eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

REFACTOR

Software ingeniaritza

Joritz Arocena eta Mikel Carrasco

25-26 ikasturtea

AURKIBIDEA

1."Write short units of code" (2. kapituloa)

1.BookRide() metodoa 15 lerrotara mugatu

1.1 Hasierako kodea

```java
public boolean bookRide(String username, Ride ride, int seats, double desk) {
    try {
        db.getTransaction().begin();

        Traveler traveler = getTraveler(username);
        if (traveler == null) {
            return false;
        }

        if (ride.getnPlaces() < seats) {
            return false;
        }

        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        double availableBalance = traveler.getMoney();
        if (availableBalance < ridePriceDesk) {
            return false;
        }

        Booking booking = new Booking(ride, traveler, seats);
        booking.setTraveler(traveler);
        booking.setDeskontua(desk);
        db.persist(booking);

        ride.setnPlaces(ride.getnPlaces() - seats);
        traveler.addBookedRide(booking);
        traveler.setMoney(availableBalance - ridePriceDesk);
        traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
        db.merge(ride);
        db.merge(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

1.2 Errefaktorizatutako kodea

```java
private Booking bookingErazagutu(Ride ride, Traveler traveler, int seats, double desk) {
    Booking b= new Booking(ride, traveler, seats);
    b.setTraveler(traveler);
    b.setDeskontua(desk);
    return b;
}
private Traveler travelerAldaketak(Traveler t, Booking b, double rideprice, double balance) {
    t.addBookedRide(b);
    t.setMoney(balance-rideprice);
    t.setIzoztatutakoDirua(t.getIzoztatutakoDirua() + rideprice);
    return t;
}

public boolean bookRide(String username, Ride ride, int seats, double desk) {
    try {
        db.getTransaction().begin();
        Traveler t = getTraveler(username);
        if (t == null || ride.getnPlaces() < seats) return false;
        double price = (ride.getPrice() - desk) * seats;
        if (t.getMoney() < price) return false;
        Booking b = bookingErazagutu(ride, t, seats, desk);
        db.persist(b);
        ride.setnPlaces(ride.getnPlaces() - seats);
        travelerAldaketak(t, b, price, t.getMoney());
        db.merge(ride); db.merge(t);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace(); db.getTransaction().rollback();
        return false;
    }

}
```

1.3 Errefaktorizazioaren deskribapena

Metodo honek ez zuen betetzen 15 lerro baino gutxiago izateko baldintza, beraz metodo honen lerro kopurua murriztu da. Horretarako aparteko bi metodo sortu ditut BookingErazagutu eta traveler aldaketak. Horrela aurreko bertsioan bookRide barnean egiten ziren aldaketa horiek guztiak kanpo metodo batean egiten dira, horrela kode kopurua murriztuz metodoan.

Bestalde, hasiera bi if elkarren segida dauden eta biak bateratu egin ditut, horrela lau lerro hartzen zituena bakar batera pasatu dugu. Aldaketa hauek eginda 15 lerro baino gutxiago izatea lortu dugu bookRide() metodoan.

1.4 Egilea

Joritz Arocena Zuriarrain

2. cancelRide(Ride) metodoa 15 kode lerrotara mugatu

2.1 Hasierako egoera

```
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();

        for (Booking booking : ride.getBookings()) {
            if (booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined")) {
                double price = booking.prezioaKalkulatu();
                Traveler traveler = booking.getTraveler();
                double frozenMoney = traveler.getIzoztatutakoDirua();
                traveler.setIzoztatutakoDirua(frozenMoney - price);

                double money = traveler.getMoney();
                traveler.setMoney(money + price);
                db.merge(traveler);
                db.getTransaction().commit();
                addMovement(traveler, "BookDeny", price);
                db.getTransaction().begin();
            }
            booking.setStatus("Rejected");
            db.merge(booking);
        }
        ride.setActive(false);
        db.merge(ride);

        db.getTransaction().commit();
    } catch (Exception e) {
        if (db.getTransaction().isActive()) {
            db.getTransaction().rollback();
        }
        e.printStackTrace();
    }
}
```

2.2 Errefaktorizatutako kodea

```
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();
        for (Booking booking : ride.getBookings()) {
            if (booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined")) {
                acceptOrNotdefined(booking);
            }
            booking.setStatus("Rejected");
            db.merge(booking);
        }
        ride.setActive(false);
        db.merge(ride);

        db.getTransaction().commit();
    } catch (Exception e) {
        salbuespenaCancelRide(e);
    }
}
public void acceptOrNotdefined(Booking booking) {
    double price = booking.prezioaKalkulatu();
    Traveler traveler = booking.getTraveler();
    double frozenMoney = traveler.getIzoztatutakoDirua();
    traveler.setIzoztatutakoDirua(frozenMoney - price);

    double money = traveler.getMoney();
    traveler.setMoney(money + price);
    db.merge(traveler);
    db.getTransaction().commit();
    addMovement(traveler, "BookDeny", price);
    db.getTransaction().begin();
}
public void salbuespenaCancelRide(Exception e) {
    if (db.getTransaction().isActive()) {
        db.getTransaction().rollback();
    }
    e.printStackTrace();

}
```

2.3 Errefaktorizazioaren deskribapena

Aukeratutako cancelRide(Ride ride) metodoak, 15 kode-lerro baino gehiago zituen.

Hau arazo bat da, izan ere, kodearen matenua zailtzen du. Hori ekiditeko,cancelRide metodoak beste bi metodori egingo dio deia, bi kasu zehatzetan; lehendabiziko kasua, erreserbaren egoera accepted edo NotDefined denean. Bigarren kasuan berriz salbuespena altzatu behar denean.

2.4 Egilea

Mikel Carrasco Tajo

2. "Write simple units of code" (3. kapituloa)

1. deleteUser metodoa

    1.1 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();
    }
}
```

1.2 Errefaktorizatutako kodea

```
public void deleteUser(User erabiltzailea) {
    try {
        db.getTransaction().begin();

        if (erabiltzailea.getMota().equals("Driver")) {
            gidariaEzabatu(erabiltzailea);
        } else {
            bidaiariaEzabatu(erabiltzailea);
        }

        erabiltzailea = db.merge(erabiltzailea);
        db.remove(erabiltzailea);

        db.getTransaction().commit();
    } catch (Exception e) {
        if (db.getTransaction().isActive()) db.getTransaction().rollback();
        e.printStackTrace();
    }
}
```

```
private void gidariaEzabatu(User erabiltzailea) {
    List<Ride> ibilaldiak = getRidesByDriver(erabiltzailea.getUsername());
    if (ibilaldiak != null) {
        for (Ride ibilaldia : ibilaldiak) {
            cancelRide(ibilaldia);
        }
    }

    Driver gidaria = getDriver(erabiltzailea.getUsername());
    if (gidaria != null && gidaria.getCars() != null) {
        for (Car autoa : new ArrayList<>(gidaria.getCars())) {
            deleteCar(autoa);
        }
    }
}
```

```
private void bidaiariaEzabatu(User erabiltzailea) {
    List<Booking> erreserbak = getBookedRides(erabiltzailea.getUsername());
    if (erreserbak != null) {
        for (Booking erreserba : erreserbak) {
            erreserba.setStatus("Rejected");
            Ride ibilaldia = erreserba.getRide();
            ibilaldia.setnPlaces(ibilaldia.getnPlaces() + erreserba.getSeats());
            db.merge(ibilaldia);
        }
    }

    List<Alert> alertak = getAlertsByUsername(erabiltzailea.getUsername());
    if (alertak != null) {
        for (Alert alerta : alertak) {
            deleteAlert(alerta.getAlertNumber());
        }
    }
}
```

1.3 Errefaktorizazioaren deskribapena

Kasu honetan deleteUser metodoak ez zuen betetzen metodo bakoitzak responsabilitate bat eta bakarra edukitzea. Beraz, egin dudana da bi metodo laguntzaile sortu, batek traveler borratzeko balio du eta bezteak driver-ak borratzeko. Horrela metodo bakoitzak responsabilitate bat edukiko du.

1.4 Egilea

Joritz Arocena Zuriarrain

2.UpdateAlertaAurkituak metodoa

2.1 Hasierako kodea

```
public boolean updateAlertaAurkituak(String username) {
    try {
        db.getTransaction().begin();

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
                Alert.class);
        alertQuery.setParameter("username", username);
        List<Alert> alerts = alertQuery.getResultList();

        TypedQuery<Ride> rideQuery = db
                .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
        List<Ride> rides = rideQuery.getResultList();

        for (Alert alert : alerts) {
            boolean found = false;
            for (Ride ride : rides) {
                if (UtilDate.datesAreEqualIgnoringTime(ride.getDate(), alert.getDate())
                        && ride.getFrom().equals(alert.getFrom()) && ride.getTo().equals(alert.getTo())
                        && ride.getnPlaces() > 0) {
                    alert.setFound(true);
                    found = true;
                    if (alert.isActive())
                        alertFound = true;
                    break;
                }
            }
            if (!found) {
                alert.setFound(false);
            }
            db.merge(alert);
        }

        db.getTransaction().commit();
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

2.2 Errefaktorizatutako kodea

```java
public boolean updateAlertaAurkituak(String username) {
    try {
        db.getTransaction().begin();

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
                Alert.class);
        alertQuery.setParameter("username", username);
        List<Alert> alerts = alertQuery.getResultList();

        TypedQuery<Ride> rideQuery = db
                .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
        List<Ride> rides = rideQuery.getResultList();

        alertFound=alertenProzesaketa(alerts,rides);
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}

public boolean alertenProzesaketa(List<Alert> alerts, List<Ride> rides) {
    boolean alertFound = false;
    for (Alert alert : alerts) {
        boolean found = findMatchingRide(alert, rides);
        alert.setFound(found);
        if (found && alert.isActive()) alertFound = true;
        db.merge(alert);
    }
    return alertFound;
}
public boolean findMatchingRide(Alert alert, List<Ride> rides) {
    for (Ride ride : rides) {
        if (UtilDate.datesAreEqualIgnoringTime(ride.getDate(), alert.getDate())
                && ride.getFrom().equals(alert.getFrom()) && ride.getTo().equals(alert.getTo())
                && ride.getnPlaces() > 0) return true;
    }
    return false;
}
```

2.3 Errefaktorizazioaren deskribapena

UpdateAlertaAurkituak izeneko metodoan, konplexutasun ziklomatikoa lau baino handiagoa da. Hori konpontzeko, bi metodo gehigarri sortu ditugu; alertenProzesaketa, alerta bakoitza hartu eta hurrengo metodoari(findMatchRide) deituta, alerta horri dagokion bidaiarik aurkitu duen esango digu. Lehen metodoak bigarrengoari egiten dio dei eta era berean metodo nagusiak(updateAlerta Aurkituak) lehen metodoari egingo dio dei.

2.4 Egilea

Mikel Carrasco Tajo

3.“Duplicate code” (4. kapituloa)

1. Kode duplikazioak ekidin alertekin

1.1 Hasierako kodea

```
public boolean updateAlertaAurkituak(String username) {
    try {
        db.getTransaction().begin();

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
                Alert.class);
        alertQuery.setParameter("username", username);
        List<Alert> alerts = alertQuery.getResultList();

        TypedQuery<Ride> rideQuery = db
                .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
        List<Ride> rides = rideQuery.getResultList();

        alertFound=alertenProzesaketa(alerts,rides);
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

1.2 Errefaktorizatutako kodea

```
public boolean updateAlertaAurkituak(String username) {
    try {
        db.getTransaction().begin();

        boolean alertFound = false;

        List<Alert> alerts =getAlertsByUsername(username);

        TypedQuery<Ride> rideQuery = db
                .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
        List<Ride> rides = rideQuery.getResultList();

        alertFound=alertenProzesaketa(alerts,rides);
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

1.3 Errefaktorizazioaren deskribapena

Aurreko kapituluan eraldatu dugun metodo honek, beste metodo batean egikarituta dagoen query bat burutzen du. Ondorioz, kode errepikatua edo “Duplicate code” delakoa sortzen da. Konpontzeko, getAlertsByUsername-i deitu zaio, metodo horrek erabiltzailearen izen emanda bere alerta guztiak lortzen baititu.

1.4 Egilea

Mikel Carrasco Tajo

2. Updaten kode errepikana ekidin

2.1 Hasierako kodea

```
public void updateTraveler(Traveler traveler) {
    try {
        db.getTransaction().begin();
        db.merge(traveler);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void updateDriver(Driver driver) {
    try {
        db.getTransaction().begin();
        db.merge(driver);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void updateUser(User user) {
    try {
        db.getTransaction().begin();
        db.merge(user);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

2.2 Errefaktorizatutako kodea

```
private <T> void updateEntity(T entity) {
    try {
        db.getTransaction().begin();
        db.merge(entity);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        if (db.getTransaction().isActive()) db.getTransaction().rollback();
    }
}

public void updateTraveler(Traveler traveler) {
    updateEntity(traveler);
}

public void updateDriver(Driver driver) {
    updateEntity(driver);
}

public void updateUser(User user) {
    updateEntity(user);
}
```

2.3 Errefaktorizazioaren azalpena

Kasu honetan metodo berdin asko errepikatzen ziren update-ak egiterakoan. Beraz metodo generiko bat sortu dugu eta horrela kodearen errepikapena ekiditen dugu.

2.4 Egilea

Joritz Arocena Zuriarrain

4."Keep unit interfaces small" (5. kapituloa)

Oharra: Bad smell zati honetako adibide bakarra aurkitu dugu.

1.createRide metodoa

1.1 Hasierako kodea

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

        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) {
        // TODO Auto-generated catch block
        return null;
    }


}
```

1.2 Errefaktorizatutako kodea

Metodoa:

```
public Ride createRide(RideRequest parameterObject)
                        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
                logger.info(
                                        ">> DataAccess: createRide=> from= " + parameterObject.getFrom() + " to= "
+ parameterObject.getTo() + " driver=" + parameterObject.getDriverName() + " date " +
parameterObject.getDate());
                if (parameterObject.getDriverName()==null) return null;
                try {
                        if (new Date().compareTo(parameterObject.getDate()) > 0) {
                                logger.info("ppppp");
                                throw new RideMustBeLaterThanTodayException(

ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
                        }
                        db.getTransaction().begin();
                        Driver driver = db.find(Driver.class, parameterObject.getDriverName());
                        if (driver.doesRideExists(parameterObject.getFrom(), parameterObject.getTo(),
parameterObject.getDate())) {
                                db.getTransaction().commit();
                                throw new RideAlreadyExistException(

ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
                        }
Ride ride = driver.addRide(parameterObject.getFrom(), parameterObject.getTo(), parameterObject.getDate(),
parameterObject.getnPlaces(), parameterObject.getPrice());
                        // next instruction can be obviated
                        db.persist(driver);
                        db.getTransaction().commit();
                        return ride;
                } catch (NullPointerException e) {
```

```
                                // TODO Auto-generated catch block
                                return null;
                        }

                }
```

Objektu berria:

```
public class RideRequest {
        private String from;
        private String to;
        private Date date;
        private int nPlaces;
        private float price;
        private String driverName;
        public RideRequest(String from, String to, Date date, int nPlaces, float price, String driverName) {
                this.from = from;
                this.to = to;
                this.date = date;
                this.nPlaces = nPlaces;
                this.price = price;
                this.driverName = driverName;
        }
        public String getFrom() {
                return from;
        }
        public void setFrom(String from) {
                this.from = from;
        }
        public String getTo() {
                return to;
        }
        public void setTo(String to) {
                this.to = to;
        }
        public Date getDate() {
                return date;
        }
        public void setDate(Date date) {
                this.date = date;
        }
        public int getnPlaces() {
                return nPlaces;
        }
        public void setnPlaces(int nPlaces) {
                this.nPlaces = nPlaces;
        }
        public float getPrice() {
                return price;
        }
        public void setPrice(float price) {
                this.price = price;
        }
        public String getDriverName() {
                return driverName;
        }
        public void setDriverName(String driverName) {
                this.driverName = driverName;
        }
}
```

1.3 Errefaktorizazioaren deskribapena

Metodo honek ez du jarraitzen 5.kapituluko baldintza, hau da, ez dauzka 4 parametro baino gutxiago. Beraz, hori konpontzeko metodoari parametro

bezala objektu bat pasatuko diogu eta objektu horretan gordeko ditugu parametro guztiak. Horrela baldintza beteko du.

1.4 Egilea

Joritz Arocena Zuriarrain