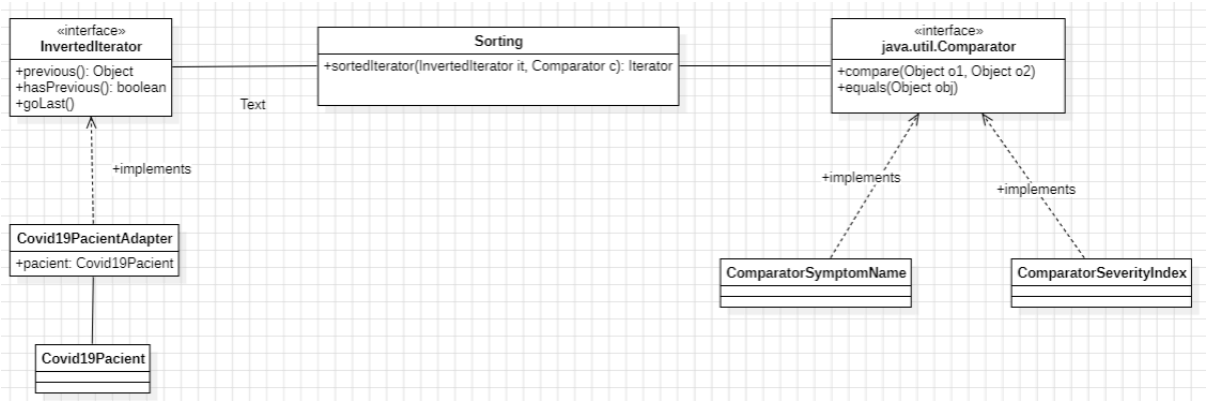

```java
public class ShowPacientTableGUI extends JFrame implements Observer{

    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));
    }
```

ShowPacienTableGUI klasearen eraikitzailean parametroz Observable bat pasa diogu, eta Observer interfazea inplementatzen jarri dugu.

Update metodoan, Covid19Pacient aldatu ondoren, taula berriz sortzen da, horrela eguneratu egiten da gaixotasun berriekin edo kentzen zaienekin.

Adapter Iterator eta Patroiak

1. UML diagrama aplikazioaren diseinuarekin.



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

```
public class ComparatorSymptomName implements Comparator<Object>{
    Object o;

    @Override
    public int compare(Object o1, Object o2){
        Symptom s1 = (Symptom) o1;
        Symptom s2 = (Symptom) o2;

        return s1.getName().compareTo(s2.getName());
    }
```

```
public class ComparatorSeverityIndex implements Comparator<Object> {
    Object o;

    @Override
    public int compare(Object o1, Object o2){
        Symptom s1 = (Symptom) o1;
        Symptom s2 = (Symptom) o2;

        return s1.getSeverityIndex() - s2.getSeverityIndex();
    }
```

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

```java
public class Covid19PacientAdapter implements InvertedIterator{
    Covid19Pacient pacient;
    List<Symptom> symptoms;
    int posizioa;

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

    public Object previous() {
        Symptom emaitza = symptoms.get(posizioa);
        posizioa--;
        return emaitza;
    }

    public boolean hasPrevious() {
        return posizioa >= 0;
    }

    public void goLast() {
        posizioa = symptoms.size() - 1;
    }
}
```

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.

```java
public class Main {

    public static void main(String[] args) {
        Covid19Pacient p=new Covid19Pacient("Ane", 29);
        p.addSymptom(new RespiratorySymptom("A", 10, 3), 1);
        p.addSymptom(new RespiratorySymptom("E", 10, 2), 2);
        p.addSymptom(new DigestiveSymptom("C", 10, 1), 3);
        p.addSymptom(new NeuroMuscularSymptom("B", 10, 4), 4);
        p.addSymptom(new NeuroMuscularSymptom("D", 10, 5), 5);

        Covid19PacientAdapter a=new Covid19PacientAdapter(p);
        Comparator<Object> c1=new ComparatorSymptomName();
        Comparator<Object> c2=new ComparatorSeverityIndex();

        Iterator<Object> i=p.iterator();
        System.out.println("Ordenatu gabe: ");
        while (i.hasNext()) {
            System.out.println(i.next());
        }
        System.out.println("\n");

        System.out.println("Izenarekin ordenatuta: ");
        Iterator<Object> i1 = Sorting.sortedIterator(a, c1);
        while (i1.hasNext()) {
            System.out.println(i1.next());
        }
        System.out.println("\n");

        System.out.println("Severity index-arekin ordenatuta: ");
        Iterator<Object> i2 = Sorting.sortedIterator(a, c2);
        while (i2.hasNext()) {
            System.out.println(i2.next());
        }

    }

}
```

```
Ordenatu gabe:
C
D
B
A
E


Izenarekin ordenatuta:
A
B
C
D
E


Severity index-arekin ordenatuta:
C
E
A
B
D
```