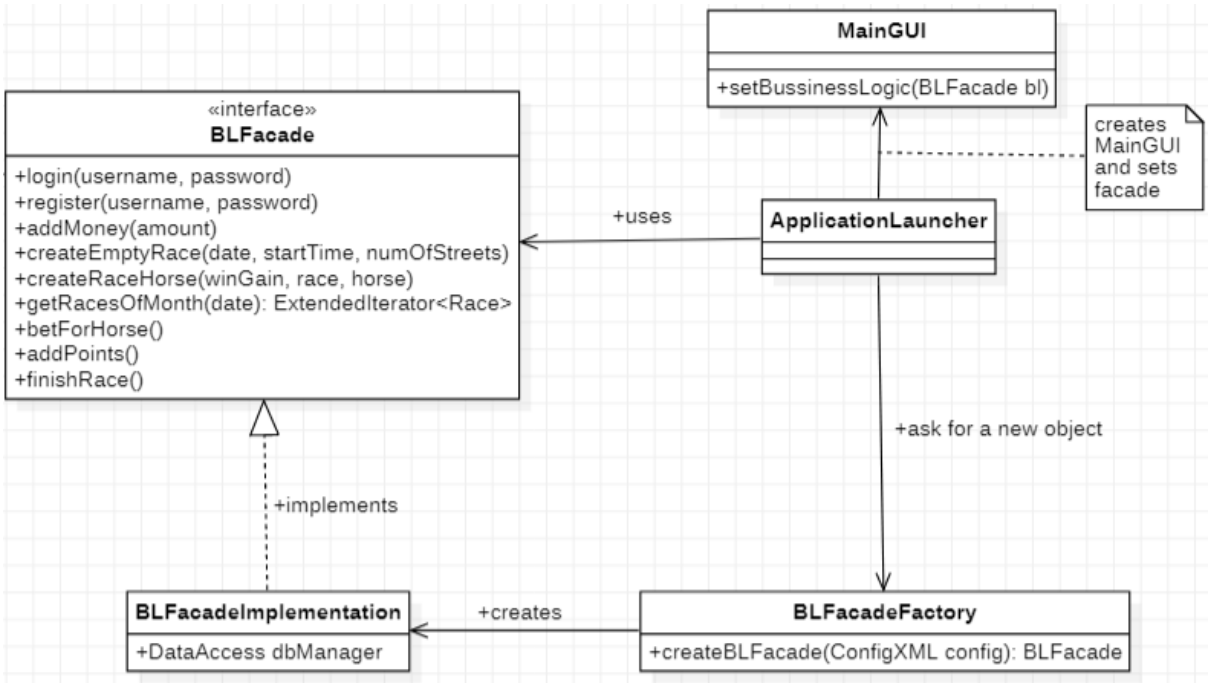
PROIEKTUAREN DISEINU PATROIAK

(Julen Larrañaga) GITHUB

Factory Method Patroia



Factory patroia burutu ahal izateko egin dudan lehen gauza *BLFacadeFactory* klasea sortzea izan da. Metodo horrek, BLFacade implementatzen duten objetuak sortzeko createBLFacade metodoa du bere barnean:

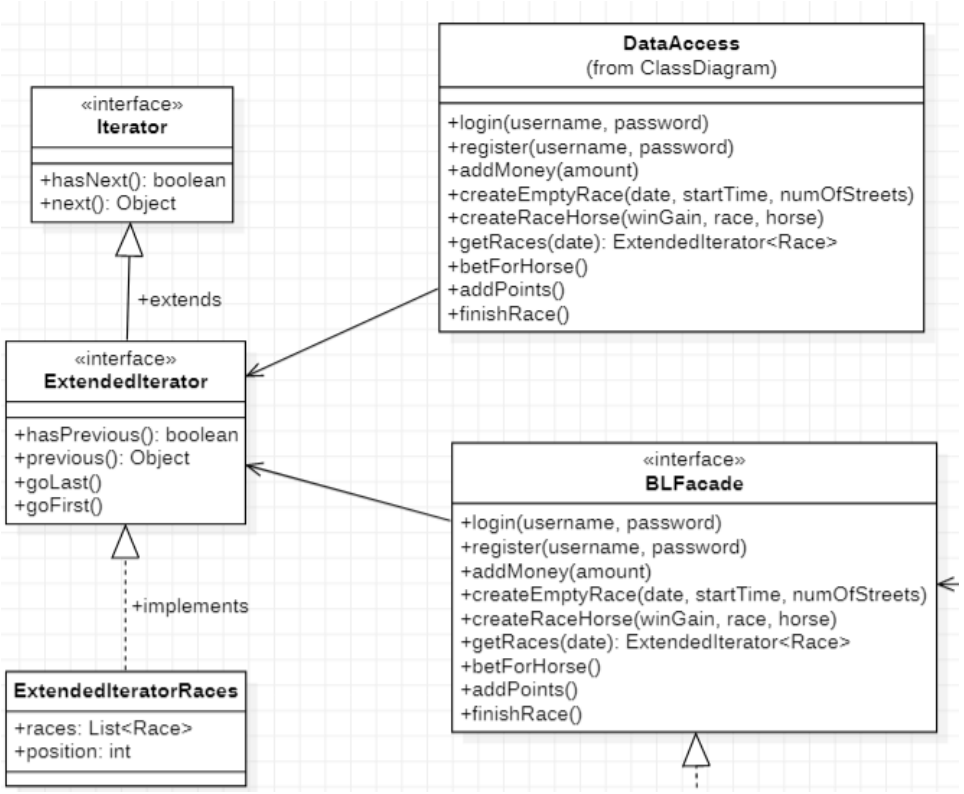
```
public BLFacade createBLFacade(ConfigXML config) throws MalformedURLException {

    BLFacade appFacadeInterface;

    if(config.isBusinessLogicLocal()) {
        DataAccess da= new DataAccess(config.getDataBaseOpenMode().equals("initialize"));
        appFacadeInterface = new BLFacadeImplementation(da);
    }else {
        String serviceName= "http://"+config.getBusinessLogicNode() +":"+ config.getBusinessLogicPort()+"/ws/"+config.getBusinessLogicName()+"?wsdl";
        URL url = new URL(serviceName);
        QName qname = new QName("http://businessLogic/", "BLFacadeImplementationService");
        Service service = Service.create(url, qname);
        appFacadeInterface = service.getPort(BLFacade.class);
    }
    return appFacadeInterface;

}
```

Argazkiak ikus daitekeen bezala, zuzenean ConfigXML ren instantzia parametro moduan pasatzea erabaki dut, horri esker metodoak jaso behar dituen parametro kopurua murrizten delako eta ondorengo argazkian ikusten den bezala, applicationLauncher klasean ere kodea sinplifikatzen delako configurazio lokala den ala ez konprobatu behar izan gabe.

```
MainGUI gui=new MainGUI();
gui.setVisible(true);

try {
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");

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

    MainGUI.setBussinessLogic(appFacadeInterface);

}catch (Exception e) {
    log.addLine("Error in ApplicationLauncher: "+e.toString());
}
```

Iterator Patroia

Nire proiektuan, Event motako objetuak beharrean lasterketak (Race) erabiltzen ditudanez, getRaces(Date date) metodoan egingo ditut aldaketak. Vector beharrean adierazitako ExtendedIterator<Race> itzul dezan. Aipatutako getRaces metodoa eta getEvents(Date date) baliokideak izango lirateke.



Hasteko, dokumentazioak adierazten duen *ExtendedIterator* interfazea implementatzen duen *ExtendedIteratorRaces* izeneko klasea sortu dut.

Klase horren inplementazioa egin ahal izateko, bi atributu erabiliko ditut: Bat races izenekoa List<Race> motakoa. Bestea position izenekoa zeinek iteradorearen burukoa dagoen posizioa gordeko duen.

```java
public class ExtendedIteratorRaces implements ExtendedIterator<Race>{

    List<Race> races;
    int position=0;

    public ExtendedIteratorRaces (List<Race> races) {
        this.races = races;
    }

    @Override
    public boolean hasNext() {
        return this.position<races.size();
    }

    @Override
    public Race next() {
        Race race = races.get(this.position);
        this.position++;
        return race;
    }

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

    @Override
    public Race previous() {
        Race race = races.get(this.position);
        this.position--;
        return race;
    }

    @Override
    public void goLast() {
        this.position = this.races.size()-1;
    }

    @Override
    public void goFirst() {
        this.position = 0;

    }
```

Argazkian ikus daitekeen bezala, *ExtendedIterator* interfazean erazagututako metodoetaz aparte, *Iteratorreko* hasNext() eta next() funtzioak ere inplementatu behar izan ditut.

Iteradorea probatzeko main metodoa:

```java
public class Main {
    public static void main(String[] args) throws MalformedURLException {
        //Facade objektua lortu lehendabiziko ariketa erabiliz
        BLFacade appFacadeInterface = new BLFacadeFactory().createBLFacade(ConfigXML.getInstance());

        ExtendedIterator<Race> i=appFacadeInterface.getRaces(new Date());
        Race race;
        i.goLast();
        while (i.hasPrevious()){
            race=i.previous();
            System.out.println("Beherantz: " + race.getDate());
        }
        //Nahiz eta suposatu hasierara ailegatu garela, eragiketa egiten dugu.
        i.goFirst();
        while(i.hasNext()){
            race=i.next();
            System.out.println("Gorantz: " + race.getDate());
        }
    }
}
```

main metodoaren exekuzioa:

(data tartea handiagoa ezarri dut, dagoeneko sortutako bi lasterketak erabiltzeko berririk sortu gabe)

```
<terminated> Main (4) [Java Application] C:\Program Files\
Beherantz: Fri Sep 17 00:00:00 CEST 2021
Beherantz: Wed Nov 17 00:00:00 CET 2021
Gorantz: Wed Nov 17 00:00:00 CET 2021
Gorantz: Fri Sep 17 00:00:00 CEST 2021
```

Bukatzeko, iteradoreak ondo egiten duela bere lana ikusita, getRaces metodoa eraldatzea geratzen da, iteradorea inplementa dezan. Horretarako:

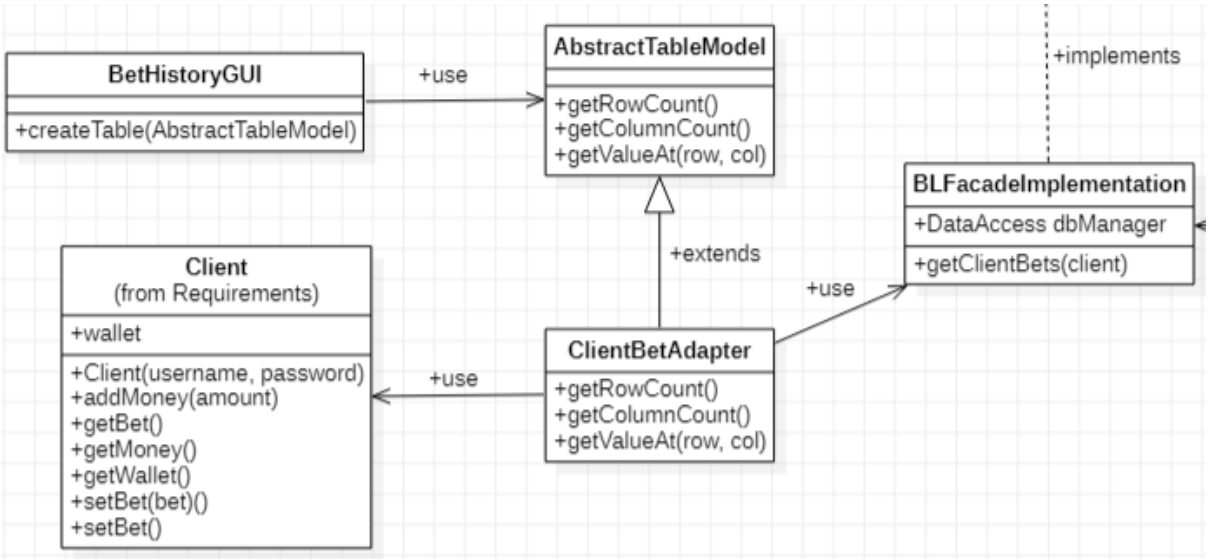
DataAccess klasean:

```
public ExtendedIterator<Race> getRaces(Date date){
    Date firstDayMonthDate = UtilDate.newDate(2010, 12, 10);
    Date lastDayMonthDate = UtilDate.lastDayMonth(date);
    TypedQuery<Race> query = db.createQuery("SELECT FROM Race rc WHERE rc.date BETWEEN ?1 and ?2",Race.class);
    query.setParameter(1, firstDayMonthDate);
    query.setParameter(2, lastDayMonthDate);
    return new ExtendedIteratorRaces(query.getResultList());
}
```

BLFacadeInplementation klasean:

```
public ExtendedIterator<Race> getRaces(Date date){
    dbManager.open(false);
    ExtendedIterator<Race> races = dbManager.getRaces(date);
    dbManager.close();
    return races;
}
```

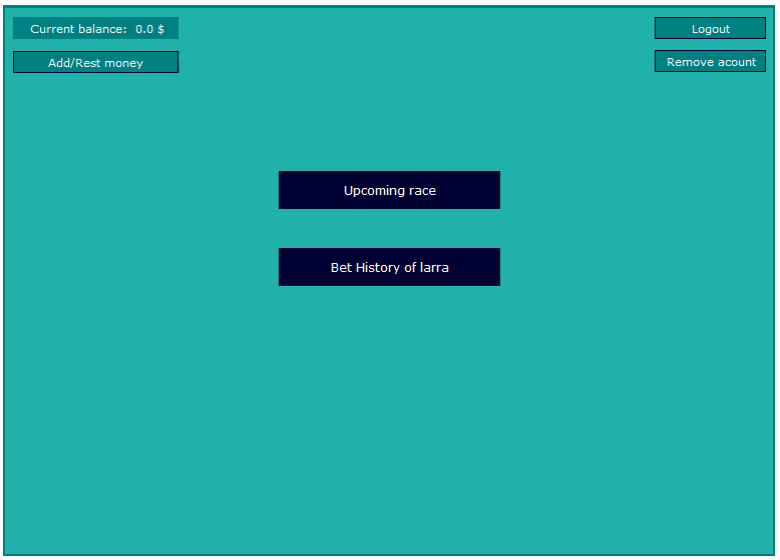
Adapter Patroia



Hasteko, nire proiektuko Client motako objetuetan gordetzen diren Bet-ak momentuan egindakoak direnez eta ez denboran zehar egindako guztiak, Facade eta DataAccess klaseetan getClientBets metodo berria inplementatu dut. Metodo horrek pasatako bezeroaren apustu guztiak lortu eta itzultzen ditu.

```
public List<Bet> getClientBets(Client client){
    TypedQuery<Bet> query = db.createQuery("SELECT FROM Bet b WHERE b.client = ?1",Bet.class);
    query.setParameter(1, client);
    return query.getResultList();
}
```

ClientGUI interfazean “Bet History of username” botoi bat txertatu dut, sortuko dudan leiho berrira eramateko:



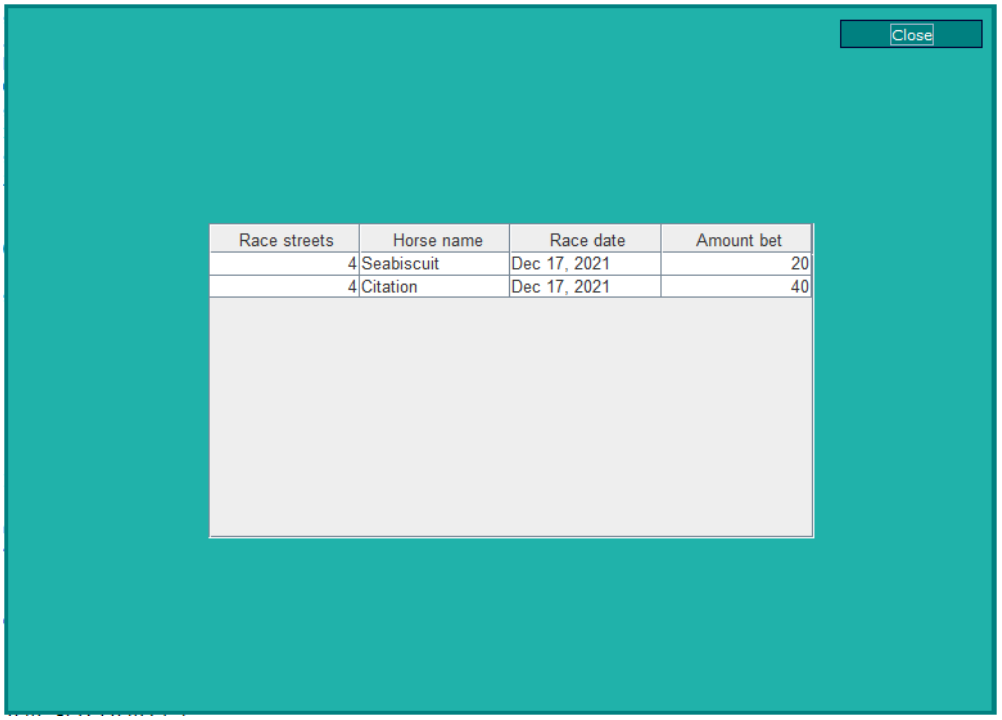
Ondoren, AbstractTableModel heredatzen duen ClientBetAdapter klasea inplementatu dut, JTable-rekin apustu taula sortzeko beharko diren metodoekin:

```
public class ClientBetAdapter extends AbstractTableModel {

    private transient List<Bet> bets;
    private String[] colNames = {"Race streets", "Horse name", "Race date", "Amount bet"};

    public ClientBetAdapter(Client client, BLFacade blf) {
        this.bets = blf.getClientBets(client);
    }

    @Override
    public int getRowCount() {
        System.out.println("a");
        if(bets == null) {
            return 0;
        }else {
            return bets.size();
        }
    }

    @Override
    public int getColumnCount() {
        System.out.println("b");
        return colNames.length;
    }

    @Override
    public String getColumnName(int col) {
        System.out.println("c");
        return colNames[col];
    }

    @Override
    public Class<?> getColumnClass(int col) {
        System.out.println("d");
        if (col == 0) {
            return Integer.class;
        }else if(col == 1){
            return String.class;
        }else if(col == 2){
            return Date.class;
        }else {
            return Double.class;
        }
   }

    @Override
    public Object getValueAt(int rowIndex, int columnIndex) {
        System.out.println("e");
        Bet bet = bets.get(rowIndex);
        Object returnObj = null;
        if(columnIndex == 0) {
            returnObj = Integer.valueOf(bet.getRaceHorse().getRace().getNumOfStreets());
        }else if(columnIndex == 1) {
            returnObj = bet.getRaceHorse().getHorse().getName();
        }else if(columnIndex == 2) {
            returnObj = bet.getRaceHorse().getRace().getDate();
        }else if(columnIndex == 3) {
            returnObj = Double.valueOf(bet.getAmount());
        }
        System.out.println(returnObj);
        return returnObj;
    }
}
```

Azkenik, taula azalduko den lehioa definitu eta bertan JTable sortu dut ClientBetAdapter parametro moduan pasata, taula automatikoki sor dezan:

```
JTable table = new JTable(new ClientBetAdapter(client, facade));
JScrollPane scrollPane=new JScrollPane(table);
scrollPane.setBounds(144, 154, 428, 222);
table.setBounds(400, 200, 200, 50);
contentPane.add(scrollPane);
```

Adapter patroia badoala ikusteko adibidea:



DataAcces-eko initialize metodoan pare bat apustu sortu ditut adibide bezala