# Patroiak Proiektua

Ander Arruti, Xabier Saiz eta Adei Tamayo

# Factory Patroia

Factory patroia aplikatzeko hainbat aldaketa egin behar dira.

Hasteko BLFactory klasea sortu behar dugu, klase hau gure factorya izango da eta getBLFacade metodoa du barruan.
Metodo honek parametro batekin lokala edo remote den jasoko du , baina c.isBusinessLogicLocal()-ekin ere jaso ahalko genu metodo barruan zuzenean, parametrorik erabili gabe.
hemen instantzia nulua bada instantzia berria sortzera pasako da, eta lehen ApplicationLauncher metodoan zegoen kodeaz abiatuz lokala edo remotoa sortuko da.
Bukatzeko instantzia bueltatzen du.
hurrengoa da kodea:

```
public class BLFactory {
   private static BLFacade instance = null;  // Singleton BLFacade
   private BLFactory() {}  //pribatua zuzenean ez instantziatzeko
    public static BLFacade getBLFacade(boolean isLocal) {
       if (instance == null) {
           try {
            ConfigXML c = ConfigXML.getInstance();
              if (isLocal) {

                  DataAccess da = new DataAccess();
                  instance = new BLFacadeImplementation(da);  //  local
              } else {
                  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);
                  instance = service.getPort(BLFacade.class);  // remote
              }
           } catch (Exception e) {
              e.printStackTrace();
           }
       }
       return instance;
   }
}
```

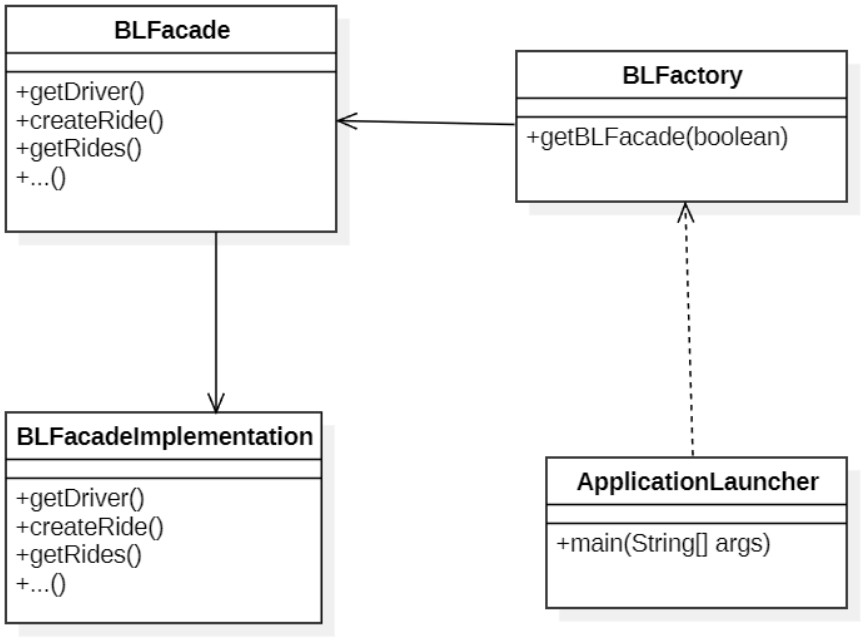
Honez gain beharrezkoa da ApplicationLauncher aldatzea egindako aldaketak erabiltzeko, honetarako Factory-an sortutako kodea kenduko dugu eta zuzenean honi deituko diogu.
Horrela geratuko litzake kodea:

```
public class ApplicationLauncher {
	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 {

			UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
			BLFacade appFacadeInterface =
BLFactory.getBLFacade(c.isBusinessLogicLocal());

			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());
			e.printStackTrace();
		}
		// a.pack();
	}
}
```

Azkenik hurrengoa da UML berriztua Factory patroiari dagokionez.


BLFacade
+getDriver()
+createRide()
+getRides()
+...()
BLFactory
+getBLFacade(boolean)
BLFacadeImplementation
+getDriver()
+createRide()
+getRides()
+...()
ApplicationLauncher
+main(String[] args)

## Iterator Patroia Inplementatzeko Kodearen Aldaketak

Lehenik eta behin, `Iterator` patroia gehitzeko hiru klase berri sortu dira:

- **ExtendedIterator**: `Iterator` interfazean oinarritutako interfaze bat, aurreraka eta atzeraka mugitzea ahalbidetzen dutena.
- **ExtendedIteratorImpl**: Interfaze honen inplementazioa, `List` edo `Vector` objektu bat iteratzeko aukera ematen du.
- **IteratorMain**: Iteradorearen funtzionamendua probatzeko programa, hirien zerrenda lehenik azkenetik lehenengora eta ondoren lehenetik azkenera inprimatzen duena.

### 1. ExtendedIterator

Hona hemen **ExtendedIterator** interfazean gehitutako metodoak eta bere deskribapena:

- `previous()`: Aurreko elementua itzultzen du.
- `hasPrevious()`: Aurreko elementurik badagoen egiaztatzen du.
- `goFirst()`: Lehen elementuan kokatzen da.
- `goLast()`: Azken elementuan kokatzen da.

**ExtendedIterator** interfazearen kodea honela geratzen da:

```
package iterator;
import java.util.Iterator;
public interface ExtendedIterator<Object> extends Iterator<Object> {
   Object previous();
   boolean hasPrevious();
   void goFirst();
   void goLast();
}
```

### 2. ExtendedIteratorImpl

**ExtendedIteratorImpl** klaseak `ExtendedIterator` interfazearekin bat egiten du, `Vector` baten gainean iteratuz. Kode honek beharrezko metodoak definitzen ditu iterazioa aurrera eta atzera egiteko, baita posizioaren kudeaketa egiteko ere (`goFirst` eta `goLast` metodoak).

Kasu honetan, hurrengo aldaketak nabarmentzen dira:

- `hasNext()`: Oraindik elementurik geratzen den itzultzen du.
- `next()`: Hurrengo elementua itzultzen du eta posizioa handitzen du.
- `previous()`: Aurreko elementua itzultzen du eta posizioa txikitzen du.
- `goFirst()` eta `goLast()`: Posizioa hasieran edo amaieran kokatzen dute.

**ExtendedIteratorImpl** kodea honela geratzen da:

```
package iterator;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;
public class ExtendedIteratorImpl<T> implements ExtendedIterator<T> {
  private List<T> elements;
  private int currentPosition;
  public ExtendedIteratorImpl(Vector<T> elements) {
     this.elements = elements;
     this.currentPosition = 0;
  }
  @Override
  public boolean hasNext() {
     return currentPosition < elements.size();
  }
  @Override
  public T next() {
     return elements.get(currentPosition++);
  }
  @Override
  public T previous() {
     return elements.get(--currentPosition);
  }
  @Override
  public boolean hasPrevious() {
     return currentPosition > 0;
  }
  @Override
  public void goFirst() {
     currentPosition = 0;
  }
  @Override
  public void goLast() {
     currentPosition = elements.size();
  }
}
```

### 3. IteratorMain

**IteratorMain** programak BLFacade bidez hirien zerrenda lortzen du eta bi norabidetan korritzen du zerrenda hori:

- **Atzetik aurrera** `goLast()` erabiliz, eta `previous()` metodoarekin atzeraka eginez.
- **Aurretik atzera** `goFirst()` erabiliz eta `next()` metodoarekin aurrera eginez.

**IteratorMain** klasearen kodea honela geratzen da:

```
package iterator;
import businesslogic.BLFacadeImplementation;
public class IteratorMain {
  public static void main(String[] args) {
        BLFacadeImplementation facade = new BLFacadeImplementation();
    ExtendedIterator<String> iterator = facade.getDepartingCitiesIterator();

    System.out.println("_____");
    System.out.println("FROM LAST TO FIRST");
    iterator.goLast();
    while (iterator.hasPrevious()) {
       String city = iterator.previous();
       System.out.println(city);
    }
    System.out.println();
    System.out.println("_____");
    System.out.println("FROM FIRST TO LAST");
    iterator.goFirst();
    while (iterator.hasNext()) {
       String city = iterator.next();
       System.out.println(city);
    }
  }
}
```

### 4. BLFacade Klasean Metodorik Berria: getDepartingCitiesIterator()

Azkenik, **BLFacade** klasean metodo berri bat gehitu da: `getDepartingCitiesIterator()`. Metodo honek `getDepartCities()` erabiliz hirien zerrenda eskuratzen du, eta `ExtendedIteratorImpl` erabiliz iteradore bat sortzen du. Hau **ExtendedIterator** interfazean oinarrituta dago, eta `List` bat `Vector` bihurtuz `ExtendedIteratorImpl`-i pasatzen zaio.

Metodoaren kodea honako hau da:

```
/**
         * {@inheritDoc}
         */
        @WebMethod
        public ExtendedIterator<String> getDepartingCitiesIterator() {
           List<String> citiesList = getDepartCities();
           Vector<String> cities = new Vector<>(citiesList);
           return new ExtendedIteratorImpl<>(cities);
        }
```

## Ondorioa

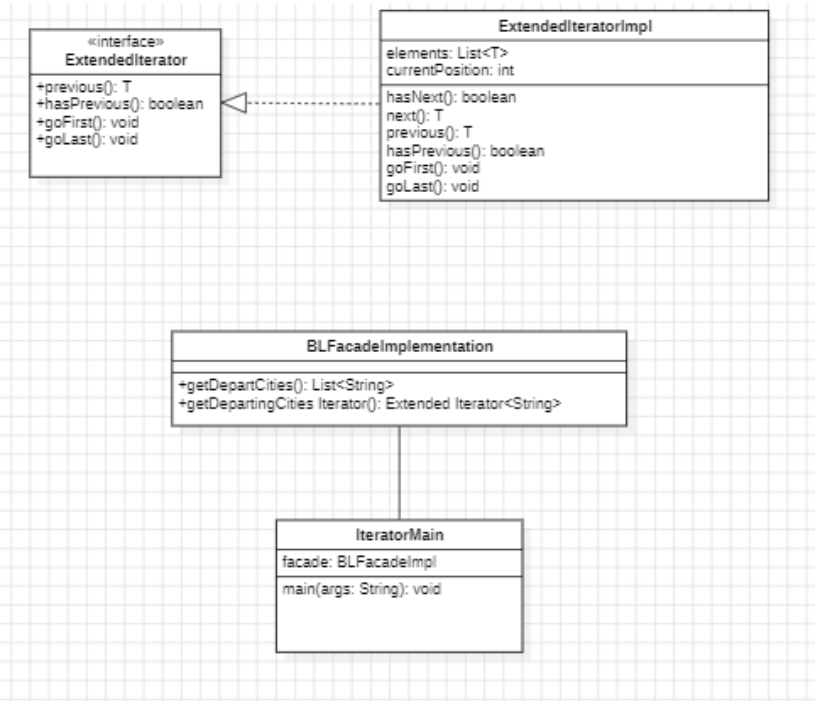
Aldaketa hauei esker, iteradorea bi norabidetan mugitzeko aukera izan dugu, `ExtendedIterator` interfaze eta `ExtendedIteratorImpl` klaseen bidez. Horrela, **IteratorMain** programan hiriak aurreraka eta atzeraka inprima ditzakegu, `goLast()` eta `goFirst()` metodoak erabiliz, eta `previous()` eta `next()` metodoen bidez iterazioa kudeatuz.

# Programeraren funtzionamenduaren irudia:

```
Read from config.xml:    businessLogicLocal=true        databaseLocal=true      dataBaseInitialized=true
File deleted
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true
DataAcess closed
DataAccess opened => isDatabaseLocal: true
Database connection opened.
Retrieved cities: [Donostia, Irun, Madrid]
DataAcess closed
Database connection closed.
_____________________
FROM LAST TO FIRST
Madrid
Irun
Donostia

_____________________
FROM FIRST TO LAST
Donostia
Irun
Madrid
```

### Implementazioaren UMLa



# Adapter patroia

Adapter patroiari hasiera emateko, klase berri batzuk sortu behar ditugu, adapter klase bat (DriverAdapter) eta taula sortzeko beste klase bat (DriverTable).

DriverAdapter klasea sortzeko, driver-a eta bere bidaiak gordeko ditugu atributuetan, eta bidai kopurua edota zutabe kopurua lortzeko metodoak definitu behar izan dira.
Klase hau egin behar izan dugu geroago taularatzean erabiltzeko.

```
public class DriverAdapter extends AbstractTableModel {
    private Driver driver;
    private List<Ride> rides; // Driveraren bidai lista

    public DriverAdapter(Driver driver) {
        this.driver = driver;
        this.rides = driver.getCreatedRides(); // Bidai lista bueltatu
    }

    @Override
    public int getRowCount() {
        return rides.size(); // Bidai kopurua
    }

    @Override
    public int getColumnCount() {
        return 3; // Erakutsi nahi diren zutabe kopurua
    }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        Ride ride = rides.get(rowIndex);
        switch (columnIndex) {
            case 0:
                return ride.getFrom();
            case 1:
                return ride.getTo();
            case 2:
                return ride.getDate();
            default:
                return null;
        }
    }

    @Override
    public String getColumnName(int column) {
        switch (column) {
```

```
            case 0:
                return "Origin";
            case 1:
                return "Destination";
            case 2:
                return "Date";
            default:
                return "";
        }
    }
}
```

Proba egiteko, enuntziatuan eskuratutako DriverTable erabiliko dugu eta main programa bat eginez, ikusi dezakegu Urtzi gidariaren bidai guztiak lortu ditugula taula batean, adapter patroia inplementatuz.


Bukatzeko, hau izan da egin den adapter patroi hau errepresentatzen duen UML diseinua: