

# PROIEKTUAREN DISEINU PATROIAK

Jon Reboiro, Estibaliz Leza eta Irantzu Loero

## Aurkibidea:

<b>1. Factory Method Patroia.....</b>	<b>2</b>
Egindakoaren deskribapena:.....	2
Aldatutako kodearen lerro garrantzitsuenak:.....	2
UML:.....	3
<b>2. Iterator Patroia.....</b>	<b>4</b>
UML:.....	4
Aldatutako kodearen lerro garrantzitsuenak:.....	4
Exekuzioaren irudia:.....	6
<b>3. Adapter Patroia.....</b>	<b>7</b>
UML:.....	7
Aldatutako kodearen lerro garrantzitsuenak:.....	7
Exekuzioaren irudia:.....	8

# 1. Factory Method Patroia

## Egindakoaren deskribapena:

Factory Method patroiarene implementazioan, BLFactory klase bat sortu dugu, Creator gisa jokatzeko duena. Haren funtzioa, konfigurazioaren arabera, zein BLFacade objektua (lokala edo urrunekoa) erabili behar den erabakitzea da. BLFacade interfazeak Product rola jokatzeko du, bi implementazioek bete beharreko eragiketak definitzen baititu. Azkenik, BLFacadeImplementation-ek eta BusinessLogicServer-ek ConcreteProduct rolak jokatzeko dituzte, Creator-ek (BLFactory) ezarritako konfigurazioaren arabera itzultzen dituen BLFacade bertsio zehatzak baitira.

## Aldatutako kodearen lerro garrantzitsuenak:

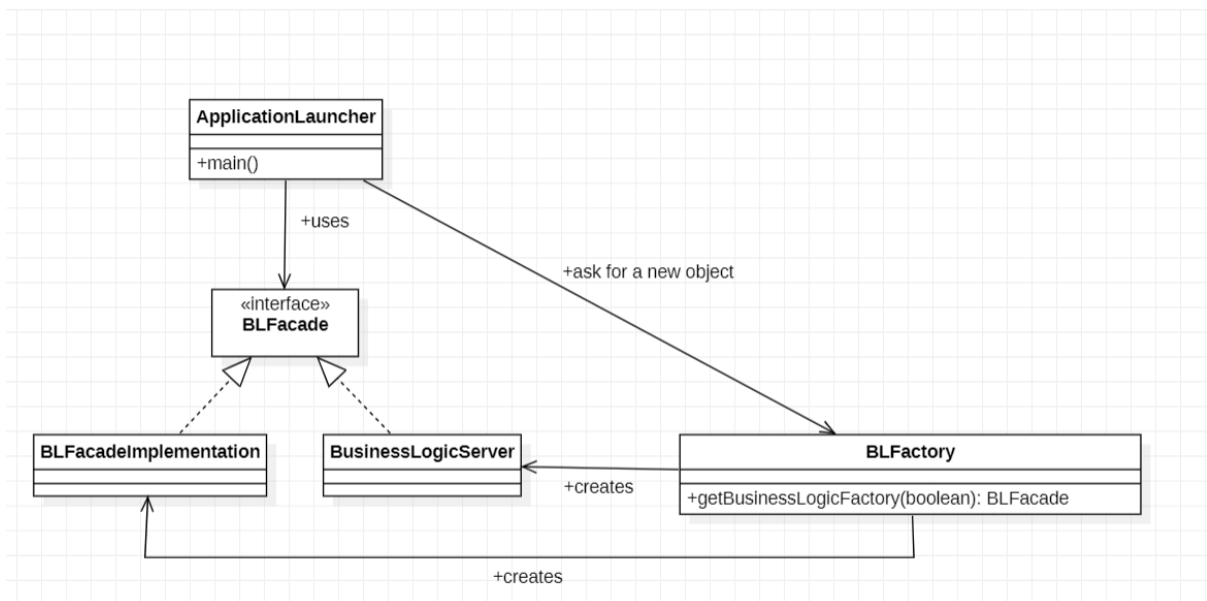
BLFactory klase bat sortu dugu businessLogic klasearen barruan. Klase honek aurreko bertsioan main metodoan zegoen kodea barneratzen du, pixka bat aldatuz. if/else kode zatia BLFactory klaseko metodo batean barneratzen du, getBusinessLogicFactory metodoan. isLocal aldagairen balioaren arabera, zein BLFacade objektua (lokala edo betse makina batean) erabili behar den erabakitzen du.

```
public class BLFactory {  
  
    public BLFacade getBusinessLogicFactory(boolean isLocal) {  
        if (isLocal) {  
            DataAccess da = new DataAccess();  
            return new BLFacadeImplementation(da);  
        } else {  
            try {  
                ConfigXML config = ConfigXML.getInstance();  
                String serviceName = "http://" + config.getBusinessLogicNode() + ":"  
                    + config.getBusinessLogicPort() + "/ws/"  
                    + config.getBusinessLogicName() + "?wsdl";  
                URL url = new URL(serviceName);  
                QName qname = new QName("http://businessLogic/",  
                    "BLFacadeImplementationService");  
                Service service = Service.create(url, qname);  
                return service.getPort(BLFacade.class);  
            } catch (MalformedURLException e) {  
                System.err.println("Malformed URL: " + e.getMessage());  
                return null;  
            }  
        }  
    }  
}
```

Gero ApplicationLauncher klasean kodea ere moldatu dugu, BLFactory klasea erabiltzeko. try barruko kodea aldatu dugu:

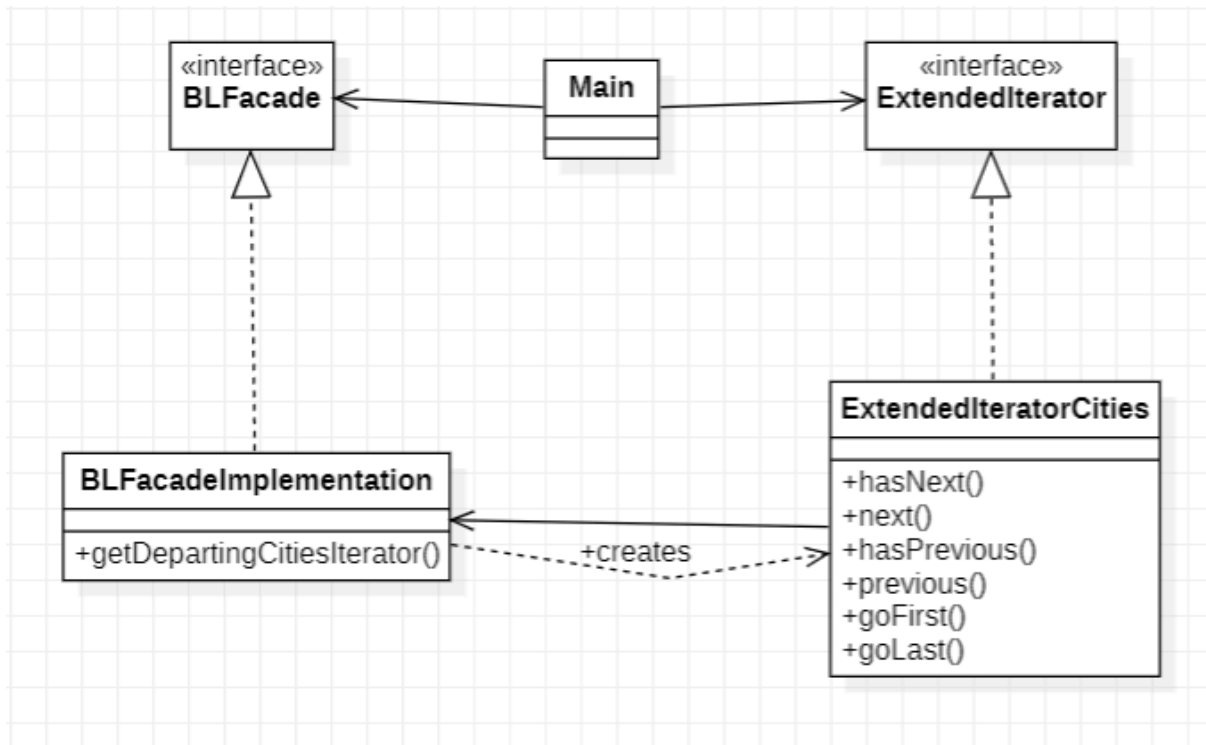
```
public static void main(String[] args) {
    ConfigXML c = ConfigXML.getInstance();
    System.out.println(c.getLocale());
    Locale.setDefault(new Locale(c.getLocale()));
    System.out.println("Locale: " + Locale.getDefault());
    try {
        boolean isLocal = c.isBusinessLogicLocal();
        BLFacade appFacadeInterface = new BLFactory().getBusinessLogicFactory(isLocal);
        MainGUI.setBussinessLogic(appFacadeInterface);
        MainGUI a = new MainGUI();
        a.setVisible(true);
    } catch (Exception e) {
        // a.jLabelSelectOption.setText("Error: "+e.toString());
        // a.jLabelSelectOption.setForeground(Color.RED);
        System.out.println("Error in ApplicationLauncher: " + e.toString());
    }
    // a.pack();
}
```

UML:



## 2. Iterator Patroia

UML:



### Aldatutako kodearen lerro garrantzitsuenak:

ExtendedIterator interfaze berria sortu da, ohiko Iterator-ren ezaugarriak zabalduz eta bi norabidetan elementuak korritzeko aukera emanez. Interfaze honek metodo hauek barne hartzen ditu: previous() (aurreko elementura mugitzen da), hasPrevious() (aurreko elementua dagoen egiaztatzen du), goFirst() (lehenengo elementura joaten da), eta goLast() (azken elementura joaten da).

Ondoren, ExtendedIteratorCities klasea inplementatu da, ExtendedIterator interfazearekin bat egiten duena. Klase honek List<String> batean gordetako hiriak aurreraka eta atzeraka korritzeko aukera eskaintzen du. Hemen lerro garrantzitsuenen azalpena:

- **goFirst() eta goLast() metodoak:** goFirst() metodoak currentIndex -1 balioan ezartzen du, aurreranzko iterazioa lehen elementutik hasteko; goLast() metodoak, aldiz, currentIndex cities.size() balioan ezartzen du, atzeranzko iterazioa azken elementutik hasteko.
- **next() eta previous() metodoak:** next() metodoak currentIndex handitzen du eta hurrengo elementua itzultzen du, eta previous() metodoak currentIndex txikitzen du aurreko elementua itzultzeko. Bi metodoek muga egiaztapenak dituzte, NoSuchElementException salbuespena jaurtiz elementurik ez dagoenean.

```
public class ExtendedIteratorCities implements ExtendedIterator<String> {
    private List<String> cities;
    private int currentIndex;
    public ExtendedIteratorCities(List<String> cities) {
```

```

        this.cities = cities;
        this.currentIndex = -1; // Start before the first element for forward
traversal
    }
    @Override
    public boolean hasNext() {
        return currentIndex < cities.size() - 1;
    }
    @Override
    public String next() {
        if (!hasNext()) throw new NoSuchElementException("No next element.");
        currentIndex++;
        return cities.get(currentIndex);
    }
    @Override
    public String previous() {
        if (!hasPrevious()) throw new NoSuchElementException("No previous
element.");
        currentIndex--;
        return cities.get(currentIndex);
    }
    @Override
    public boolean hasPrevious() {
        return currentIndex > 0;
    }
    @Override
    public void goFirst() {
        currentIndex = -1;
    }
    @Override
    public void goLast() {
        currentIndex = cities.size();
    }
}

```

BLFacadeImplementation klasean, getDepartCitiesIterator() metodoa gehitu da, ExtendedIteratorCities objektu bat itzultzeko, datu-basean gordetako hiriak aurreraka eta atzeraka korritzeko aukera emanez.

```

public ExtendedIterator<String> getDepartCitiesIterator() {
    dbManager.open();
    List<String> departLocations = dbManager.getDepartCities();
    dbManager.close();
    return new ExtendedIteratorCities(departLocations);
}

```

## Exekuzioaren irudia:

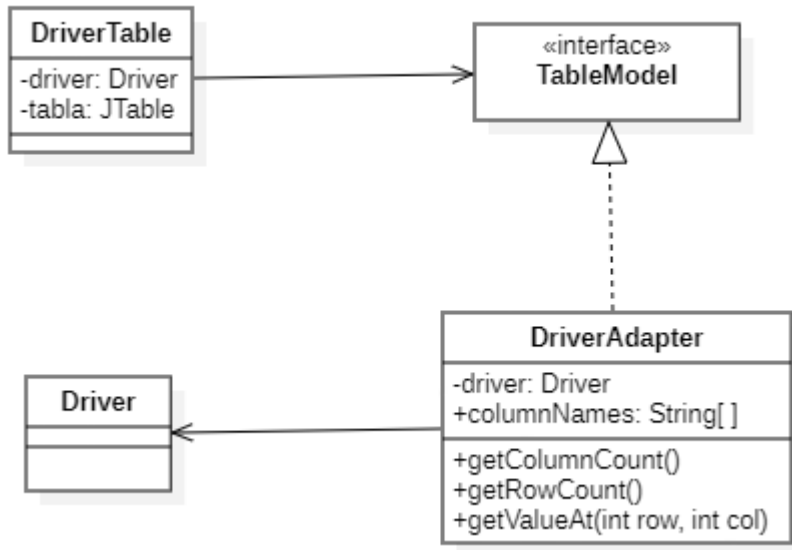
```
nov 06, 2024 6:18:31 P. M. businesslogic.BLFacadeImplementation
INFO: Creating BLFacadeImplementation instance
Read from config.xml:    businessLogicLocal=true          databa
DataAccess opened => isDatabaseLocal: true
DataAccess created => isDatabaseLocal: true isDatabaseInitializ
DataAccess closed
DataAccess opened => isDatabaseLocal: true
DataAccess closed
```

FROM	LAST	TO	FIRST
Madrid			
Iruña			
Gasteiz			
Donostia			
Barakaldo			

FROM	FIRST	TO	LAST
Barakaldo			
Donostia			
Gasteiz			
Iruña			
Madrid			

### 3. Adapter Patroia

UML:



Aldatutako kodearen lerro garrantzitsuenak:

DriverAdapter klasea sortu dugu Driver objektu batek dituen datuak JTable erabiltzen duen DriverTable interfaze grafikoan txertatu ahal izateko.

```
public class DriverAdapter extends AbstractTableModel {
    private Driver driver;
    private String[] columnNames = {"From", "To", "Date", "Places", "Price"};
    public DriverAdapter(Driver driver) {
        this.driver = driver;
    }
    @Override
    public int getColumnCount() {
        return columnNames.length;
    }
    @Override
    public int getRowCount() {
        // Driver klaseko bidaien lista erabili
        return driver.getCreatedRides().size();
    }
    @Override
    public Object getValueAt(int row, int col) {
        // Lerro jakin baten bidaiak lortu
        Ride ride = driver.getCreatedRides().get(row);
        // Zutabe bakoitzerako dagokion balioa bueltatu
        switch (col) {
            case 0: return ride.getFrom();
```



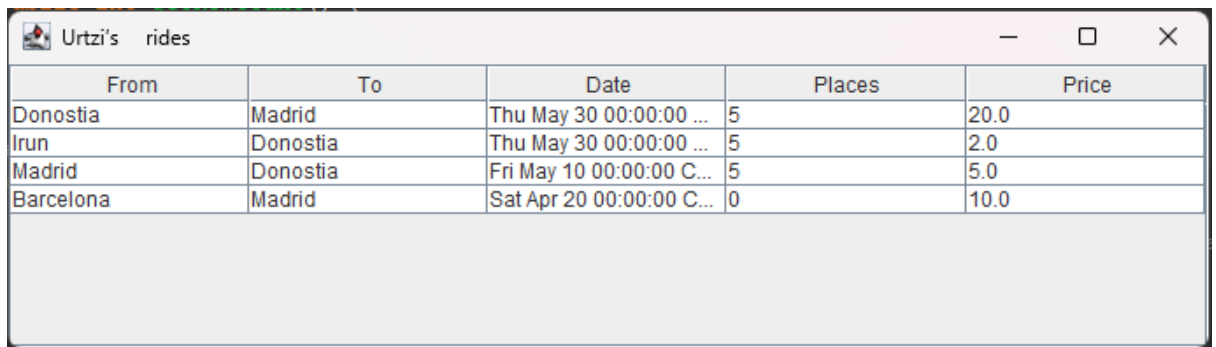
```

        case 1: return ride.getTo();
        case 2: return ride.getDate();
        case 3: return ride.getnPlaces();
        case 4: return ride.getPrice();
        default: return null;
    }
}
@Override
public String getColumnName(int col) {
    return columnNames[col];
}
}

```

getRowCount() metodoarekin lortuko ditugu gidariak sortutako bidai kopurua, eta getColumnCount() metodoak bidai batek gordetzen dituen datu kopurua itzuliko du, kopuru horrek adieraziko du taularen zutabe kopurua. getValueAt(int row, int col) metodoak itzuliko du taularen lauki bakoitzari dagokion balioa eta getColumnName() erabiliz zutabe bakoitzari izen bat esleituko diegu, columnNames aldagaian gordetakoak.

Exekuzioaren irudia:



From	To	Date	Places	Price
Donostia	Madrid	Thu May 30 00:00:00 ...	5	20.0
Irun	Donostia	Thu May 30 00:00:00 ...	5	2.0
Madrid	Donostia	Fri May 10 00:00:00 C...	5	5.0
Barcelona	Madrid	Sat Apr 20 00:00:00 C...	0	10.0