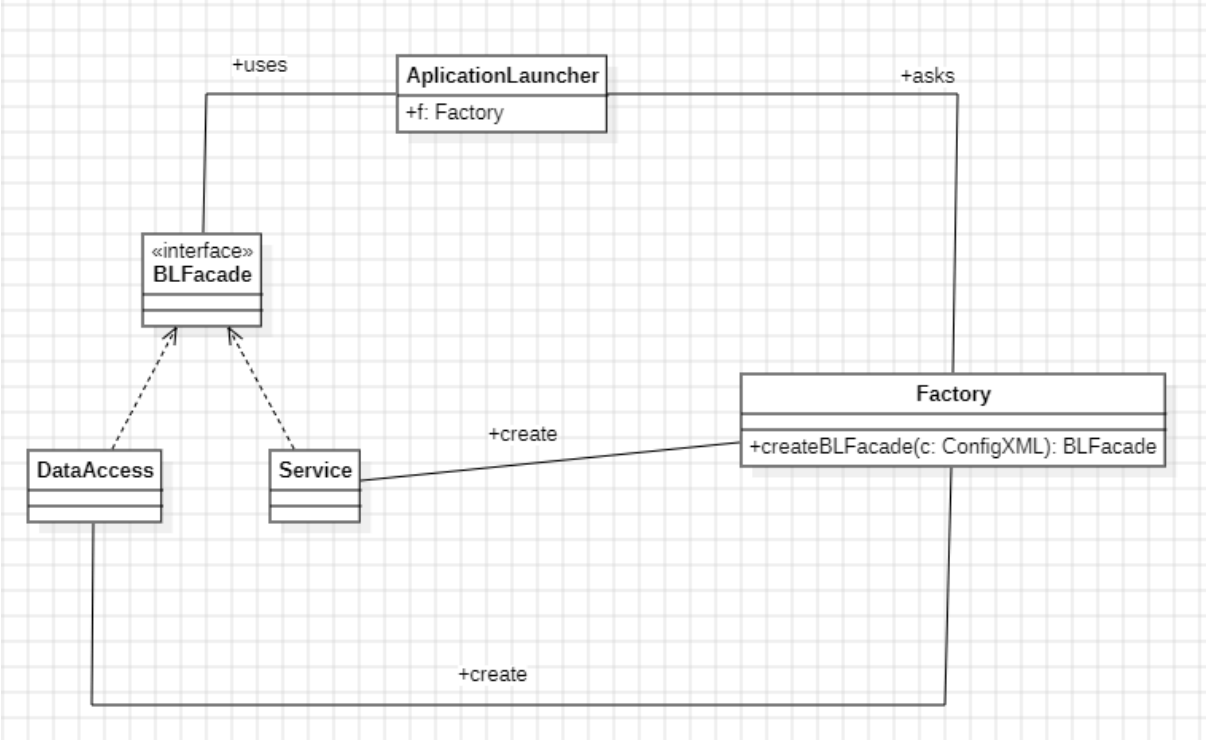
PatroiakProiektua24

Egileak: Ibai Oñatibia, Iker Galarraga eta Jon Olea

METHOD FACTORY

Method factory patroia rides proiektuan inplementatzeko honakoak izan dira kodean egindako aldaketak. Lehenik eta behin Factory.java klase berri bat sortu dut, bertan sortzeko ApplicationLauncher klaseak eskatutako BLFacade interfazea inplementatzen duten concrete product objektuak.
Kasu honetan product BLFacade izango litzateke eta concrete product objektuak isLocal atributuaren menpe dauden DataAccess (lokalean) eta Service (urrunean) dira creator klasea factory izanik.

Honako hau da ebazpenaren UML-a:



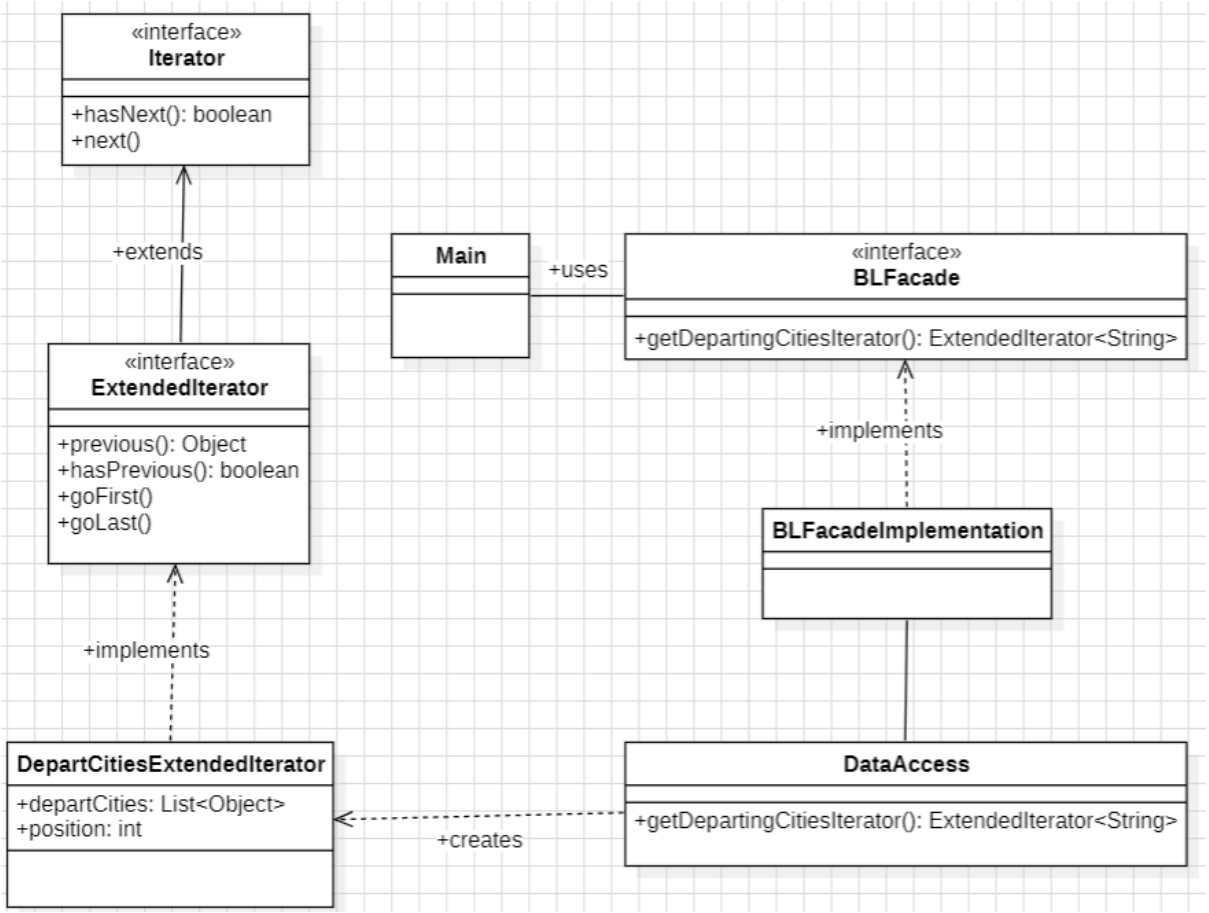
Kodea:

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

    Factory f = new Factory();

    ConfigXML c = ConfigXML.getInstance();

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

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

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

    try {

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

        appFacadeInterface = f.createBLFacade(c);

        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();

}
```

```java
public BLFacade createBLFacade(ConfigXML c) throws MalformedURLException {
       if (c.isBusinessLogicLocal()) {
           DataAccess da = new DataAccess();
           return new BLFacadeImplementation(da);
       } else {
           String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() +
           "/ws/" + c.getBusinessLogicName() + "?wsdl";

           URL 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);
       }
}
```

OHARRA: DataAccess eta Service klaseak ez dute inplementatzen BLFacade interfazea.

ITERATOR



Lehenengo, ExtendedIterator interfazea sortu, dut Iterator-ek dituen metodoak heredatuz, eta metodo berriak sortuz.
Ondoren, DeparCitiesExtendedIterator sortu dut, klase honek hirientzako lista bat eta posizio atributu bat du (Eraikitzailean hasieratzen dira), gainera ExtendedIterator-en metodo guztiak inplementatzen ditu.

DataAcces-en eta BLFacade-n getDepartCitiesIterator metodoa sortu dut, hirien ExtendedIterator bat itzultzeko.

Kodea:

```java
public class DepartCitiesExtendedIterator implements ExtendedIterator {

    List<Object> departCities;
    int position = 0;

    public DepartCitiesExtendedIterator(List<Object> cities) {
        this.departCities = cities;
    }

    //return the actual element and go to the previous
    public Object previous() {
        String city = (String) departCities.get(position);
        position--;
        return city;
    }

    //true if there is a previous element
    public boolean hasPrevious() {
        return position >= 0;
    }

    //It is placed in the first element
    public void goFirst() {
        this.position = 0;
    }

    //It is placed in the last element
    public void goLast() {
        this.position = departCities.size() - 1;
    }
```

```java
public boolean hasNext() {
    return position < departCities.size();
}

public Object next() {
    String city = (String) departCities.get(position);
    position++;
    return city;
}
```

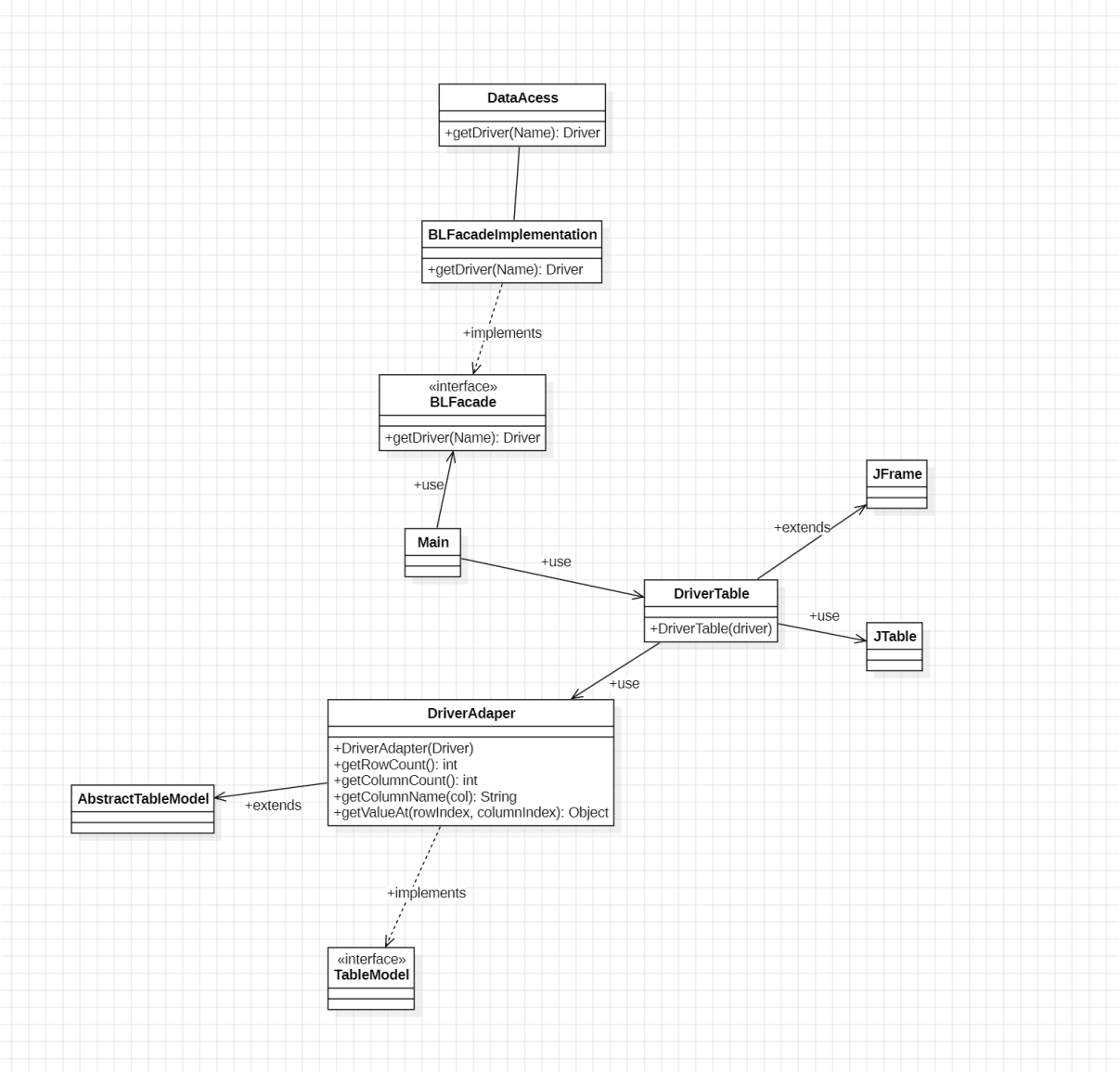
(DataAccess, Baita ere BLFacade-n eta BLFacadaImplementation-en)

```java
public ExtendedIterator<String> getDepartingCitiesIterator() {
    TypedQuery<Object> query = db.createQuery("SELECT DISTINCT r.from FROM Ride r ORDER BY r.from", Object.class);
    List<Object> cities = query.getResultList();
    return new DepartCitiesExtendedIterator(cities);
}
```

```java
public class Main {

    public static void main(String[] args) {
        //The BL is local
        ConfigXML conf = ConfigXML.getInstance();
        Factory f = new Factory();
        BLFacade appFacadeInterface = null;
        try {
            appFacadeInterface = f.createBLFacade(conf);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        ExtendedIterator<String> i = appFacadeInterface.getDepartingCitiesIterator();
        String c;
        System.out.println("_____");
        System.out.println("FROM LAST TO FIRST");
        i.goLast(); // Go to last element
        while (i.hasPrevious()) {
            c = i.previous();
            System.out.println(c);
        }
        System.out.println();
        System.out.println("_____");
        System.out.println("FROM FIRST TO LAST");
        i.goFirst(); // Go to first element
        while (i.hasNext()) {
            c = i.next();
            System.out.println(c);
        }
    }
```

```
_____
FROM LAST TO FIRST
Madrid
Irun
Donostia
Barcelona


_____
FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid
```

ADAPTER



Lehenik DriverTable izeneko klasea sortu dut, hau JFrame-ren azpiklasea da, soilik eraikitzailea dauka, bertan gidari bat eta JTable bat jasotzen ditu taula sortuz.
Ondoren DriverAdapter klaseak TableModel implementatzen du eta AbstractTableModel-ren azpiklasea da, honetan TableModel-ek dauzkan metodoak implementatzen dira getRowCount,getColumnCount etab.
Azkenik Main klasea sortu da taularen proba egin ahal izateko eta emaitza lortzeko.

Kodea:

```java
import java.awt.BorderLayout;

public class DriverTable extends JFrame {
    private Driver driver;
    private JTable tabla;

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

```java
import domain.Driver;

public class DriverAdapter extends AbstractTableModel implements TableModel {
    protected Driver driver;
    protected String[] columnNames = new String[] { "from", "to", "Date", "places","price" };
    public DriverAdapter(Driver driver) {
        this.driver = driver;
    }
    @Override
    public int getRowCount() {
        return driver.getCreatedRides().size();
    }

    @Override
    public int getColumnCount() {
        return columnNames.length;
    }

    public String getColumnName(int col) {
        return columnNames[col];
     }


    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Ride Ride = driver.getCreatedRides().get(rowIndex);
        switch (columnIndex) {
        case 0:
            return (Object) Ride.getFrom();

        case 1:
            return (Object) Ride.getTo();
        case 2:
            return (Object) Ride.getDate();
        case 3:
            return (Object) Ride.getnPlaces();
        case 4:
            return (Object) Ride.getPrice();
        }
        return null;
    }
}
```

```java
public class Main {

    public static void main(String[] args) {
        ConfigXML conf = ConfigXML.getInstance();
        Factory f = new Factory();
        BLFacade appFacadeInterface = null;
        try {
            appFacadeInterface = f.createBLFacade(conf);
        } catch (MalformedURLException e) {
            e.printStackTrace();
        }
        Driver d = appFacadeInterface.getDriver("Urtzi");
        DriverTable dt = new DriverTable(d);
        dt.setVisible(true);
    }

}
```