REFACTOR

Ingenieritza Informatikoa EHU/UPV

Jon Reboiro, Irantzu Loero, Estibaliz Leza

Aurkibidea:

Irantzu Loero

Write short units of code:

1. Hasierako kodea

```
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(R);
			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. Errefaktorizatutako kodea

```
public void cancelRide(Ride ride) {
	try {
		db.getTransaction().begin();
		for (Booking booking : ride.getBookings()) {
			diruaItzuli(booking);
		}
		ride.setActive(false);
		db.merge(ride);
		db.getTransaction().commit();
	} catch (Exception e) {
```

```
            if (db.getTransaction().isActive()) {
                db.getTransaction().rollback();
            }
            e.printStackTrace();
        }
}
public void diruaItzuli(Booking booking) {
        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(R);
        db.merge(booking);
}
```

3. Egindako errefaktorizazioren deskribapena

cancelRide() metodoa 29 linea zituen. Bi metodoetan banatu dut, alde batetik cancelRide() mantendu egin dut, eta diruaItzuli() metodo berri bat sortu dut, ezabatu den bidaiaren erreserba bakoitzerako diru mugimenduak kontrolatzeko.

Write simple units of code

1. 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;
        }
}
```

2. Errefaktorizatutako kodea

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

private void saldoaBerritu(User user, double amount, boolean deposit) {
     double currentMoney = user.getMoney();
     if (deposit) {
        user.setMoney(currentMoney + amount);
     } else {
        user.setMoney(Math.max(0, currentMoney - amount));
     }
}
```

3. Egindako errefaktorizazioren deskribapena

gauzatuEragiketa() metodoa 5eko konplexutasun ziklomatikoa zuen. Metodoa bitan banatu dut, saldoa berritzeko egin beharreko eragiketak beste metodo batean inplementatuz.

Duplicate code:

1. Hasierako kodea

```
Movement m1 = new Movement(traveler1, "BookFreeze", 20);
Movement m2 = new Movement(traveler1, "BookFreeze", 40);
Movement m3 = new Movement(traveler1, "BookFreeze", 5);
Movement m4 = new Movement(traveler2, "BookFreeze", 4);
Movement m5 = new Movement(traveler1, "BookFreeze", 3);
```

2. Errefaktorizatutako kodea

```
Movement m1 = new Movement(traveler1, BF, 20);
Movement m2 = new Movement(traveler1, BF, 40);
Movement m3 = new Movement(traveler1, BF, 5);
Movement m4 = new Movement(traveler2, BF, 4);
Movement m5 = new Movement(traveler1, BF, 3);
```

3. Egindako errefaktorizazioren deskribapena

“BookFreeze”-erako konstante bat definitu dut (`public static final String BF = "BookFreeze";`) hainbat alditan errepikatzen zelako.

Keep unit interfaces small:

Ez daude bestelakorik aurkitu.

Jon Reboiro

Write short units of code:

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(R);

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. Errefaktorizatutako kodea.

```
public void deleteUser(User us) {
		try {
			if (us.getMota().equals("Driver")) {
				deleteDriver(us);
			} else {
				deleteTraveler(us);
```

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

private void deleteTraveler(User us) {
        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());
                }
        }
}

private void deleteDriver(User us) {
        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);
                }
        }
}
```

3. Egindako errefaktorizazioren deskribapena.

Delete User metodoa 23 lerroko kodea zuen, horregatik, if baldintza bakoitzeko banatu dezakegu, driver-a denean zer egin behar duen eta traveler-a denean zer egin behar duen zehazteko metodo bat sortuz (deleteTraveler eta deleteDriver). Eta datubasea deleteUser-en egin.

Write simple units of code:

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. Errefaktorizatuko kodea.

```
public void updateAlert(Alert alert) {
      try {
            db.getTransaction().begin();
            db.merge(alert);
            db.getTransaction().commit();
            } catch (Exception e) {
                    e.printStackTrace();
                    db.getTransaction().rollback();
            }
    }
    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 = processAlerts(alerts, rides);
                    db.getTransaction().commit();
                    return alertFound;
            } catch (Exception e) {
                    e.printStackTrace();
                    db.getTransaction().rollback();
                    return false;
            }
    }
    private boolean processAlerts(List<Alert> alerts, List<Ride> rides) {
            boolean alertFound = false;
            for (Alert alert : alerts) {
                    boolean found = findMatchingRide(alert, rides);
                    if (!found) {
                            alert.setFound(false);
                    }
                    db.merge(alert);
                    if (found && alert.isActive()) {
                            alertFound = true;
                    }
            }
            return alertFound;
    }
    private 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) {
                            alert.setFound(true);
                            return true;
                    }
            }
            return false;
```

```
}
```

3. Egindako errefaktorizazioren deskribapena.

Metodoaren konplexutasun ziklomatikoa altuegia zen. Horregatik, hainbat metodotan zatitu dut metodo berdina, bakoitzak funtzionalitate batekin. updateAlertaAurkituak orain metodo nagusia da, datubasearekin lehenengo queryak eta emaitza nagusia lortzen duena. Bestalde, proccessAlerts-ek alerten for-a egiten du, eta matchingRides metodoak alerta bakoitzeko ride bat eskuragarri dagoen begiratuko du.

Duplicate code:

1- Hasierako kodea.

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

2. Errefaktorizatutako kodea.

```
private static final String R = "Rejected";
public static final String A = "Accepted";
      book1.setStatus(A);
      book2.setStatus(R);
      book3.setStatus(A);
      book4.setStatus(A);
      book5.setStatus(A);
```

3. Egindako errefaktorizazioren deskribapena.

“Accepted” eta “Rejected” String-ak lau-bost alditan agertzen dira klase osoan. Hau zuzentzeko, klasean konstate bat definitu dezakegu beharrezkoa den guztietan konstante hori zuzenean atzitzeko.

Keep unit interfaces small:

Ez dugu bestelakorik aurkitu.

Estibaliz Leza

Write short units of code

1- Hasierako kodea

```
public List<Ride> getRidesByDriver(String username) {
        try {
                db.getTransaction().begin();
                TypedQuery<Driver>  query  =  db.createQuery("SELECT  d  FROM  Driver  d
WHERE d.username = :username",
                                Driver.class);
                query.setParameter(USERNAME, username);
                Driver driver = query.getSingleResult();
                List<Ride> rides = driver.getCreatedRides();
                List<Ride> activeRides = new ArrayList<>();
                for (Ride ride : rides) {
                        if (ride.isActive()) {
                                activeRides.add(ride);
                        }
                }
                db.getTransaction().commit();
                return activeRides;
        } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
                return null;
        }
}
```

2. Errefaktorizatutako kodea

```
public List<Ride> getRidesByDriver(String username) {
        try {
                db.getTransaction().begin();
                TypedQuery<Driver>  query  =  db.createQuery("SELECT  d  FROM  Driver  d
WHERE d.username = :username",
                                Driver.class);
                query.setParameter(USERNAME, username);
                Driver driver = query.getSingleResult();
                List<Ride> activeRides = getActiveRides(driver);
                db.getTransaction().commit();
                return activeRides;
        } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
                return null;
        }
}

public List<Ride> getActiveRides (Driver driver) {
        List<Ride> rides = driver.getCreatedRides();
        List<Ride> activeRides = new ArrayList<>();
        for (Ride ride : rides) {
                if (ride.isActive()) {
                        activeRides.add(ride);
                }
        }
        return activeRides;
}
```

3. Egindako errefaktorizazioren deskribapena

getRidesByDriver() metodoa 22 linea zituen. Bi metodoetan banatu dut, alde batetik getRidesByDriver() mantendu egin dut, eta getActiveRides() metodo berri bat sortu dut, gidariak sortutako bidaietatik aktibo daudenak itzultzen dituena. Bigarren bertsio honekin 13 lerroko eta 10 lerroko bi metodo lortu ditut. Horrela, kodearen errepikapena ekiditen da, eta kode modularragoa eta ulertzeko errazagoa bihurtzen da.

Write simple units of code

1- Hasierako kodea

```
/**
 * This method books a ride for a traveler
 *
 * @param username username of the traveler
 * @param ride     ride to book
 * @param seats    number of seats to book
 * @param desk     discount to apply
 * @return true if the booking is successful, false otherwise
 */
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;
        }
}
```

2. Errefaktorizatuko kodea

```
/**
 * This method books a ride for a traveler
 *
 * @param username username of the traveler
 * @param ride     ride to book
 * @param seats    number of seats to book
 * @param desk     discount to apply
 * @return true if the booking is successful, false otherwise
 */
public boolean bookRide(String username, Ride ride, int seats, double desk) {
   try {
       db.getTransaction().begin();

       if (!canTravelerBook(username, ride, seats, desk)) {
           return false;
       }

       return createBooking(username, ride, seats, desk);
   } catch (Exception e) {
            e.printStackTrace();
            db.getTransaction().rollback();
            return false;
   }
}

/**
 * Checks if the traveler can book the ride based on the provided criteria.
 *
```

```java
 * @param username username of the traveler
 * @param ride     ride to book
 * @param seats    number of seats to book
 * @param desk     discount to apply
 * @return true if the traveler can book the ride, false otherwise
 */
private boolean canTravelerBook(String username, Ride ride, int seats, double desk) {
   Traveler traveler = getTraveler(username);

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

/**
 * Creates a booking for the traveler.
 *
 * @param username username of the traveler
 * @param ride     ride to book
 * @param seats    number of seats to book
 * @param desk     discount to apply
 * @return true if booking was successful, false otherwise
 */
private boolean createBooking(String username, Ride ride, int seats, double desk) {
   Traveler traveler = getTraveler(username);
   double ridePriceDesk = (ride.getPrice() - desk) * seats;
   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(traveler.getMoney() - ridePriceDesk);
   traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
   db.merge(ride);
   db.merge(traveler);
   db.getTransaction().commit();
   return true;
}
```

3. Egindako errefaktorizazioren deskribapena

Eskakizunetan metodoak 4 baldintza soilik izan behar dituela eskatzen da, baina soilik bi metodo aurkitu ditugu 4 baldintza baino gehaigo dituztenak. Beraz, 4 baldintzako metodo bat aukeratu dut eta lau baino baldintza gutxiagoko metodoetan banatu dut.

Errefaktorizazioa egin eta gero, metodo nagusiak (bookRide) canTravelerBook metodoari deitzen dio baliozkotzeak egiteko, eta guztiak betetzen badira createBooking metodoaren bitartez bidaia erreserbatzen du. Hasier batean try bat eta hiru if zeuden kodean, eta orain, 3 metodo desberdinetan banatzerakoan, metodo nagusiak soilik try eta if bat dauzka.

Duplicate code

1- Hasierako kodea

```
query.setParameter("username", erab);
travelerQuery.setParameter("username", erab);
driverQuery.setParameter("username", erab);
query.setParameter("username", erab);
query.setParameter("username", erab);
driverQuery.setParameter("username", erab);
travelerQuery.setParameter("username", erab);
query.setParameter("username", username);
query.setParameter("username", username);
query.setParameter("username", username);
query.setParameter("username", username);
alertQuery.setParameter("username", username);
```

2. Errefaktorizatutako kodea

```
query.setParameter(USERNAME, erab);
travelerQuery.setParameter(USERNAME, erab);
driverQuery.setParameter(USERNAME", erab);
query.setParameter(USERNAME, erab);
query.setParameter(USERNAME, erab);
driverQuery.setParameter(USERNAME, erab);
travelerQuery.setParameter(USERNAME, erab);
query.setParameter(USERNAME, username);
query.setParameter(USERNAME, username);
query.setParameter(USERNAME, username);
query.setParameter(USERNAME, username);
alertQuery.setParameter(USERNAME, username);
```

3. Egindako errefaktorizazioren deskribapena

"username" String-a 10+ alditan agertzen da klase osoan. Hau zuzentzeko, atributu konstate bat definitu dezakegu beharrezkoa den guztietan konstante hori zuzenean atzitzeko.

Keep unit interfaces small

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(Complaint erreklamazioa) {
                try {
                        db.getTransaction().begin();
                        db.persist(erreklamazioa);
                        db.getTransaction().commit();
                        return true;
                } catch (Exception e) {
                        e.printStackTrace();
                        db.getTransaction().rollback();
                        return false;
                }
}
```

BLFacadeImplementation klasean:

```
@Override
        public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking book,
String textua, boolean aurk) {
                dbManager.open();
                Complaint erreklamazioa = new Complaint(nor, nori, gaur, book, textua,
aurk);
                boolean sent = dbManager.erreklamazioaBidali(erreklamazioa);
                dbManager.close();
                return sent;
        }
```

3. Egindako errefaktorizazioren deskribapena

erreklamazioaBidali metodoko parametro guztiak Complaint klaseko objektuaren barruan pasa ditut. BLFacadeImplementation klaseko erreklamazioaBidali klasean bertan parametroekin objektua sortu eta zuzenan obketua bidali dut atributu bezala.