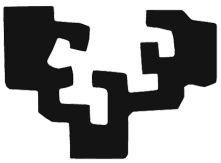
eman ta zabal zazu



Universidad del País Vasco

Euskal Herriko Unibertsitatea

Izenburua

Errefaktorizazioa

Informatika ingeniaritza gradua

Egileak:

Kepa Gaztañaga

Miriam Rodriguez

2024-ko urriak 10a

gitHubeko Repositorio esteka:
https://github.com/Kepa8/Rides24-Miriam-Kepa-.git

AURKIBIDEA

SARRERA

Lan honen helburua, Rides24 proiektuko DataAccess klasean agertzen diren "Bad Smell" edo “issue” batzuk ezabatzea da, errefaktorizazio batzuk aplikatuz. Zehazki, hurrengo Bad smell-ak ezabatu behako dira:

1. **"Write short units of code":** Metodoak 15 linea baino gehiagokoak izatea
2. **"Write simple units of code":** Metodoen konplexutasun ziklomatikoa 4 baino gehiagokoa ez izatea
3. **“Duplicate code”:** Kodigoan dauden gauza errepikatuak sahiestea
4. **"Keep unit interfaces small":** Metodoak erabiltzen dituzten parametroak 4 baino gehiago ez izatea

Beraz, taldekide bakoitzak 4 errefaktorizazio egin behar izan ditu, bad smell bakoitzeko bat. Aldaketak egon diren ala ez ikusi ahal izateko memoria honetan hurrengo azpiatalak agertuko dira atal bakoitzeko:

1. Hasierako kodea
2. Errefaktorizatuko kodea
3. Egindako errefaktorizazioren deskribapena
4. Egilea

Kepa-k konpondutako arazoak

1. "Write short units of code"

1.1. Hasierako kodea

```
public List<Booking> getBookingFromDriver(String username) {
	try {
		db.getTransaction().begin();
		TypedQuery<Driver> query =
db.createQuery(QUERY_DRIVER_BY_USERNAME, Driver.class);
		query.setParameter(US, username);
		Driver driver = query.getSingleResult();
		List<Ride> rides = driver.getCreatedRides();
		List<Booking> bookings = new ArrayList<>();
		for (Ride ride : rides) {
			if (ride.isActive()) {
				bookings.addAll(ride.getBookings());
			}
		}
		db.getTransaction().commit();
		return bookings;
	} catch (Exception e) {
		e.printStackTrace();
		db.getTransaction().rollback();
		return null;
	}
}
```

1.2. Errefaktorizatuko kodea

```
public List<Booking> getBookingFromDriver(String username) {
	try {
		db.getTransaction().begin();
		Driver driver = getDriverByUsername(username);
		List<Ride> rides = driver.getCreatedRides();
		List<Booking> bookings = getActiveBookingsFromRides(rides);
		db.getTransaction().commit();
		return bookings;
	} catch (Exception e) {
		e.printStackTrace();
		db.getTransaction().rollback();
		return null;
	}
}
private Driver getDriverByUsername(String username) {
	TypedQuery<Driver> query =
db.createQuery(QUERY_DRIVER_BY_USERNAME, Driver.class);
	query.setParameter(US, username);
```

```
        return query.getSingleResult();
}
private List<Booking> getActiveBookingsFromRides(List<Ride> rides) {
        List<Booking> bookings = new ArrayList<>();
        for (Ride ride : rides) {
                if (ride.isActive()) {
                        bookings.addAll(ride.getBookings());
                }
        }
        return bookings;
}
```

1.3. Egindako errefaktorizazioren deskribapena

Funtzioak 15 linea baino gehiago zituen hasiera batean. Funtzioaren kodea bi funtzio laguntzailetan banatu da hau konpontzeko. “getDriverByUsername” funtzioa non gidari bat lortzen da bere izenetik abiatuta eta **“**getActiveBookingsFromRides” funtzioak zeinek gidariaren rutetaz baliatuta zeintzuk dauden aktibo begiratzen du eta ondoren aktibo daudenetatik erreserbak eskuratu eta zerrenda batean itzultzen ditu. Honela funtzio nagusiak bi funtzio laguntzaileei esker gidari baten erreserbak itzultzen ditu.

2. "Write simple units of code"

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(US, 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. Errefaktorizatuko kodea

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

        List<Alert> alerts = getAlertsByUsername(username);
        List<Ride> rides = getActiveRides();

        boolean alertFound = processAlerts(alerts, rides);

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


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

private boolean processAlerts(List<Alert> alerts, List<Ride> rides) {
    boolean alertFound = false;

    for (Alert alert : alerts) {
        boolean found = checkAlertInRides(alert, rides);
        alert.setFound(found);
        if (found && alert.isActive()) {
            alertFound = true;
        }
        db.merge(alert);
    }
```

```
    return alertFound;
}

private boolean checkAlertInRides(Alert alert, List<Ride> rides) {
    for (Ride ride : rides) {
        if (isMatchingRide(ride, alert)) {
            return true; // Alert found
        }
    }
    return false; // Alert not found
}

private boolean isMatchingRide(Ride ride, Alert alert) {
    return UtilDate.datesAreEqualIgnoringTime(ride.getDate(), alert.getDate())
            && ride.getFrom().equals(alert.getFrom())
            && ride.getTo().equals(alert.getTo())
            && ride.getnPlaces() > 0;
}
```

2.3. Egindako errefaktorizazioren deskribapena

Funtzioaren konplexutasuna 5-ekoa zenez funtzio ezberdinetan banatu izan behar da bad smell a konpontzeko:

- updateAlertaAurkituak: Erabiltzailearen alerta aktiboak eguneratzen ditu, datu-basean dauden oharren arabera. Hainbat bidairekin konparatzen ditu, eta aurkitzen badu, alerta eguneratzen du.
  Konplexutasuna: 1
- getAlertsByUsername: Datu-basean, erabiltzailearen izenarekin lotutako alertak lortzen ditu.
  Konplexutasuna: 0
- getActiveRides: Datu-basean aktiboak diren eta eguna baino geroagoko datak dituzten bidaien zerrenda lortzen du.
  Konplexutasuna: 0
- processAlerts: Alertak eta bidaien zerrenda bat hartzen du, eta alerta bakoitzaren egoera eguneratzen du, bidaiekin bat datorren ikertuz.
  Konplexutasuna: 2.5. checkAlertInRides: Alertak bidaien zerrendan aurkitzen diren aztertzen du, alerta batek bidaia batekin bat datorren egiaztatzeko.
  Konplexutasuna: 1
- isMatchingRide: Alerta batek bidaia batekin bat datorren egiaztatzen du, data, jatorria eta helmuga konparatuz.
  Konplexutasuna: 1

3. “Duplicate code”

3.1. Hasierako kodea

```
public User getUser(String erab) {
        TypedQuery<User> query = db.createQuery("SELECT u FROM User u
WHERE u.username = :username", User.class);
        query.setParameter(US, erab);
```

```
        return query.getSingleResult();
}

public Driver getDriver(String erab) {
        TypedQuery<Driver> query =
db.createQuery(QUERY_DRIVER_BY_USERNAME, Driver.class);
        query.setParameter(US, 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(US, erab);
        List<Traveler> resultList = query.getResultList();
        if (resultList.isEmpty()) {
            return null;
        } else{
            return resultList.get(0);
        }
}
```

3.2. Errefaktorizatuko kodea

```
private String buildSelectQuery(String entityName, String whereClause) {
    return "SELECT e FROM " + entityName + " e WHERE " + whereClause;
}


public User getUser(String erab) {
    String whereClause = "e.username = :username";
    String queryString = buildSelectQuery("User", whereClause);
    TypedQuery<User> query = db.createQuery(queryString, User.class);
    query.setParameter("username", erab);
    return query.getSingleResult();
}


public Driver getDriver(String erab) {
        String whereClause = "e.username = :username";
        String queryString = buildSelectQuery("Driver", whereClause);
        TypedQuery<Driver> query = db.createQuery(queryString,
Driver.class);
        query.setParameter("username", erab);
        List<Driver> resultList = query.getResultList();
         return resultList.isEmpty() ? null : resultList.get(0);
}

public Traveler getTraveler(String erab) {
        String whereClause = "e.username = :username";
```

```
        String queryString = buildSelectQuery("Traveler", whereClause);
        TypedQuery<Traveler> query = db.createQuery(queryString,
Traveler.class);
        query.setParameter("username", erab);
        List<Traveler> resultList = query.getResultList();
        return resultList.isEmpty() ? null : resultList.get(0);
}
```

3.3. Egindako errefaktorizazioren deskribapena

DataAccess klasean SelecQuery ugari daude hauek behin eta berriro ez idaztearren buildSelectQuery() funtzioa sortu dut. Honekin stringak behin eta berriro idaztea saihesten dugu

4. "Keep unit interfaces small"

4.1. 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.ErrorRi
deMustBeLaterThanToday"));
                        }

                        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.RideAlread
yExist"));
                        }
                        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;
                }
        }


public Ride createRide(String from, String to, Date date, int nPlaces, float
price, String driverName)
        throws RideMustBeLaterThanTodayException,
RideAlreadyExistException {

    dbManager.open();
    Ride ride = dbManager.createRide(from, to, date, nPlaces, price,
driverName);
    dbManager.close();
    return ride;
}
```

4.2. Errefaktorizatuko kodea

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

ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRi
deMustBeLaterThanToday"));
        }

        db.getTransaction().begin();
        Driver driver = db.find(Driver.class, driverName);
        if (driver.doesRideExists(rideDetails.getFrom(), rideDetails.getTo(),
rideDetails.getDate())) {
          db.getTransaction().commit();
          throw new RideAlreadyExistException(

ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlread
yExist"));
        }

        Ride ride = driver.addRide(rideDetails.getFrom(), rideDetails.getTo(),
rideDetails.getDate(),
              rideDetails.getNPlaces(), rideDetails.getPrice());
        db.persist(driver);
        db.getTransaction().commit();

        return ride;
```

```
    } catch (NullPointerException e) {
        return null;
    }
}

public Ride createRide(String from, String to, Date date, int nPlaces, float
price, String driverName)
                throws RideMustBeLaterThanTodayException,
RideAlreadyExistException {
            dbManager.open();
            RideDetails rideDetails = new RideDetails(from, to, date,
nPlaces, price);
            Ride ride = dbManager.createRide(rideDetails, driverName);
            dbManager.close();
            return ride;
        }
package dataAccess;
import java.util.Date;
public class RideDetails {
  private String from;
  private String to;
  private Date date;
  private int nPlaces;
  private float price;
  public RideDetails(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;
  }
  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;
  }
}
```

4.3. Egindako errefaktorizazioren deskribapena

createRide() metodoan 6 parametro zituen, beraz funtzio hau errefaktorizatu da. Klase berri bat sortu da, klasean parametroa jarriz parametroak gutxitu dira.

Miriam-ek konpondutako arazoak

1. "Write short units of code"

1.1. Hasierako kodea

```
public void cancelRide(Ride ride) {
	try {
		db.getTransaction().begin();
		for (Booking booking : ride.getBookings()) {
			if (booking.getStatus().equals(ACC) ||
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(REJ);
			db.merge(booking);
		}
		ride.setActive(false);
		db.merge(ride);
		db.getTransaction().commit();
	} catch (Exception e) {
		if (db.getTransaction().isActive()) {
			db.getTransaction().rollback();
		}
		e.printStackTrace();
	}
}
```

1.2. Errefaktorizatuko kodea

```
public void cancelRide(Ride ride) {
        try {
                db.getTransaction().begin();
                for (Booking booking : ride.getBookings()) {
                        if (booking.getStatus().equals(ACC) ||
booking.getStatus().equals("NotDefined")) {
                                refundTraveler(booking);
                        }
                        booking.setStatus(REJ);
                        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 refundTraveler(Booking booking) {
        double price = booking.prezioaKalkulatu();
        Traveler traveler = booking.getTraveler();
         traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() - price);
         traveler.setMoney(traveler.getMoney() + price);
         db.merge(traveler);
         boolean currentTransactionActive = db.getTransaction().isActive();
         if (currentTransactionActive) {
            db.getTransaction().commit();
         }
         addMovement(traveler, "BookDeny", price);
         if (currentTransactionActive) {
            db.getTransaction().begin();
         }
}
```

1.3. Egindako errefaktorizazioren deskribapena

Hasierako kodean 15 linea baina gehiago zerenez, erabilitako linea gutxiagotzea lortu da errefaktorizazioarekin beste metodo laguntzailea eginez. Hau da, extracting a Method from Code aukera baliatuz. Hau egindakoan refaundTraveler() izena duen metodo laguntzailea sortu da eta gero transaction-ak zuzendu dira addMovement() metodoaren deia baitago jarraian.

2. "Write simple units of code"

2.1. 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;
        }
}
```

2.2. Errefaktorizatuko kodea

```
public boolean gauzatuEragiketa(String username, double amount, boolean
deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null) {
            updateUserMoney(user, amount, deposit);
            db.merge(user);
            db.getTransaction().commit();
            return true;
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
private void updateUserMoney(User user, double amount, boolean deposit) {
    double currentMoney = user.getMoney();
    if (deposit) {
        user.setMoney(currentMoney + amount);
    } else {
        user.setMoney(Math.max(0, currentMoney - amount));
    }
}
```

2.3. Egindako errefaktorizazioren deskribapena

Hemen ere taktika berdina erabili da arazoa zuzentzeko. Konplexutasun ziklomatikoa jaitsi zedin, metodo laguntzailea erabili da. Hasieran VG=5 zen eta orain, if-a eta else-a kentzerakoan VG=3 koa da, branch kopurua jaitsi baita.

3. "Duplicate code"

3.1. Hasierako kodea

```
public void initializeDB() {
        db.getTransaction().begin();
        try {
                //Kodea
                 book1.setStatus ("Accepted");
                  book2.setStatus("Rejected");
                  book3.setStatus("Accepted");
                  book4.setStatus("Accepted");
                  book5.setStatus("Accepted");
        //Kodea
```

3.2. Errefaktorizatuko kodea

```
private static final String ACC = "Accepted";
private static final String REJ = "Rejected";

public void initializeDB() {
        db.getTransaction().begin();
        try {
                //Kodea
                 book1.setStatus(ACC);
                  book2.setStatus(REJ);
                  book3.setStatus(ACC);
                  book4.setStatus(ACC);
                  book5.setStatus(ACC);
        //Kodea
```

3.3. Egindako errefaktorizazioren deskribapena

"Accepted" eta "Rejected" string-ak aldagai konstanteak bezala deklaratu da errefaktorizazioa erabiliz. Extract Constant aukeraren bidez.

4. "Keep unit interfaces small"

4.1. 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;
        }
}
```

4.2. Errefaktorizatuko kodea

```
public boolean erreklamazioaBidali(ErreklamazioaBidaliInfo parameterObject,
Booking booking, boolean aurk) {
        try {
                db.getTransaction().begin();
                Complaint erreklamazioa = new Complaint(parameterObject.nor,
parameterObject.nori, parameterObject.gaur, booking,
parameterObject.textua, aurk);
                db.persist(erreklamazioa);
                db.getTransaction().commit();
                return true;
        } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
                return false;
        }
}
public class ErreklamazioaBidaliInfo {
        public String nor;
        public String nori;
        public Date gaur;
        public String textua;

        public ErreklamazioaBidaliInfo(String nor, String nori, Date gaur, String
textua) {
                this.nor = nor;
                this.nori = nori;
                this.gaur = gaur;
                this.textua = textua;
        }
}
```

4.3. Egindako errefaktorizazioren deskribapena

ErreklamazioBidali() metodoan 6 parametro zeuden eta gutxitzeko errefaktorizazioa erabili da. Klase berri bat sortu da parametroak jarri ordez, klasea jarri ahal izateko. Klaseak metodo nagusitik kendu diren parametroak egongo daude definituta.