GithubEsteka: https://github.com/etxeg/Rides24

Errefaktorizazioa

1. “Write short units of code”

1)

```
public boolean createReservation(Traveler traveler, Ride ride, int nPlaces) {
    try {
        db.getTransaction().begin();
        Calendar today = Calendar.getInstance();
        int month = today.get(Calendar.MONTH);
        int year = today.get(Calendar.YEAR);
        int day = today.get(Calendar.DAY_OF_MONTH);
        Reservation erreserbaClass = new Reservation(traveler, ride, UtilDate.newDate(year, month, day), ride.getPrice() * nPlaces);
        if (erreserbaClass != null) {

            float prezioa = nPlaces * ride.getPrice();
            float travelerMoney = traveler.getMoney();
            if (travelerMoney >= prezioa) {
                ride.addPendingReservation(erreserbaClass);
                traveler.addReservation(erreserbaClass);
                traveler.setMoney(travelerMoney - prezioa);
                Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
                Movement move = new Movement(traveler.getMovementsCount(), prezioa, false, date);
                traveler.addMovement(move);
                erreserbaClass.setnSeats(nPlaces);
                ride.deleteNChairs(nPlaces);

                db.persist(erreserbaClass);
                db.merge(ride);
                db.merge(traveler);
                System.out.println("Erreserba zenbakia: " + erreserbaClass.getErreserbaZenbakia());
            } else {
                System.out.println("Dirua falta da");
            }
        }
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

Hobetzeko kodea zatitan hautsi dugu eta azpi-funtzioak sortu ditugu:

```
public boolean createReservation(Traveler traveler, Ride ride, int nPlaces) {
    try {
        db.getTransaction().begin();
        Reservation res = createNewReservation(traveler, ride, nPlaces);
        if (res != null && hasEnoughMoney(traveler, res)) {
            processReservation(traveler, ride, res, nPlaces);
        }
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}

private Reservation createNewReservation(Traveler t, Ride r, int nPlaces) {
    Calendar today = Calendar.getInstance();
    return new Reservation(t, r, UtilDate.newDate(today.get(Calendar.YEAR),
        today.get(Calendar.MONTH), today.get(Calendar.DAY_OF_MONTH)), r.getPrice() * nPlaces);
}

private boolean hasEnoughMoney(Traveler t, Reservation res) {
    return t.getMoney() >= res.getPrezioa();
}

private void processReservation(Traveler t, Ride r, Reservation res, int nPlaces) {
    r.addPendingReservation(res);
    t.addReservation(res);
    t.setMoney(t.getMoney() - r.getPrice());
    Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
    Movement move = new Movement(t.getMovementsCount(), r.getPrice(), false, date);
    t.addMovement(move);
    res.setnSeats(nPlaces);
    r.deleteNChairs(nPlaces);

    db.persist(res);
    db.merge(r);
    db.merge(t);
    System.out.println("Erreserba zenbakia: " + res.getErreserbaZenbakia());
}
```

2)

```
public List<Ride> checkAlerts(Traveler traveler) {
    db.getTransaction().begin();
    List<Ride> rideResult = new ArrayList<Ride>();
    List<Alert> removeAlerts = new ArrayList<Alert>();
    for (Alert actualAlert : traveler.getAlerts()) {
        TypedQuery<Ride> query = db.createQuery("SELECT r FROM Ride r WHERE r.from=?1 AND r.to=?2 AND r.date=?3",
                Ride.class);
        query.setParameter(1, actualAlert.getNundik());
        query.setParameter(2, actualAlert.getNora());
        query.setParameter(3, actualAlert.getData());
        List<Ride> rideList = query.getResultList();
        if (rideList.isEmpty() == false) {
            rideResult.addAll(rideList);
            removeAlerts.add(actualAlert);
        }
    }
    traveler.removeAlertList(removeAlerts);
    for (Alert alert:removeAlerts) {
        Alert managed = db.merge(alert);
        db.remove(managed);
    }
    db.merge(traveler);
    db.getTransaction().commit();
    return rideResult;
}
```

Hobetzeko kodea zatitan hautsi dugu eta azpi-funtzioak sortu ditugu:

```
public List<Ride> checkAlerts(Traveler traveler) {
    db.getTransaction().begin();
    List<Ride> rides = new ArrayList<>();
    List<Alert> toRemove = findMatchedAlerts(traveler, rides);
    removeAlerts(traveler, toRemove);
    db.getTransaction().commit();
    return rides;
}

private List<Alert> findMatchedAlerts(Traveler traveler, List<Ride> ridesOut) {
    List<Alert> toRemove = new ArrayList<>();
    for (Alert alert : traveler.getAlerts()) {
        List<Ride> matches = queryRides(alert);
        if (!matches.isEmpty()) {
            ridesOut.addAll(matches);
            toRemove.add(alert);
        }
    }
    return toRemove;
}

private List<Ride> queryRides(Alert alert) {
    TypedQuery<Ride> q = db.createQuery(
        "SELECT r FROM Ride r WHERE r.from = :from AND r.to = :to AND r.date = :date",
        Ride.class
    );
    q.setParameter("from", alert.getNundik());
    q.setParameter("to",   alert.getNora());
    q.setParameter("date", alert.getData());
    return q.getResultList();
}

private void removeAlerts(Traveler traveler, List<Alert> toRemove) {
    traveler.removeAlertList(toRemove);
    for (Alert alert : toRemove) {
        db.remove(db.merge(alert));
    }
    db.merge(traveler);
}
```

3)

```
public boolean rejectReservation(Reservation erreserba) {
    try {
        db.getTransaction().begin();
        erreserba.getRide().removePendingReservation(erreserba);
        erreserba.getTraveler().removeReservation(erreserba);
        erreserba.getRide().addNChairs(erreserba.getnSeats());

        float addMoney = erreserba.getTraveler().getMoney() + erreserba.getPrezioa();
        erreserba.getTraveler().setMoney(addMoney);
        Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
        Movement move = new Movement(erreserba.getTraveler().getMovementsCount(), addMoney, true, date);
        erreserba.getTraveler().addMovement(move);

        db.merge(erreserba.getTraveler());
        db.merge(erreserba.getRide());
        // Erreserba datu basetik ezabatu
        erreserba = db.merge(erreserba);
        db.remove(erreserba);

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

Hau hobetzeko kodea hainbat zatitan banatu dugu eta azpi-funtzioak sortu ditugu.

```
public boolean rejectReservation(Reservation erreserba) {
    try {
        db.getTransaction().begin();

        updateRideAndTraveler(erreserba);
        refundMoney(erreserba);
        refundMovement(erreserba);
        persistChange(erreserba);

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

public void updateRideAndTraveler(Reservation res) {

    res.getRide().removePendingReservation(res);
    res.getTraveler().removeReservation(res);
    res.getRide().addNChairs(res.getnSeats());

}
```

```
public void refundMoney(Reservation res) {
    Traveler t = res.getTraveler();
    float refund = t.getMoney() + res.getPrezioa();
    t.setMoney(refund);

}

public void refundMovement(Reservation res) {
    Traveler t = res.getTraveler();
    float amount = t.getMoney();
    Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
    Movement move = new Movement(t.getMovementsCount(), amount, true, date);
    t.addMovement(move);

}
```

4)

```
public void initializeDB() {

    db.getTransaction().begin();

    try {

        Calendar today = Calendar.getInstance();

        int month = today.get(Calendar.MONTH);
        int year = today.get(Calendar.YEAR);
        if (month == 12) {
            month = 1;
            year += 1;
        }

        // Create drivers
        Driver driver1 = new Driver("driver1@gmail.com", "Aitor Fernandez", "123");
        Driver driver2 = new Driver("driver2@gmail.com", "Ane Gaztañaga", "123");
        Driver driver3 = new Driver("driver3@gmail.com", "Test Driver", "123");
/       Admin admin1 = new Admin("admin1@gmail.com", "Test Admin", "123");
/       db.persist(admin1);
        // Create travelers
        Traveler traveler1 = new Traveler("traveler1@gmail.com", "Test Traveler", "123");
        traveler1.setMoney(500);
        Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
        Movement move = new Movement(traveler1.getMovementsCount(), 500, true, date);
        traveler1.addMovement(move);
        db.persist(move);
        Traveler traveler2 = new Traveler("traveler2@gmail.com", "Test Traveler2", "123");
        traveler2.setMoney(500);
        Date date2 = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
        Movement move2 = new Movement(traveler2.getMovementsCount(), 500, true, date2);
        traveler2.addMovement(move2);
        db.persist(move2);

        Car car1 = new Car("ABC", driver1, 4, "Toyota");
        driver1.addCar(car1);
        Car car2 = new Car("CBA", driver2, 4, "Toyota");
        driver2.addCar(car2);
        Car car3 = new Car("ABA", driver3, 4, "Toyota");
        driver3.addCar(car3);


       // Create rides
       driver1.addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 15), 7, car1);
       driver1.addRide("Donostia", "Gazteiz", UtilDate.newDate(year, month, 6), 8, car1);
       driver1.addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 25), 4, car1);
       //Ride ride1 = driver1.addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 1), 5, car1);

       driver1.addRide("Donostia", "Iruña", UtilDate.newDate(year, month, 7), 8, car1);

       driver2.addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 15), 3, car2);
       driver2.addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 25), 5, car2);
       driver2.addRide("Eibar", "Gasteiz", UtilDate.newDate(year, month, 6), 5, car2);

       driver3.addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 14), 3, car3);

       /*Reservation erreserba = new Reservation(traveler1, ride1, UtilDate.newDate(year, month, 1),5);
       erreserba.baieztatu();
       ride1.addAcceptedReservations(erreserba);
       traveler1.addReservation(erreserba)

       db.persist(erreserba);
       db.merge(ride1);*/

       db.persist(car1);
       db.persist(car2);
       db.persist(car3);
       db.persist(driver1);
       db.persist(driver2);
       db.persist(driver3);
       db.persist(traveler1);
       db.persist(traveler2);


       db.getTransaction().commit();
       System.out.println("Db initialized");
    } catch (Exception e) {
       e.printStackTrace();
    }
}
```

Hau hobetzeko asmoarekin, kodea hainbat zatitan banatu dugu eta azpi funtzioak sortu ditugu.

```
public void initializeDB() {

    db.getTransaction().begin();

    try {
        int[] yearMonth = getYearMonth();
        List<Driver> drivers = createDrivers();
        List<Car> cars = createCars(drivers);
        List<Traveler> travelers = createTravelers();
        createRides(drivers, cars, yearMonth[0], yearMonth[1]);

        /*Reservation erreserba = new Reservation(traveler1, ride1, UtilDate.newDate(year, month, 1),5);
        erreserba.baieztatu();
        ride1.addAcceptedReservations(erreserba);
        traveler1.addReservation(erreserba)

        db.persist(erreserba);
        db.merge(ride1);*/

        db.getTransaction().commit();
        System.out.println("Db initialized");
    } catch (Exception e) {
        e.printStackTrace();
    }
}


private  int[] getYearMonth(){
    Calendar today = Calendar.getInstance();

    int month = today.get(Calendar.MONTH);
    int year = today.get(Calendar.YEAR);
    if (month == 12) {
        month = 1;
        year += 1;
    }
    return new int[] {year, month};
}

public List<Driver> createDrivers() {
    List<Driver> drivers = new ArrayList <Driver>();
    Driver driver1 = new Driver("driver1@gmail.com", "Aitor Fernandez", "123");
    drivers.add(driver1);
    Driver driver2 = new Driver("driver2@gmail.com", "Ane Gaztañaga", "123");
    drivers.add(driver2);
    Driver driver3 = new Driver("driver3@gmail.com", "Test Driver", "123");
    drivers.add(driver3);
    Admin admin1 = new Admin("admin1@gmail.com", "Test Admin", "123");
    db.persist(admin1);

    db.persist(driver1);
    db.persist(driver2);
    db.persist(driver3);

    return drivers;
}

public List<Car> createCars(List<Driver> drivers) {
    List<Car> cars = new ArrayList<Car>();
    Car car1 = new Car("ABC", drivers.get(0), 4, "Toyota");
    drivers.get(0).addCar(car1);
    cars.add(car1);

    Car car2 = new Car("CBA", drivers.get(1), 4, "Toyota");
    drivers.get(1).addCar(car2);
    cars.add(car2);

    Car car3 = new Car("ABA", drivers.get(2), 4, "Toyota");
    drivers.get(2).addCar(car3);
    cars.add(car3);

    db.persist(car1);
    db.persist(car2);
    db.persist(car3);

    return cars;
}
```

```java
private Traveler createTraveler(String email, String name, float money) {
    Traveler traveler = new Traveler(email, name, "123");
    traveler.setMoney(money);
    Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
    Movement move = new Movement(traveler.getMovementsCount(), money, true, date);
    traveler.addMovement(move);
    db.persist(move);
    db.persist(traveler);
    return traveler;
}

public List<Traveler> createTravelers() {
    List<Traveler> travelers = new ArrayList<>();

    travelers.add(createTraveler("traveler1@gmail.com", "Test Traveler", 500));
    travelers.add(createTraveler("traveler2@gmail.com", "Test Traveler2", 500));

    return travelers;
}

public void createRides(List<Driver> drivers, List<Car> cars, int year, int month ) {

    drivers.get(0).addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 15), 7, cars.get(0));
    drivers.get(0).addRide("Donostia", "Gazteiz", UtilDate.newDate(year, month, 6), 8, cars.get(0));
    drivers.get(0).addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 25), 4, cars.get(0));
    //Ride ride1 = driver1.addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 1), 5, car1);
    drivers.get(0).addRide("Donostia", "Iruña", UtilDate.newDate(year, month, 7), 8, cars.get(0));

    drivers.get(1).addRide("Donostia", "Bilbo", UtilDate.newDate(year, month, 15), 3, cars.get(1));
    drivers.get(1).addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 25), 5, cars.get(1));
    drivers.get(1).addRide("Eibar", "Gasteiz", UtilDate.newDate(year, month, 6), 5, cars.get(1));

    drivers.get(2).addRide("Bilbo", "Donostia", UtilDate.newDate(year, month, 14), 3, cars.get(2));

}
```

2. “Write simple units of code”

1)

Arazoa: Adarketa ugari (if, else if, try-catch ugari), adibidez register() metodoetan.
Helburua: Adar kopurua ≤ 4 mantentzea.

```java
public User register(String name, String email, String password, String mota) {
    try {
        db.getTransaction().begin();
        TypedQuery<User> query = db.createQuery("SELECT r FROM User r WHERE r.name=?1 OR r.email=?2",User.class);
        query.setParameter(1, name);
        query.setParameter(2, email);
        List<User> userList = query.getResultList();

        if (userList.isEmpty()) {
            if (mota==ResourceBundle.getBundle("Etiquetas").getString("RegisterGUI.Driver")) {
                Driver dri = new Driver(email, name, password);
                db.persist(dri);
                db.getTransaction().commit();
                return(dri);

            }else if (mota==ResourceBundle.getBundle("Etiquetas").getString("RegisterGUI.Traveler")) {
                Traveler tra = new Traveler(email, name, password);
                db.persist(tra);
                db.getTransaction().commit();
                return(tra);
            } else if(mota==ResourceBundle.getBundle("Etiquetas").getString("RegisterGUI.Admin")){
                Admin adm = new Admin(email, name, password);
                db.persist(adm);
                db.getTransaction().commit();
                return(adm);
            }else {
                return(null);
            }


        } else {
            db.getTransaction().commit();
            return(null);
        }

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

Logika banatu da hiru laguntzaileetan:

`userAlreadyExists()` → erabiltzailea dagoen egiaztatzen du.
`createUserByType()` → mapa baten bidez mota bakoitza sortzen du.
`getString()` → itzulpenak zentralizatzen ditu.
Metodoa laburragoa eta irakurgarriagoa bihurtu da, adar kopurua 5tik 2ra jaitsi da.

```java
public User register(String name, String email, String password, String mota) {
    try {
        db.getTransaction().begin();

        if (userAlreadyExists(name, email)) {
            db.getTransaction().commit();
            return null;
        }

        User newUser = createUserByType(mota, email, name, password);
        if (newUser == null) {
            db.getTransaction().commit();
            return null;
        }

        db.persist(newUser);
        db.getTransaction().commit();
        return newUser;

    } catch (Exception e) {
        e.printStackTrace();
        if (db.getTransaction().isActive()) db.getTransaction().rollback();
        return null;
    }
}

/**
 * Laguntzailea: erabiltzaile existitzen den egiaztatzen du
 */
private boolean userAlreadyExists(String name, String email) {
    TypedQuery<User> query = db.createQuery("SELECT r FROM User r WHERE r.name=?1 OR r.email=?2", User.class);
    query.setParameter(1, name);
    query.setParameter(2, email);
    return !query.getResultList().isEmpty();
}

    /**
     * Laguntzailea: erabiltzaile mota baten arabera objektua sortzen du
     */
    private User createUserByType(String mota, String email, String name, String password) {
        String driverKey = getString("RegisterGUI.Driver");
        String travelerKey = getString("RegisterGUI.Traveler");
        String adminKey = getString("RegisterGUI.Admin");

        Map<String, Supplier<User>> userFactories = new HashMap<>();
        userFactories.put(driverKey, () -> new Driver(email, name, password));
        userFactories.put(travelerKey, () -> new Traveler(email, name, password));
        userFactories.put(adminKey, () -> new Admin(email, name, password));

        return userFactories.getOrDefault(mota, () -> null).get();
    }

    /**
     * Laguntzailea: ResourceBundle-etik testuak lortzeko
     */
    private String getString(String key) {
        return ResourceBundle.getBundle("Etiquetas").getString(key);
    }
```

2)

Arazoa: if else asko daude deleteUser metodoan
Helburua: Adar kopurua ≤ 4 mantentzea.

```
public boolean deleteUser(User user) {
    try {
        db.getTransaction().begin();

        if (user instanceof Traveler) {
            Traveler t = (Traveler) user;
            for (Reservation reservation:t.getReservations()) {
                reservation.cancelReservation();
            }
        } else if (user instanceof Driver) {
            Driver d = (Driver) user;
            List<Ride> ridesCopy = new ArrayList<>(d.getRides());
            for (Ride ride:ridesCopy) {
                cancelRides(ride);
            }
        } else if (user instanceof Admin) {
            System.out.println("Ezin da administratzaile bat ezabatu.");
            db.getTransaction().commit();
            return true;
        } else {
            db.getTransaction().commit();
            return false;
        }
        User managedUser = db.find(User.class, user);
        db.remove(managedUser);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        db.getTransaction().rollback();
        e.printStackTrace();
        return false;
    }
}
```

Metodoan erabiltzaile mota begiratzen duen kode zatia metodo berri bat bezala atera dugu:

```
public boolean deleteUser(User user) {
    try {
        db.getTransaction().begin();

        if(!userMota(user)) {
            return false;
        }
        User managedUser = db.find(User.class, user);
        db.remove(managedUser);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        db.getTransaction().rollback();
        e.printStackTrace();
        return false;
    }
}

private boolean userMota(User user) {
    if (user instanceof Traveler) {
        Traveler t = (Traveler) user;
        for (Reservation reservation:t.getReservations()) {
            reservation.cancelReservation();
        }
        return true;
    } else if (user instanceof Driver) {
        Driver d = (Driver) user;
        List<Ride> ridesCopy = new ArrayList<>(d.getRides());
        for (Ride ride:ridesCopy) {
            cancelRides(ride);
        }
        return true;
    } else if (user instanceof Admin) {
        System.out.println("Ezin da administratzaile bat ezabatu.");
        db.getTransaction().commit();
        return true;
    } else {
        db.getTransaction().commit();
        return false;
    }
}
```

EZ DITUGU “WRITE SIMPLE UNITS OF CODE” MOTAKO BAD-SMELL GEHIAGORIK AURKITU

3. Duplicate Code

```
public boolean cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();
        ride.cancelRide();
        for (Reservation reservation : ride.getAcceptedReservations()) { // Dirua bueltatu
            float money = reservation.getTraveler().getMoney() + reservation.getPrezioa();
            reservation.getTraveler().setMoney(money);
            Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
            Movement move = new Movement(reservation.getTraveler().getMovementsCount(), reservation.getPrezioa(),
                    true, date);
            reservation.getTraveler().addMovement(move);

            reservation.cancel();

            db.merge(reservation);
            db.merge(reservation.getTraveler());
        }

        Driver driver = ride.getDriver();
        driver.cancelRide(ride);

        db.merge(ride);
        db.merge(driver);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

```
public boolean cancelRides(Ride ride) {
    try {
        ride.cancelRide();
        for (Reservation reservation : ride.getAcceptedReservations()) { // Dirua bueltatu
            float money = reservation.getTraveler().getMoney() + reservation.getPrezioa();
            reservation.getTraveler().setMoney(money);
            Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
            Movement move = new Movement(reservation.getTraveler().getMovementsCount(), reservation.getPrezioa(),
                    true, date);
            reservation.getTraveler().addMovement(move);

            reservation.cancel();

            db.merge(reservation);
            db.merge(reservation.getTraveler());
        }

        Driver driver = ride.getDriver();
        driver.cancelRide(ride);

        db.merge(ride);
        db.merge(driver);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        return false;
    }
}
```

cancelRide() eta cancelRides() funtzioen kodea ia berdinak dira. Dagoen kodigo kopurua murrizteko hurrengoa egin dugu: Hasteko identifikatu bi funtzioen desberdintasuna (cancelRide() transakzioak begin eta commit egiten du eta cancelRides()-ek aldiz ez), ondoren kode berdina funtzio batean sartu dugu. Bien arteko desberdintasuna logika sinple batekin ezarri dugu. cancelRide() true pasako du eta honela transakzioa hasi eta commit egingo du eta aldiz cancelRides()-ek false pasako du eta hau ezta egingo.

```java
public boolean cancelRide(Ride ride) {
    // cancelRide kasuan transakzioa barruan kudeatzen da
    return cancelRideInternal(ride, true);
}

public boolean cancelRides(Ride ride) {
    // cancelRides kasuan transakzioa kanpoan kudeatzen da
    return cancelRideInternal(ride, false);
}

//transakzioa da bien arteko desberdintasuna eta booleanaren bidez identifikatuko da
private boolean cancelRideInternal(Ride ride, boolean withTransaction) {
    try {
        if (withTransaction) db.getTransaction().begin();

        // Ride bertan behera uztea
        ride.cancelRide();

        // Erreserba onartutakoak itzultzea
        for (Reservation reservation : ride.getAcceptedReservations()) {
            Traveler traveler = reservation.getTraveler();
            float money = traveler.getMoney() + reservation.getPrezioa();
            traveler.setMoney(money);

            Date date = Date.from(LocalDate.now().atStartOfDay(ZoneId.systemDefault()).toInstant());
            Movement move = new Movement(traveler.getMovementsCount(), reservation.getPrezioa(), true, date);
            traveler.addMovement(move);

            reservation.cancel();

            db.merge(reservation);
            db.merge(traveler);
        }

        // Driver eguneratzea
        Driver driver = ride.getDriver();
        driver.cancelRide(ride);

        db.merge(ride);
        db.merge(driver);

        if (withTransaction) db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        if (withTransaction && db.getTransaction().isActive()) db.getTransaction().rollback();
        return false;
    }
}
```

EZ DITUGU “DUPLICATE CODE” MOTAKO BAD-SMELL GEHIAGORIK AURKITU

4. Keep unit interfaces small

1)

Metodo batzuek 5-6 parametro dituzte, adibidez: createRide(String from, String to, Date date, float price, String driverEmail, Car pCar).

Helburua: Gehienez 4 parametro, eta parametroak objektu batean biltzea. Hau egiteko. Arazoa da aldaketa hau egiteko, aldaketa dataAccess.java-n egitea ez zela nahikoa izango eta aldaketa asko egin beharko litzatekela kodigoan. Aldaketa sinplena kasu honetan String guztiak batzea izan da eta funtzioan bertan separatzea. Berez, konplikazio bat da, baina interfazearen txikizioa erakustearren egingo dugu.

```
public Ride createRide(String from, String to, Date date, float price, String driverEmail, Car pCar)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {

    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverEmail
            + " date " + date);
    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, driverEmail);
        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, price, pCar);
        // next instruction can be obviated
        db.persist(driver);
        db.getTransaction().commit();

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

}
```


Egin duguna lortzeko BLFacade-n deitzen denean stringaren batura egin dugu eta ondoren txikitu dugu interfaze berriarekin deitu diogu funtzioari. Funtzioan bertan separatuko da. Errepikatzen dut hau berez ez dala praktikoa baina interfazea txikitu dela erakusteko egin dugula.

```
@WebMethod
public Ride createRide(String from, String to, Date date, float price, String driverEmail, Car pCar)
        throws RideMustBeLaterThanTodayException, RideAlreadyExistException {

    dbManager.open();
    String rideInfo = from+"/"+to+"/"+driverEmail;
    Ride ride = dbManager.createRide(rideInfo, date, price, pCar);
    dbManager.close();
    return ride;
};

/**
```

```
public Ride createRide(String rideInfo, Date date, float price, Car pCar)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    String[] parts = rideInfo.split("/");
    if (parts.length != 3) {
        throw new IllegalArgumentException("Input must be in the format: from/to/driverEmail");
    }
    String from = parts[0];
    String to = parts[1];
    String driverEmail = parts[2];
    System.out.println(">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverEmail
            + " date " + date);
    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, driverEmail);
        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, price, pCar);
        // next instruction can be obviated
        db.persist(driver);
        db.getTransaction().commit();

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

}
```

2)

getRideByParams metodoa parametro gehiegi jasotzen ditu, bere funtzioa betetzeko behar dituenakl bino gehiago alegia: getRideByParams(Driver driver, **int** nPlaces, **float** price, String from, String to, Date date)

```
public Ride getRideByParams(Driver driver, int nPlaces, float price, String from, String to, Date date) {
    TypedQuery<Ride> query = db
            .createQuery("SELECT r FROM Ride r WHERE r.driver=?1 AND r.nPlaces=?2 AND r.price=?3", Ride.class);
    query.setParameter(1, driver);
    query.setParameter(2, nPlaces);
    query.setParameter(3, price);

    List<Ride> rideList = query.getResultList();
    if (!rideList.isEmpty()) {
        Ride actualRide = rideList.get(0);
        return actualRide;

    } else {
        return null;
    }

}
```

Arazo hau konpontzeko egin beharreko errefaktorizazio bakarra erabiltzen ez diren parametroak ezabatzea da:

```java
public Ride getRideByParams(Driver driver, int nPlaces, float price) {
    TypedQuery<Ride> query = db
            .createQuery("SELECT r FROM Ride r WHERE r.driver=?1 AND r.nPlaces=?2 AND r.price=?3", Ride.class);
    query.setParameter(1, driver);
    query.setParameter(2, nPlaces);
    query.setParameter(3, price);

    List<Ride> rideList = query.getResultList();
    if (!rideList.isEmpty()) {
        Ride actualRide = rideList.get(0);
        return actualRide;

    } else {
        return null;
    }

}
```

EZ DITUGU “KEEP UNIT INTERFACES SMALL” MOTAKO BAD-SMELL GEHIAGORIK AURKITU