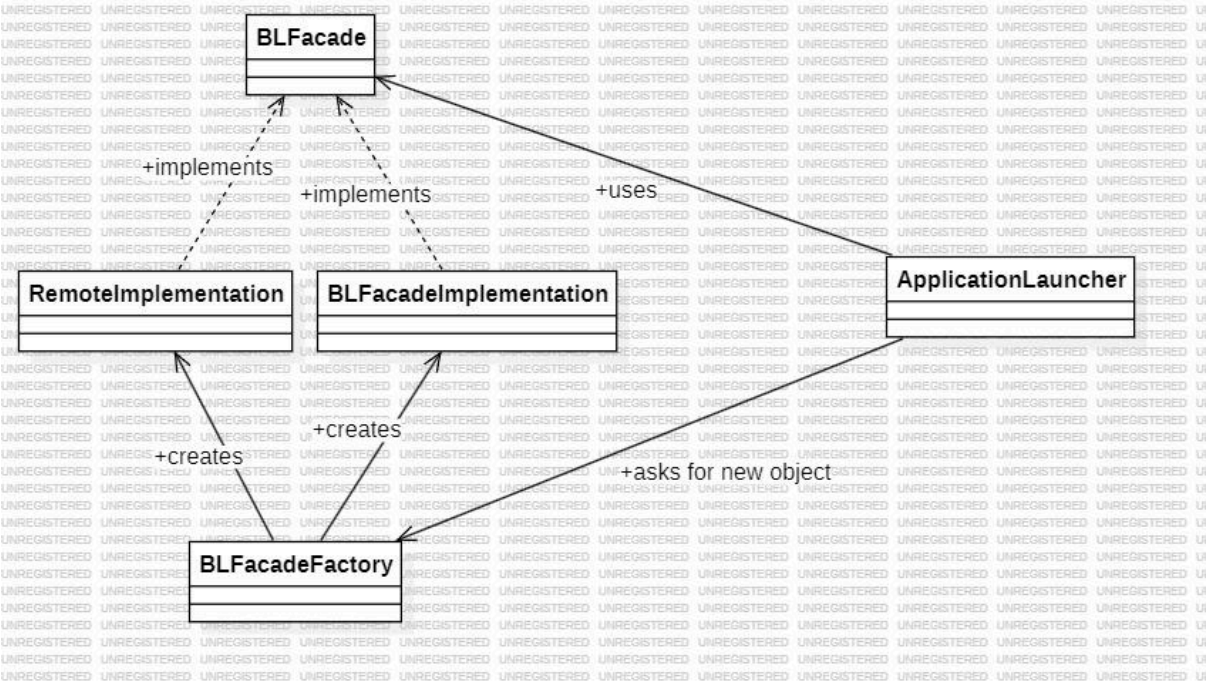
IS-2 / PROYECTO

PATRONES DE DISEÑO

GRUPO 5 / PABLO ALCOLEA, IKER GARCÍA, UNAI LEÓN

---

1. FACTORY



Para generalizar la creación de objetos BLFacade, hemos creado la clase BLFacadeFactory y definido el método createBLFacade. El método recibe un objeto ConfigXML para decidir el tipo de BLFacade que debe crear.

En el caso de que la base de datos sea local, crea el objeto BLFacadeImplementation pasándole un objeto DataAccess de forma local. En el caso contrario, utiliza la clase Service de Java para generar un acceso al servidor remoto donde se encuentra la implementación de BLFacade (representado en el UML como RemoteImplementation).

En la clase ApplicationLauncher, hemos sustituido el código que decidía qué implementación utilizar por una llamada al método createBLFacade de la factoría.

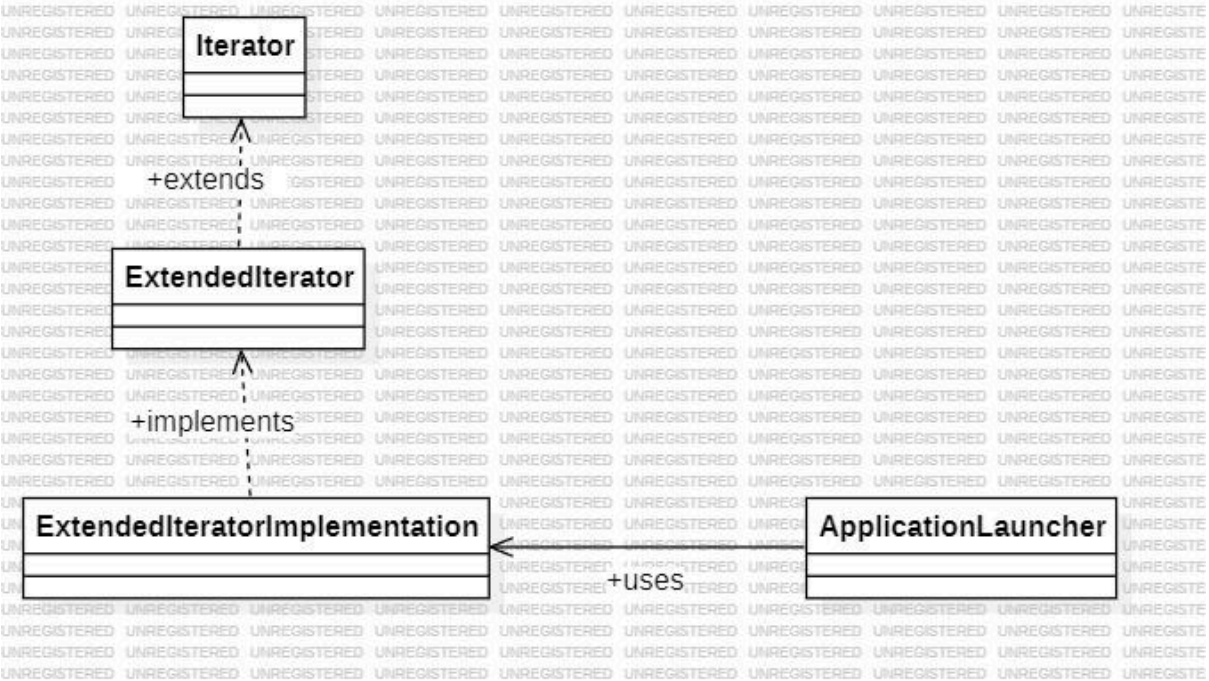
```
public static void main(String[] args) {

    ConfigXML c = ConfigXML.getInstance();

    System.out.println(c.getLocale());
    Locale.setDefault(new Locale(c.getLocale()));
    System.out.println("Locale: " + Locale.getDefault());

    BLFacade appFacadeInterface = new BLFacadeFactory().createBLFacade(c);

    if (appFacadeInterface != null) {
        MainGUI.setBussinessLogic(appFacadeInterface);
        MainGUI a = new MainGUI();
        a.setVisible(true);
    }
```

En el caso de que el método lance una excepción (probablemente por posibles fallos de conexión si la base de datos es remota), el objeto devuelto será null. Por ello, hemos añadido una comprobación para que no se cree la interfaz en ese caso.

El método createBLFacade de la clase BLFacadeFactory hace lo mismo que antes se hacía en la clase ApplicationLauncher. Se obtiene del objeto ConfigXML la información necesaria para decidir si la base de datos es local, y después se crea y devuelve el objeto correspondiente. En el caso de que sea local, se devuelve un objeto BLFacadeImplementation que implementa la interfaz; y si no es local, se devuelve un objeto de acceso al servicio remoto que también implementa la interfaz.

```
public BLFacade createBLFacade(ConfigXML c) {

    try {

        if (c.isBusinessLogicLocal()) {
            DataAccess da = new DataAccess();
            return new BLFacadeImplementation(da); // return "appFacadeInterface"
        }

        else {

            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); // return "appFacadeInterface"

        }

    } catch (Exception e) {

        // a.jLabelSelectOption.setText("Error: "+e.toString());
        // a.jLabelSelectOption.setForeground(Color.RED);

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

}
```

2. ITERATOR



Se ha creado la clase ExtendedIteratorImplementation que implementa la interfaz ExtendedIterator con los métodos para recorrer las estructuras en orden natural e inverso. A su vez, la interfaz ExtendedIterator extiende el iterator de java.util.

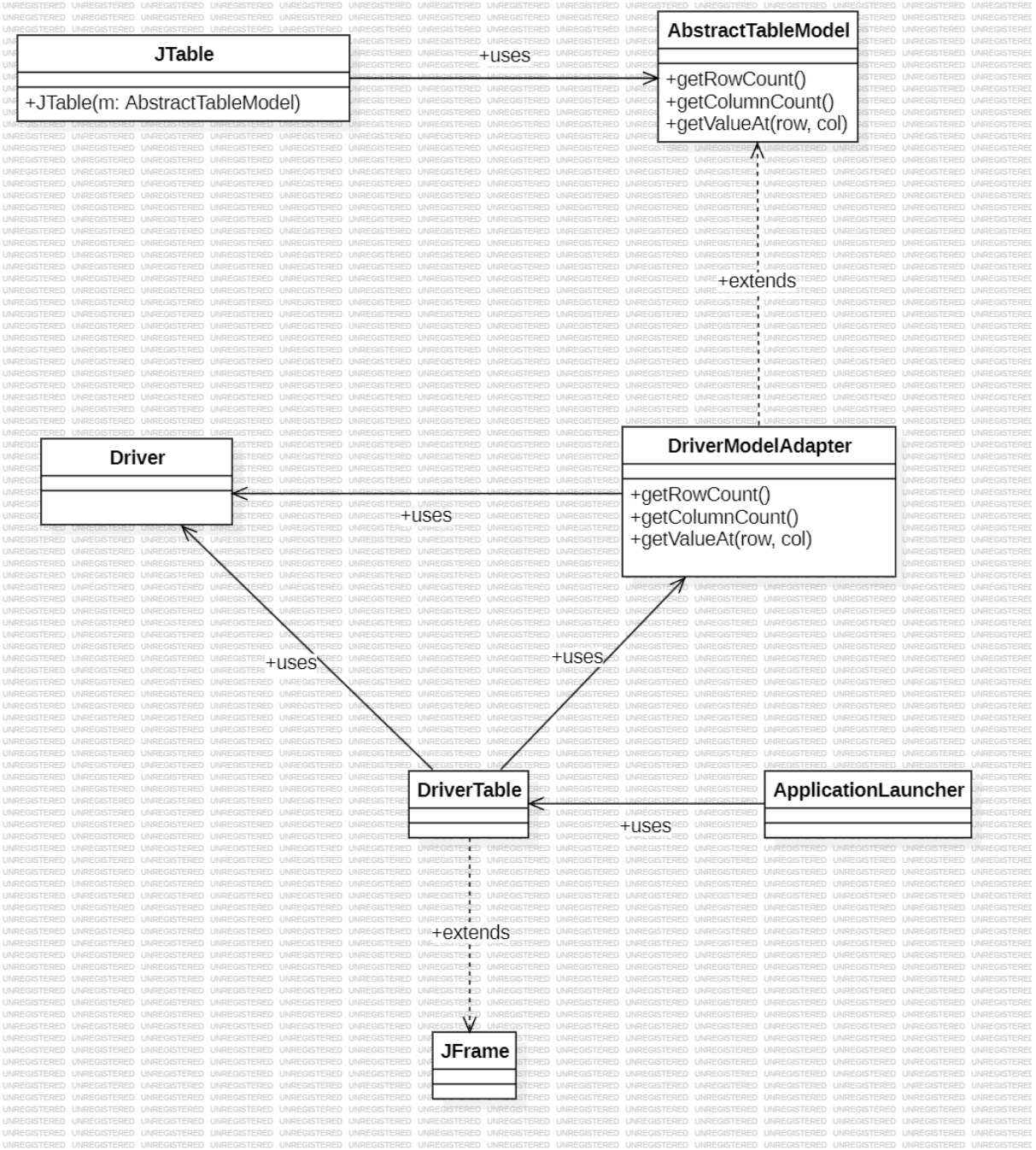
En el BLFacadeImplentation he añadido el método getDepartCitiesIterator(), que devuelve un objeto extended iterator llamando a la misma función de data access que devuelve una lista.

```
@WebMethod
public ExtendedIterator<String> getDepartCitiesIterator() {
    dbManager.open();

    List<String> departLocations = dbManager.getDepartCities();

    dbManager.close();

    return new ExtendedIteratorImplementation<String>(departLocations);

}
```

En la clase ApplicationLauncher, he realizado el main que imprime las ciudades en orden natural e inverso, aquí una imagen de la ejecución:

```
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true
DataAcess closed
Nov 04, 2024 3:16:56 PM businessLogic.BLFacadeImplementation <init>
INFO: Creating BLFacadeImplementation instance with DataAccess parameter
DataAccess opened => isDatabaseLocal: true
DataAcess closed
businessLogic.ExtendedIterator2@5e21e98f
_____________________
FROM    LAST    TO      FIRST
Toledo
Paris
Madrid
Donostia
Barcelona

_____________________
FROM    FIRST   TO      LAST
Barcelona
Donostia
Madrid
Paris
Toledo
```

3. ADAPTER



Se ha creado la clase DriverModelAdapter, que extiende la clase AbstractTableModel con los métodos que devuelven información relativa a la tabla. Además, se ha hecho Override sobre los métodos heredados:

```java
@Override
public int getRowCount() {
    return datalist.size();
}

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

@Override
public Object getValueAt(int rowIndex, int columnIndex) {
    Ride r = datalist.get(rowIndex);

    switch(columnIndex) {
    case 0:
        return r.getFrom();
    case 1:
        return r.getTo();
    case 2:
        return r.getDate();
    case 3:
        return r.getnPlaces();
    case 4:
        return r.getPrice();
    default:
        return null;
    }
}
```

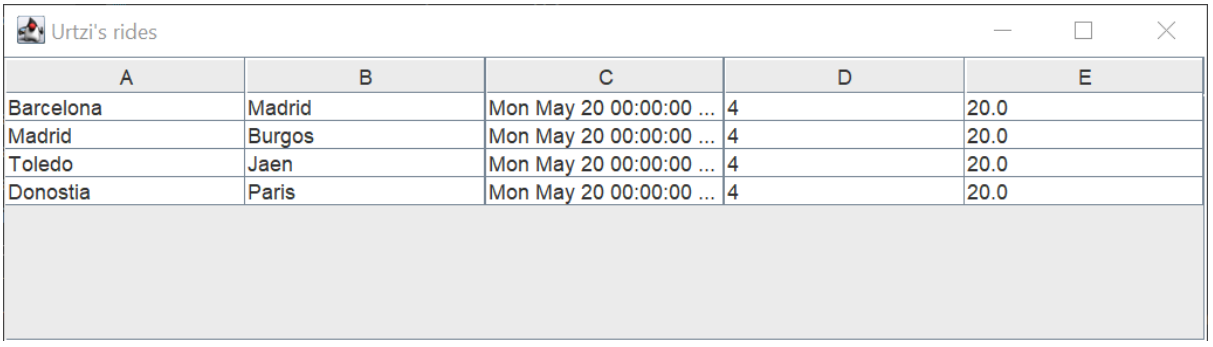
También se ha creado la clase DriverTable que extiende JFrame. En ella, se ha creado una constructora que genera la tabla que se mostrará en pantalla con los datos de los viajes realizados por el conductor deseado:

```java
public DriverTable(BLFacade blf, Driver driver) {

    super(driver.getUsername() + "'s rides ");
    this.blf = blf;
    this.setBounds(100, 100, 700, 200);
    this.driver = driver;
    DriverModelAdapter adapt = new DriverModelAdapter(blf,driver);
    tabla = new JTable(adapt);
    tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));
    JScrollPane scrollPane = new JScrollPane(tabla);
    getContentPane().add(scrollPane, BorderLayout.CENTER);
}
```

Por último, se ha modificado la clase ApplicationLauncher para que muestre por pantalla la tabla con los viajes una vez se ejecuta la aplicación con los viajes del Driver “Urtzi”:

```java
boolean isLocal = true;
c.setBusinessLogicLocal(isLocal); //He tenido que hacer que ConfigXML sea public y generar el setter.
BLFacade blf = new BLFacadeFactory().createBLFacade(c);
Driver d = blf.getDriver("Urtzi");
DriverTable dt = new DriverTable(blf, d);
dt.setVisible(true);
```

Esta es la ejecución resultante:



A	B	C	D	E
Barcelona	Madrid	Mon May 20 00:00:00 ...	4	20.0
Madrid	Burgos	Mon May 20 00:00:00 ...	4	20.0
Toledo	Jaen	Mon May 20 00:00:00 ...	4	20.0
Donostia	Paris	Mon May 20 00:00:00 ...	4	20.0