ERREFAKTORIZAZIO LABORATEGIA

Beñat Zubizarreta eta Xabier Abaunz

- 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;
        }
}
```

  - Amaierako kodea:

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

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

  - Deskribapena:
    ErreklamazioaBidali() funtzioan lehen zeuzkan parametroak erabili beharrean, klase berri bat sortu da parametro horiek atribututzat hartzen dituena. Ondorioz, funtzioari mota horretako objektu bat pasatzen zaio, jasotzen dituen paramtetroak 1era murriztuz. Ondoren, metodo honi deitzen zaion lekuetan ErreklamazioaBidaliParameter motako objektu bat sortu behar da, eta hori pasa funtzioari.

- Egilea: Xabier Abaunz Gallastegui

2.

- 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) {
                        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;
        }


}
```

- Amaierako kodea:

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

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

        return ride;
    } catch (NullPointerException e) {
        // TODO Auto-generated catch block
        return null;
    }

}
```

- Deskribapena:
  createRide() funtzioan lehen zeuzkan parametroak erabili beharrean, klase berri bat sortu da parametro horiek atribututzat hartzen dituena. Ondorioz,

funtzioari mota horretako objektu bat pasatzen zaio, jasotzen dituen paramtetroak 1era murriztuz. Ondoren, metodo honi deitzen zaion lekuetan CreateRideParameter motako objektu bat sortu behar da, eta hori pasa funtzioari.

- Egilea: Beñat Zubizarreta Regillaga

- Write short units of code

1.

- Hasierako kodea:

```
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;
        }
}
```

- Amaierako kodea:

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

        Traveler traveler = getTraveler(username);

        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        double availableBalance = traveler.getMoney();

        if (traveler == null || ride.getnPlaces() < seats || availableBalance < ridePriceDesk) {
            return false;
        }

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

        setTravelerBookRide(traveler, ridePriceDesk, availableBalance, booking);
        setRideBookRide(ride, seats);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        handleException(e);
        return false;
    }
}
```

```java
private void handleException(Exception e) {
    e.printStackTrace();
    db.getTransaction().rollback();
}

private void setRideBookRide(Ride ride, int seats) {
    ride.setnPlaces(ride.getnPlaces() - seats);
    db.merge(ride);
}
private void setTravelerBookRide(Traveler traveler, double ridePriceDesk, double availableBalance,
        Booking booking) {
    traveler.addBookedRide(booking);
    traveler.setMoney(availableBalance - ridePriceDesk);
    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
    db.merge(traveler);
}

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

- Deskribapena:
  BookRIde() metodoak 15 lerro baino gehiago zituenez, handleException(), setRideBookRide(), setTravelerBookRIde eta createBooking() metodoak kanpora atera dira. Horrela hasierako metodoak horiei deitu eta ez du luzera maximoa gainditzen.

- Egilea: Xabier Abaunz Gallastegui

2.

- 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("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();
    }
}
```

- Amaierako kodea:

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

- Deskribapena:

cancelRide() metodoak 15 lerro baino gehiago zituenez, bitan banatu dugu. Horretarako, adjustTravelerFound() metodoa sortzen dugu, bidaia erreserbatu duten bezeroei dirua bueltatuko diona. Horrela, candelRide() metodoak adjustTravelerFound() metodoari deituko dio, metodo hau laburtuz.

- Egilea: Beñat Zubizarreta Regillaga

- Write simple 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("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();
	}
}
```

- Amaierako kodea:

```java
public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {
            handleDriverDeletion(us);
        } else {
            handleUserDeletion(us);
        }
        removeUserFromDatabase(us);
    } catch (Exception e) {
        e.printStackTrace();
    }
}

private void handleDriverDeletion(User us) {
    cancelDriverRides(us);
    deleteDriverCars(us);
}

private void cancelDriverRides(User us) {
    List<Ride> rides = getRidesByDriver(us.getUsername());
    if (rides != null) {
        for (Ride ride : rides) {
            cancelRide(ride);
        }
    }
}

private void deleteDriverCars(User us) {
    Driver driver = getDriver(us.getUsername());
    List<Car> cars = driver.getCars();
    if (cars != null) {
        for (int i = cars.size() - 1; i >= 0; i--) {
            deleteCar(cars.get(i));
        }
    }
}
```

```java
private void handleUserDeletion(User us) {
    rejectUserBookings(us);
    deleteUserAlerts(us);
}

private void rejectUserBookings(User us) {
    List<Booking> bookings = getBookedRides(us.getUsername());
    if (bookings != null) {
        for (Booking booking : bookings) {
            booking.setStatus("Rejected");
            booking.getRide().setnPlaces(booking.getRide().getnPlaces() + booking.getSeats());
        }
    }
}

private void deleteUserAlerts(User us) {
    List<Alert> alerts = getAlertsByUsername(us.getUsername());
    if (alerts != null) {
        for (Alert alert : alerts) {
            deleteAlert(alert.getAlertNumber());
        }
    }
}

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

- Deskribapena:
  DeleteUser() metodoak baldintza gehiegi zituenez, metodo desberdinak atera dira kanpora baldintza horiek banatzeko. Horrela, amaierako metodoan if bat eta try bat bakarrik gelditukо dira. Errefaktorizazio honek, metodoaren luzera 15 lerrotik jaistea ere ahalbidetzen du.

2.

- Hasierako 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();

        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) {
        handleException(e);
        return false;
    }
}
```

- Amaierako kodea:

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

        List<Alert> alerts = getAlertsByUsername(username);

        List<Ride> rides = getActiveRides();

        boolean alertFound = updateAlerts(alerts, rides);

        db.getTransaction().commit();
        return alertFound;
    } catch (Exception e) {
        handleException(e);
        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 updateAlerts(List<Alert> alerts, List<Ride> rides) {
    boolean alertFound = false;

    for (Alert alert : alerts) {
        boolean found = checkIfRideMatchesAlert(rides, alert);

        alert.setFound(found);
        if (found && alert.isActive()) {
            alertFound = true;
        }
        db.merge(alert);
    }

    return alertFound;
}
```

```java
private boolean checkIfRideMatchesAlert(List<Ride> rides, Alert alert) {
    for (Ride ride : rides) {
        if (isMatchingRide(ride, alert)) {
            return true;
        }
    }
    return false;
}

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;
}
```

- Deskribapena:
  updateAlertaAurkituak() metodoak baldintza gehiegi zituenez, metodo desberdinak atera dira kanpora baldintza horiek banatzeko, dagoeneko sortutik zeuden metodoak berrerabiltzeaz gain. Horrela, amaierako metodoan try/catch bakarrarekin gelditu gara. Errefaktorizazio honek, metodoaren luzera 15 lerrotik jaistea ere ahalbidetzen du.

- Egilea: Beñat Zubizarreta Regillaga

- Duplicate code:

1.

- Hasierako kodea:

```
public List<Booking> getBookingFromDriver(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<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;
        }
}
```

```
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;
        }
}
```

- Amaierako kodea:

```java
public List<Booking> getBookingFromDriver(String username) {
    try {
        db.getTransaction().begin();
        Driver driver = this.getDriver(username);

        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) {
        handleException(e);
        return null;
    }
}
```

```java
public List<Ride> getRidesByDriver(String username) {
    try {
        db.getTransaction().begin();
        Driver driver = this.getDriver(username);

        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) {
        handleException(e);
        return null;
    }
}
```

- Deskribapena:
  GetBookingFromDriver() eta gerRidesByDriver() metodoetan, bilaketa berdina egiten zen datu-basean. Gainera, horretarako getDriver() metodoa sortuta zegoenez, metodo horri deitzen zaio bilaketa egiteko, kodearen bikoizketa kenduz.
- Egilea: Xabier Abaunz Gallastegui

2.

- Hasierako kodea:

```
public List<Ride> getRides(String from, String to, Date date) {
    logger.info(">> DataAccess: getActiveRides=> from= " + from + " to= " + to + " date " + date);

    List<Ride> res = new ArrayList<>();
    TypedQuery<Ride> query = db.createQuery(
            "SELECT r FROM Ride r WHERE r.from = ?1 AND r.to = ?2 AND r.date = ?3 AND r.active = true", Ride.class);
    query.setParameter(1, from);
    query.setParameter(2, to);
    query.setParameter(3, date);
    List<Ride> rides = query.getResultList();
    for (Ride ride : rides) {
        res.add(ride);
    }
    return res;
}
```

```
public List<Date> getThisMonthDatesWithRides(String from, String to, Date date) {
    logger.info(">> DataAccess: getThisMonthActiveRideDates");

    List<Date> res = new ArrayList<>();

    Date firstDayMonthDate = UtilDate.firstDayMonth(date);
    Date lastDayMonthDate = UtilDate.lastDayMonth(date);

    TypedQuery<Date> query = db.createQuery(
            "SELECT DISTINCT r.date FROM Ride r WHERE r.from=?1 AND r.to=?2 AND r.date BETWEEN ?3 and ?4 AND r.active = true",
            Date.class);

    query.setParameter(1, from);
    query.setParameter(2, to);
    query.setParameter(3, firstDayMonthDate);
    query.setParameter(4, lastDayMonthDate);
    List<Date> dates = query.getResultList();
    res.addAll(dates);

    return res;
}
```

- Amaierako kodea:

```
public List<Ride> getRides(HerriakEtaDataParameter parametroak) {
    logger.info(">> DataAccess: getActiveRides=> from= " + parametroak.from + " to= " + parametroak.from + " date " + parametroak.date);

    List<Ride> res = new ArrayList<>();
    TypedQuery<Ride> query = db.createQuery(
            "SELECT r FROM Ride r WHERE r.from = ?1 AND r.to = ?2 AND r.date = ?3 AND r.active = true", Ride.class);
    query.setParameter(1, parametroak.from);
    query.setParameter(2, parametroak.to);
    query.setParameter(3, parametroak.date);
    List<Ride> rides = query.getResultList();
    for (Ride ride : rides) {
        res.add(ride);
    }
    return res;
}
```

```
public List<Date> getThisMonthDatesWithRides(HerriakEtaDataParameter parametroak) {
    logger.info(">> DataAccess: getThisMonthActiveRideDates");

    List<Date> res = new ArrayList<>();

    Date firstDayMonthDate = UtilDate.firstDayMonth(parametroak.date);
    Date lastDayMonthDate = UtilDate.lastDayMonth(parametroak.date);

    TypedQuery<Date> query = db.createQuery(
            "SELECT DISTINCT r.date FROM Ride r WHERE r.from=?1 AND r.to=?2 AND r.date BETWEEN ?3 and ?4 AND r.active = true",
            Date.class);

    query.setParameter(1, parametroak.from);
    query.setParameter(2, parametroak.to);
    query.setParameter(3, firstDayMonthDate);
    query.setParameter(4, lastDayMonthDate);
    List<Date> dates = query.getResultList();
    res.addAll(dates);

    return res;
}
```

- Deskribapena:

  getRides() eta getThisMonthWithRides() metodoek parametro berdinak erabiltzen zituztenez, atribututzat parametro hoiek dituen klase bat sortu dut.

Horrela, biek berrerabiliko dute kodea. Gainera, parametro kodurua murrizten dugu.

- Egilea: Beñat Zubizarreta Regillaga