ERREFAKTORIZAZIOAK

EGILEA: Oier Unanue

- "Write short units of code":

Hasierako kodea:

```
public boolean addCar(String username, Car kotxe) {
    try {
        boolean b = isAdded(username, kotxe.getMatrikula());
        if (!b) {
            db.getTransaction().begin();
            Driver dri = getDriver(username);
            dri.addCar(kotxe);
            db.persist(dri);
            db.getTransaction().commit();
        }
        return !b;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

Errefaktorizatutako kodea:

```java
public boolean addCar(String username, Car kotxe) {
    try {
        if (isCarAlreadyAdded(username, kotxe.getMatrikula())) {
            return false;
        }

        addCarToDriver(username, kotxe);
        return true;
    } catch (Exception e) {
        handleTransactionRollback();
        e.printStackTrace();
        return false;
    }
}

private boolean isCarAlreadyAdded(String username, String matrikula) {
    return isAdded(username, matrikula);
}

private void addCarToDriver(String username, Car kotxe) {
    db.getTransaction().begin();
    Driver driver = getDriver(username);
    driver.addCar(kotxe);
    db.persist(driver);
    db.getTransaction().commit();
}

private void handleTransactionRollback() {
    if (db.getTransaction().isActive()) {
        db.getTransaction().rollback();
    }
}
```

Egindako errefaktorizazioren deskribapena:

Kode honekin egin dudana addCar() metodoa moztu izan da 15 lerro baino gutxiago edukitzeko. hau lortzeko hainbat metodo auxiliar sortu ditut eta horrela addCar() metodoa moztea lortu dut. Horrela errazagoa izango da kodea ulertzea eta testak egitea.

- "Write simple units of code":

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;
    }
```

Errefaktorizatutako kodea:

```
public List<Booking> getBookingFromDriver(String username) {
    try {
        Driver driver = getDriverByUsername(username);
        return getActiveBookings(driver);
    } catch (Exception e) {
        e.printStackTrace();
        rollbackTransaction();
        return null;
    }
}

private Driver getDriverByUsername(String username) {
    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();
    db.getTransaction().commit();
    return driver;
}

private List<Booking> getActiveBookings(Driver driver) {
    List<Booking> bookings = new ArrayList<>();
    for (Ride ride : driver.getCreatedRides()) {
        if (ride.isActive()) {
            bookings.addAll(ride.getBookings());
        }
    }
    return bookings;
}

private void rollbackTransaction() {
    if (db.getTransaction().isActive()) {
        db.getTransaction().rollback();
    }
}
```

Egindako errefaktorizazioren deskribapena:

Hemen egin dudana, konplexutasuna maneiatzeko bifurkazio puntuak murriztu ditut. Logika hainbat metodo desberdinetan banatzean, bifurkazio puntuak murriztu egiten dira eta horrela maneiatzeko errezagoak bihurtzen dira.

- “Duplicate code”:

Hasierako kodea:

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

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

Errefaktorizatutako kodea:

```
// Método genérico para buscar usuarios
private <T> T findUser(String username, Class<T> userType) {
    TypedQuery<T> query = db.createQuery("SELECT u FROM " + userType.getSimpleName() + " u WHERE u.username = :username
    query.setParameter("username", username);
    List<T> resultList = query.getResultList();
    return resultList.isEmpty() ? null : resultList.get(0);
}

// Refactorización en getDriver, getTraveler
public Driver getDriver(String erab) {
    return findUser(erab, Driver.class);
}

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

Egindako errefaktorizazioren deskribapena:

Metodo bakoitzean bilaketa bat egin ordez, bilaketa hori egiten duen metodoa sortu kodigoa ez duplikatzeko.

- "Keep unit interfaces small":

Hasierako kodea:

```
  public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua,
          boolean aurk) {
      try {
          db.getTransaction().begin();

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

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

Errefaktorizatutako kodea:

```
  public boolean erreklamazioaBidali(ComplaintRequest request) {
      try {
          db.getTransaction().begin();
          Complaint erreklamazioa = new Complaint(request.getSender(), request.getRecipient(), request.getDate(), request.g
          db.persist(erreklamazioa);
          db.getTransaction().commit();
          return true;
      } catch (Exception e) {
          e.printStackTrace();
          db.getTransaction().rollback();
          return false;
      }
  }
```

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

Egindako errefaktorizazioren deskribapena: Metodo honetan parametro gehiegi pasatzen ziren, 4 baino gehiago. Beraz, kudeatzeko errezagoa izateko eta errezago ulertzeko beste klase bat sortu dut parametro guzti hauek jasotzen dituenak eta horrela soilik parametro bat pasatzen diogu metodoari.

EGILEA: Manex Ormazabal

- "Write short units of code":

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();
    }
}
```

Errefaktorizatutako kodea:

```
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();
        for (Booking booking : ride.getBookings()) {
            cancelBooking(booking);
        }
        deactivateRide(ride);
        db.getTransaction().commit();
    } catch (Exception e) {
        handleTransactionFailure(e);
    }
}

private void cancelBooking(Booking booking) {
    if (booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined")) {
        refundTraveler(booking);
    }
    booking.setStatus("Rejected");
    db.merge(booking);
}

private void refundTraveler(Booking booking) {
    Traveler traveler = booking.getTraveler();
    double price = booking.prezioaKalkulatu();

    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() - price);
    traveler.setMoney(traveler.getMoney() + price);
    db.merge(traveler);

    addMovement(traveler, "BookDeny", price);
}

private void deactivateRide(Ride ride) {
    ride.setActive(false);
    db.merge(ride);
}
```

Egindako errefaktorizazioren deskribapena: Errefaktorizazio honetan egin dudana, cancelRide() metodoa, metodo txikiagoetan banatzea izan da. canceBooking(), refundTraveler(),... sortu ditut, eta horrela errazagoa da metodoa ulertzea eta akatsik egon ezkero, azpimetodoen artean errazago bilatuko dugu akatsa.

- "Write simple units of code":

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;
    }
}
```

Errefaktorizatutako kodea:

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

        Traveler traveler = getTraveler(username);
        if (!isTravelerValid(traveler) || !isRideAvailable(ride, seats) || !isBookingAffordable(traveler, ride, seats, desk))
            return false;
        }

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

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

private boolean isTravelerValid(Traveler traveler) {
    return traveler != null;
}

private boolean isRideAvailable(Ride ride, int seats) {
    return ride.getnPlaces() >= seats;
}

private boolean isBookingAffordable(Traveler traveler, Ride ride, int seats, double desk) {
    double ridePriceDesk = (ride.getPrice() - desk) * seats;
    return traveler.getMoney() >= ridePriceDesk;
}


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

private void updateRideAndTraveler(Ride ride, Traveler traveler, Booking booking, int seats) {
    ride.setnPlaces(ride.getnPlaces() - seats);
    traveler.addBookedRide(booking);
    double availableBalance = traveler.getMoney();
    traveler.setMoney(availableBalance - ((ride.getPrice() - booking.getDeskontua()) * seats));
    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ((ride.getPrice() - booking.getDeskontua()) * seats));
    db.merge(ride);
    db.merge(traveler);
}
```

Egindako errefaktorizazioren deskribapena: Errefaktorizazio honetan egin dudana, bifurkazio puntuak murriztea da, metodo printzipala sinlifikatzeko, eta mantainagarriagoa eta ulertzeko errazagoa izateko.

- “Duplicate code”:

Hasierako kodea:

```
public void updateTraveler(Traveler traveler) {
    try {
        db.getTransaction().begin();
        db.merge(traveler);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void updateDriver(Driver driver) {
    try {
        db.getTransaction().begin();
        db.merge(driver);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void updateUser(User user) {
    try {
        db.getTransaction().begin();
        db.merge(user);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

Errefaktorizatutako kodea:

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

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

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

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

Egindako errefaktorizazioren deskribapena: Kasu honetan, hiru metodok kode bera ibiltzen zuten, entitate desberdinentzako, eta egin dudana, metodo general bat sortzea izan da, edozein entitaterako, eta azpimetodoekin banatu entitateen kasuak, denek metodo berdina erabiliz.

- "Keep unit interfaces small":

Hasierako kodea:

```
public void addMovement(User user, String eragiketa, double amount) {
    try {
        db.getTransaction().begin();

        db.persist(movement);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

Errefaktorizatutako kodea:

```
public void addMovement(Movement movement) {
    try {
        db.getTransaction().begin();

        db.persist(movement);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

Egindako errefaktorizazioren deskribapena: Errefaktorizazio honetan, hiru parametro zituen metodoa, movement parametroagatik ordezkatu dut. Honetarako, BLFacadeImplementationen ere metodoa aldatu beharra izan dut, eta horrela egoki funtzionatzea lortu dut. Horrela addMovement metodoa sinpleagoa da eta errazago erabili daiteke.

EGILEA: Aretz Gurrutxaga

- "Write short units of code":

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;
    }
}
```

Errefaktorizatutako kodea:

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

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

    for (Alert alert : alerts) {
        boolean found = false;
        for (Ride ride : rides) {
            if (isRideMatchingAlert(ride, alert)) {
                alert.setFound(true);
                found = true;
                if (alert.isActive()) {
                    alertFound = true;
                }
                break;
            }
        }
        if (!found) {
            alert.setFound(false);
        }
        db.merge(alert);
    }

    return alertFound;
}

private List<Alert> findAlertsByUsername(String username) {
    TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username", Alert.class);
    alertQuery.setParameter("username", username);
    return alertQuery.getResultList();
}


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

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

          List<Alert> alerts = findAlertsByUsername(username);
          List<Ride> rides = findUpcomingRides();

          boolean alertFound = updateAlertsStatus(alerts, rides);

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

Egindako errefaktorizazioren deskribapena: Errefaktorizazio honetan,updateAlertaAurkituak() metodoa sinplifikatu dut beste 4 metodo sortuta. isRideMatchingAlert, updateAlertStatus, findAlertByUsername eta findUpcomingRides metodoak erabiliz lehen luzeagoa zen metodoa motzago eta ulergarriago egin dut

- "Write simple units of code":

Hasierako kodea:

```
public void deleteUser(User us) {
    try {
        if (us.getMota().equals("Driver")) {
            List<Ride> rl = getRidesByDriver(us.getUsername());
            if (rl != null) {
                for (Ride ri : rl) {
                    cancelRide(ri);
                }
            }
            Driver d = getDriver(us.getUsername());
            List<Car> cl = d.getCars();
            if (cl != null) {
                for (int i = cl.size() - 1; i >= 0; i--) {
                    Car ci = cl.get(i);
                    deleteCar(ci);
                }
            }
        } else {
            List<Booking> lb = getBookedRides(us.getUsername());
            if (lb != null) {
                for (Booking li : lb) {
                    li.setStatus("Rejected");
                    li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
                }
            }
            List<Alert> la = getAlertsByUsername(us.getUsername());
            if (la != null) {
                for (Alert lx : la) {
                    deleteAlert(lx.getAlertNumber());
                }
            }
        }
        db.getTransaction().begin();
        us = db.merge(us);
        db.remove(us);
        db.getTransaction().commit();
    } catch (Exception e) {

        e.printStackTrace();
    }
}
```

Errefaktorizatutako kodea:

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

    private void deleteAlerts(List<Alert> alerts) {
        if (alerts != null) {
            for (Alert alert : alerts) {
                deleteAlert(alert.getAlertNumber());
            }
        }
    }

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

    private void handleUserDeletion(User user) {
        List<Booking> bookings = getBookedRides(user.getUsername());
        rejectBookings(bookings);

        List<Alert> alerts = getAlertsByUsername(user.getUsername());
        deleteAlerts(alerts);
    }
```

```java
private void deleteCars(List<Car> cars) {
    if (cars != null) {
        for (int i = cars.size() - 1; i >= 0; i--) {
            deleteCar(cars.get(i));
        }
    }
}

private void cancelRides(List<Ride> rides) {
    if (rides != null) {
        for (Ride ride : rides) {
            cancelRide(ride);
        }
    }
}

private void handleDriverDeletion(User user) {
    List<Ride> rides = getRidesByDriver(user.getUsername());
    cancelRides(rides);

    Driver driver = getDriver(user.getUsername());
    deleteCars(driver.getCars());
}

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

Egindako errefaktorizazioren deskribapena: deleteUser metodoa 7 metodo desberdinetan banatu dut metodoaren erabaki puntuak gutxituz.Honekin metodoa argiagoa eta irakurgarriagoa da. Hortaz aparte, modu honetara metodo bakoitza testeatu dezakegu banan bana eta gañera metodo hauetako bakoitza berrerabilia izan daiteke beste zerbait implementatzerakoan.

- “Duplicate code”:

Hasierako kodea:

```
public void deleteDiscount(Discount dis) {
    try {
        db.getTransaction().begin();
        db.remove(dis);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void updateDiscount(Discount dis) {
    try {
        db.getTransaction().begin();
        db.merge(dis);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

Errefaktorizatutako kodea:

```
private void executeTransaction(Runnable operation) {
    try {
        db.getTransaction().begin();
        operation.run();
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}

public void deleteDiscount(Discount discount) {
    executeTransaction(() -> db.remove(discount));
}

public void updateDiscount(Discount discount) {
    executeTransaction(() -> db.merge(discount));
}
```

Egindako errefaktorizazioren deskribapena: executeTransaction() metodoa sortu dut ondoren deleteDiscount() eta updateDiscount() metodoetan erabiltzeko. Horrela bikoiztutako kodea ezabatzen dugu eta honek mantenugarritasuna hobetzen du.

- "Keep unit interfaces small":

Hasierako kodea:

```
public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
        throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
    System.out.println(
            ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
    if (driverName==null) return null;
    try {
        if (new Date().compareTo(date) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(
                    ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.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;
    }


}
```

Errefaktorizatutako kodea:

```java
package domain;

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;
    }

    // Getters
    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;
```

```java
  public Ride createRide(RideDetails rideDetails, String driverName)
          throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
      System.out.println(">> 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) {
              throw new RideMustBeLaterThanTodayException(
                      ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
          }

          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.RideAlreadyExist"));
          }

          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;
      }
  }
```

Egindako errefaktorizazioren deskribapena: Errefaktorizazio honetan, metodo bat 6 atributu edukitzetik 2 atributu edukitzera pasa dut. Horretarako RideDetails klasea sortu dut eta mota hortako atributu batekin ordezkatu ditut createRide metodoko 5 atributu. Ondo funtzionatzeko honek BLFacade eta BLFacadeImplementetion klaseetan ere aldaketak egin ditut. Errefaktorizazio honen bitartez metodoaren argitasuna, mantenugarritasuna eta bererabilgarritasuna hobetzen ditugu.