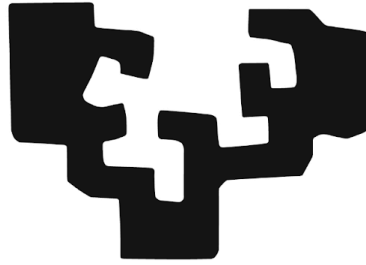


eman ta zabal zazu



Universidad
del País Vasco

Euskal Herriko
Unibertsitatea

Patrones de diseño proyecto

Profesor:
Jon Iturrioz

Ingeniería del software II

Presentan los alumnos:

Daniel Gutiérrez Gómez

Hugo Muñoz Rodríguez

11/11/2024

PROYECTO RIDES

PATRONES DE DISEÑO

Trabajo en grupo (0.5 puntos)

Introducción

En este documento se presentan una serie de modificaciones a vuestro proyecto de reserva de viajes de coche que implica el re-diseño de la aplicación. Deberéis plantearos cómo realizar cada modificación tanto a nivel de diseño como sus implicaciones en el código. Con este documento se adjunta otro describe de manera general los patrones a utilizar en cada uno de los ejercicios propuestos.

IMPORTANTE. Para inicializar la BD, el fichero config.xml ubicado en la carpeta resources tiene que tener el valor `initialize="true"`.

1. Patrón Factory Method

La aplicación está diseñada en una arquitectura de 3 niveles, donde la presentación puede acceder a un objeto de la lógica de negocio ubicado bien de forma local o bien a un objeto distribuido a través de servicios web. En ambos casos, si bien los objetos comparten la misma interfaz (BLFacade), la implementación varía mucho.

En la aplicación actual, la clase ApplicationLauncher, decide cual de las 2 implementaciones utilizar. Concretamente en el método main(), teniendo en cuenta el valor obtenido en el método isBusinessLogicLocal de la clase ConfigXML, se genera el objeto adecuado a cada circunstancia y se asigna a la variable appFacadeInterface.

Modifica la aplicación para que la obtención del objeto de la lógica de negocio se centralice en un objeto factoría, y sean las clases de la presentación las que decidan cuál de las implementaciones de la lógica de negocio utilizar. Diseña e implementa la solución indicando qué clases juegan el papel de Creator, Product y ConcreteProduct. *NOTA: El ejemplo main que se encuentra al final hace uso de la factoría y os puede ayudar a entender el uso de la factoría solicitada.*

Podemos decir que la aplicación es Mockito-friendly, ya que en la clase factoría podríamos añadir otro **ConcreteProduct** que fuese un objeto mock de clase BLFacade.

Identificamos, en el diseño que existe un **producto** que en este caso es la **fachada**, dos **productos concretos**, uno es la **fachada** corriendo en la **web** y el otro de manera **local**. De igual manera, identificamos que el **Creator o cliente**, es el **ApplicationLauncherGUI.java**.

El diseño refactorizado se encuentra en el [siguiente link](#).

Después de realizar el diseño, pasamos a su implementación, abajo encontramos cada una de las implementaciones del diseño refactorizado.

```

LocalBusinessLogic.java ×
1 package factoryFacade;
2
3 import businessLogic.BLFacadeImplementation;
4
5
6 public class LocalBusinessLogic{
7     public DataAccess dataAccess;
8     public BLFacadeImplementation BLFacadeImpl;
9
10    public void createLocalBusinessLogic() {
11        try {
12            this.dataAccess = new DataAccess();
13            this.BLFacadeImpl = new BLFacadeImplementation(this.dataAccess);
14        }
15        catch (Exception e){
16            System.out.println("Error in ApplicationLauncher: " + e.toString());
17        }
18    }
19    public BLFacadeImplementation getBLFacade(){
20        return this.BLFacadeImpl;
21    }
22 }

```

**Figura 1. Implementación del producto concreto
Fachada local**

```

WebBusinessLogic.java ×
1 package factoryFacade;
2
3 import java.net.URL;
4
5
6 public class WebBusinessLogic{
7     public ConfigXML c;
8     public BLFacade BLFacadeInterface;
9     public WebBusinessLogic(ConfigXML c){
10         this.c=ConfigXML.getInstance();
11     }
12     public BLFacade getBLFacade(){
13         return this.BLFacadeInterface;
14     }
15     public void createWebBusinessLogic(){
16         try {
17             String serviceName = "http://" + c.getBusinessLogicNode() + ":" + c.getBusinessLogicPort() + "/ws/"
18                 + c.getBusinessLogicName() + "?wsdl";
19
20             URL url = new URL(serviceName);
21
22             // 1st argument refers to wsdl document above
23             // 2nd argument is service name, refer to wsdl document above
24             QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
25
26             Service service = Service.create(url, qname);
27
28             this.BLFacadeInterface = service.getPort(BLFacade.class);
29         }
30         catch (Exception e) {
31             System.out.println("Error in ApplicationLauncher: " + e.toString());
32         }
33     }
34 }

```

**Figura 2. Implementación del producto concreto
FachadaWeb**

```

1 package factoryFacade;
2
3 import java.util.Locale;
4
5
6 public class FactoryFacade{
7     public ConfigXML c;
8     public MainGUI a;
9
10    public FactoryFacade (MainGUI a) {
11        this.c=ConfigXML.getInstance();
12        Locale.setDefault(new Locale(c.getLocale()));
13        this.a=new MainGUI();
14        a.setVisible(true);
15    }
16    public BLFacade createBLFacade(String BLFacadeType){
17        BLFacade appFacadeInterface;
18        try {
19            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
20
21            if (BLFacadeType.equals("local")) {
22                LocalBusinessLogic lbl = new LocalBusinessLogic();
23                lbl.createLocalBusinessLogic();
24                appFacadeInterface = lbl.getBLFacade();
25            }
26
27            else { // If remote
28                WebBusinessLogic wbl = new WebBusinessLogic(c);
29                wbl.createWebBusinessLogic();
30                appFacadeInterface = wbl.getBLFacade();
31            }
32
33            MainGUI.setBusinessLogic(appFacadeInterface);
34            MainGUI a = new MainGUI();
35            a.setVisible(true);
36
37            return appFacadeInterface;
38        }
39    }
40 }

```

```

44     } catch (Exception e) {
45         // a.jLabelSelectOption.setText("Error: "+e.toString());
46         // a.jLabelSelectOption.setForeground(Color.RED);
47
48         System.out.println("Error in ApplicationLauncher: " + e.toString());
49     }
50     // a.pack();
51     return null;
52 }
53 }

```

Figuras 3-4. Implementación de clase FactoryFaca

```

Tasks Console × Coverage SonarLint Report Type Hierarchy SonarLint Rule Description Search Call Hierarchy
<terminated> ApplicationLauncher [Java Application] C:\Users\PC\p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_22.0.2.v2024
Read from config.xml: businessLogicLocal=true databaseLocal=true dataBaseInitialized=false
DataAccess opened => isDatabaseLocal: true
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: false
DataAccess closed
Oct 26, 2024 12:52:07 PM businessLogic.BLFacadeImplementation <init>
INFO: Creating BLFacadeImplementation instance with DataAccess parameter
businessLogic.BLFacadeImplementation@9ebe38b
Locale: es
Oct 26, 2024 12:52:24 PM gui.MainGUI$1 actionPerformed
INFO: Locale: en
Locale: es
Locale: eus
Locale: es
DataAccess opened => isDatabaseLocal: true
DataAccess closed
DataAccess opened => isDatabaseLocal: true
DataAccess closed
DataAccess opened => isDatabaseLocal: true
>> DataAccess: getThisMonthActiveRideDates
DataAccess closed
DataAccess opened => isDatabaseLocal: true
DataAccess closed
DataAccess opened => isDatabaseLocal: true
>> DataAccess: getThisMonthActiveRideDates
DataAccess closed
DataAccess opened => isDatabaseLocal: true
DataAccess closed
DataAccess opened => isDatabaseLocal: true
>> DataAccess: getThisMonthActiveRideDates
DataAccess closed
DataAccess opened => isDatabaseLocal: true
>> DataAccess: getThisMonthActiveRideDates
DataAccess closed

```

Figuras 5. Ejecución de clase FactoryFacade

2. Patrón Iterator

Queremos implementar otro método de la clase BLFacadeImplementation que haga las tareas de

```
public Vector<String> getDepartingCities()
```

pero que tenga la siguiente signatura:

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

de manera que, en vez de devolvernos un Vector de ciudades nos devuelva un Iterator de ciudades. Sin embargo, este nuevo "Iterator extendido", además de poder recorrer los elementos de la forma tradicional (secuencialmente hacia adelante), puede ir secuencialmente hacia atrás, o bien posicionarse en el primer o último elemento. La signatura de la interfaz ExtendedIterator es la siguiente:

```

public interface ExtendedIterator<Object> extends Iterator<Object>
{ //return the actual element and go to the previous
    public Object previous();

    //true if ther is a previous element
    public boolean hasPrevious();

    //It is placed in the first element
    public void goFirst();

    // It is placed in the last element
    public void goLast();
}

```

Los dos métodos deben coexistir. Evita la repetición de código excesiva y reutiliza al máximo el código ya que los dos métodos tienen una parte común.

Se pide implementar el Iterator extendido, y realizar un programa similar al del ejemplo que visualice todos los eventos en orden creciente y decreciente.

Un ejemplo de programa principal (adaptado a los cambios ya realizados del apartado 1) donde recorremos todos los eventos en secuencia inversa, y a continuación en secuencia tradicional sería:

```
public static void main(String[] args) {
    // the BL is local
    boolean isLocal = true;
    BLFacade blFacade = new BLFactory().getBusinessLogicFactory(isLocal);

    ExtendedIterator<String> i =
    blFacade.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);
    }
}
```

El resultado que se debería obtener es:



```
<terminated> Application Launcher (4) [Java Application] S:\library\java\java\VirtualMachines\jdk-11.0.8-jdk\Contents\Home\bin\java [Oct 22, 2024, 12:47:36 PM - 12:47:37 PM]
Read from config.xml:    businessLogicLocal=true    databaseLocal=true    dataBaseInitialized=true
File deleted
DataAccess opened => isDatabaseLocal: true
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
Madrid
Irun
Donostia
Barcelona

FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid
```

Después de realizar un diseño adaptado a la parte del Iterator extendido con la clase ExtendedIterator, hemos llegado al siguiente diseño, donde se muestra cómo es que hemos aplicado el patrón de diseño Iterator a cada una de las funcionalidades necesarias para lograr un recorrido en inverso y de manera incremental.

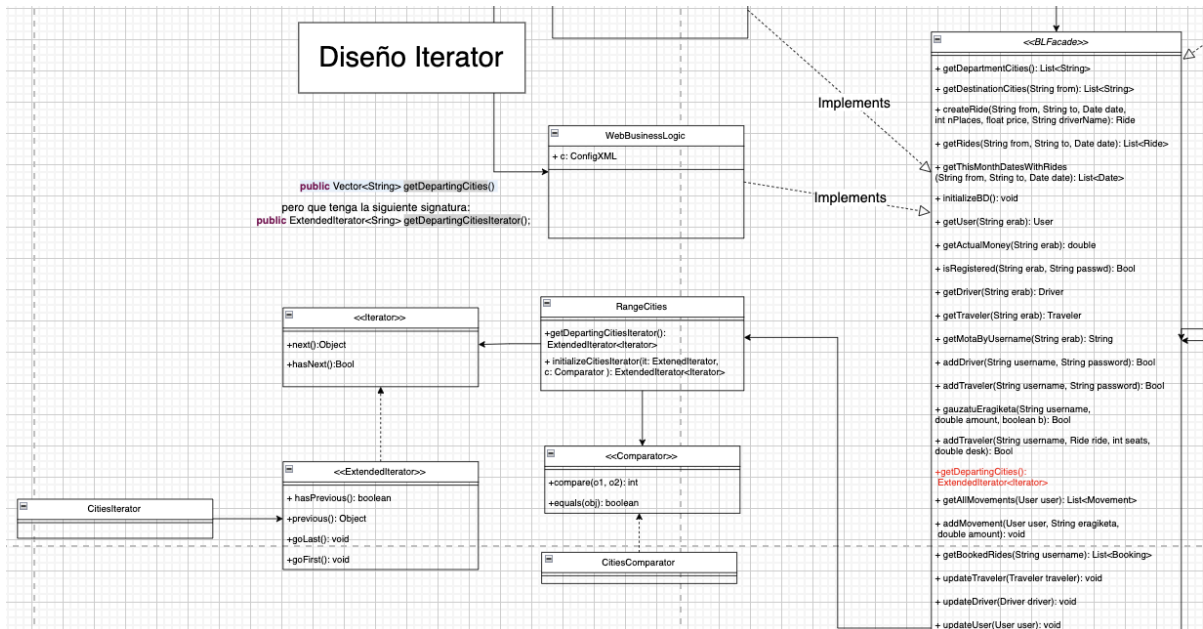


Figura 6 que muestra las clases

Agregadas así como la función modificada en BLFacade

Ahora, para cada una de las clases:

La clase de RangeCities se encarga de: Se crea un extended iterator y la lista que implementaremos para la solución, al iterador extendido le pasamos con lo que va a trabajar que en este caso será la lista inicializada como cities.

La interfaz de Comparator se encarga de: Establecer un contrato que deberá de ser implementado por un comparador customizable.

La interfaz de ExtendedIterator se encarga de: Extender la funcionalidad del Iterator de java por default.

La clase de CitiesIterator se encarga de: Implementar la funcionalidad del ExtendedIterator.

Los cambios realizados en BLFacade, BLFacadeImplementation y DataAccess, han sido realizados con el propósito de obtener un iterador de vuelta al momento de realizar una llamada al endpoint.

Finalmente, en ApplicationLauncher, se realizó lo especificado, donde se muestra el funcionamiento claro del iterador de manera que este va en reversa primero después va de manera incremental.

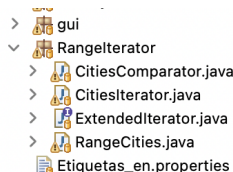
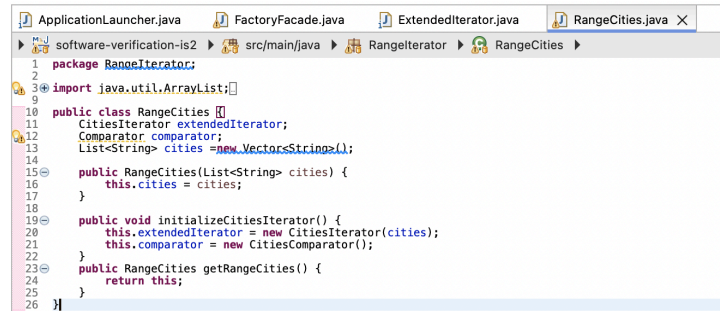


Figura 7 paquete añadido

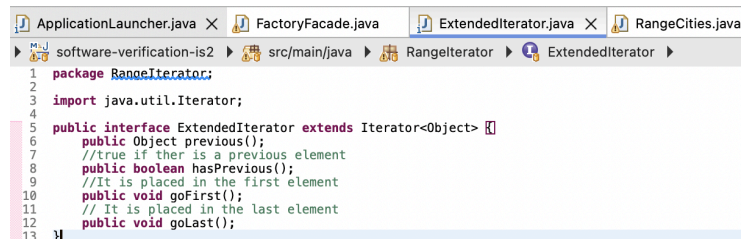


```

1 package RangeIterator;
2
3 import java.util.ArrayList;
4
5 public class RangeCities implements CitiesIterator {
6     Comparator comparator;
7     List<String> cities = new Vector<String>();
8
9     public RangeCities(List<String> cities) {
10         this.cities = cities;
11     }
12
13     public void initializeCitiesIterator() {
14         this.extendedIterator = new CitiesIterator(cities);
15         this.comparator = new CitiesComparator();
16     }
17
18     public RangeCities getRangeCities() {
19         return this;
20     }
21 }

```

**Figura 8 file de implementación de
RangeCities.java**

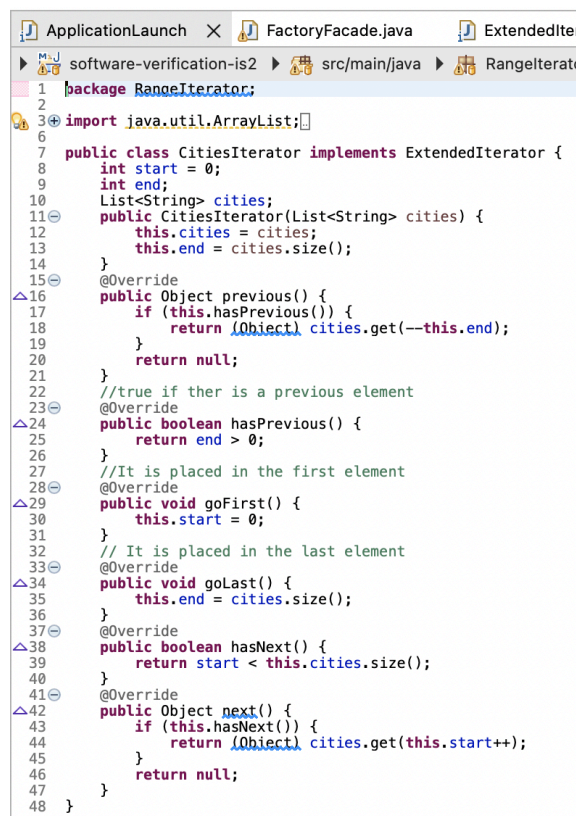


```

1 package RangeIterator;
2
3 import java.util.Iterator;
4
5 public interface ExtendedIterator extends Iterator<Object> {
6     Object previous();
7     //true if ther is a previous element
8     boolean hasNext();
9     //It is placed in the first element
10    void goFirst();
11    // It is placed in the last element
12    void goLast();
13 }

```

**Figura 9 Interfaz de
ExtendedIterator.java**



```

1 package RangeIterator;
2
3 import java.util.ArrayList;
4
5 public class CitiesIterator implements ExtendedIterator {
6     int start = 0;
7     int end;
8     List<String> cities;
9
10    public CitiesIterator(List<String> cities) {
11        this.cities = cities;
12        this.end = cities.size();
13    }
14
15    @Override
16    public Object previous() {
17        if (this.hasNext()) {
18            return (Object) cities.get(--this.end);
19        }
20        return null;
21    }
22    //true if ther is a previous element
23    @Override
24    public boolean hasNext() {
25        return end > 0;
26    }
27    //It is placed in the first element
28    @Override
29    public void goFirst() {
30        this.start = 0;
31    }
32    // It is placed in the last element
33    @Override
34    public void goLast() {
35        this.end = cities.size();
36    }
37    @Override
38    public boolean hasNext() {
39        return start < this.cities.size();
40    }
41    @Override
42    public Object next() {
43        if (this.hasNext()) {
44            return (Object) cities.get(this.start++);
45        }
46        return null;
47    }
48 }

```

**Figura 10 file de implementación
De iterador ExtendedIterator**


```

1 package RangeIterator;
2
3 import java.util.Comparator;
4
5 public class CitiesComparator implements Comparator {
6     @Override
7     public int compare(Object o1, Object o2) {
8         String city1 = (String)o1;
9         String city2 = (String)o2;
10        return city1.compareTo(city2);
11    }
12 }

```

Figura 11 file de implementación del comparador

Extendido (no fue utilizado pero sería factible para escalabilidad)

```

151 public void deleteCar(Car car);
152
153 public boolean erreklamazioaBidali(Complaint complaint);
154
155 public void updateComplaint(Complaint erreklamazioa);
156
157 public void createDiscount(Discount di);
158
159 public List<Discount> getAllDiscounts();
160
161 public void deleteDiscount(Discount dis);
162
163 public void updateDiscount(Discount dis);
164
165 public Discount getDiscount(String desk);
166
167 public List<User> getUserList();
168
169 public void deleteUser(User us);
170
171 public List<Alert> getAlertsByUsername(String username);
172
173 public Alert getAlert(int alertNumber);
174
175 public void updateAlert(Alert alert);
176
177 public boolean updateAlertaAurkituak(String username);
178
179 public boolean createAlert(Alert newAlert);
180
181 public boolean deleteAlert(int alertNumber);
182
183 public Complaint getComplaintsByBook(Booking bo);
184
185 public ExtendedIterator getDepartingCitiesIterator();
186
187 }
188
189

```

Figura 12 File la fachada BLFacade.java

Método getDepartingCitiesIterator()

```

72 * {@inheritDoc}
73 */
74 @WebMethod
75 public ExtendedIterator getDepartingCitiesIterator() {
76     dbManager.open();
77
78     ExtendedIterator departLocations = dbManager.getDepartingCities();
79
80     dbManager.close();
81
82     return departLocations;
83 }
84

```

Figura 13 File la implementación de la fachada

(se agregó un método que regresa un Extendediterator)

```

195 List<String> cities = query.getResultList();
196 return cities;
197
198 }
199
200 public ExtendedIterator getDepartingCities() {
201     TypedQuery<String> query = db.createQuery("SELECT DISTINCT r.from FROM Ride r ORDER BY r.from", String.class);
202     List<String> cityList = query.getResultList();
203     CitiesIterator citiesIterator = new CitiesIterator(cityList);
204     return citiesIterator;
205 }
206
207 /**
208  * This method returns all the arrival destinations, from all rides that depart
209  * from a given city
210  * @param from the depart location of a ride
211  * @return all the arrival destinations
212  */
213 public List<String> getArrivalCities(String from) {
214     TypedQuery<String> query = db.createQuery("SELECT DISTINCT r.to FROM Ride r WHERE r.from=?1 ORDER BY r.to",
215         String.class);
216     query.setParameter(1, from);
217     List<String> arrivingCities = query.getResultList();
218     return arrivingCities;
219 }
220
221 }

```

Figura 14 File de DataAccess.java

Método getArrivalCities()


```

ApplicationLauncher.java
src/main/java gui ApplicationLauncher main(String[]) : void
1 package gui;
2
3 import RangeIterator.ExtendedIterator;
4
5 public class ApplicationLauncher {
6
7     public static void main(String[] args) {
8         String isLocal = "local";
9         BLFacade bLFacade = new FactoryFacade().createBLFacade(isLocal);
10        System.out.println(bLFacade);
11        ExtendedIterator i = bLFacade.getDepartingCitiesIterator();
12        String c;
13        System.out.println("FROM LAST TO FIRST");
14        i.golast(); // Go to last element
15        while (i.hasPrevious()) {
16            c = (String) i.previous();
17            System.out.println(c);
18        }
19        System.out.println();
20        System.out.println("FROM FIRST TO LAST");
21        i.goFirst(); // Go to first element
22        while (i.hasNext()) {
23            c = (String) i.next();
24            System.out.println(c);
25        }
26    }
27 }

```

Figura 15 File de ApplicationLauncher.java

```

Console X
ApplicationLauncher [Java Application] /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java (Nov 4, 2024,
Read from config.xml: businessLogicLocal=true databaseLocal=true databaseInitialized=true
2024-11-04 15:42:13.184 java[8022:371057] +[IMKClient subclass]: chose IMKClient_Legacy
2024-11-04 15:42:13.184 java[8022:371057] +[IMKInputSession subclass]: chose IMKInputSession_Legacy
File deleted
DataAccess opened => isDatabaseLocal: true
Db initialized
DataAccess created => isDatabaseLocal: true isDatabaseInitialized: true
DataAccess closed
Nov 04, 2024 3:42:14 PM businessLogic.BLFacadeImplementation <init>
INFO: Creating BLFacadeImplementation instance with DataAccess parameter
businessLogic.BLFacadeImplementation@221a3fa4
DataAccess opened => isDatabaseLocal: true
DataAccess closed

FROM LAST TO FIRST
Madrid
Irun
Donostia
Barcelona

FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid

```

```

1 package gui;
2
3 import RangeIterator.ExtendedIterator;
4
5 public class ApplicationLauncher {
6
7     public static void main(String[] args) {
8         String isLocal = "local";
9         BLFacade bLFacade = new FactoryFacade().createBLFacade(isLocal);
10        System.out.println(bLFacade);
11        ExtendedIterator i = bLFacade.getDepartingCitiesIterator();
12        String c;
13        System.out.println("FROM LAST TO FIRST");
14        i.golast(); // Go to last element
15        while (i.hasPrevious()) {
16            c = (String) i.previous();
17            System.out.println(c);
18        }
19        System.out.println();
20        System.out.println("FROM FIRST TO LAST");
21        i.goFirst(); // Go to first element
22        while (i.hasNext()) {
23            c = (String) i.next();
24            System.out.println(c);
25        }
26    }
27 }

```

Figura 16 Resultado de aplicación del patrón

De diseño Iterator

El diseño refactorizado se encuentra en el siguiente [link](#).

3- Patrón Adapter¹

Se desea crear una tabla donde se muestren todos los viajes de un conductor. En la siguiente figura se muestran los viajes del conductor “Urtzi”.

Urtzi's rides				
from	to	date	places	price
Donostia	Madrid	Thu May 30 00:00:00 CEST 2024	5	20.0
Irun	Donostia	Thu May 30 00:00:00 CEST 2024	5	2.0
Madrid	Donostia	Fri May 10 00:00:00 CEST 2024	5	5.0
Barcelona	Madrid	Sat Apr 20 00:00:00 CEST 2024	0	10.0

La siguiente clase crea una ventana (JFrame) con un driver y muestra sus viajes en un JTable.

```

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);
    }
}

```

- ¹
- Una implementación similar a este ejercicio se presenta en la siguiente dirección:
<https://www.informit.com/articles/article.aspx?p=332278&seqNum=2>

El programa principal es:

```

public static void main(String[] args) {
    // the BL is local
    boolean isLocal = true;
    BLFacade blFacade = new
    BLFactory().getBusinessLogicFactory(isLocal); Driver d= blFacade.
    getDriver("Urtzi");
    DriverTable dt=new DriverTable(d);
    dt.setVisible(true);
}

```

Se pide implementar una clase adaptadora **DriverAdapter** de tipo TableModel, para poder visualizar un objeto de Driver (sus rides) en una JTable (Tambien hay que entregar el diagrama UML indicando qué papel juega cada clase en el patrón adapter)

DIAGRAMA UML:

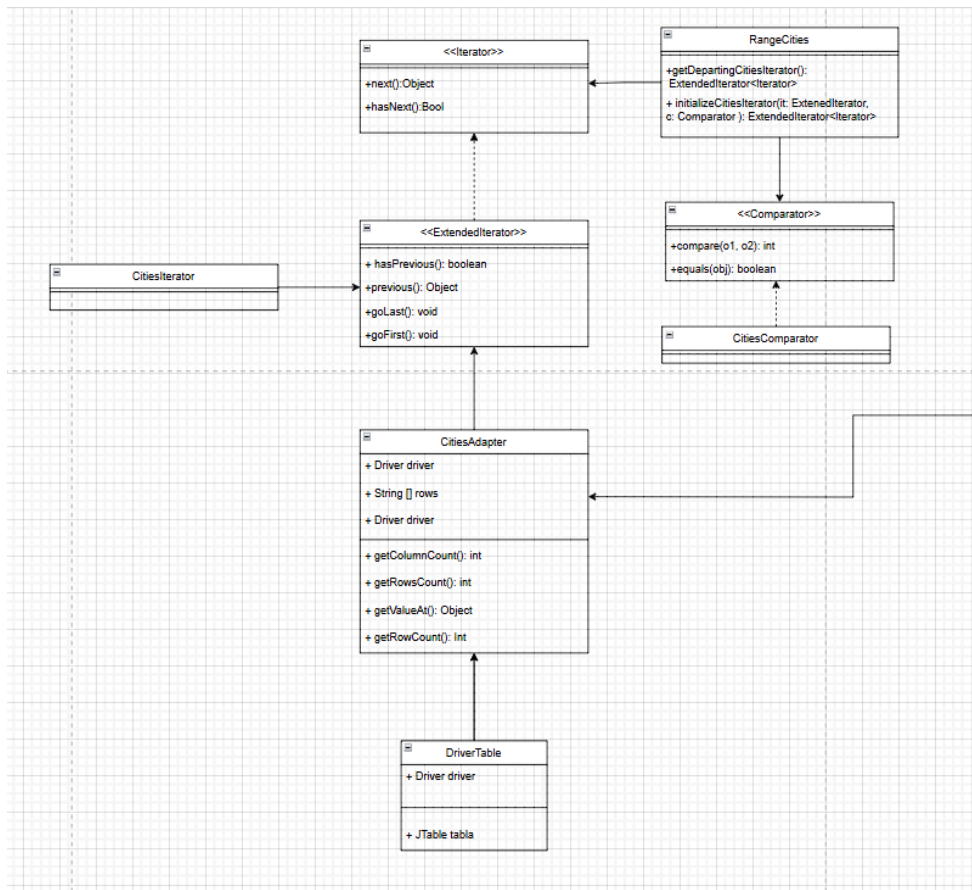


Figura 17 Resultado de aplicación del patrón

De diseño Iterator

En este diagrama implementamos nuestro Adaptador **CitiesAdapter** esta clase extiende o implementa **AbstractTableModel**, y actúa como un adaptador que permite presentar los datos de un **Driver** en una tabla de interfaz gráfica.

Urtzi's rides				
from	to	date	places	price
Donostia	Madrid	Sat Nov 30 00:00:00 CE...	5	20.0
Irun	Donostia	Sat Nov 30 00:00:00 CE...	5	2.0
Madrid	Donostia	Sun Nov 10 00:00:00 CE...	5	5.0
Barcelona	Madrid	Wed Nov 20 00:00:00 C...	0	10.0

Figura 18 Resultado de aplicación del patrón

Adapter

```

package adapter;

import businessLogic.BLFacade;
import domain.Driver;
import factoryFacade.FactoryFacade;

public class Main{
    public static void main(String[] args) {
        // the BL is local
        boolean isLocal = true;
        BLFacade blFacade = new FactoryFacade().createBLFacade("local");
        Driver d= blFacade. getDriver("Urtzi");
        DriverTableGUI dt=new DriverTableGUI(d);
        dt.setVisible(true);
    }
}

```

Figura 19 Clase de main

Contiene el resultado de implementar al adaptador

Nuestro main configura una fachada de negocio local para obtener un objeto Driver llamado "Urtzi" y luego muestra su información en una interfaz gráfica (DriverTableGUI).

```

package adapter;

import java.awt.BorderLayout;

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

    public DriverTableGUI(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);
    }
}

```

Figura 20 Creación de la tabla (Jscrollpane) agregando Jtable y el contenido

Nuestra **DriverTableGUI** muestra las "rides" de un Driver especificado, usando un adaptador (DriverAdapter) para poblar la tabla.

```

public class DriverAdapter extends AbstractTableModel {
    protected Driver driver;
    protected String [] columnNames = new String[] {"from", "to","date","places","price"};
    private final List<Ride> rides;
    public DriverAdapter(Driver d) {
        this.driver = d;
        rides = new ArrayList<Ride>(d.getCreatedRides());
    }

    public int getColumnCount(){
        return 5;
    }
    public int getRowCount(){
        return rides.size();
    }
    public Object getValueAt(int row, int col){
        switch (col) {
            //case 0: return ((Object) rides.get(row));
            case 0: return ((Object) rides.get(row).getFrom());
            case 1: return ((Object) rides.get(row).getTo());
            case 2: return ((Object) rides.get(row).getDate());
            case 3: return ((Object) rides.get(row).getnPlaces());
            case 4: return ((Object) rides.get(row).getPrice());
        }
        return null;
    }
    public String getColumnName(int col){
        switch (col) {
            case 0: return "from";
            case 1: return "to";
            case 2: return "date";
            case 3: return "places";
            case 4: return "price";
        }
        return null;
    }
}

```

***Figura 21 Contiene la implementación
Del adaptador del Driver (desplegar la tabla)***

Ahora implementamos el adaptador (DriverAdapter) que extiende AbstractTableModel para permitir que los datos de un Driver y sus "rides" (viajes) se muestren en una tabla (JTable). El adaptador toma una instancia de Driver y extrae una lista de sus viajes (rides), asignando a cada viaje las columnas from, to, date, places y price.

A continuación el link al [diagrama UML](#).

Documentación a entregar

Únicamente la URL del proyecto. En la raíz del repositorio habrá un documento patterns.pdf. En este documento habrá un apartado para cada ejercicio, indicando la siguiente información:

- a) Un diagrama UML extendido, indicando los cambios realizados y la finalidad de cada clase/interfaz.
- b) El código que habéis modificado, describiendo las líneas más significativas.
- c) Para los patrones Iterator y Adapter, una captura de imagen que muestre su ejecución.

