Rides proiektuaren errefaktorizaioa

Egileak:Egoitz Ladron, Julen Oyarzabal eta Hodei Jauregi

Aurkibidea

1.SARRERA

Dokumentu honetan Rides proiektuaren gainean egindako errefaktorizazioak azalduko dira. Errefaktorizazioen ondoren lortutako emaitzak ondorengo GitHub-eko link-an daude: https://github.com/egoitz-ehu/Rides25

2.WRITE SHORT UNITS OF CODE

Gehienez 15 lerroko kode atalak onartzen dira, hau da, metodoen luzerak ez du gainditu behar 15 lerro. Ez bada baldintza hori betetzen banatu egin beharko dugu kodea metodo laguntzaileetan.

2.1.Egoitz Ladron

DataAccess klaseko kantzelatuBidaia metodoak 25 lerro dauzka. Kodea honakoa da:

```
public void kantzelatuBidaia(Ride r, String dMail) {
    db.getTransaction().begin();
    List<Erreserba> erreserbaList = r.getErreserbak();
    r.setEgoera(RideEgoera.KANTZELATUTA);
    Driver  d = db.find(Driver.class, dMail);
    for(Erreserba err:erreserbaList) {
        Erreserba e = db.find(Erreserba.class, err.getEskaeraNum());
        ErreserbaEgoera egoera = e.getEgoera();
        e.setEgoera(ErreserbaEgoera.KANTZELATUA);
        e.setKantzelatuData(new Date());
        Traveler t = e.getBidaiaria();
        double prezioa = e.getPrezioa();
        if(egoera.equals(ErreserbaEgoera.ONARTUA)) {
            d.removeFrozenMoney(prezioa);
            t.diruaSartu(prezioa);
            d.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_GIDARI);
            t.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_BIDAIARI);
        } else if(egoera.equals(ErreserbaEgoera.ZAIN)) {
            t.removeFrozenMoney(prezioa);
            t.diruaSartu(prezioa);
            t.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_BIDAIARI);
        }
    }
    db.merge(r);
    db.persist(d);
    db.getTransaction().commit();
}
```

Esan dugun moduan ‘bad smell’ hau zuzentzeko metodo laguntzaileetara banatu beharko dugu kodea. Ikus daiteke for aginduaren barnean dauden aginduak kanpoko metodo batera atera ditzakegu. Horretarako **“Extracting a Method from Code”** errefaktorizazio erabiliko dugu. Hori egin ondoren sortutako metodo laguntzailea horrelako izango da,

kantzelatuErreserba deitu diogu:

```
private void kantzelatuErreserba(Driver d, Erreserba err) {
    Erreserba e = db.find(Erreserba.class, err.getEskaeraNum());
    ErreserbaEgoera egoera = e.getEgoera();
    e.setEgoera(ErreserbaEgoera.KANTZELATUA);
    e.setKantzelatuData(new Date());
    Traveler t = e.getBidaiaria();
    double prezioa = e.getPrezioa();
    if(egoera.equals(ErreserbaEgoera.ONARTUA)) {
        d.removeFrozenMoney(prezioa);
        t.diruaSartu(prezioa);
        d.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_GIDARI);
        t.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_BIDAIARI);
    } else if(egoera.equals(ErreserbaEgoera.ZAIN)) {
        t.removeFrozenMoney(prezioa);
        t.diruaSartu(prezioa);
        t.addMugimendua(prezioa, MugimenduMota.BIDAIA_KANTZELATU_BIDAIARI);
    }
}
```

Metodo laguntzaile erabiliz honela geratuko da kantzelatuBidaia hasierako metodoa:

```
public void kantzelatuBidaia(Ride r, String dMail) {
    db.getTransaction().begin();
    List<Erreserba> erreserbaList = r.getErreserbak();
    r.setEgoera(RideEgoera.KANTZELATUTA);
    Driver  d = db.find(Driver.class, dMail);
    for(Erreserba err:erreserbaList) {
        kantzelatuErreserba(d, err);
    }
    db.merge(r);
    db.persist(d);
    db.getTransaction().commit();
}
```

2.2. Julen Oyarzabal

DataAccess klaseko sortuErreserba metodoak guztira 26 lerro dauzka. Honakoa da kodea:

```
public boolean sortuErreserba(Traveler t, int rNumber, int kop, String from, String to) throws EserlekurikLibreEzException, ErreserbaAlreadyExistsException,
DiruaEzDaukaException, DatuakNullException {
    if(kop>0) {
        db.getTransaction().begin();
        Ride r = db.find(Ride.class, rNumber);
        Traveler tr = db.find(Traveler.class, t.getEmail());
        if(r==null || tr==null) {
            db.getTransaction().commit();
            throw new DatuakNullException("Datuak null dira");
        }
        double kostua = r.prezioaKalkulatu(from, to);
        if(!tr.existBook(r)) {
            if(tr.diruaDauka(kostua)) {
                if(r.eserlekuakLibre(kop)) {
                    Erreserba erreserbaBerria = tr.sortuErreserba(r, kop, from, to, kostua);
                    r.gehituErreserba(erreserbaBerria);
                    tr.addMugimendua(kostua,MugimenduMota.ERRESERBA_SORTU);
                    db.persist(r);
                    db.getTransaction().commit();
                    return true;
                } else {
                    throw new EserlekurikLibreEzException("Ez dago nahiko eserlekurik libre");
                }
            } else {
                throw new DiruaEzDaukaException("Ez dauka dirurik");
            }
        } else {
            throw new ErreserbaAlreadyExistsException("Dagoeneko erreserba bat du erabiltzaile honek bidaia honetan");
        }
    }
    return false;
}
```

Ride eta Traveler datu basean dauden analizatu ondoren dagoen kode zatia beste metodo laguntzaile batera atera dezakegu, non bere helburua erreserba sortzea izango da. ErreserbaSortuEtaGehitu izeneko metodo laguntzaile bat sortu dugu, non 13 lerro dituen. Honakoa da kodea:

```
private boolean erreserbaSortuEtaGehitu(int kop, String from, String to, Ride r, Traveler tr)
        throws EserlekurikLibreEzException, DiruaEzDaukaException, ErreserbaAlreadyExistsException {
    double kostua = r.prezioaKalkulatu(from, to);
    if(!tr.existBook(r)) {
        if(tr.diruaDauka(kostua)) {
            if(r.eserlekuakLibre(kop)) {
                Erreserba erreserbaBerria = tr.sortuErreserba(r, kop, from, to, kostua);
                r.gehituErreserba(erreserbaBerria);
                tr.addMugimendua(kostua,MugimenduMota.ERRESERBA_SORTU);
                db.persist(r);
                db.getTransaction().commit();
                return true;
            } else throw new EserlekurikLibreEzException("Ez dago nahiko eserlekurik libre");
        } else throw new DiruaEzDaukaException("Ez dauka dirurik");
    } else throw new ErreserbaAlreadyExistsException("Dagoeneko erreserba bat du erabiltzaile honek bidaia honetan");
}
```

Hasierako metodoa honela geratu da:

```
public boolean sortuErreserba(Traveler t, int rNumber, int kop, String from, String to) throws EserlekurikLibreEzException, ErreserbaAlreadyExistsException,
DiruaEzDaukaException, DatuakNullException {
    if(kop>0) {
        db.getTransaction().begin();
        Ride r = db.find(Ride.class, rNumber);
        Traveler tr = db.find(Traveler.class, t.getEmail());
        if(r==null || tr==null) {
            db.getTransaction().commit();
            throw new DatuakNullException("Datuak null dira");
        }
        return erreserbaSortuEtaGehitu(kop, from, to, r, tr);
    }
    return false;
}
```

2.3. Hodei Jauregi

ErreserbaUkatu metodoak ere 28 kode lerro dituenez errefaktorizatu egingo dugu. Hau da kodea:

```java
public void erreserbaUkatu(int erreserbaNum, int rNumber) throws DatuakNullException, ErreserbaEgoeraEzDaZainException {
    db.getTransaction().begin();
    Erreserba e = db.find(Erreserba.class, erreserbaNum);
    if(e == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Erreserba null da");
    }
    if(e.getEgoera() != ErreserbaEgoera.ZAIN) {
        db.getTransaction().commit();
        throw new ErreserbaEgoeraEzDaZainException();
    }
    Traveler t = db.find(Traveler.class, e.getBidaiariaEmail());
    if(t == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Bidaiaria null da");
    }
    Ride r = db.find(Ride.class, rNumber);
    if(r == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Ride null da");
    }
    double kop = e.getPrezioa();
    int eserKop = e.getPlazaKop();

    e.setEgoera(ErreserbaEgoera.UKATUA);
    t.removeFrozenMoney(kop);
    t.diruaSartu(kop);
    r.itzuliEserlekuak(eserKop);
    t.addMugimendua(kop, MugimenduMota.ERRESERBA_UKATU);

    db.getTransaction().commit();
}
```

Metodo hau motzagoa egiteko if guztiak ateratzen saiatuko gara, eta ondoren erreserba ukatuta bezala jartzeko beharrezkoak direnak beste batera aterako ditugu. Hori egiteko lehen bi if-ak metodo batera aterako ditugu *erreserbarekinArazoak* izenekoa, beste bi if-ak *travelerEdoRideArazoak* izenekoa eta erreserba ukatzeko behar direnak hirugarren *erreserbaUkatu* metodora.

Lau if-ak bi metodotan banatu behar dira lehen biek erreserbarekin zerikusia dutelako eta azken biek erreserbaren egoerarekin, beraz lehen erreserbaren egoera aztertu behar da, ez dena batera (bestela ez du funtzionatzen Refactor->Extract Method funztioak eclipsen).

Horrela geratuko litzateke metodo berria:

```java
public void erreserbaUkatu(int erreserbaNum, int rNumber) throws DatuakNullException, ErreserbaEgoeraEzDaZainException {
    Erreserba e = db.find(Erreserba.class, erreserbaNum);
    erreserbarekinArazoak(e);
    Traveler t = db.find(Traveler.class, e.getBidaiariaEmail());
    Ride r = db.find(Ride.class, rNumber);

    travelerEdoRideArazoak(t, r);
    double kop = e.getPrezioa();
    int eserKop = e.getPlazaKop();
    erreserbaUkatuDB(e, t, r, kop, eserKop);
}
```

```java
private void erreserbaUkatuDB(Erreserba e, Traveler t, Ride r, double kop, int eserKop) {
    db.getTransaction().begin();
    e.setEgoera(ErreserbaEgoera.UKATUA);
    t.removeFrozenMoney(kop);
    t.diruaSartu(kop);
    r.itzuliEserlekuak(eserKop);
    t.addMugimendua(kop, MugimenduMota.ERRESERBA_UKATU);
    db.getTransaction().commit();
}
```

```java
private void travelerEdoRideArazoak(Traveler t, Ride r) throws DatuakNullException {
    if(t == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Bidaiaria null da");
    }
    if(r == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Ride null da");
    }
}
```

```java
private void erreserbarekinArazoak(Erreserba e) throws DatuakNullException, ErreserbaEgoeraEzDaZainException {
    if(e == null) {
        db.getTransaction().commit();
        throw new DatuakNullException("Erreserba null da");
    }
    if(e.getEgoera() != ErreserbaEgoera.ZAIN) {
        db.getTransaction().commit();
        throw new ErreserbaEgoeraEzDaZainException();
    }
}
```

3.WRITE SIMPLE UNITS OF CODE

Gehienez 4 adar puntu edukitzea gomendatzen da, hau da, if,for... motako aginduak gehienez 4 aldiz agertzea. Agindu horien bidez kodeak bide desberdinak hartu ditzake, ondorioz kodea ulertzea zailagoa izango da. Hori konpontzeko gomendatzen da beste metodo batzuetara mugitzea kode zati batzuk. Horrela kodea ulertzea eta mantentzea errezagoa izango da.

3.1. Egoitz Ladron

DataAccess klaseko lortuBalorazioErreserbak metodoak 5 adar puntu dauzka, zehazki, 3 if eta 2 for. Honakoa da kodea:

```
public List<TravelerErreserbaContainer> lortuBalorazioErreserbak(User u){
    if(u == null) return new LinkedList<TravelerErreserbaContainer>();
    List<Erreserba> erreserbak;
    if(u instanceof Traveler) {
        TypedQuery<Erreserba> q1 = db.createQuery(
                "SELECT e FROM Erreserba e WHERE e.bidaiaria = :u AND (e.egoera=:j OR e.egoera=:k)", Erreserba.class);
        q1.setParameter("u", u);
        q1.setParameter("j", ErreserbaEgoera.BAIEZTATUA);
        q1.setParameter("k", ErreserbaEgoera.EZEZTATUA);
        erreserbak = q1.getResultList();

    } else {
        TypedQuery<Erreserba> q1 = db.createQuery("SELECT e FROM Erreserba e JOIN e.ride r WHERE r.driver=:u AND (e.egoera=:j OR e.egoera=:k)",Erreserba.class);
        q1.setParameter("u", u);
        q1.setParameter("j", ErreserbaEgoera.BAIEZTATUA);
        q1.setParameter("k", ErreserbaEgoera.EZEZTATUA);
        erreserbak = q1.getResultList();
    }
    TypedQuery<Erreserba> q2 = db.createQuery(
            "SELECT b.erreserba FROM Balorazioa b WHERE b.nork = :u", Erreserba.class);
    q2.setParameter("u", u);
    List<Erreserba> baloratuak = q2.getResultList();
    List<Erreserba> gabe = new ArrayList<>();
    for (Erreserba e : erreserbak) {
        if (!baloratuak.contains(e)) {
            gabe.add(e);
        }
    }
    List<TravelerErreserbaContainer> containerList = new LinkedList<TravelerErreserbaContainer>();
    for(Erreserba e:gabe) {
        containerList.add(new TravelerErreserbaContainer(e,e.getBidaiaria()));
    }
    return containerList;
}
```

Bigarren if agindua(if eta else atala) beste metodo laguntzaile batera atera dezakegu. Metodo horrek erreserba lista bat itzuliko du. Horretarako **“Extracting a Method from Code”** errefaktorizazioa erabiliko dugu. Honelako izango da metodo laguntzailea:

```
private List<Erreserba> lotuErabiltzaileErreserbak(User u) {
    List<Erreserba> erreserbak;
    if(u instanceof Traveler) {
        TypedQuery<Erreserba> q1 = db.createQuery(
                "SELECT e FROM Erreserba e WHERE e.bidaiaria = :u AND (e.egoera=:j OR e.egoera=:k)", Erreserba.class);
        q1.setParameter("u", u);
        q1.setParameter("j", ErreserbaEgoera.BAIEZTATUA);
        q1.setParameter("k", ErreserbaEgoera.EZEZTATUA);
        erreserbak = q1.getResultList();

    } else {
        TypedQuery<Erreserba> q1 = db.createQuery("SELECT e FROM Erreserba e JOIN e.ride r WHERE r.driver=:u AND (e.egoera=:j OR e.egoera=:k)",Erreserba.class);
        q1.setParameter("u", u);
        q1.setParameter("j", ErreserbaEgoera.BAIEZTATUA);
        q1.setParameter("k", ErreserbaEgoera.EZEZTATUA);
        erreserbak = q1.getResultList();
    }
    return erreserbak;
}
```

LortuBalorazioakErreserba metodoa horrela geratuko da:

```java
public List<TravelerErreserbaContainer> lortuBalorazioErreserbak(User u){
    if(u == null) return new LinkedList<TravelerErreserbaContainer>();
    List<Erreserba> erreserbak = lotuErabiltzaileErreserbak(u);
    TypedQuery<Erreserba> q2 = db.createQuery(
            "SELECT b.erreserba FROM Balorazioa b WHERE b.nork = :u", Erreserba.class);
    q2.setParameter("u", u);
    List<Erreserba> baloratuak = q2.getResultList();
    List<Erreserba> gabe = new ArrayList<>();
    for (Erreserba e : erreserbak) {
        if (!baloratuak.contains(e)) {
            gabe.add(e);
        }
    }
    List<TravelerErreserbaContainer> containerList = new LinkedList<TravelerErreserbaContainer>();
    for(Erreserba e:gabe) {
        containerList.add(new TravelerErreserbaContainer(e,e.getBidaiaria()));
    }
    return containerList;
}
```

3.2. Julen Oyarzabal

DataAccess klaseko sortuAlerta metodoak 5 adar puntu dauzka, zehazki, 5 if agindu. Honakoa da kodea:

```java
public void sortuAlerta(String email, String from, String to, Date d) throws BadagoRideException, ErreserbaAlreadyExistsException, AlertaAlreadyExistsException {
    if(email!=null) {
        if(from!=null && to!=null) {
            db.getTransaction().begin();
            Traveler t = db.find(Traveler.class, email);
            if(!t.baduAlertaBerdina(from, to, d)) {
                if(!t.baduErreserba(from, to, d)) {
                    if(getThisMonthDatesWithRides(from,to,d).isEmpty()) {
                        t.sortuAlerta(from, to, d);
                    } else {
                        db.getTransaction().commit();
                        throw new BadagoRideException();
                    }
                } else {
                    db.getTransaction().commit();
                    throw new ErreserbaAlreadyExistsException();
                }
            } else {
                db.getTransaction().commit();
                throw new AlertaAlreadyExistsException();
            }
            db.getTransaction().commit();
        }
    }
}
```

Kode zati hori ulertzeko zaila da. Hori konpontzeko bi metodo laguntzaileetara mugituko ditugu balidazioak. Bat sarrerako balioak egokiak diren analizatzeko eta bestea alerta sortzea egokia izango den aztertzeko(dagoeneko badagoen bat sortua, bidaiarik existitzen den...). Honakoa da bi metodo laguntzaile horien kodea:

```java
private boolean sortuAlertaSarreraEgokiak(String email, String from, String to) {
    return (email!=null && from!=null && to!=null);
}
```

```
private boolean sortuAlertaSortuBehar(Traveler t, String from, String to, Date d) throws BadagoRideException,
ErreserbaAlreadyExistsException, AlertaAlreadyExistsException {
    if(t.baduAlertaBerdina(from, to, d)) {
        throw new AlertaAlreadyExistsException();
    }
    if(t.baduErreserba(from, to, d)) {
        throw new ErreserbaAlreadyExistsException();
    }
    if(!getThisMonthDatesWithRides(from,to,d).isEmpty()) {
        throw new BadagoRideException();
    }
    return true;
}
```

Bi metodo horiez erabiliz honela geratuko da hasierako metodoa:

```
public void sortuAlerta(String email, String from, String to, Date d) throws BadagoRideException,
ErreserbaAlreadyExistsException, AlertaAlreadyExistsException {
    if(sortuAlertaSarreraEgokiak(email, from, to)) {
        Traveler t = db.find(Traveler.class, email);
        sortuAlertaSortuBehar(t,from,to,d);
        db.getTransaction().begin();
        t.sortuAlerta(from, to, d);
        db.getTransaction().commit();
    }
}
```

Orain kodea ulertzea/aldatzea errazagoa da.

3.3. Hodei Jauregi

Proiektuan ez dugu aurkitu beste metodorik non 4 adar puntu baino gehiago dituen. Baina DataAccess paketean badago metodo bat non ulertzea zaila den. getArrivalCities metodoak ere 3 for bigizta eta if baldintza bat jasotzen du, beraz komeni da hauetakoren bat beste metodo batera ateratzea.

Hasierako metodoa:

```
public List<String> getArrivalCities(String from){
    TypedQuery<Ride> query = db.createQuery("SELECT r FROM Ride r",Ride.class);
    List<Ride> rides = query.getResultList();
    List<String> cities = new LinkedList<String>();
    for(Ride r:rides) {
        List<Geldialdia> gList = r.getGeldialdiak();
        List<String> geldialdiak = new LinkedList<String>();
        for(Geldialdia g:gList) {
            geldialdiak.add(g.getHiria());
        }
        if(geldialdiak.contains(from)) {
            int i = geldialdiak.indexOf(from);
            for(int j=i+1;j<geldialdiak.size();j++) {
                cities.add(geldialdiak.get(j));
            }
        }
    }
    return cities;
}
```

Kasu honetan metodoa bi zatitan banatuko dugu. Batetik lehen bi for bigiztak mantenduko ditugu, eta if barrukoa beste metodo batera pasako dugu *addArrivalCities* izenekoa. Hau berriz ere **"Extracting a Method from Code"** errefaktorizazioa erabiliz egin dezakegu.

Horrela geratuko litzateke metodo berria:

```
public List<String> getArrivalCities(String from){
    TypedQuery<Ride> query = db.createQuery("SELECT r FROM Ride r",Ride.class);
    List<Ride> rides = query.getResultList();
    List<String> cities = new LinkedList<String>();
    for(Ride r:rides) {
        List<Geldialdia> gList = r.getGeldialdiak();
        List<String> geldialdiak = new LinkedList<String>();
        for(Geldialdia g:gList) {
            geldialdiak.add(g.getHiria());
        }

        addArrivalCities(from, cities, geldialdiak);
    }
    return cities;
}
```

```
private void addArrivalCities(String from, List<String> cities, List<String> geldialdiak) {
    if(geldialdiak.contains(from)) {
        int i = geldialdiak.indexOf(from);
        for(int j=i+1;j<geldialdiak.size();j++) {
            cities.add(geldialdiak.get(j));
        }
    }
}
```

4.DUPLICATE CODE

Sonar-eko analisiak esaten du ez dagoela kode errepikaturik DataAccess klasean:

Hori dela eta beste klase batzuetako kode errepikatuak konponduko ditugu.

Horrelako egoera baten aurrean gaudenean egokiena kode hori metodo generiko batera ateratzea da. Horrela leku desberdinetatik erabili dezakegu, eta errore bat aurkitzea edo aldaketa bat egin behar denean leku bakar batean egitearekin nahikoa izango da.

Proiektua aztertu ondoren errepikatuta dagoen kode atal bakarra GUI sorrerari dagokiona da, GUIa sortzean erabilitako interfaze grafikoak sortutako kodea. Kode hori ez errepikatzea oso zaila da. Irakaslearekin hitz egin ondoren erabaki da ez dela aldaketarik egingo.

5.KEEP UNIT INTERFACES SMALL

Metodo batek jasotzen dituen parametro kopurua 4era mugatzea gomendatzen da. Horrela kodea ulertzeko errazagoa izango da. 4 baino handiagoa denean parametro kopurua parametroak objektuetan elkartzea gomendatzen da.

5.2.Egoitz Ladron

DataAccess klaseko register metodoak 5 parametro ditu bere signaturan. Honakoa da kodea:

```java
public boolean register(String email, String name, String surname, String password, String type) {
    User u=db.find(User.class, email);
    if (u==null) {
        User user;
        if(type.equals("driver")) {
            user = new Driver(email,password,name,surname);
        } else {
            user = new Traveler(email,password,name,surname);
        }
        db.getTransaction().begin();
        db.persist(user);
        db.getTransaction().commit();
        return true;
    } else {
        return false;
    }
}
```

Email, name, surname eta password parametroak elkartu ditzazkegu User klase batean. Gure domeinuan dagoeneko sortutako dago User klase bat baina klase abstraktu bat da. Ondorioz ez digu balio, UserData izeneko klase bat sortuko dugu informazio hori parametro moduan pasatzeko. Honako itxura dauka:

```java
public class UserData {
    public String email;
    public String password;
    public String name;
    public String surname;

    public UserData(String email, String pass, String name, String surname) {
        this.email = email;
        this.password = pass;
        this.name = name;
        this.surname = surname;
    }
}
```

Hortaz parametro moduan User erabiliz, horretarako “Change method signature” errefaktorizazio erabili dezakegu. Honakoa izango da kodea errefaktorizazioaren ondoren:

```
public boolean register(UserData us, String type) {
    User u=db.find(User.class, us.email);
    if (u==null) {
        User user;
        if(type.equals("driver")) {
            user = new Driver(us.email,us.password,us.name,us.surname);
        } else {
            user = new Traveler(us.email,us.password,us.name,us.surname);
        }
        db.getTransaction().begin();
        db.persist(user);
        db.getTransaction().commit();
        return true;
    } else {
        return false;
    }
}
```

Baina BLFacade klasean metodo hori erabiltzen duen metodoan arazo bera daukagu. Arazoa konpontzeko berdina egingo dugu. Hasierako kodea:

```
@WebMethod
public boolean register(String email, String name, String surname, String password, String type) {
    dbManager.open();
    boolean b = dbManager.register(null, type);
    dbManager.close();
    return b;
}
```

Ondoren:

```
@WebMethod
public boolean register(UserData u, String type) {
    dbManager.open();
    boolean b = dbManager.register(u, type);
    dbManager.close();
    return b;
}
```

Orain metodo hau erabiltzeko User motako klase bat sortu beharko da eta ondoren parametro moduan pasa. Adibidez:

```
boolean b =bussinessLogic.register(new UserData(mail, name, surname, pass), type);
```

5.2 Julen Oyarzabal

DataAccess klaseko sortuErreserba metodoak 5 parametro ditu bere signaturan. Honakoa da kodea:

```
public boolean sortuErreserba(Traveler t, int rNumber, int kop, String from, String to)
        throws EserlekurikLibreEzException, ErreserbaAlreadyExistsException,
DiruaEzDaukaException, DatuakNullException {
    if(kop>0) {
        db.getTransaction().begin();
        Ride r = db.find(Ride.class, rNumber);
        Traveler tr = db.find(Traveler.class, t.getEmail());
        if(r==null || tr==null) {
            db.getTransaction().commit();
            throw new DatuakNullException("Datuak null dira");
        }
        return erreserbaSortuEtaGehitu(kop, from, to, r, tr);
    }
    return false;
}
```

rNumber, kop, from eta to parametroak elkartu ditzazkegu erreserba informazioa gordetzen duen klase batean. Gure domeinuan dagoeneko sortutako dago Erreserba klase bat, baina digu balio beste balio batzuk ere behar dituela(loturak, prezioak...). Ondorioz egokiena beste klase bat sortzea da, adibidez ErreserbaData. Honakoa da bere kodea:

```
public class ErreserbaData {
    public int rNumber;
    public String from;
    public String to;
    public int kop;

    public ErreserbaData(int rNumber, String from, String to, int kop) {
        this.rNumber = rNumber;
        this.from = from;
        this.to = to;
        this.kop = kop;
    }
}
```

Hortaz parametro moduan BidaiaDate erabiliz, horretarako “Change method signature” errefaktorizazio erabili dezakegu. Honakoa izango da kodea errefaktorizazioaren ondoren:

```
public boolean sortuErreserba(Traveler t, ErreserbaData erreData)
        throws EserlekurikLibreEzException, ErreserbaAlreadyExistsException,
DiruaEzDaukaException, DatuakNullException {
    if(erreData.kop>0) {
        db.getTransaction().begin();
        Ride r = db.find(Ride.class, erreData.rNumber);
        Traveler tr = db.find(Traveler.class, t.getEmail());
        if(r==null || tr==null) {
            db.getTransaction().commit();
            throw new DatuakNullException("Datuak null dira");
        }
        return erreserbaSortuEtaGehitu(erreData.kop, erreData.from, erreData.to, r, tr);
    }
    return false;
}
```

Baina hemen ere metodoa deitzen den bakoitzean metodoaren signatura aldatu denez, deia ere aldatu egin behar da.

Adibidez, lehen BLFacadeImplementation klasean horrela zegoen:

```
    @WebMethod
    public boolean sortuErreserba(Traveler t, int rNumber, int kop, String from, String to)
    throws EserlekurikLibreEzException, ErreserbaAlreadyExistsException, DiruaEzDaukaException, DatuakNullException {
        dbManager.open();
        boolean b = dbManager.sortuErreserba(t, rNumber, kop, from, to);
        dbManager.close();
        return b;
    }
```

Eta orain aldaketa hau egin zaio:

```
@WebMethod
public boolean sortuErreserba(Traveler t, ErreserbaData eData) throws
EserlekurikLibreEzException, ErreserbaAlreadyExistsException, DiruaEzDaukaException, DatuakNullException {
    dbManager.open();
    boolean b = dbManager.sortuErreserba(t, eData);
    dbManager.close();
    return b;
}
```

BLFacadeko metodoa erabiltzen duten lekuetan ere aldatu beharko dugu. ErresrbaData klasearen instantzia bat sortu beharko dugu, parametro moduan pasatzeko. Adibidez, ErreserbaEskaeraGUI klasean:

```
ErreserbaData eData = new ErreserbaData(selectedRide.getRideNumber(),
    (String) jComboBoxOrigin.getSelectedItem(),
    (String)jComboBoxDestination.getSelectedItem(), kop);
boolean b = WelcomeGUI.getBusinessLogic().sortuErreserba(t, eData);
```

Metodoaren signatura aldatu dugunez test-ak ere aldatu beharra dago. Hortaz sotuErreserba metodoaren inguruko test guztiak aldatu beharko dira, ErreserbaEskaeraGUI klasean egin dugun moduan.

5.3. Hodei Jauregi

DataAccess klaseko *sortuKotxea* metodoak ere 5 parametro jasotzen ditu bere signaturan, eta ulerterrazagoa egiteko objektu bat pasako diogu hauen ordez. Hau da hasierako kodea:

```
public boolean sortuKotxea(String matrikula, int eserKop, String kolorea, String mota, Driver d) {
    Car kotxea = db.find(Car.class, matrikula);
    if(kotxea!=null) return false;
    db.getTransaction().begin();

    Car c=d.addCar(matrikula,eserKop,kolorea,mota);
    db.persist(c);
    db.merge(d);
    db.getTransaction().commit();
    return true;
}
```

Kasu honetan matrikula, eserKop, kolorea, mota eta d (gidaria) guztiak objektu batean bildu nahi ditugu, eta gure domeinuko Car klaseak iada hori egiten du. Hortaz zuzenean Car klasea erabil dezakegu, mota horretako objetua sortu eta sortuKotxea metodoan parametro moduan pasa.

Baina hau egiten bada, kontuan izan behar da metodo barruan hainbat gauza aldatu behar direla. Esaterako, metodoan matrikula, eserKop, kolorea, mota eta d parametroak erabiltzen dira, baina orain ez dira metodoarekin batera pasatzen. Hala ere, orain Car motako objektua pasatzen da, eta honek baditu barruan parametro hauek. Hortaz, hau konpontzeko besterik gabe Car klaseko getterrak erabiliko ditugu.

Hau da kode berria:

```
public boolean sortuKotxea(Car kotxeBerria) {
    Car kotxea = db.find(Car.class, kotxeBerria.getMatrikula());
    if(kotxea!=null) return false;
    db.getTransaction().begin();

    Driver d = kotxeBerria.getJabea();
    Car c = d.addCar(kotxeBerria.getMatrikula(), kotxeBerria.getEserKop(),
            kotxeBerria.getKolorea(), kotxeBerria.getModeloa());
    db.persist(c);
    db.merge(d);
    db.getTransaction().commit();
    return true;
}
```

Baina hemen ere metodoa deitzen den bakoitzean metodoaren signatura aldatu denez, deia ere aldatu egin behar da.

Adibidez, lehen BLImplementation klasean horrela zegoen:

```java
@WebMethod
public boolean sortuKotxea(String matrikula, int eserKop, String kolorea, String mota, Driver d) {
    dbManager.open();
    boolean b = dbManager.sortuKotxea(matrikula,eserKop,kolorea, mota,d);
    dbManager.close();
    return b;
}
```

Eta orain aldaketa hau egin zaio:

```java
@WebMethod
public boolean sortuKotxea(Car kotxea) {
    dbManager.open();
    boolean b = dbManager.sortuKotxea(kotxea);
    dbManager.close();
    return b;
}
```