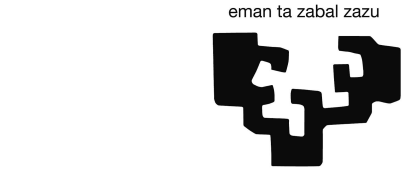
SOFTWARE INGENIARITZA II
DISEINU PATROIAK



EGILEAK: Oier Elola eta Unax Lazkanotegi

SARRERA

Dokumentu honetan Software Ingenieritza II irakasgaian diseinu patroien inguruko laborategian egindako lana aurkeztuko da. Hiru ariketa egongo dira bertan azalduta: Simple Factory, Observer eta Adapter izango dira. Ariketa bakoitzean azalpen bat eta kodea aurkitu ahal izango da, eta UML diagramak ikusi ahal izateko GitHub-eko errepositorioan aurkituko dituzue, ariketa bakoitzarentzat bat egongo da eta izenean adierazita (elab5-ariketaizena.mdj da UML diagramen fitxategien izen formatua). Diagramak dokumentu honetan atxikitu beharrean fitxategi horiek ireki eta bertan UML diagramak ikusi ahal izango dituzue. Enuntziatua elab5-patterns-2021 fitxategian aurkitzen da, zeina GitHub-eko errepositorioan eskuragarri egongo den.

SIMPLE FACTORY

Ariketa honetan eskatzen dena, laburbilduz honakoa da: sortu klase bat Symptom-ak sortzea ahalbidetzen diguna; izan ere, orain aurkitzen den moduan Medicament klasean zein Covid19Pacient klasean metodo berdina dago Symptom-ak sortzeko.

Beraz, SymptomSortzailea izeneko klasea sortuko dugu horretarako, non honen barruan egongo den Symptom objektua sortzeko metodoa, createSymptom() izenekoa, non parametro gisa sintomaren izena emango den. Hemen ikusi dezakezue SymptomSortzailea klasearen inplementazioa:

```
public class SymptomSortzailea {
	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","vomitos","congestion 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", "vomitos", "diarrea");
		List<String> neuroMuscularSymptom =
Arrays.asList("fiebre","astenia","cefalea","mialgia", "escalofrios");
		List<String> respiratorySymptom =
Arrays.asList("tos seca","expectoracion","disnea","dolor de garganta","congestiÃ³n 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;
	}

}
```

Baina, SymptomSortzailea erabiltzeko, bai Covid19Pacient eta Medicament klaseetan aldaketa bat egin behar dugu: beraien klaseetan atributu bat izatea SymptomSortzailea motakoa, eta beraz eraikitzailean SymptomSortzailea motako objektu bat ematea ekartzen du horrek. Jarraian ikusi ditzakezue aldaketak bi klaseetan (urdinez agertzen dira egindako aldaketak).

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

	public Covid19Pacient(String name, int years, SymptomSortzailea ss) {
		this.name = name;
		this.age = years;
		this.ss=ss;
	}
	public Symptom addSymptomByName(String symptom, Integer w){
		Symptom s=getSymptomByName(symptom);
		if (s==null) {
			s=ss.createSymptom(symptom);
			symptoms.put(s,w);
			setChanged();
			notifyObservers();
		}
		return s;
	}
}


public class Medicament {
	private String name;
	private List<Symptom> symptoms=new ArrayList<Symptom>();
	private SymptomSortzailea ss;

	public Medicament(String name,SymptomSortzailea ss) {
		super();
		this.name = name;
		this.ss=ss;
	}

	public Symptom addSymptomByName(String symptom){
		Symptom s2=null;
		Symptom s=getSymptomByName(symptom);
		if (s==null) {
			s2=ss.createSymptom(symptom);
			symptoms.add(s2);
		}
		return s2;
	}
}
```

Aldaketa horiek eginik, programa nagusira joko dugu; izan ere, orain aldaketak jasan ditu eta: aldaketa izango litzateke Covid19Pacient motako objektua edota Medicament motako objektua sortzerakoan SymptomSortzailea motako objektua eman beharko diogula.

```
public class Main {

	public static void main(String[] args) {
		Covid19Pacient p1=new Covid19Pacient("aitor", 35,new SymptomSortzailea());
		PacientSymptomGUI p1gui=new PacientSymptomGUI(p1);

		Medicament m=new Medicament("Ibuprofeno", new SymptomSortzailea());
		MedicalGUI mgui=new MedicalGUI(m);
		}
}
```

Bi probak batera jarri dira, baina gomendatzen da GUI bat probatzen bada bestea komentaturik uztea.

OHARRA: Kodeketa guztia bi klasetan banaturik aurkitzen da: Domain klasean aurkitzen dira klase guztiak eta Factory klasean bi GUI interfazeak eta Main klasea. UML diagrama ikusteko, elab5-Factory.mdj fitxategia ireki behar da.

OBSERVER

Observer ariketak hiru atal ditu:

- ➢ Lehendabiziko atalean eskatzen da Observer-Observable patroia ezartzea. Behin hori eginik, programa nagusi batean 2 Covid19Pacient sortu behar dira, non bakoitzak bere PacientSymptomGUI izango duen
- ➢ Bigarren prototipaketan, lehendabizi PacientThermometerGUI klasea gehituko dugu Observer gisa eta, aurrekoan bezala, programa nagusian 3 Covid19Pacient sortu behar dira bakoitzak bere interfazeak izanik.
- ➢ Azken prototipoan, SemaphorGUI klasea sortu eta garatu beharko dugu Observable gisa, eta programa nagusian Covid19Pacient bat sortuta interfaze guztiak izan beharko ditu.

Esan beharra dago bai lehenengo atala zein bigarrena burutzerako orduan jarraitu beharreko pauso guztiak ematen direla, beraz bertan emandako pausoei jarraituz eginda dago ariketa hau. Behin hiru atalak zeintzuk diren esanik, goazen bakoitzean sakontzera.

LEHEN ATALA

Atal honetan Observer-Observable patroia ezartzea da helburu eta frogatzea ongi funtzionatzen duela. Jarraitu beharreko prozedura enuntziatuan datorrenez, ez dugu sakonduko horretan (fitxategia, sarreran aipatu bezala GitHub-eko repositorioan aurkitu dezakezue) eta zentratuko gara programa nagusia azaltzen, azkenean lehenengo atal honen helburua hori baita: programa nagusi bat sortzea 2 Covid19Pacient sortzeko bakoitza bere PacientSymptomGUI interfazearekin.

```
public class Main1 {
	public static void main(String[] args) {
		Observable pacient=new Covid19Pacient("aitor", 35, new SymptomSortzailea());
		PacientObserverGUI pacientGUI1= new PacientObserverGUI (pacient);
		PacientSymptomGUI frame1 = new PacientSymptomGUI ((Covid19Pacient)pacient);
	}
}
```

BIGARREN ATALA

Atal honetan, lehenengoan bezala, programa nagusi bat sortu behar da amaieran 3 Covid19Pacient sortzeko bakoitza bere interfazeekin. Baina horraino heldu aurretik, PacientThermometerGUI interfazea inplementatu eta Covid19Pacient-en Observable birhutu behar dugu (kasu honetan ere enuntziatuak ematen ditu pausu guztiak nola egin prozesu hori). Beraz, aurrekoan bezala, programa nagusian zentratuko gara eta azalduko dugu nola egin dugun.

```
public class Main2 {
	public static void main(String[] args) {
		Observable pacient1=new Covid19Pacient("aitor",35,new SymptomSortzailea());
		PacientObserverGUI pacientGUI= new PacientObserverGUI(pacient1);
		PacientSymptomGUI frame = new PacientSymptomGUI ((Covid19Pacient)pacient1);
		PacientThermometerGUI termometro=new PacientThermometerGUI((Covid19Pacient)pacient1);

		Observable pacient2=new Covid19Pacient("karlos",24,new SymptomSortzailea());
		PacientObserverGUI pacientGUI2= new PacientObserverGUI(pacient2);
		PacientSymptomGUI frame2 = new PacientSymptomGUI ((Covid19Pacient)pacient2);
		PacientThermometerGUI termometro2= new PacientThermometerGUI((Covid19Pacient)pacient2);

		Observable pacient3=new Covid19Pacient("jose",47,new SymptomSortzailea());
		PacientObserverGUI pacientGUI3=new PacientObserverGUI(pacient3);
		PacientSymptomGUI frame3 =new PacientSymptomGUI ((Covid19Pacient)pacient3);
		PacientThermometerGUI termometro3= new PacientThermometerGUI((Covid19Pacient)pacient3);
	}
}
```

HIRUGARREN ATALA

Atal honetan, bigarren atalaren antzekoa da: kasu honetan SemaphorGUI interfazea sortu behar dugu eta Observable bihurtu eta ondoren programa nagusi bat sortu Covid19Pacient bat sortuz eta honi dagozkion interfazeak erakutsi. Baina, aurrekoetan ez bezala, enuntziatuak ez dizkigu jarraitu beharreko pausoak ematen, baina badakigu PacientThermometerGUI egiterakoan jarraitutako prozedura berdina dela. Beraz, jarraian lehendabizi esplikatuko dugu nola egin dugun eta ondoren joango gara programa nagusiarekin.

Hasieran, SemaphorGUI klaseak honako kodeketa du:

```
public class SemaphorGUI extends JFrame {
	public SemaphorGUI(){
		setSize(100,100);
		setLocation(350,10);
		Color  c=Color.green;
		getContentPane().setBackground(c); repaint(); setVisible(true);
	}
}
```

Noski, Observer klase bat izango denez, `implements Observer` gehituko diogu klasearen definizioan, eta eraikitzailean Observable motako objektu bat emango diogu, kasu honetan Covid19Pacient bat izango da.

```
public SemaphorGUI(Observable o){
	setSize(100,100);
	setLocation(350,10);
	Color  c=Color.green;
	getContentPane().setBackground(c); repaint(); setVisible(true);
}
```

Ez hori bakarrik, Covid19Pacient bat eguneratzean interfazea eguneratzeko update() metodo bat sortuko dugu, non Covid19Pacient klaseko covidImpact() metodoan lortzen den integer balioaren arabera semaforoak kolore bat edo beste erakutsiko duen:

- 5 baino txikiagoa bada, kolore berdea
- 5 eta 10 arteko balioa bada, kolore horia
- 10 baino handiagoa bada, kolore gorria

Beraz, hori kontutan izanik, update() metodoaren kodeketa honakoa da:

```
public void update(Observable o,Object arg){
	Covid19Pacient p=(Covid19Pacient) o;
	int farenheit= (int) p.covidImpact();
	if (farenheit<5){
		getContentPane().setBackground(Color.green);
	}else if (farenheit<10){
		getContentPane().setBackground(Color.yellow);
	}else{
		getContentPane().setBackground(Color.red);
	}
}
```

Beraz, update metodoaren bitartez eguneratzen joango da Covid19Pacient klasean aldaketaren bat gertatzen bada.

SemaphorGUI interfazea inplementatuta izanik, honako hau izango litzateke programa nagusia probatzeko:

```
public class Main3 {
	public static void main(String[] args) {
		Observable pacient1=new Covid19Pacient("aitor",35,new SymptomSortzailea());
		PacientObserverGUI pacientGUI= new PacientObserverGUI (pacient1);
		PacientSymptomGUI frame = new PacientSymptomGUI ((Covid19Pacient)pacient1);
		PacientThermometerGUI termometro= new PacientThermometerGUI((Covid19Pacient)pacient1);
		SemaphorGUI semaforo= new SemaphorGUI((Covid19Pacient) pacient1);
	}

}
```

OHARRA: Kodeketa guztia Observer klasean aurkitzen da. UML diagrama ikusteko, elab5-Observer.mdj fitxategia ireki behar da.

ADAPTER

Ariketa honetan eskatzen dena honakoa da: sortu behar dugu programa nagusi batean Covid19Pacient objektu bat bost Symptom dituena, eta ondoren Symptom horiek bi modutara ordenatu behar dira: alde batetik, symptomName bitartez ordenatuak eta bestetik severityIndex bitartez ordenatuak. Hori bai, Covid19Pacient eta Sorting klaseak ezin dira aldaketak egin kodean hori burutzeko.

Hori kontutan izanik, eta enuntziatuak ematen dizkigun baliabideak kontutan hartuta, bi konparatzaile klase sortuko ditugu: compareSeverityIndex eta comparateSymptomName, eta bakoitzak konparaketak egingo ditu bi Symptom-en artean, bakoitzean dagokion atributua. Bi klaseen java.Util.Comparator interfazea inplementatzen dute, eta jarraian kodea erakutsiko dizuegu:

```
public class compareSeverityIndex implements Comparator<Object> {
	@Override
	public int compare(Object o1, Object o2) {
		Symptom syp1 = (Symptom) o1;
		Integer i1 = syp1.getSeverityIndex();
		Symptom syp2 = (Symptom) o2;
		Integer i2 = syp2.getSeverityIndex();
		return i1.compareTo(i2);
	}
}

public class comparateSymptomName implements Comparator<Object> {
	@Override
	public int compare(Object o1, Object o2) {
		Symptom syp1 = (Symptom) o1;
		String s1 = syp1.getName();
		Symptom syp2 = (Symptom) o2;
		String s2 = syp2.getName();
		return s1.compareTo(s2);
	}
}
```

Ez hori bakarrik, Covid19PacientInvertedIterator klasea kodetuko dugudugu, zeinak InvertedIterator klasea inplementatuko duen. Honako hau da klasearen kodeketa:

```
public class Covid19PacientInvertedIterartor implements InvertedIterator {
	private List<Object> list;
	private int posizioa;
	public Covid19PacientInvertedIterartor(List<Object> list) {
		this.list = list;
		posizioa = 0;
	}
	public Object previous() {
		posizioa++;
		return list.get(posizioa - 1);
	}
```

```
	public boolean hasPrevious() {
		return posizioa < list.size();
	}
	public void goLast() {
		posizioa = 0;
	}

}
```

Behin kodeketa guztia, programa nagusia inplementatuko dugu, non bertan Sorting.sort() metodoari deituz, parametro gisa Covid19PacientInvertedIterator objektu bat eta sortutako konparatzaile bat emanik, iteratzaile batean gordeko da emaitza eta hori pantailaratuko da. Jarraian ikusi dezakezue kodeketa.

```
public class Main {
	public static void main(String[] args) {
		Covid19Pacient p = new Covid19Pacient("Unax", 20, new SymptomSortzailea());
		p.addSymptom(new Symptom("s1", 10, 11), 1);
		p.addSymptom(new Symptom("s2", 10, 12), 2);
		p.addSymptom(new Symptom("s3", 10, 8), 3);
		p.addSymptom(new Symptom("s4", 10, 2), 4);
		p.addSymptom(new Symptom("s5", 10, 7), 5);
		Set<Symptom> giltzak = p.getSymptoms();
		Iterator giltzIt = giltzak.iterator();
		ArrayList<Object> keys = new ArrayList<Object>();
		Symptom sy;
		while (giltzIt.hasNext())
			keys.add(giltzIt.next());
		Sorting s = new Sorting();
		Iterator<Object> it1 = s.sortedIterator(new Covid19PacientInvertedIterartor(keys), new
comparateSymptomName());

		System.out.println("Emaitza, string kasurako:\n");
		Symptom s1;
		while (it1.hasNext()) {
			s1 = (Symptom) it1.next();
			System.out.println("Izena: " + s1.getName() + " Severity: " + s1.getSeverityIndex());
		}

		Iterator<Object> it2 = s.sortedIterator(new Covid19PacientInvertedIterartor(keys), new
compareSeverityIndex());

		System.out.println("Emaitza, severity kasurako:\n");
		Symptom s2;
		while (it2.hasNext()) {
			s2 = (Symptom) it2.next();
			System.out.println("Izena: " + s2.getName() + " Severity: " + s2.getSeverityIndex());
		}
	}
}
```

OHARRA:Kodeketa guztia Adapter klasean aurkitzen da. UML diagrama ikusteko, elab5-SortingAdapter.mdj fitxategia ireki behar da.

GITHUB REPOSITORIOA

Atal honetan GitHub-eko errepositoriora link-a aurkitu dezakezue, non bertan Eclipse-ko proiektua, UML diagramak ariketa bakoitzarenak eta bi pdf: bata dokumentu hau bera izango da eta bestea enuntziatuari dagokiona izango da.

GitHub-eko esteka: https://github.com/Elola27/labpatterns