

SOFTWARE INGENIARITZA



Diseinu Patroiak - Rides24

GitHub-eko kodea:

<https://github.com/Braian-PC/Rides24Complete>

Egileak, Github helbidea:

Braian Porta, Braian-PC (bporta001@ikasle.ehu.eus)

Martín Etxeberria, MEtxebe (metxeberria045@ikasle.ehu.eus).

Diseinu patroiak

Dokumentu honetan Rides proiektuaren gainean egindako hainbat aldaketa ageri dira, aplikazioaren egituraren gainean eraldaketa ugari dakartenak. Aldaketa bakoitza nola egin pentsatu behar izan da, bai diseinuren, bai kodearen dituen ondorioak azalduz.

Diseinu patroiak programazio munduan erabili ohi diren irtenbide orokorrak dira, softwarearen diseinu arazo komunak konpontzeko. Diseinu patroiak software programak garatzerako orduan erabili daitezkeen praktika onak definitzen dituzte, eta proiektu desberdinetan errepikatu daitezkeen arazoentzako konponbide estandarizatuak eskaintzen dituzte.

Hiru motakoak izan daitezke:

- Sorketa patroiak Objektu eta klaseen sorketa, hasieraketa eta konfiguraketarekin erlazionatuta.
- Egitura patroiak: Klase eta objektuen banaketa interfaze eta inplementazioarekiko.
- Portaera patroiak objektu eta klaseen arteteko elkar eragiketak eta Sistemar baten portaera nola banatzen den klase eta objektuen artean

Dokumentu honetan **Factory**, **Iterator** eta **Adapter** patroiak implementatuko dira Rides24Complete proiektuaren gainean.

Eskatzen da: Proiektuaren URL-a entregatu behar da soilik. Errepositorioan, patterns.pdf dokumentu bat egon beharko da, ariketa bakoitzeko hurrengo informazioarekin:

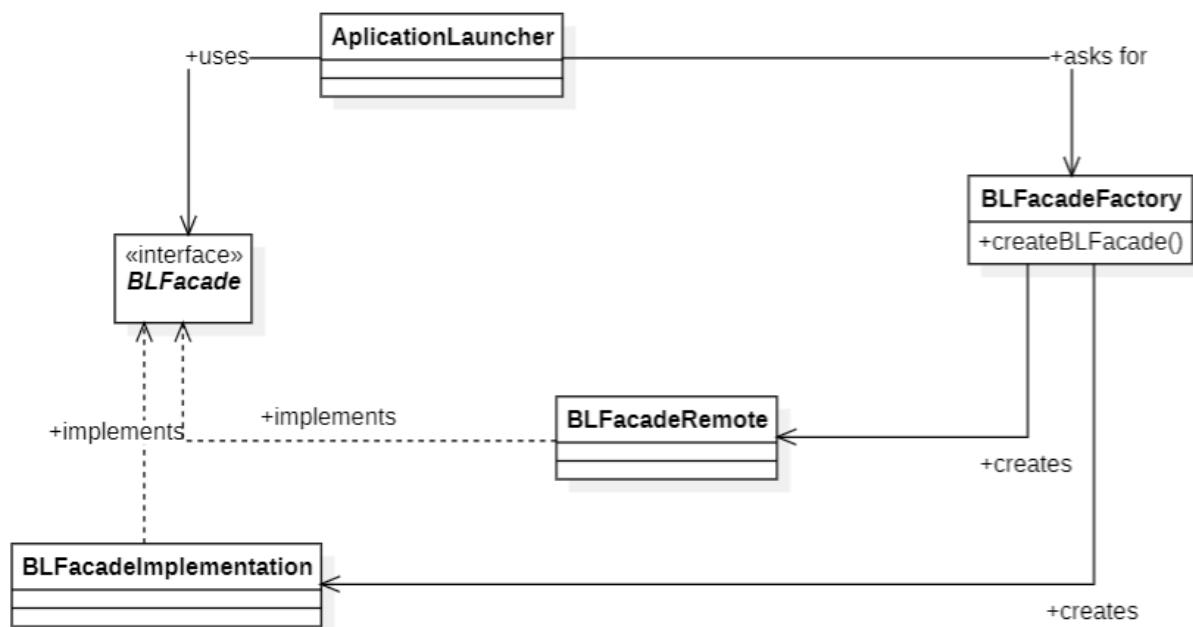
- a) UML diagrama hedatua egin dituzun aldaketak aurkeztuz.
- b) Aldatu duzun kodea, lerro garrantzitsuenak azalduz.
- c) Iterator eta Adapter patroientzako, exekuzioaren irudi bat

Factory Method Patroia

Eskatzen da: Aplikazioa aldatu negozio logikako objektuaren lorpena faktoria objektu batean zentralizatuta egoteko, eta aurkezpenak zein negozio logikako implementazio erabili erabaki dezatela. Diseina eta implementatu ebazpena Creator, Product eta ConcreteProduct jokatzen duten klaseen rola garbi aurkeztuz.

Implementazioa:

Factory patroia "ApplicationLauncher"-ean aplikatu ahal izateko BLFacadeFactory klase bat sortuko dugu nun ConfigXML.java klasearen edukiaren arabera BLFacadeImplementation (dataAccess lokala) edo BLFacadeRemote (konexio bidezko sarrera) sortuko duen.



```

public class BLFacadeFactory {
    public static BLFacade createBLFacade() {
        ConfigXML c = ConfigXML.getInstance();

        if (c.isBusinessLogicLocal()) {
            // lógica local
            DataAccess da= new DataAccess(c.isDatabaseInitialized()); //c.getDataBaseOpenMode().equals("initialize")
            return new BLFacadeImplementation(da);
        } else {
            // lógica remota
            try {
                return new BLFacadeRemote(c);
            } catch (Exception e) {
                System.out.println("Error creating remote BLFacade: " + e.toString());
                return null;
            }
        }
    }
}

```

BLFacadeImplementation klasean ez dira aldaketa nabarmenik burutu beharko baina patroia honi egitura egoki bat emateko BLFacadeRemote klasea ere sortu dugu. Hau BLFacade interfazearen implementazio bat da (ConfigXML c objektua eraikitzaile gisa jasotzen duena) eta zerbitzariarekin konexio bat ezartzeaz arduratuko da (eta BLFacade-ren instantzia itzultzeaz).

Kontuan eduki zerbitzaria arduratuko dela BLFacade interfazeko metodoak zehazteko arduraduna, “*service.getPort(BLFacade.class)*” eginez jasoko dugu BLFacade-ren instantzia (eta beraz ez gera hortaz arduratu behar).

```
public class BLFacadeRemote implements BLFacade{

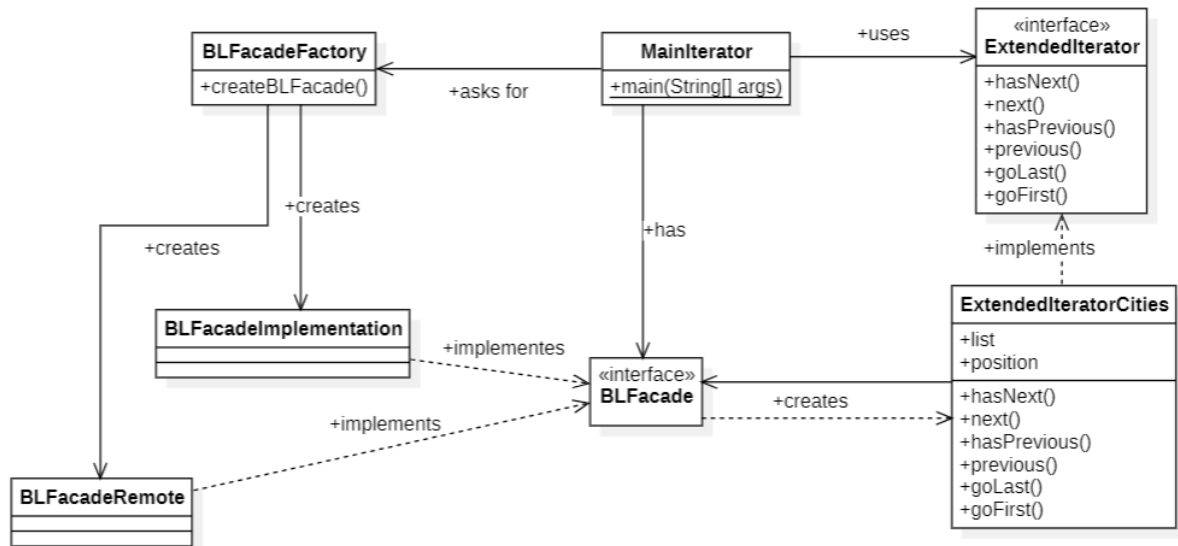
    private BLFacade remoteFacade;

    public BLFacadeRemote(ConfigXML c) {
        try {
            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);
            this.remoteFacade=service.getPort(BLFacade.class);
        }
        catch(Exception e) {
            System.out.println("Error creating remote BLFacade: " + e.toString());
        }
    }

    @Override
    public List<String> getDepartCities() {
        return remoteFacade.getDepartCities();
    }
    //...|
}
```

Iterator Patroia

Eskatzen da: Iteratzaile Hedatua implementatu, eta adibidezko antzeko programa bat implementatuz, hiriak aurkeztutako ordenan inprimatu.



Implementazioa:

ExtendedIterator interfazea erabiltzen duen klasea garatu ahal itzateko ondorengoak izan dira erabili diren parametroak eta eraikitzailea; "position" atriburua indeize gisa funtzionatu du, "list"-ek aldiz objetuen zerrenda gordeko du.

```

public class ExtendedIteratorCities<T> implements ExtendedIterator<T> {
    private List<T> list;
    private int position;

    public ExtendedIteratorCities(List<T> list) {
        // Clonar la lista para evitar modificar la original
        this.list = list.copyOf(list);
        this.position = 0; // Inicializar en la primera posición
    }
}
  
```

Ondoren next(), hasNext(), previous(), hasPrevious() metodoen funtzionamendua ageri da:

```
@Override
public boolean hasNext() {
    return position < list.size();
}

@Override
public T next() {
    if (hasNext()) {
        return list.get(position++);
    }
    throw new NoSuchElementException("No more elements");
}

@Override
public boolean hasPrevious() {
    return position > 0;
}

@Override
public T previous() {
    if (hasPrevious()) {
        return list.get(--position);
    }
    throw new NoSuchElementException("No previous elements");
}
```

Jarraian goFirst() eta goLast() metodoak:

```
@Override
public void goFirst() {
    position = 0;
}

@Override
public void goLast() {
    position = list.size(); // Colocar después del último elemento
}
```

Irteera emaitza (zuzena):

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

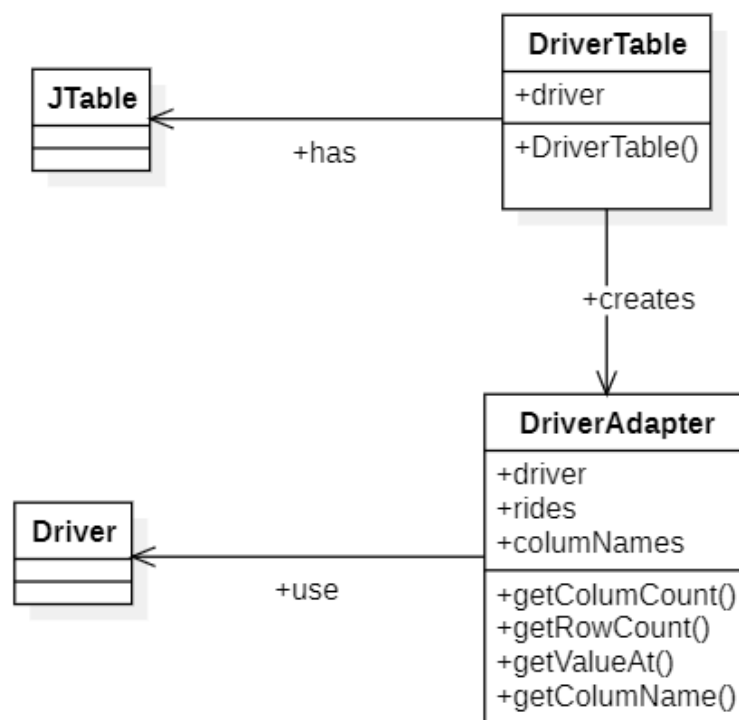
```
FROM FIRST TO LAST
Barcelona
Donostia
Irun
Madrid
```

Adapter Patroia

Eskatzen da: DriverTable klasean deitzen den DriverAdapter klasea implementatu, hasieran agertzen den taula aurkezteko (UML diseinua ere aurkeztu beharko da Adapter patroian parte hartzen duten klase guztiak aurkeztuz).

Implementazioa:

Eskaini zaigun informazioaren eta kodearen arabera ondorengoa izan beharko da kasu honetan Adapter patroiak eduki beharko duen UML diagrama. Bertan jada eskuragarri dugun DriverTable klaseak, JTable baten bidez, taula bat paintailaratuko du (nondik, nora, data... informazioarekin).



Horretarako lehenik datuak egokitu beharko dituen DriverAdapter objektu bat sortu beharko du gidariaren (honen bidaiei) informazioa DriverTable klaserako egokitu beharko duena.

Ondorengo izango da DriverAdapter klasearen implementazioa:

```
public class DriverAdapter extends AbstractTableModel {
    private Driver driver;
    private List<Ride> rides;
    private final String[] columnNames = {"from", "to", "date", "places", "price"};

    public DriverAdapter(Driver driver) {
        this.driver = driver;
        this.rides = driver.getCreatedRides();
    }

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

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

    @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();
            case 3: return ride.getNPlaces();
            case 4: return ride.getPrice();
            default: return null;
        }
    }

    @Override
    public String getColumnName(int column) {
        return columnNames[column];
    }
}
```

Klase honetan, erakusten duen erabiltzailearen bidaiaren tabla da. Hemen sortuko da tabla eta erakutsiko du bidaiaren informazioa. “addWindowListener()” metodoarekin, ahalbidetuko digu lehioa ixterakoan datubasearen exekuzioa amaitzea eta berriro “MainAdapter” testa exekutatzea (Zuzenki, System.exit(0)).

```
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);

        JScrollPane scrollPane = new JScrollPane(tabla);
        tabla.setPreferredScrollableViewportSize(new Dimension(500, 70));

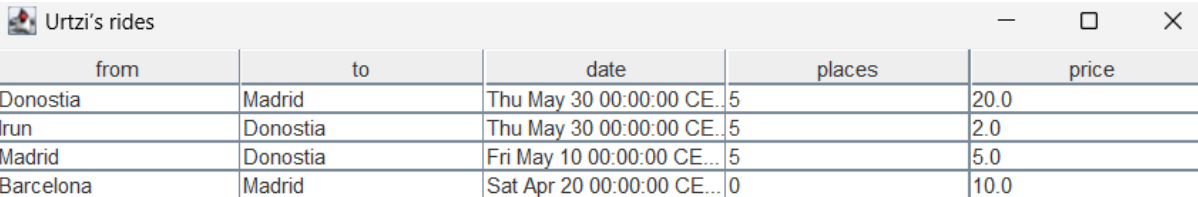
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        getContentPane().add(scrollPane, BorderLayout.CENTER);

        pack();

        this.addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                System.exit(0);
            }
        });
    }
}
```

Irteera emaitza (zuzena):



from	to	date	places	price
Donostia	Madrid	Thu May 30 00:00:00 CE...	5	20.0
Irun	Donostia	Thu May 30 00:00:00 CE...	5	2.0
Madrid	Donostia	Fri May 10 00:00:00 CE...	5	5.0
Barcelona	Madrid	Sat Apr 20 00:00:00 CE...	0	10.0