

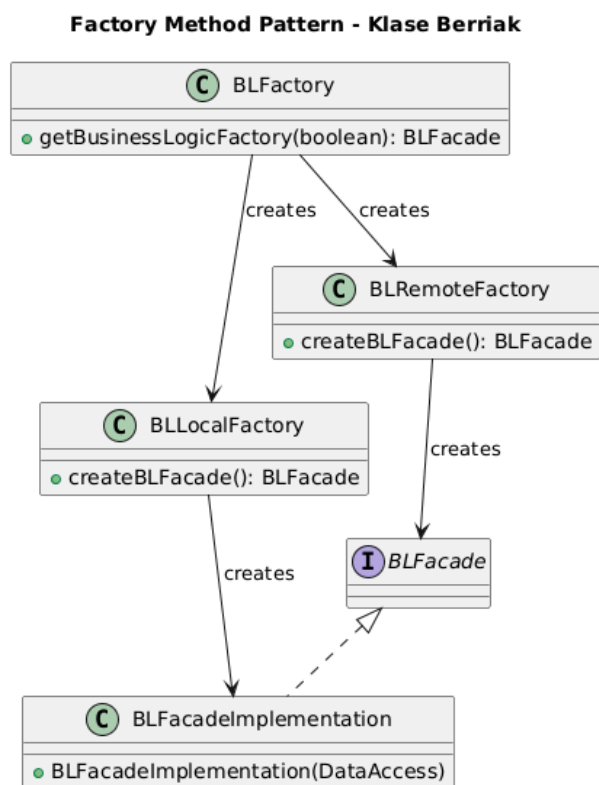
Diseinu patroia

Github esteka: <https://github.com/IkerSada/DiseinuPatroiak.git>

Talde Kideak: Iker Sada eta Qian Lizhang.

Simple Factory

UML diagrama hedatua:



Egindako aldaketak:

BLFactory (KLASE BERRIA)

```
public BLFacade getBusinessLogicFactory(boolean isLocal) {
    if (isLocal) {
        return new BLLocalFactory().createBLFacade();
    } else {
        return new BLRemoteFactory().createBLFacade();
    }
}
```

Zertarako? Erabaki zein BLFacade mota erabili (lokala edo urrunekoa).

BLLocalFactory (KLASE BERRIA)

```
public BLFacade createBLFacade() {  
    DataAccess da = new DataAccess();  
    return new BLFacadeImplementation(da);  
}
```

Zertarako? BLFacade lokalaren sorrera kapsulatuta.

BLRemoteFactory (KLASE BERRIA)

```
public BLFacade createBLFacade() {  
    try {  
        ConfigXML c = ConfigXML.getInstance();  
        String serviceName = "http://" + c.getBusinessLogicNode() + ":" +  
            c.getBusinessLogicPort() + "/ws/" +  
            c.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 (Exception e) {  
        throw new RuntimeException("Error creating remote BLFacade", e);  
    }  
}
```

Zertarako? Web zerbitzuaren konexio konplexua kapsulatuta.

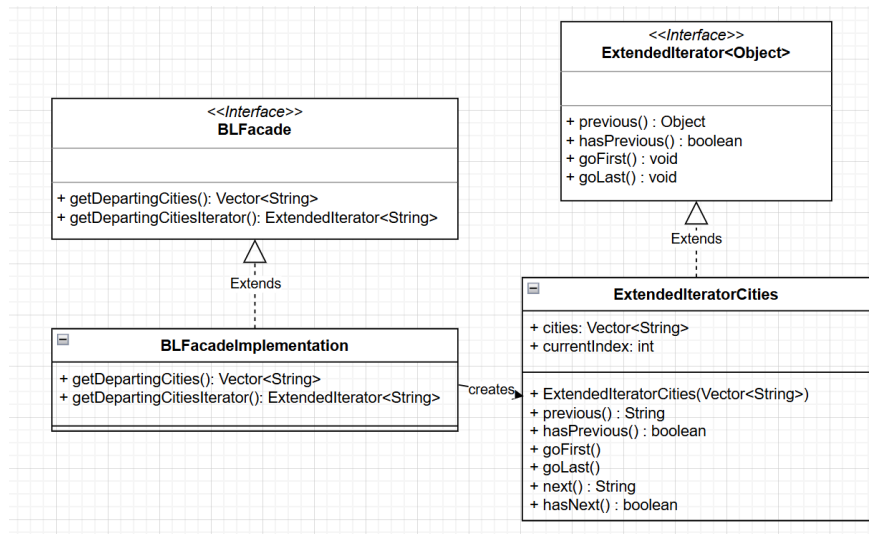
ApplicationLauncher (ALDATUTA)

```
BLFactory factory = new BLFactory();  
appFacadeInterface = factory.getBusinessLogicFactory(c.isBusinessLogicLocal());
```

Zertarako? Kodea sinplifikatu eta desakoplatu.

Iterator Patroia

UML diagrama hedatua:



Egindako aldaketak:

BLFacadeImplementation (ALDATUTA)

```
@WebMethod
public Vector<String> getDepartingCities() {
    dbManager.open();
    Vector<String> cities = new Vector<String>(dbManager.getDepartCities());
    dbManager.close();
    return cities;
}

@WebMethod
public ExtendedIterator<String> getDepartingCitiesIterator() {
    Vector<String> cities = this.getDepartingCities();
    return new ExtendedIteratorCities(cities);
}
```

BLFacade (ALDATUTA)

```
@WebMethod public Vector<String> getDepartingCities();

@WebMethod public ExtendedIterator<String> getDepartingCitiesIterator();
```

Vector erabilita dagoen getDepartCities() metodotik datuak lortzeko
ExtendedIteratorCities klasea sortzen da hirien lista pasatuz
Bi metodo berri interfazeaz eta implementazioan gehitu dira
WebService moduan eskura daude (@WebMethod anotazioa)

ExtendedIterator (KLASE BERRIA)

```
public interface ExtendedIterator<Object> extends Iterator<Object> {  
    public Object previous();  
    public boolean hasPrevious();  
    public void goFirst();  
    public void goLast();  
}
```

ExtendedIteratorCities (KLASE BERRIA)

```
public class ExtendedIteratorCities implements ExtendedIterator<String> {  
  
    public ExtendedIteratorCities(Vector<String> cities) {  
        (...) beste metodoak  
    }  
  
    @Override  
    public String next() {  
        if (!hasNext()) {  
            throw new NoSuchElementException("No more elements");  
        }  
        return cities.get(currentIndex++);  
    }  
  
    @Override  
    public String previous() {  
        if (!hasPrevious()) {  
            throw new NoSuchElementException("No previous elements");  
        }  
        return cities.get(--currentIndex);  
    }  
}
```

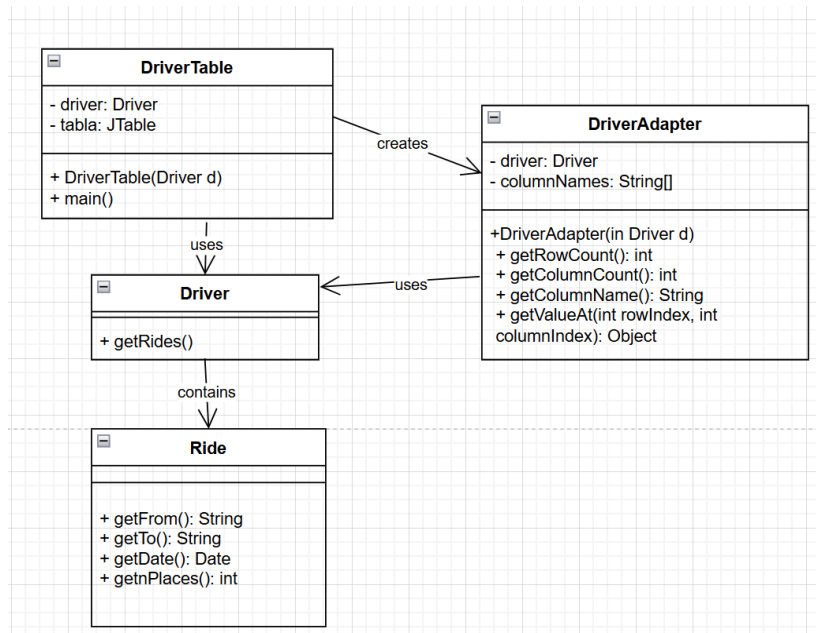
Bi klase berri sortu dira iteradorea implementatzeko. ExtendedIterator interfazeak 4 metodo gehitzen ditu. ExtendedIteratorCities klaseak interfazea implementatzen du. Vector-ean oinarrituta dago hirien lista maneiatzeko.

Exekuzioa:

```
Db initialized  
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true  
DataAccess closed  
Creating BLFacadeImplementation instance with DataAccess parameter  
DataAccess opened => isDatabaseLocal: true  
DataAccess closed  
  
-----  
FROM LAST TO FIRST  
Eibar  
Donostia  
Bilbo  
  
-----  
FROM FIRST TO LAST  
Bilbo  
Donostia  
Eibar
```

Adapter Patroia

UML diagrama hedatua:



Egindako aldaketak:

DriverAdapter(KLASE BERRIA)

```
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 getRowCount() {
        if (driver == null || driver.getRides() == null) {
            return 0;
        }
        return driver.getRides().size();
    }
    @Override
    public int getColumnCount() {
        return columnNames.length;
    }
    @Override
    public String getColumnName(int column) {
        return columnNames[column];
    }
    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        if (driver == null || driver.getRides() == null || rowIndex >= driver.getRides().size()) {
            return null;
        }
        List<Ride> rides = driver.getRides();
        Ride ride = rides.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return ride.getFrom();
            case 1:
                return ride.getTo();
            case 2:
                return ride.getDate();
            case 3:
                return ride.getnPlaces(); // Zure Ride klasean getnPlaces() dago
            case 4:
                return String.format("%.1f", ride.getPrice()); // Zure Ride klasean getPrice() dago
            default:
                return null;
        }
    }
}
```

DriverAdapter klasea sortu dugu, ariketak eskatzen duen taula aurkezteko. Gure **Driver** klasea ez dator bat **TableModel** interfazearekin. Hortaz, Adapter patroiak erabili dugu hori gauzako.

DriverAdapter-ek AbstractTableModel heredatzen du, TableModel interfazea inplementatzeko. Gero, Driver objektuaren bidaiak hartu eta taula formatura moldatzen ditu. Taularen zutabea definitu (**colNames**), gidariaren bidaia eskatzen duen informazioak erakusteko. Taula osatzeko behar diren metodoak inplementatu. **getValueAt()** metodoak switch bat erabiltzen du zutabe bakoitzarentzako datu egokia lortzeko

DriverTable (KLASE BERRIA)

```
public DriverTable(Driver driver){
    super(driver.getIzena()+"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);
}

public static void main(String[] args) {
    // the BL is local

    String driverEmail = "driver1@gmail.com";

    boolean isLocal = true;
    BLFacade bFacade = new
    BLFactory().getBusinessLogicFactory(isLocal);
    Driver d= (Driver) bFacade.erabiltzaileaBilatu(driverEmail);
    DriverTable dt=new DriverTable(d);
    dt.setVisible(true);
}
```

DriverTable klase berria sortu dugu, ariketak emandako kodearekin. DriverTable eta programa nagusian dagokion aldaketak egin ditugu gure proiektuan ditugun metodoa erabiliz. Programa nagusian datubasean dagoen gidari baten email-a hartu dugu, exekutzioa erakusteko.

Exekuzioa:

Aitor Fernandez's rides				
From	To	Date	Places	Price
Donostia	Bilbo	Sat Nov 15 00:00:00 C...	4	7.0
Donostia	Gasteiz	Thu Nov 06 00:00:00 ...	4	8.0
Bilbo	Donostia	Tue Nov 25 00:00:00 ...	4	4.0
Donostia	Iruña	Fri Nov 07 00:00:00 C...	4	8.0