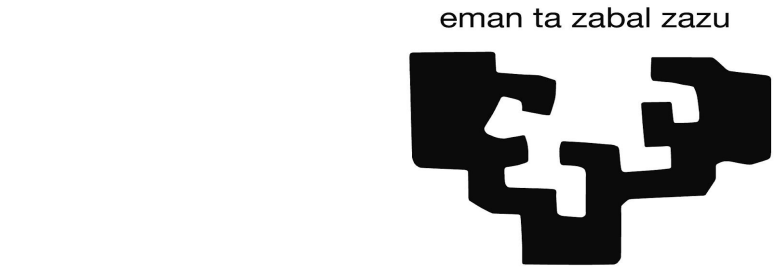
SOFTWARE INGENIERITZA II
DISEINU PATROIAK PROIEKTUAN APLIKATUTA



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Egileak: Oier Elola eta Unax Lazkanotegi

SARRERA

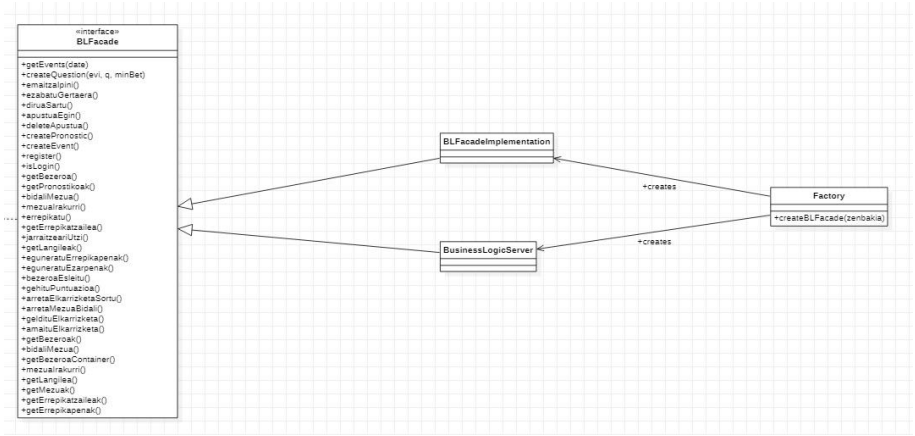
Dokumentu honetan Software Ingenieritza I irakasgaian landutako Bets21 proiektuaren gainean diseinu patroiak aplikatuz eginiko ariketak nola egin diren azalduko da: bakoitzean helburu nagusia zein den, nola egin den eta zenbait irudi ikusiko dira (irudi horiek UML diagramak edota exekuzio argazkiak izan daitezke). Ariketak zehazten dituen dokumentua GitHub-eko errepositorioan aurki dezakezue (bere izena elab6-patroiak-Apustuak.pdf da), eta laguntzarako emaniko Power Point fitxategia ere (elab6-Bets.pptx izenekoa). UML diagramei dagokienez, Bets21_DiseinuPatroiak.mdj izena du fitxategiak, eta bertan Class Diagram atalean ikusi dezakezue UML diagrama hedatua.

FACTORY

Ariketa honen helburu nagusia honakoa da: negozio logika sortzeko prozedura Factory patroira egokitzea. Hori aurrera eramateko, atal bakoitza identifikatu behar dugu:

- ➢ Creator lana egingo duena guk sortuko dugun klasea izango da; bere izena Factory da.
- ➢ Product interfazea BLFacade izango da.
- ➢ ConcreteProduct kasu honetan bi aukera egongo dira: BLFacadeInplementation lokalean exekutatzen bada negozio logika, eta BusinessLogicServer baldin eta urruneko negozio logika exekutatzen bada.

Beraz, atal guztiak identifikatuta izanik, UML diagraman bere errepresentazioa zein den jarraian ikusi dezakezue:



Behin diagrama ikusirik, orain inplementatzera joango gara; izan ere, Factory klaseak sortuko ditu biak. Hau izango da Factory klasearen inplementazioa:

```
public class Factory {

public static BLFacade createBLFacade(int zenbakia) {
        if (zenbakia == 0) {
                return new BLFacadeImplementation();
        }if (zenbakia==1) {
                try {
                        ConfigXML c = ConfigXML.getInstance();
                        String serviceName = "http://" + c.getBusinessLogicNode() + ":" +
c.getBusinessLogicPort() + "/ws/"+ c.getBusinessLogicName() + "?wsdl";
                        URL url = new URL(serviceName);
                        // 1st argument refers to wsdl document above
                        // 2nd argument is service name, refer to wsdl document above
                        QName qname = new QName("http://businessLogic/",
"BLFacadeImplementationService");
                        Service service = Service.create(url, qname);
                        return service.getPort(BLFacade.class);
                } catch (Exception e) {
                        System.out.println("Error at retrieving the business logic: " + e.toString());
                }
        }
        return null;
}
}
```

Factory klaseak `createBLFacade()` funtzioa du, non parametro bezala zenbaki bat jasoko duen: 0 bada negozio logika lokalekoa itzuliko du eta 1 bada urrunekoa itzuliko du.

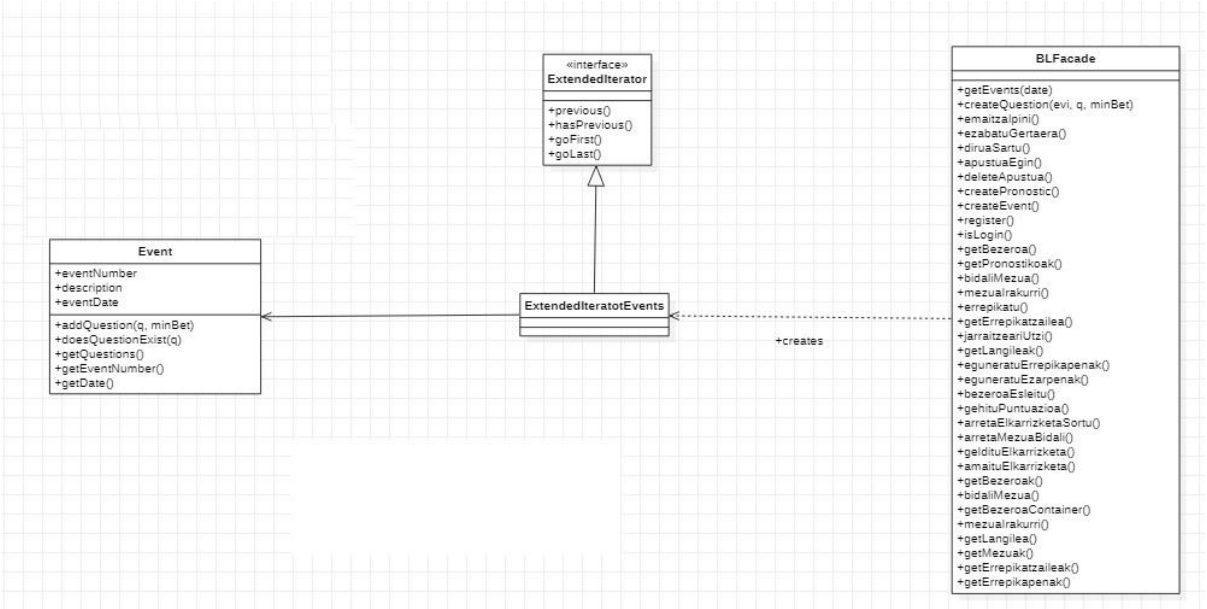
Beraz, aldaketa hori eginik, orain ApplicationLauncher.java klasean aldatuko dugu kodea Factory klasea erabil dadin negozio logika sortzeko. Zein negozio logika erabili isBusinessLogicLocal() funtzioak emango digu: baldin eta true bada lokalekoa `createBLFacade(0)` deia egingo zaio Factory-ri eta honek BLFacadeInplementation objetkua emango dio; false izanez gero BusinessLogicServer-ek jaurtiriko zerbitzua lortzen saiatuko da `createBLFacade(1)` Factory deiaren bidez. Honela geratuko litzateke aldaketak egin ostean:

```
public class ApplicationLauncher {
public static void main(String[] args) {

	ConfigXML c=ConfigXML.getInstance();

	System.out.println(c.getLocale());

	Locale.setDefault(new Locale(c.getLocale()));

	System.out.println("Locale: "+Locale.getDefault());

	MainGUI a=new MainGUI();
	a.setVisible(true);

	try {

		BLFacade appFacadeInterface;
		UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

		if (c.isBusinessLogicLocal()) {
			appFacadeInterface=Factory.createBLFacade(0);
		}

		else { //If remote
			appFacadeInterface=Factory.createBLFacade(1);
		}
		MainGUI.setBussinessLogic(appFacadeInterface);
	}catch (Exception e) {
		a.jLabelSelectOption.setText("Error: "+e.toString());
		a.jLabelSelectOption.setForeground(Color.RED);

		System.out.println("Error in ApplicationLauncher: "+e.toString());
	}
}
}
```

ITERATOR

Ariketa honetan helburua Event klasearentzako iteradore bat sortzean datza, eta iteradore horrek aurkako noranzkoan Event objektuak erakusteko gaitasuna izan behar du. Horretarako, enuntziatuan ExtendedIterator interfazea eskaintzen zaigu, zeinak Iterator klasea extenditzen duen, eta bera baliatuz Event klasearentzako iteradore hedatu bat sortu behar dugu.

Beraz, UML diagrama batean erakutsi nahiko bagenu zein izango litzatekeen banaketa/klase bakoitzaren posizioa jarraian agertzen dena dela esan daiteke:



Behin ulerturik antolamendua, goazen ExtendedIterator interfazea inplementatzen duen iteradore hedatua sortzera, zeinak ExtendedIteratorEvents izena izango duen, eta bertan ExtendedIterator klaseko metodo guztiak gainidatziko diren gure beharretara egokituz. Hau da klasearen inplementazioa:

```
public class ExtendedIteratorEvents implements ExtendedIterator {
	private int pos;
	private Vector<Event> apus;

	public ExtendedIteratorEvents(Vector<Event> p) {
		this.pos=0;
		this.apus=p;
	}

	public Object previous() {
		// TODO Auto-generated method stub
		Event uneko=this.apus.get(pos);
		pos--;
		return uneko;
	}
```

```
	public boolean hasPrevious() {
		// TODO Auto-generated method stub
		if (this.pos>=0) {
			return true;
		}else {
			return false;
		}
	}

	public void goFirst() {
		// TODO Auto-generated method stub
		this.pos=0;
	}

	public void goLast() {
		// TODO Auto-generated method stub
		this.pos=apus.size()-1;
	}

	public boolean hasNext() {
		// TODO Auto-generated method stub
		if (this.pos<=this.apus.size()-1) {
			return true;
		}else {
			return false;
		}
	}

	public Object next() {
		// TODO Auto-generated method stub
		Event uneko=this.apus.get(pos);
		pos++;
		return uneko;
	}
}
```

Ez hori bakarrik, BLFacade eta BLFacadeImplementation klaseetan aldaketa bat egingo dugu: getEvents metodoak ez du Vector<Event> objektu bat itzuliko, baizik eta ExtendedIterator<Event> objektu bat; horrela, kodetu dugun iteradore hedatua proba dezakegu probarako sortuko dugun main programan. Honela geratuko litzateke BLFacade-n zein BLFacadeImplementation klaseetan getEvents() funtzioaren inplementazioa:

BLFacade:

```
@WebMethod public ExtendedIterator<Event> getEventsIterator(Date date);
```

BLFacadeImplementation:

```
public ExtendedIterator<Event> getEventsIterator(Date date) {
	return new ExtendedIteratorEvents(getEvents(date));
}
```

Azkenik, main programa bat sortuko dugu kodetutako guztia frogatzeko ongi funtzionatzen duela: TestExtendedIteratorEvents. java klasean egin dugun programa nagusiaren kodea honakoa da (kode hau enuntziatuan dator, beraz itsatsi besterik ez dugu egin eta ondoren exekutatu egingo dugu).

```
public class TestExtendedIteratorEvents {
	public static void main(String[] args) {
		int isLocal = 0;
		BLFacade blFacade = Factory.createBLFacade(isLocal);
		SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy");
		Date date;
		try {
			date = sdf.parse("17/12/2021"); //17 del mes que viene
			ExtendedIterator<Event> i = blFacade.getEventsIterator(date);
			Event e;
			System.out.println("_____________________");
			System.out.println("ATZETIK AURRERAKA");
			i.goLast();//Azkeneko elementuan kokatu
			while (i.hasPrevious()) {
				e = i.previous();
				System.out.println(e.toString());
			}
			System.out.println();
			System.out.println("_____________________");
			System.out.println("AURRETIK ATZERAKA");
			i.goFirst(); // Lehen elem. kokatu
			while (i.hasNext()) {
				e = i.next();
				System.out.println(e.toString());
			}
		}catch(Exception e) {
			System.out.println("Error at the procedure:" + e.toString());
		}
	}
}
```

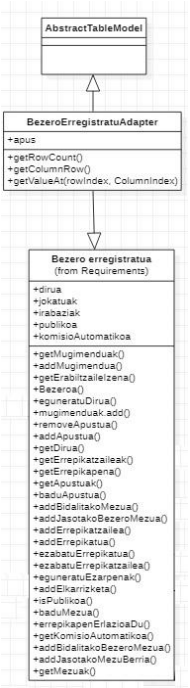
Jarraian ikusi dezakezuen argazkian probaren ondorioz lortutako emaitzak dira, eta ikusi daitekeen moduan, ongi funtzionatzen du iteradoreak: lehendabizi atzetik aurreraka erakusten ditu Event objektu bakoitzaren deskripzioak, eta ondoren aurrekoz atzera prozesu berdina, betiere guk sorturiko iteradore hedatuarekin.

```
_____________________
ATZETIK AURRERAKA
Betis-Real Madrid
Real Sociedad-Levante
Girona-Leganés
Malaga-Valencia
Las Palmas-Sevilla
Español-Villareal
Alavés-Deportivo
Getafe-Celta
Eibar-Barcelona
Atlético-Athletic

_____________________
AURRETIK ATZERAKA
Atlético-Athletic
Eibar-Barcelona
Getafe-Celta
Alavés-Deportivo
Español-Villareal
Las Palmas-Sevilla
Malaga-Valencia
Girona-Leganés
Real Sociedad-Levante
Betis-Real Madrid
```

ADAPTER

Azken ariketa honen helburua honakoa da: aplikazioa martxan jartzean botoi bat egotea pantaila nagusian, non honi klikatuz gero JTable bat erakutsiko duen Bezero Erregistratu klaseko objektu batek dituen Apustua motako objektuen informazioa erakutsiz, baina Bezero Erregistratua klasea ezin da aldatu.

Beraz, zein izango litzateke soluzioa hau burutzeko? Bada, Adapter patroia aplikatuz lor genezake hori: BezeroErregistratuAdapter klasea sortuko dugu, non azken honek AbstractTableModel extenditzen duen, eta horrela sor dezakegu taula bat. Eskuinean ikusi dezakezue UML diagrama nola izango litzateke:



Behin UML diagrama ikusirik nola geratuko litzatekeen, goazen BezeroErregistratuAdapter klasearen inplementazioa erakustera, azken finean AbstractTableModel klaseko metodoak gainidazten dira gure beharretara egokituz.

```
public class BezeroErregistratuAdapter extends AbstractTableModel{
	private Vector<Apustua> apus;

	public BezeroErregistratuAdapter(Vector<Apustua> p) {
		apus=p;
	}

	public int getRowCount() {
		// TODO Auto-generated method stub
		return apus.size();
	}

	public int getColumnCount() {
		// TODO Auto-generated method stub
		return 4;
	}

	public Object getValueAt(int rowIndex, int columnIndex) {
		// TODO Auto-generated method stub
		Apustua ap=apus.get(rowIndex);
		switch (columnIndex) {
		case 0:
			return ap.getIdentifikadorea();
		case 1:
			return ap.getKuotaTotala();
		case 2:
			return ap.getPronostikoKop();
		case 3:
			return ap.getAsmatutakoKop();
			default:
				return null;
		}
	}
};
```

Behin inplementaturik klase hau, JFrame bat sortu behar dugu ikusteko. JFrame honen izena JTableAdapter izango da, eta bertan JTable bat sortuko da emandako BezeroErregistratuAdapter-a emanik, non aldi berean BezeroErregistratuAdapter klase hau sortuko den negozio logikako getBezeroa() funtziotik lortutako Bezeroa klaseko objektutik Apustuen zerrendatik (begiratu BezeroErregistratuAdapter klasearen eraikitzailea). Jarraian ikusi dezakezue bere kodea (gure kasuan, “b” erabiltzaile izena duen Bezeroarekin egingo dugu proba, datubasea initialize moduan eginik Apustua motako objektu bat izango baitu gordeta).

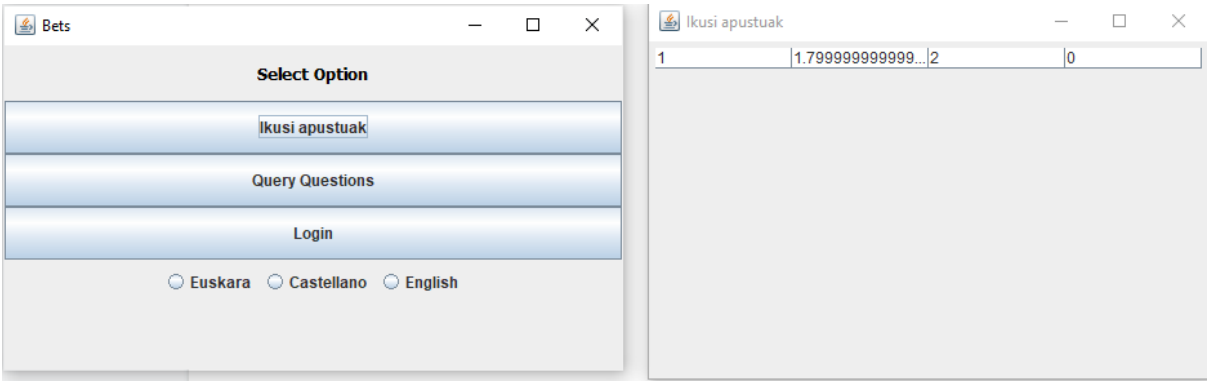
```
public JTableAdapter() {
	setTitle("Ikusi apustuak");
	BLFacade blFacade= new BLFacadeImplementation();
	user = blFacade.getBezeroa("b");
	setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	setBounds(100, 100, 450, 300);
	contentPane = new JPanel();
	contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
	contentPane.setLayout(new BorderLayout(0, 0));
	setContentPane(contentPane);

	BezeroErregistratuAdapter adaptadorea=new
	BezeroErregistratuAdapter(user.getApustuak());
	taula=new JTable(adaptadorea);
		//taula.setModel(taulaModel);
	taula.setBounds(80,80, 250, 300);
	getContentPane().add(taula,BorderLayout.NORTH);
}
```

Baina, MainGUI JFrame-ean aldaketak egin behar ditugu; izan ere, aplikazioa martxan jartzean ezin gara JTableAdapter-era heldu. Beraz, botoi berri bat sortuko dugu, eta hau sakatzean JTableAdapter klaseko lehio berri bat zabalduko digu. Botoiaren funtzionalitatea honela inplementatu dugu:

```
private JButton getBoton4() {
	if (jButtonIkusiApustuak == null) {
		jButtonIkusiApustuak = new JButton();
		jButtonIkusiApustuak.setText("Ikusi apustuak");
		jButtonIkusiApustuak.addActionListener(new java.awt.event.ActionListener() {
			public void actionPerformed(java.awt.event.ActionEvent e) {
				JFrame a = new JTableAdapter();
				a.setVisible(true);
			}
		});
	}
	return jButtonIkusiApustuak;
}
```

Beraz, orain ikusiko genukeena exekuzioa burutuz gero honakoa izango litzateke: interfaze nagusia (MainGUI) lau hiru aukerekin, eta “Ikusi Apustuak” botoian sakatuz gero,

JTableAdapter leihoa irekiko liguke, beheran ikusi ditzakezue bi JFrame-ak nola geratu diren.



OHARRAK

Hainbat ohar eman nahi dira aldez aurretik jakinarazteko:

- Factory probarako, BLFacade-n zein BLFacadeImplementation klaseetako `public ExtendedIterator<Event> getEventsIterator(Date date)` metodoa komentario gisa utzi behar da; izan ere, JAXB-k arazoak ematen ditu eta antza denez ezin du tratatu ExtendedIterator interfazea eta orduan urruneko negozio logika martxan ez du jarriko.
- Adapter ariketan taulako zutabe bakoitzak honakoa adierazten du orden honetan:
    - Apustuaren identifikadorea
    - Irabazi posibleak
    - Pronostiko kopurua
    - Asmatutako pronostiko kopurua

  Kontua da saiatu garela taularen lehenengo lerro bezala zutabeak izendatzen, baina ez dugu lortu hori egitea, nahiz eta kode egokia eskuragarri izan egiteko. Beraz, atal honetan adierazten dugu zutabe bakoitzak nori/zeri egiten dion erreferentzia.

- Beste erabiltzaile batekin probatu nahi bada Adapter ariketa, UserTableAdapter klasearen eraikitzailean dagoen `blFacade.getBezeroa()` funtzioan aldatu behar da ematen den parametroa; hau da, guk probatarako “b” erabiltzaile izena ematen diogu, eta beste erabiltzaile izen bat emateko aukera dago horrela.
- Azkenik, kontsolan errore batzuk azalduko dira; lasai egon, dira datu basean objektu konkretu bat ezin delako sartu (oraindik ez dakigu zergatik gertatzen den zehazki, eta

GITHUB-EKO ERREPOSITORIOA

Atal honetan GitHub-eko errepositoriorako link-a aurkituko duzue, non bertan hainbat fitxategi ikusteko aukera izango duzuen:

- ➢ Enuntziatua eta laguntza fitxategiak: elab6-patroiak-Apustuak.pdf eta elab6-Bets.pptx hurrenez hurren.
- ➢ Proiektuaren kodea.
- ➢ UML fitxategi eguneratua: Bets21_DiseinuPatroiak.mdj
- ➢ Iazko proiektua garatu zeneko lanak + aurtengo ikasturtean proiektuaren gainean garatutako txostenak.

GitHub-eko errepositorioaren link-a: https://www.github.com/Elola27/SI2_Integratua