

# PATROIAK RIDES PROIEKTUAN

Egileak: Julen Oyarzabal, Hodei Jauregi eta Egoitz Ladron



1.Sarrera.....	2
2.Factory Method Patroia.....	2
3. Iterator Patroia .....	5
4. Adapter Patroia.....	8

# 1.Sarrera

Dokumentu honetan proiektuaren gainean aplikatutako diseinu patroiak azalduko dira. Proiektuaren kode guztia ikusteko ondorengo gordelekura sartu: <https://github.com/egoitzehu/Rides25>.

Kontuan eduki proiektu honek 10 entitate baino gehiago dauzkala datu basean. Hortaz arazoak sortzen dira patroiak exekutatzean, beharrezkoa delako kode bat edukitzea. Hori dela eta ariketa hauetan erabiltzen ez diren hainbat entitate kendu dira datu basetik. Horrela ez du errorerik ematen baina programa atal batzuk ez dute funtzionatuko, patroia hauen kasuan ez du eraginik.

## 2.Factory Method Patroia

IBLFactory izeneko interfaze bat sortuko dugu, non createBL izeneko metodo baten signatura definituko duen. Metodo honen helburua BLFacade motako objektua bat itzultzea izango da. Kodea honako da:

```
public interface IBLFactory {  
    public BLFacade createBL() throws Exception;  
}
```

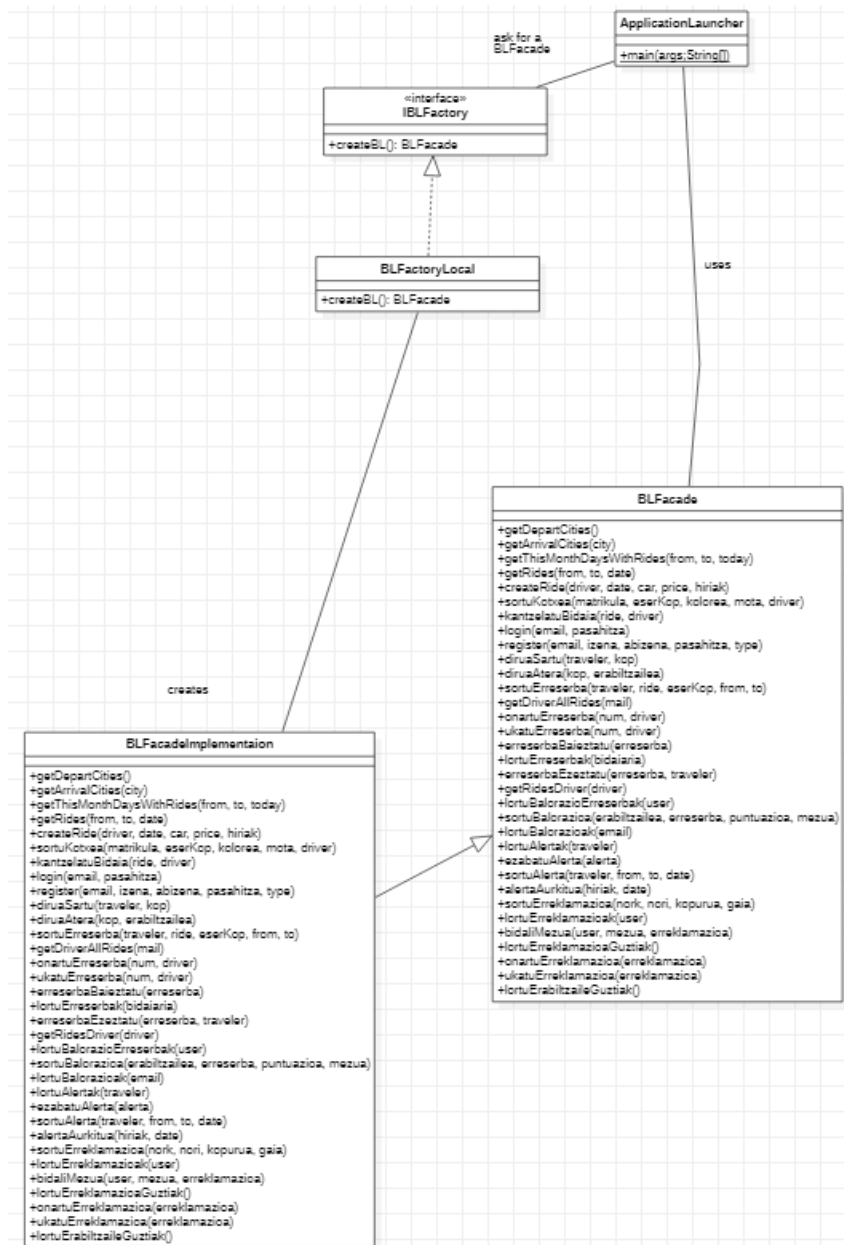
Ondoren interfaze hori implementatzen duten klase bat sortuko dugu, BLFactoryImplementation:

```
public class BLFactoryImplementation implements IBLFactory {  
    @Override  
    public BLFacade createBL(boolean isLocal) throws Exception {  
        if (isLocal) {  
            DataAccess da;  
            da = new DataAccess();  
            return new BLFacadeImplementation(da);  
        } else {  
            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);  
        }  
    }  
}
```

Ondoren ApplicationLauncher klasean config fitxategian dagoen datuaren arabera parametroa emandgo diogu. Ondoren sortutako instantziaren createBL metodoa sortuko dugu. Honakoa da kodea:

```
public static void main(String[] args) {  
    ConfigXML c=ConfigXML.getInstance();  
    Locale.setDefault(new Locale(c.getLocale()));  
  
    WelcomeGUI a=new WelcomeGUI();  
  
    try {  
        BLFacade appFacadeInterface;  
        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");  
        IBLFactory factory = new BLFactoryImplementation();  
  
        appFacadeInterface = factory.createBL(c.isBusinessLogicLocal());  
        WelcomeGUI.setBusinessLogic(appFacadeInterface);  
  
        a.setVisible(true);  
    }  
    catch (Exception e) {  
        System.out.println("Error in ApplicationLauncher: "+e.toString());  
    }  
}
```

Garatutako guztiaren diseinua ondorengo izango da:



BLFactory izeneko interfaze bat sortu dugu Factory Method patroia aplikatu nahi dugulako, horrela beste Factory mota bat sortu nahiko bagenuke BL sortzeko interfaze hori inplementatzen duen klase bat sortzearekin nahikoa da.

### 3. Iterator Patroia

ExtendedIterator interfazea inplementatzen duen klase bat definituko dugu, ExtendedIteratorImplementation. Klase honek ExtendedIterator eta Iterator interfazeetako metodoak inplementatuko ditu. Honako da kodea:

```
public class ExtendedIteratorImplementation<E> implements ExtendedIterator{
    List<E> cityList;

    int position = 0;

    public ExtendedIteratorImplementation(List<E> list) {
        this.cityList = list;
    }

    @Override
    public boolean hasNext() {
        return position<cityList.size();
    }

    @Override
    public E next() {
        E s = cityList.get(position);
        position++;
        return s;
    }

    @Override
    public E previous() {
        E s = cityList.get(position);
        position--;
        return s;
    }

    @Override
    public boolean hasPrevious() {
        return position>=0;
    }

    @Override
    public void goFirst() {
        position = 0;
    }

    @Override
    public void goLast() {
        position = cityList.size()-1;
    }
}
```

Iterator hori itzultzen duen metodoa honakoa izango da. Kontuan eduki proiektu honetan hiri anitz erabilpen kasua inplementatuta dagoela. Hori dela eta getDepartingCities metodoa erabili beharrean getStopCitiesNames metodoa erabiliko da, helburua berdina da baina tarteko hiriak ere kontuan ditu. Metodo hori erabiliz honakoa da kodea:

```
public ExtendedIterator<String> getDepartingCitiesIterator() {
    return new ExtendedIteratorImplementation<String>(this.getStopCitiesNames());
}
```

Ondoren klase hori erabiltzen duen metodo nagusi bat sortu dugu:

```
public static void main(String[] args) {
    ConfigXML c=ConfigXML.getInstance();

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

    try {

        BLFacade appFacadeInterface;
        IBLFactory factory = new BLFactoryImplementation();

        appFacadeInterface = factory.createBL(true);

        ExtendedIterator<String> i = appFacadeInterface.getDepartingCitiesIterator();
        String city;
        System.out.println("_____");
        System.out.println("FROM LAST TO FIRST");
        i.goLast();
        while(i.hasPrevious()) {
            city = i.previous();
            System.out.println(city);
        }
        System.out.println();
        System.out.println("_____");
        System.out.println("FROM FIRST TO LAST");
        i.goFirst();
        while(i.hasNext()) {
            city = i.next();
            System.out.println(city);
        }

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

}
}
```

Honakoa da kodearen irteera:

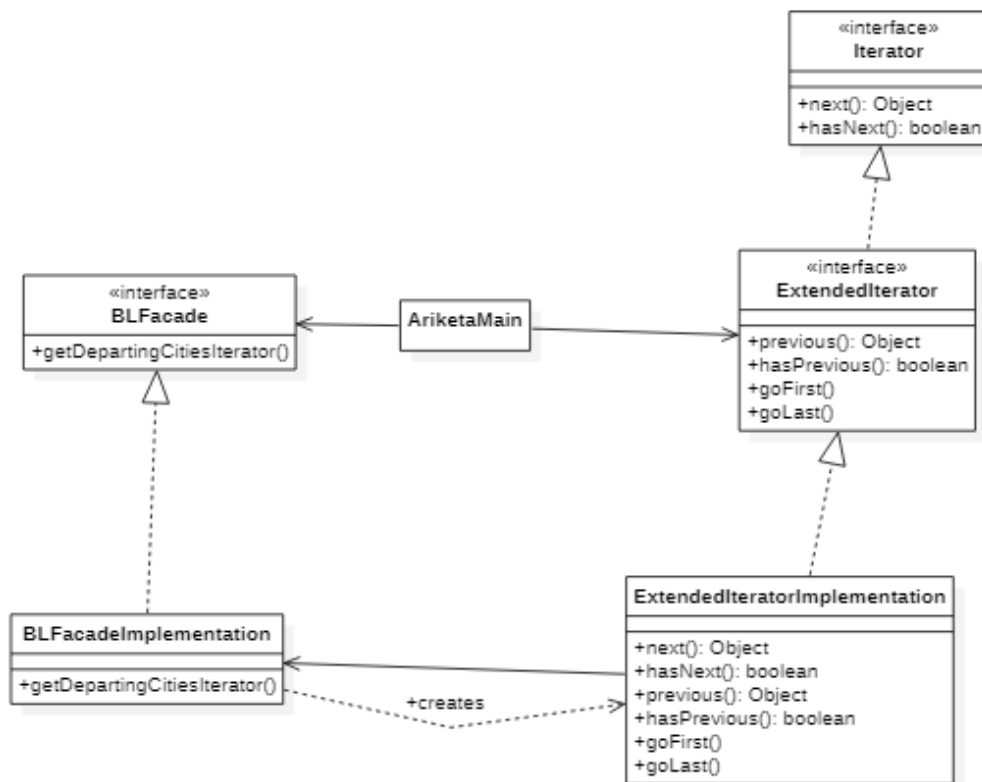
---

FROM LAST TO FIRST  
 Madrid  
 Barcelona  
 Irun  
 Bilbo

---

FROM FIRST TO LAST  
 Bilbo  
 Irun  
 Barcelona  
 Madrid

Honakoa da kodearen UML diagrama. Main funtzioa duen klaseak ExtendedIterator eta BLFacade interfazeak erabiltzen ditu. BLFacade implementatzen duen klase zehatzak ExtendedIterator bat sortu beharko du, zehazki bere implementazio bat. Diagrama honetan ez dira BLFacade interfazeko eta bere implementazioa den BLFacadeImplementation klasearen metodo guztiak agertzen, bakarrik iterator itzultzen duena.



## 4. Adapter Patroia

Patroia hau Driver klasearen gainean aplikatzean hainbat gauza hartu behar dira. Driver batek dituen bidaien datuak nahi ditugu, hasiera, amaiera, data, plaza kopurua eta prezioa. Baina gure proiektuaren kasuan hiri anitz erabilpen kasua du, ondorioz ez dago hasiera eta amaiera zehatzik. Honen aurrean erabaki dugu hasiera moduan lehenengo hiria hartzea eta amaiera moduan bigarren hiria.

Honakoa izango da DriverAdapter klasea, AbstractTableModel klasea hedatzen du. Sortutako klasearen helburua JTable klaseak behar dituen datuak ematea da.

```
public class DriverAdapter extends AbstractTableModel {
    List<Ride> rideList;

    public DriverAdapter(Driver d) {
        this.rideList = d.getRides();
    }

    @Override
    public int getRowCount() {
        return rideList.size();
    }

    @Override
    public int getColumnCount() {
        return 5;
    }

    @Override
    public String getColumnName(int columnIndex) {
        if(columnIndex==0) return "From";
        else if(columnIndex==1) return "To";
        else if(columnIndex==2) return "Date";
        else if(columnIndex==3) return "Places";
        else return "Price";
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if(columnIndex==0) return this.rideList.get(rowIndex).getGeldialdiak().get(0).getHiria();
        else if(columnIndex==1) return this.rideList.get(rowIndex).getGeldialdiak().get(1).getHiria();
        else if(columnIndex==2) return this.rideList.get(rowIndex).getDate();
        else if(columnIndex==3) return this.rideList.get(rowIndex).getnPlaces();
        else return this.rideList.get(rowIndex).prezioaKalkulatu(this.rideList.get(rowIndex).getGeldialdiak().get(0).getHiria(),
            this.rideList.get(rowIndex).getGeldialdiak().get(1).getHiria());
    }
}
```

Gure implementazioan bakarrik bidaien lista erabiltzen da hortaz ez dugu Driver instantzia guztia gorde bakarrik beharrezko datuak. Nahiko bagenuke Driver osoa gorde daiteke.

getColumnName metodoa ez da beharrezkoa taulak funtzionatu dezan, baina metodo hau definitzen badugu taula zutabeetan dagokion izena agertuko da.

Ondoren klase hori erabiliko dugu enuntziatuan emandako DriverTable klasean. Honakoa da kodea:

```

public class DriverTable extends JFrame{
    private Driver driver;
    private JTable tabla;
    public DriverTable(Driver driver){
        super(driver.getName()+"'s rides ");
        this.setBounds(100, 100, 700, 200);
        this.driver = driver;
        DriverAdapter adapt = new DriverAdapter(driver);
        tabla = new JTable(adapt);
        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
        //Creamos un JScrollPane y le agregamos la JTable
        JScrollPane scrollPane = new JScrollPane(tabla);
        //Agregamos el JScrollPane al contenedor
        getContentPane().add(scrollPane, BorderLayout.CENTER);
    }
}

```

Honakoa da lortuko den emaitza:

From	To	Date	Places	Price
Bilbo	Donostia	Fri Jun 20 00:00:00 C...	5.0	10.0
Irun	Barcelona	Fri Jun 20 00:00:00 C...	5.0	10.0

Adapter patroiarene UML diagrama honakoa izango da:

