Software Ingeniaritza II

REFACTOR lab

Iván González

Beñat Arruti

Mikel Berlanga

Eneko Santos

IVAN

"Write short units of code":

Hasierako kodea:

```
public boolean addDriver(String username, String password) {
try {
db.getTransaction().begin();
Driver existingDriver = getDriver(username);
Traveler existingTraveler = getTraveler(username);
if (existingDriver != null || existingTraveler != null) {
return false;
}

Driver driver = new Driver(username, password);
db.persist(driver);
db.getTransaction().commit();
return true;
} catch (Exception e) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}
}
```

Errefaktorizatuko kodea:

```
public boolean addDriver(String username, String password) {
    try {
        db.getTransaction().begin();

        if (!isUsernameAvailable(username)) return false;

        persistNewDriver(username, password);

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

private boolean isUsernameAvailable(String username) {
    return getDriver(username) == null && getTraveler(username) == null;
}

private void persistNewDriver(String username, String password) {
    Driver driver = new Driver(username, password);
    db.persist(driver);
}
```

Egindako errefaktorizazioren deskribapena:

Metodoa 15 lerro baino gehiago zuenez, bi funtzio laguntzaile sortu dira bad smell hau konpondu ahal izateko.

"Write simple units of code":

Hasierako kodea:w

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

        Traveler traveler = getTraveler(username);        // 1
        if (traveler == null) {                            // 2
            return false;
        }

        if (ride.getnPlaces() < seats) {                   // 3
            return false;
        }

        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        double availableBalance = traveler.getMoney();
        if (availableBalance < ridePriceDesk) {           // 4
            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;
    }
}
```

Errefaktorizatuko kodea:

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

        Traveler traveler = getTraveler(username);
        if (!isBookingValid(traveler, ride, seats, desk)) return false;

        Booking booking = createBooking(traveler, ride, seats, desk);
        updateBalancesAndSeats(traveler, ride, booking, desk);

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

private boolean isBookingValid(Traveler traveler, Ride ride, int seats, double
desk) {
    if (traveler == null) return false;
    if (ride.getnPlaces() < seats) return false;
    double totalPrice = (ride.getPrice() - desk) * seats;
    return traveler.getMoney() >= totalPrice;
}

private Booking createBooking(Traveler traveler, Ride ride, int seats, double
desk) {
    Booking booking = new Booking(ride, traveler, seats);
    booking.setTraveler(traveler);
    booking.setDeskontua(desk);
    db.persist(booking);
    return booking;
}

private void updateBalancesAndSeats(Traveler traveler, Ride ride, Booking booking,
double desk) {
    double totalPrice = (ride.getPrice() - desk) * booking.getSeats();
    ride.setnPlaces(ride.getnPlaces() - booking.getSeats());
    traveler.addBookedRide(booking);
    traveler.setMoney(traveler.getMoney() - totalPrice);
    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + totalPrice);
    db.merge(ride);
    db.merge(traveler);
}
```

Egindako errefaktorizazioren deskribapena:

Konplexutasun zinematikoa 4 baino handiagoa zenez, metodo laguntzailea sortu da, bad smell-a konpondu da.

"Duplicate code":

Hasierako kodea:

```
public Driver getDriver(String erab) {
    TypedQuery<Driver> query = db.createQuery("SELECT d FROM Driver d WHERE
d.username = :username", Driver.class);
    query.setParameter("username", erab);
    List<Driver> resultList = query.getResultList();
    if (resultList.isEmpty()) {
        return null;
    } else {
        return resultList.get(0);
    }
}


public Traveler getTraveler(String erab) {
    TypedQuery<Traveler> query = db.createQuery("SELECT t FROM Traveler t WHERE
t.username = :username", Traveler.class);
    query.setParameter("username", erab);
    List<Traveler> resultList = query.getResultList();
    if (resultList.isEmpty()) {
        return null;
    } else {
        return resultList.get(0);
    }
}
```

Errefaktorizatuko kodea:

```
private <T extends User> T getUserByUsername(Class<T> clazz, String username) {
    TypedQuery<T> query = db.createQuery(
        "SELECT u FROM " + clazz.getSimpleName() + " u WHERE u.username =
:username", clazz);
    query.setParameter("username", username);
    List<T> resultList = query.getResultList();
    return resultList.isEmpty() ? null : resultList.get(0);
}
public Driver getDriver(String erab) {
    return getUserByUsername(Driver.class, erab);
}

public Traveler getTraveler(String erab) {
    return getUserByUsername(Traveler.class, erab);
}
```

Egindako errefaktorizazioren deskribapena:

Nola bi getter-etan kodea errepikatzen zen (bad smell bat zegoen), kodea errefaktorizatu da.

" Keep unit interfaces small ":

Hasierako kodea:

```
public Ride createRide(String from, String to, Date date, int nPlaces, float
price, String driverName) throws RideAlreadyExistException,
RideMustBeLaterThanTodayException {
 validateRideCreationInputs(date, driverName);
 Driver driver = findDriver(driverName);
 db.getTransaction().begin();
 checkRideDoesNotExist(driver, from, to, date);
 Ride ride = persistNewRide(driver, from, to, date, nPlaces, price);
 db.getTransaction().commit();
 return ride;
}
```

Errefaktorizatuko kodea:

```
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 String getTo() { return to; }
    public Date getDate() { return date; }
    public int getNPlaces() { return nPlaces; }
    public float getPrice() { return price; }
    public String getDriverName() { return driverName; }
}
public Ride createRide(RideRequest request)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {

    validateRideCreationInputs(request.getDate(), request.getDriverName());
    Driver driver = findDriver(request.getDriverName());

    db.getTransaction().begin();
    checkRideDoesNotExist(driver, request.getFrom(), request.getTo(),
request.getDate());
    Ride ride = persistNewRide(driver, request.getFrom(), request.getTo(),
request.getDate(),
                               request.getNPlaces(), request.getPrice());
    db.getTransaction().commit();

    return ride;
}
```

Egindako errefaktorizazioren deskribapena:

Kodea errefaktorizatu da parametro bakarra izateko funtzioa (bad smell-ak eliminatuz)

MIKEL

"Write short units of code":

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.ErrorRideMustBeLate
rThanToday"));
}
                   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.RideAlreadyExis
t"));
}
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) {
return null;
}
}
```

Errefaktorizatuko kodea:

```
public Ride createRide(String from, String to, Date date, int nPlaces, float
price, String driverName) throws RideAlreadyExistException,
RideMustBeLaterThanTodayException {
 validateRideCreationInputs(date, driverName);
 Driver driver = findDriver(driverName);
 db.getTransaction().begin();
 checkRideDoesNotExist(driver, from, to, date);
 Ride ride = persistNewRide(driver, from, to, date, nPlaces, price);
 db.getTransaction().commit();
 return ride;
}

private void validateRideCreationInputs(Date date, String driverName) throws
RideMustBeLaterThanTodayException {
 if (driverName == null) {
       throw new IllegalArgumentException("Driver name cannot be null");
 }
 if (new Date().compareTo(date) > 0) {
       throw new RideMustBeLaterThanTodayException(
       ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMus
tBeLaterThanToday"));
 }
}

private Driver findDriver(String driverName) {
return db.find(Driver.class, driverName);
}

private void checkRideDoesNotExist(Driver driver, String from, String to, Date
date)
throws RideAlreadyExistException {
if (driver.doesRideExists(from, to, date)) {
db.getTransaction().commit();
       throw new RideAlreadyExistException(
       ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExis
t"));
 }
}

private Ride persistNewRide(Driver driver, String from, String to, Date date, int
nPlaces, float price) {
 Ride ride = driver.addRide(from, to, date, nPlaces, price);
 db.persist(driver);
 return ride;
}
```

Egindako errefaktorizazioren deskribapena:

createRide metodoak 20 lerro baino gehiago ditu. Hau konpontzeko hoberena da metodoaren kanpo, metodo pribatuak egitea. Metodo hoietan egiten dena da createRide metodoaren konprobaketak egitea kanpoan eta metodo pribatu horiek deitzea createRide metodoaren barruan ez ukitzeko hainbat lerro.

"Write simple units of code":

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());
}
}
```

```java
}
db.getTransaction().begin();
us = db.merge(us);
db.remove(us);
db.getTransaction().commit();
} catch (Exception e) {
e.printStackTrace();
}
}
```

Errefaktorizatuko kodea:

```java
public void deleteUser(User us) {

try {
if (us.getMota().equals("Driver")) {
deleteDriverUser(us);
} else {
deleteTravelerUser(us);
}
removeUserFromDatabase(us);
} catch (Exception e) {
e.printStackTrace();
}
}

private void deleteDriverUser(User user) {
String username = user.getUsername();
List<Ride> rides = getRidesByDriver(username);
if (rides != null) {
for (Ride ride : rides) {
cancelRide(ride);
}
}
Driver driver = getDriver(username);
List<Car> cars = driver.getCars();
if (cars != null) {
for (int i = cars.size() - 1; i >= 0; i--) {
deleteCar(cars.get(i));
}
}
}

private void deleteTravelerUser(User user) {
String username = user.getUsername();
List<Booking> bookings = getBookedRides(username);
if (bookings != null) {
for (Booking booking : bookings) {
rejectBooking(booking);
}
}
List<Alert> alerts = getAlertsByUsername(username);
if (alerts != null) {
for (Alert alert : alerts) {
deleteAlert(alert.getAlertNumber());
}
}
}

private void rejectBooking(Booking booking) {
      booking.setStatus("Rejected");
Ride ride = booking.getRide();
ride.setnPlaces(ride.getnPlaces() + booking.getSeats());
}

private void removeUserFromDatabase(User user) {
db.getTransaction().begin();
user = db.merge(user);
db.remove(user);
db.getTransaction().commit();
}
```

Egindako errefaktorizazioren deskribapena:

deleteUser metodo originalak adarkatze konplexua zuen, hainbat erabiltzaile motarentzako baldintza habiadun eta begiztekin. Horregatik, aparte metodoak egin dut deleteUser ekintza bakoitza egiteko eta karga kentzeko.

"Duplicate code":

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();
}
}
```

Errefaktorizatuko kodea:

```
public <T> boolean updateEntity(T entity) {
try {
db.getTransaction().begin();
db.merge(entity);
db.getTransaction().commit();
return true;
} catch (Exception e) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}
}

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

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

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

Egindako errefaktorizazioren deskribapena:

Gauza bera baina entitate desberdinekin egiten duten hiru metodo desberdin izan beharrean, mota generiko bat sartzen den metodo orokor bat egiten dugu, eta metodo horretaz gain, beste batzuk egiten ditugu zer entitate sartzen zaigun jakiteko.

"Keep unit interfaces small":

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;
}
}
```

Errefaktorizatuko kodea:

```
public boolean erreklamazioaBidali(ComplaintData data) {
try {
db.getTransaction().begin();
Complaint erreklamazioa = new Complaint(
data.getSender(),
data.getReceiver(),
data.getDate(),
data.getBooking(),
data.getText(),
data.isAurk()
);
db.persist(erreklamazioa);
db.getTransaction().commit();
return true;
} catch (Exception e) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}
}


public class ComplaintData {
private String sender;
private String receiver;
private Date date;
private Booking booking;
private String text;
private boolean aurk;
// Erakitzailea, getterrak eta setterrak
}
```

Egindako errefaktorizazioren deskribapena:

Egin duguna da klase berri bat sortu non metodoak eskatzen dituen parametro guztiak sartu ditugu. Eta gero erraklamazioBidali metodoari sartzen diogo objektu bat klase berri horretakoa eta bere getterrak eta eraikitzailea erabiltzen ditugu.

ENEKO

"Write short units of code":

Hasierako kodea.

```java
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();
    }
}
```

Errefaktorizatuko kodea:

```java
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();
        for (Booking booking : ride.getBookings()) {
            if ("Accepted".equals(booking.getStatus()) || "NotDefined".equals(booking.getStatus()))
                kudeatuOnartutakoErreserba(booking);
            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();
    }
}

private void kudeatuOnartutakoErreserba(Booking booking) {
    double price = booking.prezioaKalkulatu();
    Traveler traveler = booking.getTraveler();
    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() - price);
    traveler.setMoney(traveler.getMoney() + price);
    db.merge(traveler);
    db.getTransaction().commit();
    addMovement(traveler, "BookDeny", price);
    db.getTransaction().begin();
}
```

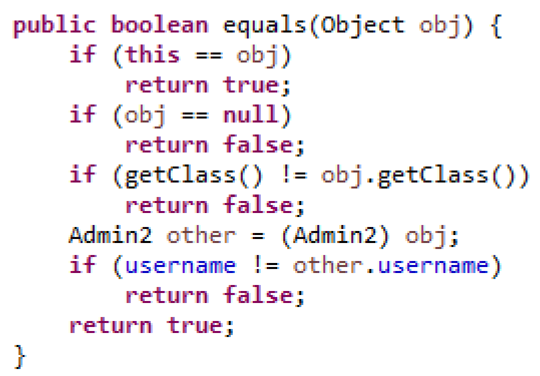
Egindako errefaktorizazioren deskribapena

15 lerro baino gehiagoko metodo bat ez edukitzeko, onartutako erreserben kudeaketa metodo laguntzaile batera mugitu dut.

“Write simple units of code”

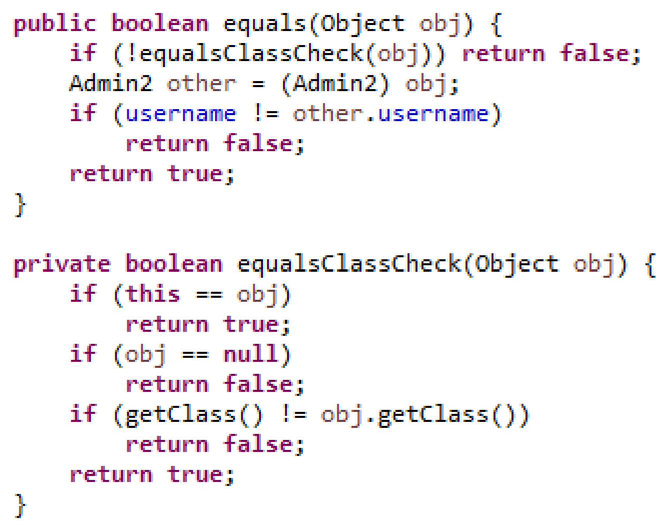
(Ez dira geratzen konplexutasun ziklomatikoa >4 eta errefaktorizatuta ez dauden metodoak DataAccess-ean, beraz, Admin2-ko equals metodoa errefaktorizatuko dut)

Hasierako kodea.



```java
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Admin2 other = (Admin2) obj;
    if (username != other.username)
        return false;
    return true;
}
```

Errefaktorizatuko kodea:



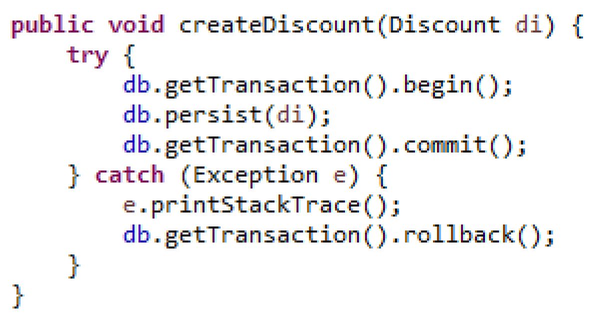
```java
public boolean equals(Object obj) {
    if (!equalsClassCheck(obj)) return false;
    Admin2 other = (Admin2) obj;
    if (username != other.username)
        return false;
    return true;
}

private boolean equalsClassCheck(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    return true;
}
```

Egindako errefaktorizazioren deskribapena

Konplexutasun ziklomatikoa 5 zen, beraz, metodo laguntzaileak erabiliz, metodoa sinplifikatu det beste metodoetan banatzen. Kasu honetan, egiaztapen basikoak beste metodo batean egin ditut.

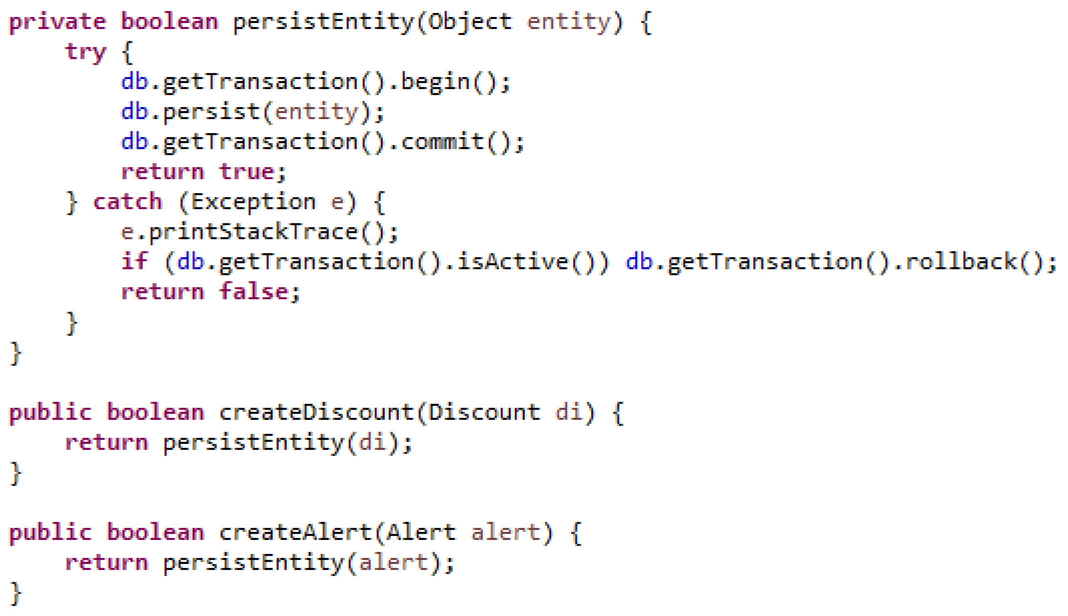
“Duplicate code”

Hasierako kodea.



```java
public void createDiscount(Discount di) {
    try {
        db.getTransaction().begin();
        db.persist(di);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public boolean createAlert(Alert alert) {
    try {
        db.getTransaction().begin();
        db.persist(alert);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

Errefaktorizatuko kodea:



```java
private boolean persistEntity(Object entity) {
    try {
        db.getTransaction().begin();
        db.persist(entity);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        if (db.getTransaction().isActive()) db.getTransaction().rollback();
        return false;
    }
}

public boolean createDiscount(Discount di) {
    return persistEntity(di);
}

public boolean createAlert(Alert alert) {
    return persistEntity(alert);
}
```

Egindako errefaktorizazioren deskribapena

Bi metodo hauetan kodea ia berdina zenez, errepikapena saihesteko metodo laguntzaile bat sortu dut.

"Keep unit interfaces small":

Kasu honetan, bad smell hau soilik bi alditan agertzen da proiektu osoan.

BEÑAT

“Write short units of code”

Hasierako kodea:

```
public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
try {
db.getTransaction().begin();
User user = getUser(username);
if (user != null) {
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) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}
}
```

<u>Errefaktorizatuko kodea:</u>

```
public boolean gauzatuEragiketa(String username, double amount,
boolean deposit) {
try {
db.getTransaction().begin();
User user = getUser(username);
if (user != null) {
return diruMugimenduak(user, amount, deposit);
}
db.getTransaction().commit();
return false;
} catch (Exception e) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}

}


public boolean diruMugimenduak(User user, double amount, boolean
deposit) {

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;


}
```

Egindako errefaktorizazioren deskribapena:

Kasu honetan, etorkizuneko errore posibleak erraztearren, gauzatuEragiketak barruan, diru maneiua egiten den atala, funtzio batean sartu eta metodo berri bat sortu da, kodea ulergarriago eginez eta erroreak errazago aurkituz.

<u>“Write simple units of code”</u>

<u>Hasierako kodea:</u>

```
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;
}
}
```

Errefaktorizatuko 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 = findAlerts(rides, alert);
alert.setFound(found);
if (alert.isActive() && found)
alertFound = true;
db.merge(alert);
}

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

public boolean findAlerts(List<Ride> rides, Alert alert) {
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;
```

```
}
```

Egindako errefaktorizazioren deskribapena:

Kodean, metodo berri bat sortu behar izan da sinpleago egiteko, baina horrez gain, kodean ere aldaketak egin behar izandu dira, eginda zegoen moduan ez zegoelako batere sinplea, eta kode zikina zegoelako tartean, kodeari buelta txiki bat emanez.

"Duplicate code"

Hasierako kodea:

```
public boolean addDriver(String username, String password) {
 try {
 db.getTransaction().begin();

 if (!isUsernameAvailable(username)) return false;

 persistNewDriver(username, password);

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

private boolean isUsernameAvailable(String username) {
 return getDriver(username) == null && getTraveler(username) == null;
}

private void persistNewDriver(String username, String password) {
 Driver driver = new Driver(username, password);
 db.persist(driver);
}



public boolean addTraveler(String username, String password) {
try {
db.getTransaction().begin();

Driver existingDriver = getDriver(username);
Traveler existingTraveler = getTraveler(username);
if (existingDriver != null || existingTraveler != null) {
return false;
}

Traveler traveler = new Traveler(username, password);
db.persist(traveler);
db.getTransaction().commit();
return true;
} catch (Exception e) {
e.printStackTrace();
db.getTransaction().rollback();
return false;
}
}
```

Errefaktorizatuko kodea:

```
public <T> boolean addUser(String username, String password, Class<T> mota) {

try {

db.getTransaction().begin();

if (!isUsernameAvailable(username)) return false;


persistNewUser(username, password, mota);

db.getTransaction().commit();

return true;

}catch(Exception e) {

db.getTransaction().rollback();

return false;

}

}


private <T> void persistNewUser(String username, String password, Class<T> mota)
throws Exception {

Constructor<T> constr = mota.getConstructor(String.class, String.class);

T user = constr.newInstance(username, password);

db.persist(user);

}

private boolean isUsernameAvailable(String username) {

 return getDriver(username) == null && getTraveler(username) == null;

}
```

Egindako errefaktorizazioren deskribapena:

AddDriver eta Add Traveler metodoak, oso antzekoak direnez, aldatuta, beraien klase mota, horregatik, mota generiko baterako kodea egin dut, bi metodoak bateratuz.

<u>**"Keep unit interfaces small":**</u>

Kasu honetan, bad smell hau soilik bi alditan agertzen da proiektu osoan.