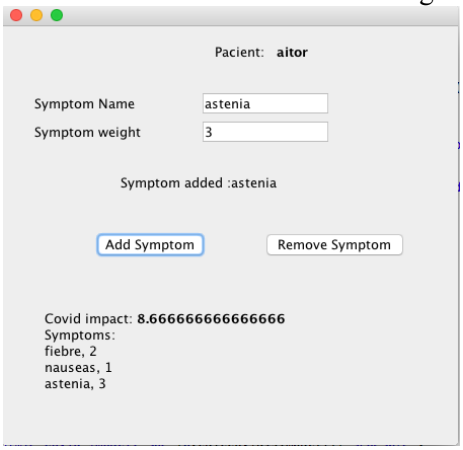
Sarrera

Laborategi honetan hurrengo diseinu patroien ariketak planteatuko dira: Strategy, Simple Factory, Observer eta Adapter.
Laborategi honen kodea (eta UML fitxategiak) hurrengo github estekan aurkitu daiteke: https://github.com/jononekin/labpatterns.

Proiektu honen kopia bat egin (fork) eta bukatzen duzunean aldaketa github-ean uzi.

Simple Factory

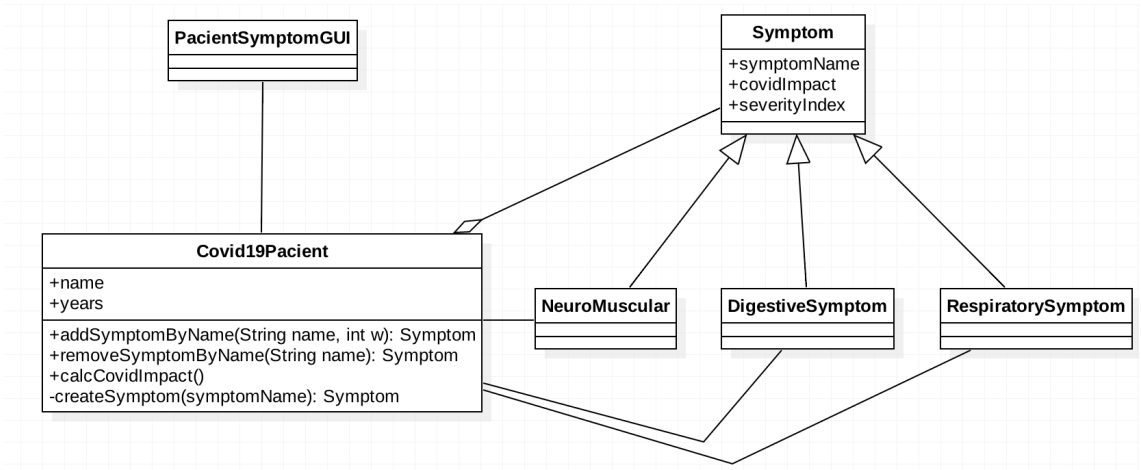
Hurrengo aplikazioak Paziente batek daukan Covid19 sintomak gehitzeko erabiltzen da.



Aplikazioa exekutatzeko Main klasean dauden hurrengo instrukzioak ipini:

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

Aplikazioaren klase diagrama hurrengo irudian agertzen da:



PacientSymptomGUI klaseak erabiltzaile baten datuak jasotzen ditu, eta **Add Symptom** eta **Remove Symptom** botoiak sakatzen direnean addSymptom eta removeSymptom metodoak deitzen dira hurrenez hurren. addSymptom kodea hurrengo da:

```
(1) public void addSymptom(Covid19Pacient p, String symptomName) {
(2)       Symptom s;
(3)       if (isNumeric(weightField.getText())) {
(4)          if (p.getSymptomByName(symptomName)==null) {
(5)             if (Integer.parseInt(weightField.getText())<=3) {
(6)                s=p.addSymptomByName(symptomName, Integer.parseInt(weightField.getText()));
(7)                if (s!=null) {
(8)                  errorLabel.setText("Symptom added :"+symptomName);
(9)                  reportLabel.setText(createReport());
(10)               } else errorLabel.setText("ERROR, Symptom "+symptomName+ " does not exist ");
(11)            } else errorLabel.setText("ERROR, Weight between [1..3]");
(12)         } else errorLabel.setText("ERROR, Symptom "+symptomName+" already assigned ");
(13)       } else errorLabel.setText("ERROR, weight must be an integer");
(14)    }
```

Metodoak lehendabizi pazienteak sintoma hori duen konprobatzen du (*getSymptomByName* metodoari deituz, (4) lerroan), eta oraindik ez badauka, sintoma berri bat sortzen du *Covid19Pacient* objektuaren addSymptomByName metodoriari deituz (6.lerroa). Metodo honen deskribapena hurrengoa da:

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

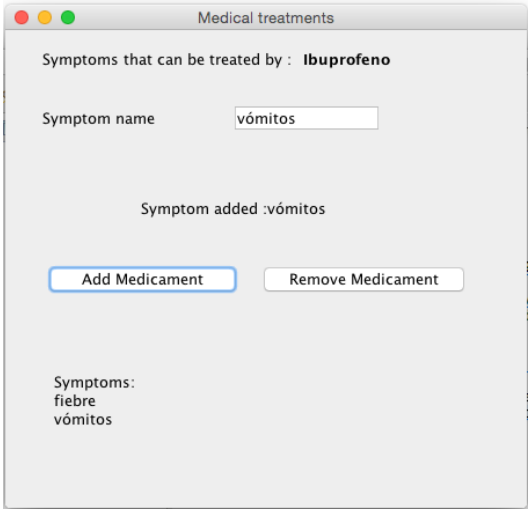
Metodoak lehendabizi pazienteak sintoma hori duen konprobatzen du (*getSymptomByName* metodoari deituz), eta oraindik ez badauka, sintoma berri bat sortzen du *createSymptom* metodo pribatuari deituz. Metodo honen deskribapena hurrengoa da:

```
private 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");
        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",
                                                "congestión nasal","hemoptisis","congestion conjuntival");
        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;
}
```

Aplikazio ondo diseinatuta egongo legoke, *Symptom* klaseko objektuak *Covid19Pacient* objektuetan soilik erabiliko balira, baina zer gertatzen da beste klase batzuk *Symptom* klaseko objektuak sortu nahi badituzte?

Suposatu aplikazioa hedatu nahi dugula beste interfaze batekin. Interfaze honetan, Medikamendu bat zein sintoma tratatzeko erabiltzen den gorde nahi dugu. Aplikazio honen interfazea hurrengo da:

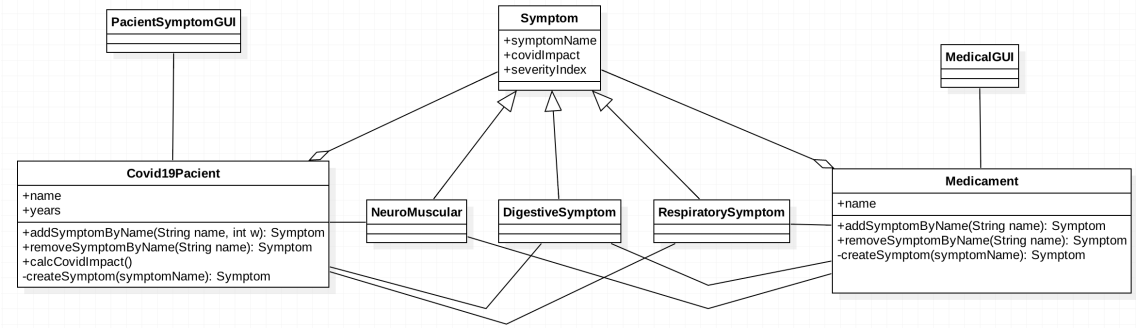


Interfazea ikusteko, main programa aldatu hurrengo instrukzioengatik:

```
public static void main(String[] args) {
                    Medicament m=new Medicament("Ibuprofeno");
                    MedicalGUI mgui=new MedicalGUI(m);

}
```

"Add Medicament" botoia sakatzen denean, *Medicament* klaseko *addSymptomByName* klaseko metodoari deitzen zaio. Metodo honek, sintoma bat sortzen du, bere *createSymptom* metodoari deituz, baina..... konturatu zarenez *Covid19Pacient* zegoen metodo berdina da, eta honek usain txar bat botatzen du. Hau da orain arteko aplikazio guztiaren diseinua:

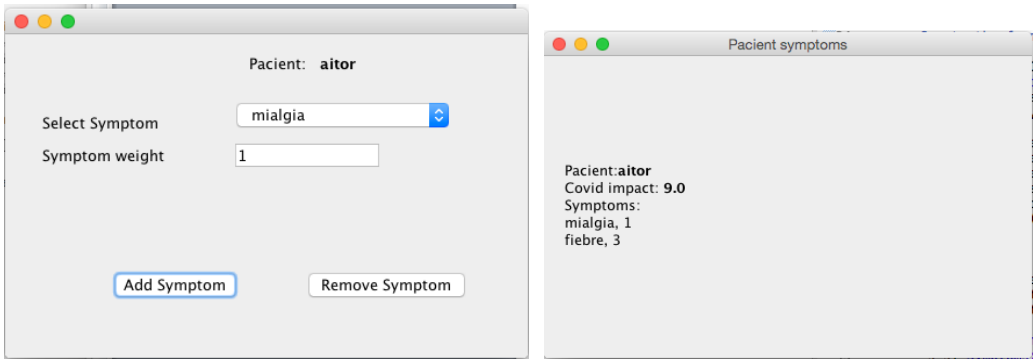


Aplikazio honek dauzkan ahultasunak ugari dira:
1. Zer gertatzen da, sintoma berri bat agertzen bada (adb: mareos)?
2. Nola sortu daiteke sintoma berri bat orain arte dauden klaseak aldatu gabe (OCP printzipioa)?
3. Zenbat erresponsabilitate dauzkate *Covid19Pacient* eta *Medicament* klaseak (SRP printzipioa)?

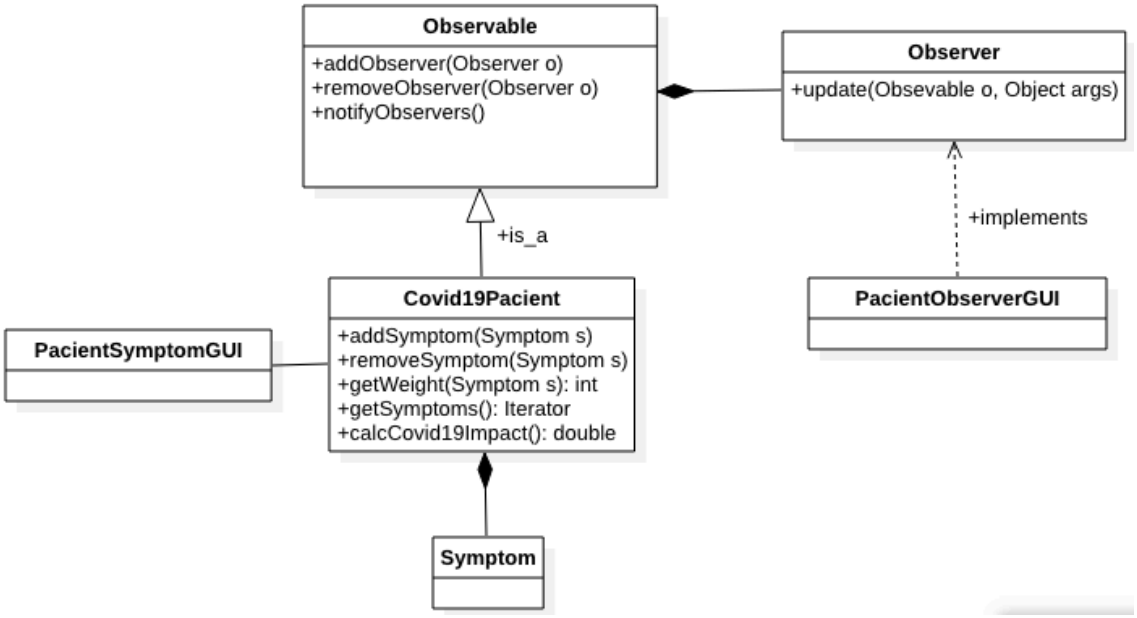
Eskatzen da:
1. Aplikazioaren diseinu berri bat egin (UML diagrama) Simple Factory patroia aplikatuz, aurreko ahultasunak ezabatzeko. Jarraian deskribatu testu batean egin dituzun aldaketak.
2. Aplikazioa inplementatu, eta "mareos" sintoma berria gehitu 1 inpaktuarekin.
3. Nola egokitu daiteke Factory klasea, *Covid19Pacient* eta *Medicament* erabiltzen dituzten Symptom objektuak berdinak izateko? Hau da, sintoma bakoitzeko objektu bakarra egon dadin.

Observer Patroia

Paziente baten sintomak gehitzeko aplikazio bat garatu nahi da eta aldi berean, sintoma bat gehitzen den bakoitzean, pazientearen sintoma pantaila ere eguneratu. Hurrengo irudian bi sintoma gehitzen direnean zer gertatzen den aurkezten da:



Horretarako *Observer* patroia jarraitzen duen aplikazio bat garatuko dugu, hurrengo diseinuak aurkezten duen bezala:



Observable klasea eta *Observer* interfazea *java.util* liburutegiak eskaintzen dizkigu Observer patroia garatzeko. *Covid19Pacient* klaseak, paziente batek daukan sintoma

guztiekin bere pisuekin gordetzen du. Klase hauen eguneraketaz beste klase batzuk interesatuta daude, beraz *Observable* klase bat izango da. *PacientObserverGUI* klaseak Paziente batek daukan sintomak aukezten ditu, eta bere eguneraketaz ohartuta egon behar da, beraz, *Observer* interfazea inplementatuko du, eguneraketaz ohartarazteko. PacientSymptomGUI klaseak paziente baten sintomak eguneratzen ditu.

Laborategi honekin eman den kodearekin, **observer** paketean, aplikazio honen lehendabiziko prototipo bat aurkitu dezakezu, baina *Observer* patroian aplikatu behar diren instrukziorik gabe.

Hauek dira aldatu behar diren 4 klaseak aplikazioa garatzeko.

1. *CovidPacient Observable* klasea.

Lehendabizi klasea Observable klasetik hedatu behar du (hau da, klase definizioan **extends** Observable gehitu). Sintoma bat gehitu edo ezabatzen den bakoitzean, bere subskribatzailei notifikatu behar zaie, beraz, *addSymptomByName* eta *removeSymptomByName* metodoetan hurrengo instrukzioak gehitu beharko dira: *setChanged(); notifyObservers();*

2. PacientObserverGUI Observer klasea.

Lau aldaketa egin behar dira:

a) *Observer* klasea inplementatu hau da, **implements** Observer klasearen definiziora gehitu.
b) Metodo eraikitzaileari "obs" izeneko *Observable* parametro bat gehitu. Parametro hau subskribatu nahi dugun objektua izango da.
c) Metodoaren eraikitzailearen bukaeran, pasatutako parametroari subskribatu nahi dela adieraziko dugu, hau da, obs.addObserver(**this**);
d) *Observer* interfazea inplementatzen duen *update* metodoa eguneratuko dugu. Metodo honetan, notifikazioak jasoko dira subskribatuta gauden objektua aldatzen denean (*Observable* o objektua). Lehendabiziko prototipoan args null izango da.

```
public void update(Observable o, Object arg) {
    Covid19Pacient p=(Covid19Pacient)o;
    Iterator<Symptom> i=p.getSymptoms();
    Symptom p2;
    String s="<html> Pacient:<b>"+p.getName()+"</b> <br>";
    s=s+"Covid impact: <b>"+p.calcCovid19Impact()+"</b><br>";
    s=s+"Symptoms: <br>";
    while (i.hasNext()) {
            p2=i.next();
            s=s+p2.toString()+", "+p.getWeight(p2)+"<br>";
      }
      s=s+"</html>";
      symptomLabel.setText(s);
}
```

Metodoak, pazientearen sintomak jasotzen ditu, eta "s" String batean HTML testu bat sortzen du, pazientearen izenarekin, bere Covid inpaktua eta bere sintomak, Bukatzeko pantailaren symptomLabel etiketan aurkezten dira.

3. PacientSymptomGUI klasea.
Klase honek *Covid19Pacient* paziente baten sintomak pantailatik jasotzen ditu, eta objektua eguneratzen du. Hauek dira egin behar diren aldaketak:

1. Metodo eraikitzaileari *Covid19Pacient* motako parámetro bat gehitu, aurreko "pacient" aldagaia eguneratzeko.
2. Bi botoien listener-etan akzioak ipini pacient objektuari síntoma bat gehitu edo ezabatzeko, hau da,

```
p.addSymptomByName(((Symptom)symptomComboBox.getSelectedItem()).get
Name(), new Integer(weightField.getText()));
```

eta

```
p.removeSymptomByName(((Symptom)symptomComboBox.getSelectedItem())
.getName();
```

4: Programa nagusia sortu.

Bukatzeko, programa nagusi bat sortuko dugu objektu guztiak konektatzeko. *Covid19Pacient* motako pacient objektu bat sortzen da, jarraian *PacientObserverGUI* klaseko objektu bat sortzen da eta pacient gertatzen diren eguneraketatan insteratuta gaudela ipintzen dugu. Bukatzeko *PacientSymptomGUI* klaseko objektu bat sortzen dugu pacient objektu baten síntomak eguneratzeko.

Kodea jarraian daukazue:

```
public class Nagusia {
        public static void main(String args[]){
                Observable pacient=new Covid19Pacient("aitor", 35);
                PacientObserverGUI pacientGUI= new PacientObserverGUI (pacient);
                PacientSymptomGUI frame = new PacientSymptomGUI (pacient);

        }
}
```

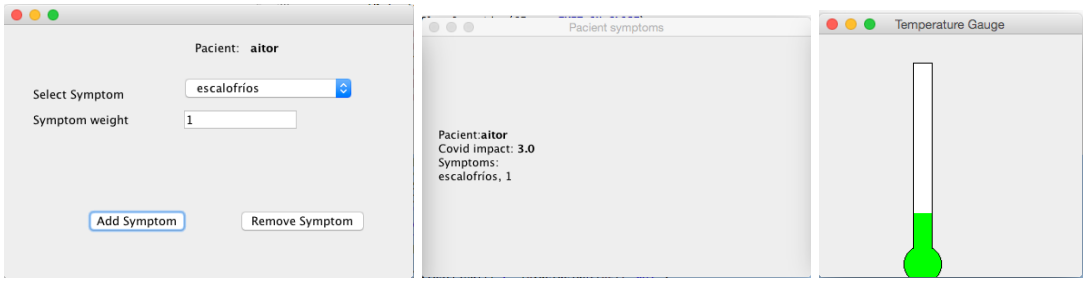
Programa nagusia exekutatu, eta frogatu aplikazioa ondo funtzionatzen duen.

Eskatzen da: Programa nagusia aldatu 2 Covid19Pacient sortzeko bere ondoko PacientSymptomGUI interfazearekin.

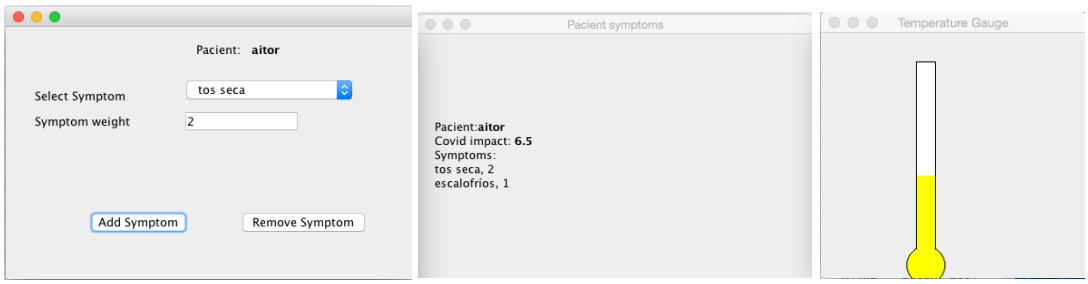
Lehendabiziko prototipoa bukatu dugu, orain beste *Observer* bat gehituko dugu

Bigarren prototipoa:
Prototipo honetan hurrengoa lortu nahi dugu: Sintoma bat gehitzen denean, aurreko interfazea eta *covid19Impact* termometroaren grafika eta kolorea eguneratzea (berdea covidImpact<5 bada, horia 5<=covidImapct<10 edo gorria 10<=covidImpact<=15) pazientearen *covid19Impact()* balioaren arabera. Hemen hiru adibide daude:

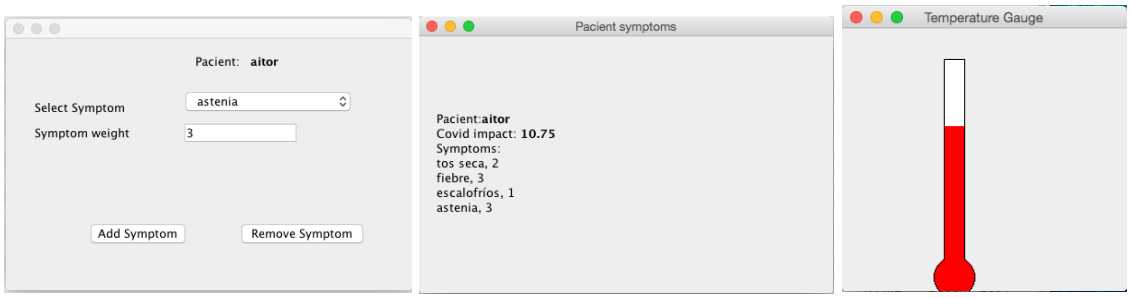
Lehendabizi sintoma bat gehitzen diogu (escalofrios, 1 pisuarekin), eta bere covidImpact 3 denez, termometroa berdez agertzen da.



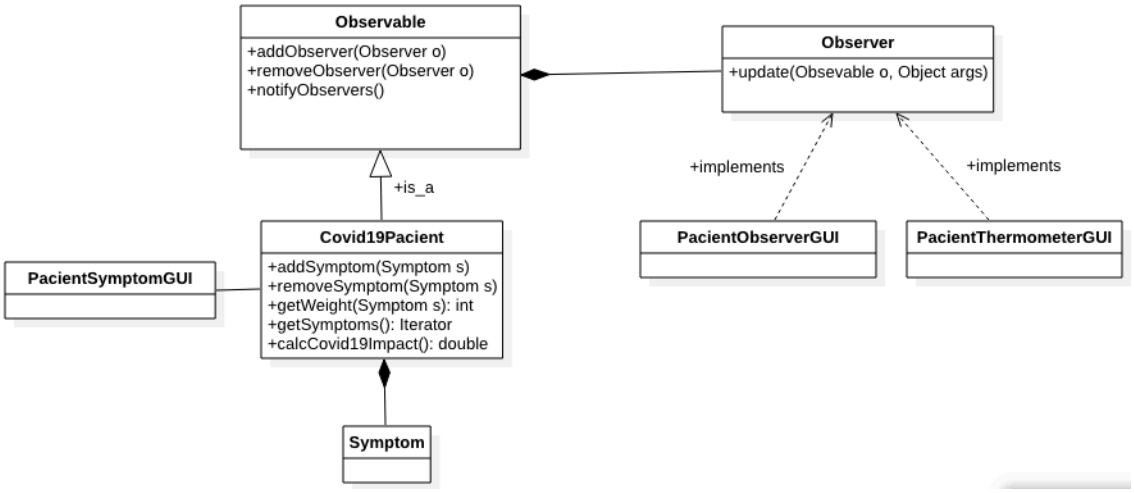
Jarraian beste sintoma bat gehitzen dugu (tos seca, 2). Orain bere covidImpact 6-ra igo da, hurrengo irudietan agertzen den bezala:



Bukatzeko, (fiebre,3) eta (astenia, 3) sintomak gehitzem ditugu:



Aplikazio hau osatzeko Observer patroiaren egituran klase berri bat sortuko dugu. Azpimagarria da Observer patroari esker, aplikazioa hedatu dezakegula, **aurrez garatutako inongo klaserik aldatu gabe**. Hau da diseinu berria:



PacientThermometerGUI klasea gehitu behar izan dugu soilik. Jarraian klase hau aplikaziori gehitzeko jarraitu behar diren pausoak aurkezten dira:

1. Observer klasea inplementatu hau da, **implements** Observer klasearen definiziora gehitu.
2. Metodo eraikitzaileari "obs" izeneko *Observable* parametro bat gehitu. Parametro hau subskribatu nahi dugun objektua izango da.
3. Metodoaren eraikitzailearen bukaeran, pasatutako parametroari subskribatu nahi dela adieraziko dugu, hau da, obs.addObserver(**this**);
4. *Observer* interfazea inplementatzen duen *update* metodoa eguneratuko dugu. Metodo honetan, notifikazioak jasoko dira subskribatuta gauden objektua aldatzen denean (*Observable* o objektua).

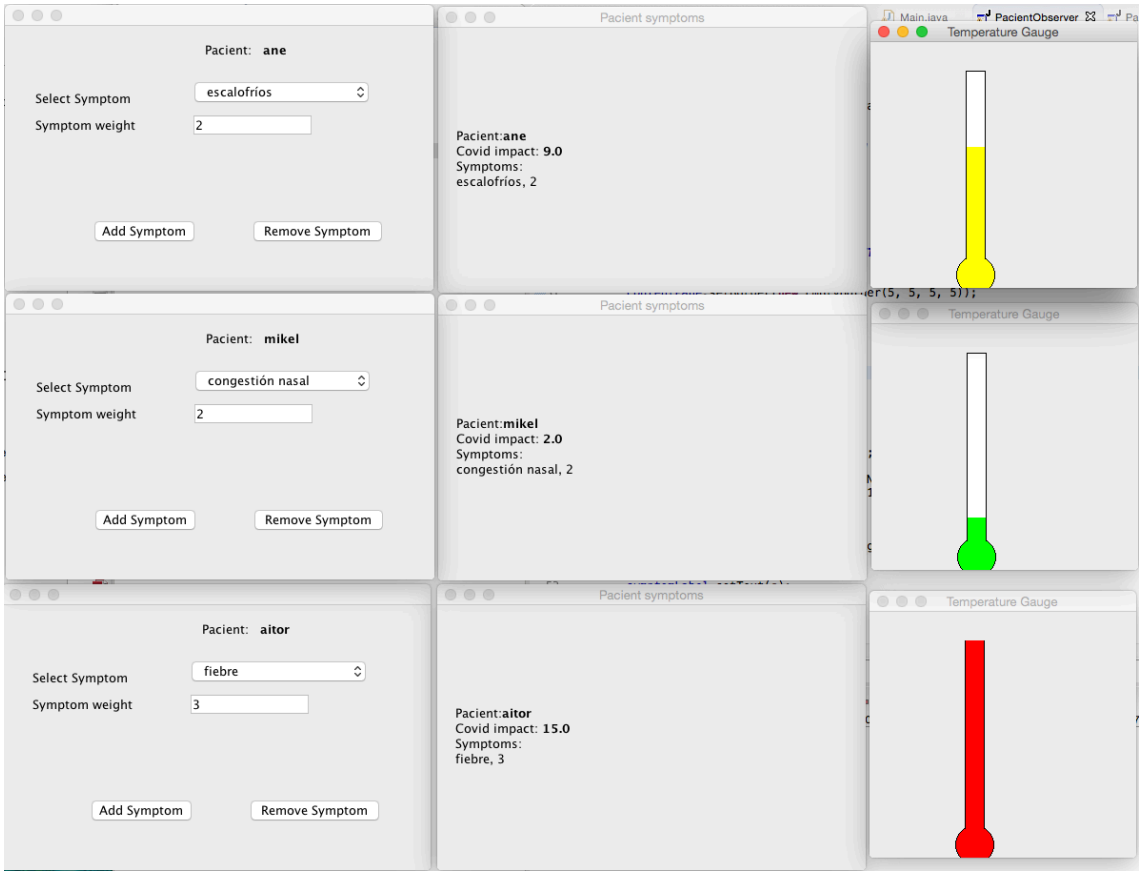
```
public void update(Observable o, Object o) {
                Covid19Pacient p=(Covid19Pacient)o;
                // Obtain the  current covidImpact to paint
                int farenheit = (int) p.calcCovid19Impact();
                // temperature gauge update
                gauges.set(farenheit);
                gauges.repaint();

        }
```

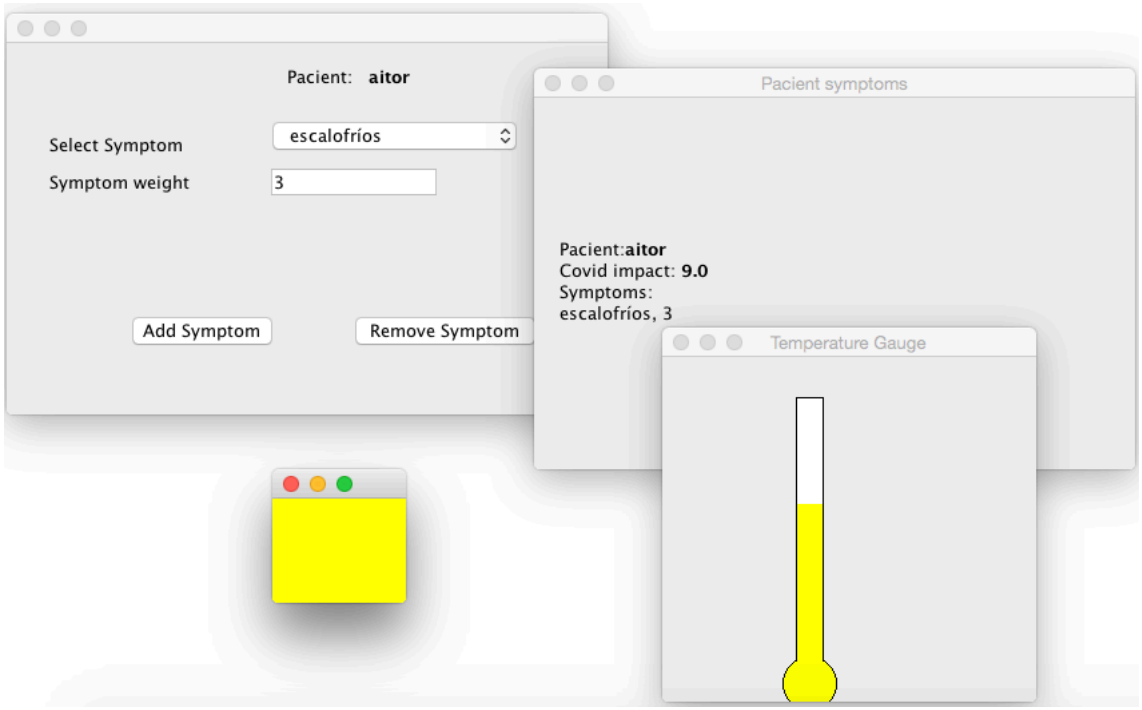
Prototipo honekin bukatzeko, **bi ariketa eskatzen dira** dira:

1. Programa nagusi bat sortu klase berri honetako objektu bat sortu pacient objetuan egiten diren aldaketak bistaratzeko bi interfaze grafikoetan. Programa exekutatu eta frogatu emaitzak.

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



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



Eskatzen da:

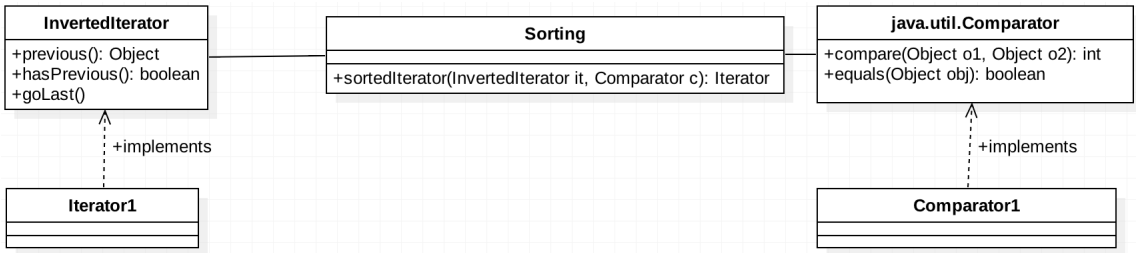
1. Aplikazioaren UML diagrama hedatuta egin dituzun aldaketan aurkeztuz.
2. Aplikazioaren inplementazioa.

Hau da SemaphorGUI klasearen hasierako kodea:

```
public class SemaphorGUI extends JFrame {
        /** stores the associated ConcreteSubject */
        public SemaphorGUI (int h, int w) {
                setSize(h, w);
                setLocation(350,10);
                Color c= Color.green;
                getContentPane().setBackground(c);
                repaint();
                setVisible(true);
        }
}
```

Adapter Patroia

Sorting klasea *Strategy* diseinu patroia erabiliz inplementatu da, hau da, `sortedIterator` metodoak, elementuak korritzeko InvertedIterator inplementatzen duen estrategia bat erabiltzen du, eta modu berean elementuak ordenatzeko Comparator inplementatzen duen estrategia bat. Hurrengo irudian aplikazioaren diseinu eta inplementazioa nolakoa den ikusi daiteke:




```
public class Sorting {
   public static Iterator sortedIterator(InvertedIterator it, Comparator comparator) {
              List list = new ArrayList();
              it.goLast();
              while (it.hasPrevious()) {
                   Symptom s=(Symptom)it.previous();
                  list.add(s);
              }
              Collections.sort(list, comparator);
              return list.iterator();
          }

}
```

Invertedterator interfazearen deskripzi:

```
public interface InvertedIterator {
    // return the actual element and go to the previous
    public Object previous();

    // true if there is a previous element
    public boolean hasPrevious();

    // It is placed in the last element
    public void goLast();
}
```

Iterator eta Comparator java.util paketean definitutako bi interfaze dira. Beren espezifikazioa hurrengoa da:

Method Summary

Interface Iterator

Modifier and Type	Method and Description
`Boolean`	**`hasNext()`** Returns `true` if the iteration has more elements.
`E`	**`next()`** Returns the next element in the iteration.

Interface Comparator

Modifier and Type	Method and Description
`int`	**`compare(T o1, T o2)`** Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the first argument is less than, equal to, or greater than the second.
`Boolean`	**`equals(Object obj)`** Indicates whether some other object is "equal to" this comparator.

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- Comparator interfazea inplementatzen dituzten bi klase definitu elementuak symptomName eta severityIndex ordenatzen dituztenak hurrenez hurren.
2- Covid19Pacient klasea InvertedIterator interfazera egokitzen duen klase adaptadorea sortu. Gogoratu metodo eraikitzaile egokia sortzea pazienteren informazioa bidaltzeko.
3- 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 pantaitik.
4- UML diagrama aplikazioaren diseinuarekin.

Laburbilduz eskatzen dena laborategi guztirako: PDF dokumentu bat. Ariketa bakoitzarako, (a) Diseinu hedatua, UML diagrama hedatua agertuko da egin diren aldatak aurkeztuz, (b) inplementazioa, garatu duzun kode azpimagarrienarekin, instrukzio garrantzitsuenak deskribatuz, eta (c) github esteka, proeiktuaren helbidearekin.

Lan Pertsonala
Balorazioa: 0.5 puntu