ERREFAKTORIZAZIOAK:

"Write short units of code"

1.1.-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();
    }
}
```

1.2.-Errefaktutako kodea:

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

1.3.-Deskribapena: Lehen metodoa oso luzea zen, if handi batekin eta honen barruan kodigo lerro asko. Hau laburtzeko bi Extract Method egin ditut, bi metodo sortuz. Batek “Driver” motako User ezabatzen ditu eta bestek beste motakoak. Honekin lehen metodoaren luzera asko jaixten da.

1.4.- Julen Etxeberria, Aimar San Martin

2.1-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) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

2.2-Errefaktutako 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();

        alertFound = alertakAurkitu(alertFound, alerts, rides);

        db.getTransaction().commit();
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
public boolean alertakAurkitu(boolean alertFound, List<Alert> alerts, List<Ride> rides) {
    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);
    }
    return alertFound;
}
```

2.3-Deskribapena:Lehen metodoa luzeegia zen eta kasu honetan bigizta bat ateratze aerabaki dugu. Modu honetan metodoa bitan banatzen dugu eta usain txarra kentzen dugu

2.4-Egilea: Julen Etxeberria, Aimar San Martin

“Write simple units of code”

1.1-Hasierako kodea:

```java
//username @Id bezala duen erabiltzaileari dirua gehitu edo kendu egiten zaio. deposit true bada amount gehituko zaio
// eta bestela dirua kenduko zaio. Erabiltzailearen diru ezin du inoiz negatiboa izan.

public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null) {  // erabiltzailea ez bada aurkitzen ez du ezer egiten eta false bueltatu.
            double currentMoney = user.getMoney();
            if (deposit) {
                user.setMoney(currentMoney + amount);  //deposit true izanda dirua gehitzen zaio
            } else {
                if ((currentMoney - amount) < 0)
                    user.setMoney(0); // dirua kentzen zaio, baina ez duenez nahikoa, 0n geratzen da ez geratzeko negatibon
                else
                    user.setMoney(currentMoney - amount);  // dirua kentzen zaio
            }
            db.merge(user);
            db.getTransaction().commit();
            return true;  // Dena errore gabe gertatu bada true bueltatzen du
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

1.2-Errefaktutako kodea:

```java
//username @Id bezala duen erabiltzaileari dirua gehitu edo kendu egiten zaio. deposit true bada amount gehituko zaio
// eta bestela dirua kenduko zaio. Erabiltzailearen diru ezin du inoiz negatiboa izan.

public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
    try {
        db.getTransaction().begin();
        User user = getUser(username);
        if (user != null) {  // erabiltzailea ez bada aurkitzen ez du ezer egiten eta false bueltatu.
            diruaAldatu(amount, deposit, user);
            db.merge(user);
            db.getTransaction().commit();
            return true;  // Dena errore gabe gertatu bada true bueltatzen du
        }
        db.getTransaction().commit();
        return false;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
public void diruaAldatu(double amount, boolean deposit, User user) {
    double currentMoney = user.getMoney();
    if (deposit) {
        user.setMoney(currentMoney + amount);  //deposit true izanda dirua gehitzen zaio
    } else {
        if ((currentMoney - amount) < 0)
            user.setMoney(0); // dirua kentzen zaio, baina ez duenez nahikoa, 0n geratzen da ez geratzeko negatibon
        else
            user.setMoney(currentMoney - amount);  // dirua kentzen zaio
    }
}
```

1.3-Deskribapena: Metodo honen konplexutasun ziklomatikoa 4 zen, usain txarra zuen. Hau aldatzeko metodoaren zati bat atera dugu beste metodo bat eginez eta konplexutasuna jaisten.

1.4-Egileak:Julen Etxeberria, Aimar San Martin

2.1- Hasierako kodea:

```
public void removeTraveler(User us) {
    List<Booking> lb = getBookedRides(us.getUsername());
    if (lb != null) {
        for (Booking li : lb) {
            li.setStatus("Rejected");
            li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
        }
    }
    List<Alert> la = getAlertsByUsername(us.getUsername());
    if (la != null) {
        for (Alert lx : la) {
            deleteAlert(lx.getAlertNumber());
        }
    }
}
```

2.2-Errefraktutako kodea:

```
public void removeTraveler(User us) {
    List<Booking> lb = getBookedRides(us.getUsername());
    deleteRides(lb);
    List<Alert> la = getAlertsByUsername(us.getUsername());
    deleteAlerts(la);
}
public void deleteAlerts(List<Alert> la) {
    if (la != null) {
        for (Alert lx : la) {
            deleteAlert(lx.getAlertNumber());
        }
    }
}
public void deleteRides(List<Booking> lb) {
    if (lb != null) {
        for (Booking li : lb) {
            li.setStatus("Rejected");
            li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
        }
    }
}
```

2.3-Deskribapena: Lehen sortutako removeTraveler metodoaren konplexutasun ziklomatikoa 4 da, altua. Hau aldatzeko bi metodo txiki eta simple egin ditugu.

2.4-Egilea:Julen Etxeberria, Aimar San Martin

“Duplicate code”

1.1-Hasierako kodea:

```
public Driver getDriver(String erab) {
    Duplication
    TypedQuery<Driver> query = db.createQuery( 1 "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);
    }
}
```

```java
public List<Booking> getBookingFromDriver(String username) {
    try {
        db.getTransaction().begin();
        Duplication
        TypedQuery<Driver> query = db.createQuery( 2  "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;
    }
}
```

```java
public List<Ride> getRidesByDriver(String username) {
    try {
        db.getTransaction().begin();
        Duplication
        TypedQuery<Driver> query = db.createQuery( 3  "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;
    }
}
```

1.2-Errefraktutako kodea:

```java
private static final String sqlDriver= "SELECT d FROM Driver d WHERE d.username = :username";
```

```java
public Driver getDriver(String erab) {

    TypedQuery<Driver> query = db.createQuery(sqlDriver, Driver.class);
    query.setParameter("username", erab);
    List<Driver> resultList = query.getResultList();
    if (resultList.isEmpty()) {
        return null;
    } else {
        return resultList.get(0);
    }
}
```

```java
public List<Booking> getBookingFromDriver(String username) {
    try {
        db.getTransaction().begin();

        TypedQuery<Driver> query = db.createQuery(sqlDriver,
                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;
    }
```

```java
public List<Ride> getRidesByDriver(String username) {
    try {
        db.getTransaction().begin();
        TypedQuery<Driver> query = db.createQuery(sqlDriver,
                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;
    }
}
```

1.3- Deskribapena: Bertan ikusten ditugun 3 funtzioetan denek erabiltzen dute sql bilaketa berdina eta denetan idazten du berriz ere orduan egin duguna da hori atributu baten gorde eta atributu hori erabili, sql bilaketa hori idazterakoan behin eta berriz erroreren bat eduki dezakegu.

1.4-Egilea:Julen Etxeberria, Aimar San Martin

2.1-Hasierako kodea:

```java
public BaloraGUI(String username) {

    setBussinessLogic(LoginGUI.getBusinessLogic());
    this.setSize(495, 290);
    Duplication
    this.setTitle(ResourceBundle.getBundle( 1  "Etiquetas").getString("BezeroGUI.Baloratu"));
    this.setResizable(false);

    // Bi aukerak aztertu.
    User us = appFacadeInterface.getUser(username);

    Duplication
    lbltxt = new JLabel(ResourceBundle.getBundle( 2  "Etiquetas").getString("BaloraGUI.Izena") + ": " + username);

    slider = new JSlider(JSlider.HORIZONTAL, 1, 5, 1);
    slider.setForeground(new Color(100, 100, 100));
    slider.setMinorTickSpacing(1);
    slider.setMajorTickSpacing(1);
    slider.setPaintLabels(true);
    slider.setPaintTicks(true);

    Duplication
    baloratu = new JButton(ResourceBundle.getBundle( 3  "Etiquetas").getString("BaloraGUI.Baloratu"));
    baloratu.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int kont = slider.getValue();
            double bal;
            int kop;
```

2.2-Errefraktutako kodea:

```
private static final String eti="Etiquetas";
```

```
public BaloraGUI(String username) {

    setBussinessLogic(LoginGUI.getBusinessLogic());
    this.setSize(495, 290);
    Duplication
    this.setTitle(ResourceBundle.getBundle( 1  eti).getString("BezeroGUI.Baloratu"));
    this.setResizable(false);

    // Bi aukerak aztertu.
    User us = appFacadeInterface.getUser(username);

    Duplication
    lbltxt = new JLabel(ResourceBundle.getBundle( 2  eti).getString("BaloraGUI.Izena") + ": " + username);

    slider = new JSlider(JSlider.HORIZONTAL, 1, 5, 1);
    slider.setForeground(new Color(100, 100, 100));
    slider.setMinorTickSpacing(1);
    slider.setMajorTickSpacing(1);
    slider.setPaintLabels(true);
    slider.setPaintTicks(true);

    Duplication
    baloratu = new JButton(ResourceBundle.getBundle(eti).getString("BaloraGUI.Baloratu"));
    baloratu.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int kont = slider.getValue();
            double bal;
```

2.3- Deskribapena: Etiquetas hitza askotan erabiltzen eta hainbestetan idatzita erroreren bat sortu daiteke beraz aldagai baten gorde dugu eta hori erabili dugu behin eta berriz idatzi beharrean.

2.4-Egilea:Julen Etxeberria, Aimar San Martin

"Keep unit interfaces small"

1.1-Hasierako kodea:

```java
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;
    }
```

1.2-Errefraktutako kodea:

```java
public class RideData {
    private String from;
    private String to;
    private Date date;
    private int nPlaces;
    private float price;
    private String driverName;

    public RideData(String from, String to, Date date, int nPlaces, float price, String driverName) {
        this.from = from;
        this.to = to;
        this.date = date;
        this.nPlaces = nPlaces;
        this.price = price;
        this.driverName = driverName;
    }


    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; }
    public String getDriverName() { return driverName; }
}
```

```java
public RideData RideData(String from, String to, Date date, int nPlaces, float price, String driverName) {
    // TODO Auto-generated method stub
    return new RideData(from, to, date, nPlaces, price, driverName);
}
```

```java
public Ride createRide(RideData rideData)throws RideAlreadyExistException, RideMustBeLaterThanTodayException {

    System.out.println(">> DataAccess: createRide => from= " + rideData.getFrom()
        + " to= " + rideData.getTo()
        + " driver= " + rideData.getDriverName()
        + " date= " + rideData.getDate());

    if (rideData.getDriverName() == null) return null;

    try {
        if (new Date().compareTo(rideData.getDate()) > 0) {
            System.out.println("ppppp");
            throw new RideMustBeLaterThanTodayException(
                ResourceBundle.getBundle("Etiquetas")
                .getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
        }

        db.getTransaction().begin();
        Driver driver = db.find(Driver.class, rideData.getDriverName());

        if (driver.doesRideExists(rideData.getFrom(), rideData.getTo(), rideData.getDate())) {
            db.getTransaction().commit();
            throw new RideAlreadyExistException(
                ResourceBundle.getBundle("Etiquetas")
                .getString("DataAccess.RideAlreadyExist"));
        }

        Ride ride = driver.addRide(
            rideData.getFrom(), rideData.getTo(), rideData.getDate(),
            rideData.getNPlaces(), rideData.getPrice());

        db.persist(driver);
        db.getTransaction().commit();

        return ride;

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

1.3- Deskribapena: Funtzio honek sarrera aldagai asko ditu beraz aldagai horiek guztiak klase batean sartu ditugu eta orduan sarrera aldagai bakarra edukiko du.

1.4-Egilea:Julen Etxeberria, Aimar San Martin

2.1-Hasierako kodea:

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

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

2.2-Errefraktutako kodea:

```java
public class KexaData {
    private String nor;
    private String nori;
    private Date gaur;
    private Booking booking;
    private String textua;
    private boolean aurk;

    public KexaData(String nor, String nori, Date gaur, Booking booking, String textua, boolean aurk) {
        this.nor = nor;
        this.nori = nori;
        this.gaur = gaur;
        this.booking = booking;
        this.textua = textua;
        this.aurk = aurk;
    }


    public String getNor() { return nor; }
    public String getNori() { return nori; }
    public Date getGaur() { return gaur; }
    public Booking getBooking() { return booking; }
    public String getTextua() { return textua; }
    public boolean isAurk() { return aurk; }
}
```

```java
public KexaData KexaData(String nor, String nori, Date gaur, Booking book, String textua, boolean aurk) {
    // TODO Auto-generated method stub
    return new KexaData(nor, nori, gaur, book, textua, aurk);
}
```

```java
public boolean erreklamazioaBidali(KexaData kdata) {
    try {
        db.getTransaction().begin();

        Complaint erreklamazioa = new Complaint(
                kdata.getNor(),
                kdata.getNori(),
                kdata.getGaur(),
                kdata.getBooking(),
                kdata.getTextua(),
                kdata.isAurk()
        );

        db.persist(erreklamazioa);
        db.getTransaction().commit();
        return true;

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

2.3- Deskribapena: Funtzio honek sarrera aldagai asko ditu beraz aldagai horiek guztiak klase batean sartu ditugu eta orduan sarrera aldagai bakarra edukiko du.

2.4-Egilea:Julen Etxeberria, Aimar San Martin