Github esteka: https://github.com/estibalizleza/labpatterns.git

**SIMPLE FACTORY**

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

- Sintoma mota berria agertzean, `Covid19Pacient` edo `Medicament` klasean aldaketak egin beharko genituzke. Baina Simple Factory patroia erabilita, aldaketa hau beharrezkoa ez litzateke izango, OCP printzipioa jarraituz.

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

- Simple Factory patroia erabiliz, sintoma mota berriak gehitu ahal izango ditugu `SymptomFactory` klasea aldatu gabe. Hori lortzeko, `Symptom` motako objektu berriak sortuko dituen Simple Factory bat sortu behar dugu.

**3. Zenbat erantzukizun dauzkate `Covid19Pacient` eta `Medicament` klaseek (SRP printzipioa)?**

- `Covid19Pacient` eta `Medicament` klaseek `Symptom` objektuak sortzearen ardura dute, hau SRP (Single Responsibility Principle) printzipioa urratuz. Simple Factory erabiliz, sortze erantzukizuna `SymptomFactory` klase batera mugatuko dugu, klase bakoitza bere zereginean espezializatuta utziz.

**Kode berria:**
Simple Factory erabiltzeko, `SymptomFactory` klase bat gehitu dugu:

```
public class SymptomFactory {
     public 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");
           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;

      }


}
```

**OBSERVER PATROIA**

**Kodea eta Inplementazioa:**

- `Covid19Pacient` klasean `Observable` heredatu eta `setChanged();` `notifyObservers();` metodoak gehitu ditugu `addSymptomByName` eta `removeSymptomByName` metodoetan:

```
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;
}
```

- `PacientObserverGUI` klaseak `Observer` interfazea implementatzen du, eta `update` metodoan GUI eguneratzen da.

```
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);
}
```

**ADAPTER PATROIA**

`Adapter` patroia erabiliz, `Covid19PacientTableModelAdapter` sortu dugu `Covid19Pacient` eta `TableModel` artean. Adapter klase honek, `Covid19Pacient` objektu baten sintomak taula batean bistaratuko ditu.

**Kodea eta Inplementazioa:**

- `Covid19PacientTableModelAdapter` klasea gehitu dugu:

```
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 pacient.getSymptoms().size();
        }
        public Object getValueAt(int row, int col) {
                // Challenge!
                Iterator<Symptom> it = pacient.getSymptoms().iterator();
                Symptom element = null;
                int currentRow = 0;
                while (it.hasNext()) {
```

```
                element = it.next();
                if (currentRow == row)
                        break;
                currentRow++;
        }
        if (element != null) {
                switch (col) {
                case 0:
                        return element.getName();
                case 1:
                        return pacient.getWeight(element);
                default:
                        return null;
                }
        }
        return null;
    }
}
```

**STRATEGY PATROIAK**

`Comparator` interfazearekin bi estrategia sortu ditugu: bata `SymptomNameComparator` eta bestea `SymptomSeverityComparator`. Horrez gain, `InvertedIterator` erabiltzen duen `Sorting` klasea estrategia aplikatzeko erabili da.

**Kodea eta Inplementazioa:**

- ComSymptomName eta ComSeverityIndex klaseak gehitu ditugu:

```
public class ComSymptomName implements Comparator<Symptom> {
        @Override
        public int compare(Symptom o1, Symptom o2) {
                return o1.getName().compareTo(o2.getName());
        }
}
public class ComSeverityIndex implements Comparator<Object> {
        @Override
        public int compare(Object o1, Object o2) {
                return ((domain.Symptom) o1).getSeverityIndex() - ((domain.Symptom)
o2).getSeverityIndex();
        }
}
```