Errefaktorizazioa

Urtzi Espinosa:

Write short units of code (gauzatuEragiketa metodoa)

1- Hasierako kodea

```
public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null && amount >= 0) { // amount negatiboa ez izatea
            double currentMoney = user.getMoney();
            if (deposit) {
                user.setMoney(currentMoney + amount);
            } else {
                if ((currentMoney - amount) < 0) {
                    user.setMoney(0);
                } else {
                    user.setMoney(currentMoney - amount);
                }
            }
            db.merge(user);
            db.getTransaction().commit();
            return true;
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        db.getTransaction().rollback();
        return false;
    }
}
```

2. Errefaktorizatuko kodea

```
public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null && amount >= 0) { // amount negatiboa ez izatea
            user = aldatuDirua(amount, deposit, user);
            db.merge(user);
            db.getTransaction().commit();
            return true;
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        db.getTransaction().rollback();
        return false;
    }
}

private User aldatuDirua(double amount, boolean deposit, User user) {
    double currentMoney = user.getMoney();
    if (deposit) {
        user.setMoney(currentMoney + amount);
    } else {
        if ((currentMoney - amount) < 0) {
            user.setMoney(0);
        } else {
            user.setMoney(currentMoney - amount);
        }
    }
    return user;
}
```

3. Egindako errefaktorizazioren deskribapena

gauzatuEragiketa-k hasieran 24 kode lerro zituen, gehiegi metodo bakarrarentzat. Hau konpontzeko, *Extract Method* egin diot kode zati bati, hori beste metodo pribatu batean exekutatzeko, eta horrela metodo originala laburrago geratu da.

Write simple units of code (equals metodoa User klasean)

1- Hasierako kodea

```java
@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    User other = (User) obj;
    if (username != other.username)
        return false;
    return true;
}
```

2. Errefaktorizatuko kodea

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (!(obj instanceof User)) return false;
    User other = (User) obj;
    return username.equals(other.username);
}
```

3. Egindako errefaktorizazioren deskribapena

User klaseko equals metodoan, lau if erabiltzen dira, konplexutasun ziklomatikoa bostekoa eginez. Hau erraz konpondu dezakegu, obj null den edo User klasekoa den ziurtatzeko bi if erabili ordez bakarrarekin nahikoa delako, eta amaierako if hori erabili ordez zuzenean baldintza betetzen den ala ez itzuli dezakeelako metodoak.

Duplicate code (bookFreeze)

1- Hasierako kodea

```
Duplication
Movement m1 = new Movement(traveler1, 1 "BookFreeze", 20);
Duplication
Movement m2 = new Movement(traveler1, 2 "BookFreeze", 40);
Duplication
Movement m3 = new Movement(traveler1, 3 "BookFreeze", 5);
Duplication
Movement m4 = new Movement(traveler2, 4 "BookFreeze", 4);
Duplication
Movement m5 = new Movement(traveler1, 5 "BookFreeze", 3);
Movement m6 = new Movement(driver1, "Deposit", 15);
Movement m7 = new Movement(traveler1, "Deposit", 168);
```

2. Errefaktorizatuko kodea

```
public class DataAccess {
    private static final String BOOK_FREEZE = "BookFreeze";
```

```
Movement m1 = new Movement(traveler1, BOOK_FREEZE, 20);
Movement m2 = new Movement(traveler1, BOOK_FREEZE, 40);
Movement m3 = new Movement(traveler1, BOOK_FREEZE, 5);
Movement m4 = new Movement(traveler2, BOOK_FREEZE, 4);
Movement m5 = new Movement(traveler1, BOOK_FREEZE, 3);
Movement m6 = new Movement(driver1, "Deposit", 15);
Movement m7 = new Movement(traveler1, "Deposit", 168);
```

3. Egindako errefaktorizazioren deskribapena

BookFreeze string-a bost aldiz agertzen denez kodean, aldagai konstante estatiko bat definitu dut DataAccess klasearentzako. Horrela, string-a aldatu nahi badugu, bost aldiz aldatu ordez behin bakarrik aldatu beharko dugu.

Keep unit interfaces small (erreklamazioaBidali metodoa)

1- Hasierako kodea

```
public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua,
        boolean aurk) {
    try {
        db.getTransaction().begin();

        Complaint erreklamazioa = new Complaint(nor, nori, gaur, booking, textua, aurk);
        db.persist(erreklamazioa);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

2. Errefaktorizatuko kodea

```
public boolean erreklamazioaBidali(ErreklamazioaBidaliParameter parameterObject) {
    try {
        db.getTransaction().begin();

        Complaint erreklamazioa = new Complaint(parameterObject.nor, parameterObject.nori, parameterObject.gaur,
                parameterObject.booking, parameterObject.textua, parameterObject.aurk);
        db.persist(erreklamazioa);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

3. Egindako errefaktorizazioren deskribapena

Hasierako kodeak 6 parametro jasotzen zituen, hau ez da oso egokia, kopuru handiegia delako, horretarako, *Introduce Parameter Object* egin dut eta parametro bakarrean bildu ditu aurreko guztiak. Honek ErreklamazioaBidaliParameter klasea sortu du dataAccess paketean, eta hemen metodoak behar dituen parametro guztiak atributu bezela gordetzen dira.

Aimar Errazkin:

Write short units of code (createRide metodoa)

1- Hasierako kodea

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName == null)
        return null;
    try {
        if (new Date().compareTo(date) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterT
        }

        db.getTransaction().begin();
        Driver driver = db.find(Driver.class, driverName);
        if (driver.doesRideExists(from, to, date)) {
            db.getTransaction().commit();
            throw new RideAlreadyExistException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
        }
        Ride ride = driver.addRide(from, to, date, nPlaces, price);
        // next instruction can be obviated
        db.persist(driver);
        db.getTransaction().commit();

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

2. Errefaktorizatuko kodea

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName == null)
        return null;
    try {
        if (new Date().compareTo(date) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterT
        }

        Ride ride = rideGorde(from, to, date, nPlaces, price, driverName);

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

3. Egindako errefaktorizazioren deskribapena

Metodo originala luzeegia zenez, ride objektua sortzen duen zatiari extract egin eta rideGorde deitu diot. Horrela 15 lerrotan geratzen da metodoa, askoz onargarriagoa den kopuru bat.

Write simple units of code (createRide metodoa)

1- Hasierako kodea

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName == null)
        return null;
    try {
        if (new Date().compareTo(date) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterT
        }

        Ride ride = rideGorde(from, to, date, nPlaces, price, driverName);

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

2. Errefaktorizatuko kodea

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName == null)
        return null;
    try {
        dataKonparatu(date);

        Ride ride = rideGorde(from, to, date, nPlaces, price, driverName);

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

3. Egindako errefaktorizazioren deskribapena

Aurreko aldaketa egin baina lehen, metodoaren konplexutasun ziklomatikoa gehiegizkoa zen. Lerro kopurua gutxitu ondoren hau hobetu dut pixka bat, baina hala ere gehiago egin daiteke. Horretarako, dataKonparatu metodoa sortu det , eta metodotik extraitu, horrela konplexutasun ziklomatikoa 4 bezala geratzen da.

Duplicate code (Accepted)

1- Hasierako kodea

```
    Duplication
    book1.setStatus("Accepted");
    book2.setStatus("Rejected");
    Duplication
    book3.setStatus("Accepted");
    Duplication
    book4.setStatus("Accepted");
    Duplication
    book5.setStatus("Accepted");
```

2. Errefaktorizatuko kodea

```
public class DataAccess {
    private static final String  ACCEPTED ="Accepted";
```

```
book1.setStatus(ACCEPTED);
book2.setStatus("Rejected");
book3.setStatus(ACCEPTED);
book4.setStatus(ACCEPTED);
book5.setStatus(ACCEPTED);
```

3. Egindako errefaktorizazioren deskribapena

“Accepted” string-a duen konstante bat sortu dut eta beste string guztiekin ordezkatu dut, string bera hainbat alditan duplikatua egon beharrean egokiagoa da horrela jartzea.

Keep unit interfaces small (createRide metodoa)

1- Hasierako kodea

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName == null)
        return null;
    try {
        dataKonparatu(date);

        Ride ride = rideGorde(from, to, date, nPlaces, price, driverName);

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

2. Errefaktorizatuko kodea

```
public Ride createRide(CreateRideParameter parameterObject)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(">> DataAccess: createRide=> from= " + parameterObject.from + " to= " + parameterObject.to + " driver=" + parameterObj
    if (parameterObject.driverName == null)
        return null;
    try {
        dataKonparatu(parameterObject.date);

        Ride ride = rideGorde(parameterObject.from, parameterObject.to, parameterObject.date, parameterObject.nPlaces, parameterObject.price,

        return ride;
    } catch (NullPointerException e) {
        if (db.getTransaction().isActive())
            db.getTransaction().rollback();
        return null;
    }

}
```

3. Egindako errefaktorizazioren deskribapena

Hasieran 6 parametro jasotzen ziren metodoan, kopuru handiegi bat. Hau konpontzeko, *Introduce Parameter Object* eginez parametro bakarrean geratu dira bilduta aurreko guztiak. Aldaketaren ondoren, dataAccess paketean CreateRideParameter klasea sortu zaigu, metodo originalaren parametro guztiak atributu bezala gordetzen dituena.

Dani Delgado:

Write short units of code eta Write simple units of code (deleteUser)

1- Hasierako kodea

```
public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {
            List<Ride> rl = getRidesByDriver(us.getUsername());
            if (rl != null) {
                for (Ride ri : rl) {
                    cancelRide(ri);
                }
            }
            Driver d = getDriver(us.getUsername());
            List<Car> cl = d.getCars();
            if (cl != null) {
                for (int i = cl.size() - 1; i >= 0; i--) {
                    Car ci = cl.get(i);
                    deleteCar(ci);
                }
            }
        } else {
            List<Booking> lb = getBookedRides(us.getUsername());
            if (lb != null) {
                for (Booking li : lb) {
                    li.setStatus("Rejected");
                    li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
                }
            }
            List<Alert> la = getAlertsByUsername(us.getUsername());
            if (la != null) {
                for (Alert lx : la) {
                    deleteAlert(lx.getAlertNumber());
                }
            }
        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

2. Errefaktorizatuko kodea

```
public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {
            cancelRides(us);
            deleteCars(us);
        } else {
            rejectRides(us);
            deleteAlerts(us);
        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public void deleteAlerts(User us) {
    List<Alert> la = getAlertsByUsername(us.getUsername());
    if (la != null) {
        for (Alert lx : la) {
            deleteAlert(lx.getAlertNumber());
        }
    }
}
```

```
882     public void rejectRides(User us) {
883         List<Booking> lb = getBookedRides(us.getUsername());
884         if (lb != null) {
885             for (Booking li : lb) {
886                 li.setStatus("Rejected");
887                 li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats()
888             }
889         }
890     }
891
892     public void deleteCars(User us) {
893         Driver d = getDriver(us.getUsername());
894         List<Car> cl = d.getCars();
895         if (cl != null) {
896             for (int i = cl.size() - 1; i >= 0; i--) {
897                 Car ci = cl.get(i);
898                 deleteCar(ci);
899             }
900         }
901     }
902
903     public void cancelRides(User us) {
904         List<Ride> rl = getRidesByDriver(us.getUsername());
905         if (rl != null) {
906             for (Ride ri : rl) {
907                 cancelRide(ri);
908             }
909         }
910     }
```

3. Egindako errefaktorizazioren deskribapena

Metodo honetan, bi bad smell zeuden. Alde batetik oso luzea zela, eta bestetik, konplexutasun ziklomatikoa 11koa zela, gomendatutakoa baino askoz handiagoa. Bi arazo hauek sahiesteko, if eta for konibinazio horiek hartu eta kodetik atera ditut, lau metodo berri sortuz.

Duplicate code (username)

1- Hasierako kodea

```
        query.setParameter( 1 "username", erab);
        Double money = query.getSingleResult();
        if (money != null) {
            return money;
        } else {
            return 0;
        }
    }

    public boolean isRegistered(String erab, String passwd) {
        TypedQuery<Long> travelerQuery = db.createQuery(
                "SELECT COUNT(t) FROM Traveler t WHERE t.username = :usern
        Duplication
        travelerQuery.setParameter( 2 "username", erab);
        travelerQuery.setParameter("passwd", passwd);
        Long travelerCount = travelerQuery.getSingleResult();

        TypedQuery<Long> driverQuery = db.createQuery(
                "SELECT COUNT(d) FROM Driver d WHERE d.username = :usernar
        Duplication
        driverQuery.setParameter( 3 "username", erab);
        driverQuery.setParameter("passwd", passwd);
        Long driverCount = driverQuery.getSingleResult();

        boolean isAdmin = ((erab.compareTo("admin") == 0) && (passwd.compa
        return travelerCount > 0 || driverCount > 0 || isAdmin;
    }

    public Driver getDriver(String erab) {
        TypedQuery<Driver> query = db.createQuery("SELECT d FROM Driver d
        Duplication
        query.setParameter( 4 "username", erab);
        List<Driver> resultList = query.getResultList();
        if (resultList.isEmpty()) {
            return null;
```

2. Errefaktorizatuko kodea

```
public class DataAccess {
    private static final String  USERNAME="username";
```

3. Egindako errefaktorizazioren deskribapena

“Username” stringa 12 aldiz errepikatzen da dataAccess osoan, beraz username atributu estatiko konstantea sortu dut, eta hau jarri dut string-ak zeuden tokian.

Keep unit interfaces small (addRide metodoa Driver klasean)

1- Hasierako kodea

```java
    /**
     * This method creates a bet with a question, minimum bet ammount and percentual
     * profit
     *
     * @param question   to be added to the event
     * @param betMinimum of that question
     * @return Bet
     */
    public Ride addRide(String from, String to, Date date, int nPlaces, float price) {
        Ride ride = new Ride(from, to, date, nPlaces, price, this);
        createdRides.add(ride);
        return ride;
    }
```

2. Errefaktorizatuko kodea

```java
    public Ride addRide(AddRideParameter parameterObject) {
        Ride ride = new Ride(parameterObject.from, parameterObject.to, parameterObject.date, parameterObject.nPlaces, parameterObject.price, this);
        createdRides.add(ride);
        return ride;
    }
```

```java
package domain;

import java.util.Date;

public class AddRideParameter {
    public String from;
    public String to;
    public Date date;
    public int nPlaces;
    public float price;

    public AddRideParameter(String from, String to, Date date, int nPlaces, float price) {
        this.from = from;
        this.to = to;
        this.date = date;
        this.nPlaces = nPlaces;
        this.price = price;
    }
}
```

3. Egindako errefaktorizazioren deskribapena

Driver klaseko addRide metodoak 5 atributu jasotzen ditu. Hau kentzeko, parameter object erabiliz, AddRideParameter klasea sortzen da, eta hau izango da addRide funtzioak jasoko duen parametro bakarra.