REFACTORIZACIÓN

INGENIERÍA DE SOFTWARE II

Iñigo Alzugaray

Aitor Gil

Pablo Jurado

13/10/2024

1.1 Código Inicial

1.2Código refactorizado

```
private static final String CITY_NAME = "Donostia";
```

```
driver1.addRide(CITY_NAME, to:"Madrid", date2, nPlaces:5, price:20); //ride1
driver1.addRide(from:"Irun", CITY_NAME, date2, nPlaces:5, price:2); //ride2
driver1.addRide(from:"Madrid", CITY_NAME, date3, nPlaces:5, price:5); //ride3
driver1.addRide(from:"Barcelona", CITY_NAME, date4, nPlaces:0, price:10); //ride4
driver2.addRide(CITY_NAME, to:"Hondarribi", date1, nPlaces:5, price:3); //ride5
```

1.3 Descripción

"Define a constant instead of duplicating this literal 'Donostia' 4 times" corresponde al "Bad Smell" de duplicación de código y está alineado con el tercer tipo de "Bad Smell" que se menciona en tu actividad: "Duplicate code".

Pablo Jurado

2.1 Código Inicial

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

        Traveler traveler = getTraveler(username);
        if (traveler == null || ride.getnPlaces() < seats || traveler.getMoney() < (ride.getPrice() - desk) * seats) {
            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(traveler.getMoney() - (ride.getPrice() - desk) * seats);
        traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + (ride.getPrice() - desk) * seats);
        db.merge(ride);
        db.merge(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

2.2 Código Refactorizado

```java
public boolean bookRide(BookingData bookingData) {
    try {
        db.getTransaction().begin();

        Traveler traveler = getTraveler(bookingData.getUsername());
        if (traveler == null || bookingData.getRide().getnPlaces() < bookingData.getSeats() || traveler.getMoney() < (bookingData.getRide().getPrice() - bookingData.geti

        saveBooking(bookingData.getRide(), bookingData.getSeats(), bookingData.getDesk(), traveler);
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

2.3 Descripción

El "Bad Smell" "Keep Unit Interfaces Small" se refiere a la necesidad de mantener las interfaces de los métodos (o unidades de código) simples y con un número reducido de parámetros. Al reducir la cantidad de parámetros del método bookRide, se mejora la claridad, la legibilidad y la mantenibilidad del código. Los test de este método siguen ejecutándose igual que antes.

Pablo Jurado

3.1 Código Inicial

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

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = db.createQuery(qlString:"SELECT a FROM Alert a WHERE a.traveler.username = :username",
                resultClass:Alert.class);
        alertQuery.setParameter(name:"username", username);
        List<Alert> alerts = alertQuery.getResultList();

        TypedQuery<Ride> rideQuery = db
                .createQuery(qlString:"SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", resultClass: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(found:true);
                    found = true;
                    if (alert.isActive())
                        alertFound = true;
                    break;
                }
            }
            if (!found) {
                alert.setFound(found:false);
            }
            db.merge(alert);
        }
        db.getTransaction().commit();
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

3.2 Código Refactorizado

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

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = db.createQuery(qlString:"SELECT a FROM Alert a WHERE a.traveler.username = :username",
                resultClass:Alert.class);
        alertQuery.setParameter(name:"username", username);
        List<Alert> alerts = alertQuery.getResultList();

        TypedQuery<Ride> rideQuery = db
                .createQuery(qlString:"SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", resultClass:Ride.class);
        List<Ride> rides = rideQuery.getResultList();

        alertFound = processAlerts(alerts, rides);

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

```java
private boolean processAlerts(List<Alert> alerts, List<Ride> rides) {
    boolean alertFound = false;
    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(found:true);
                found = true;
                if (alert.isActive())
                    alertFound = true;
                break;
            }
        }
        if (!found) {
            alert.setFound(found:false);
        }
        db.merge(alert);
    }
    return alertFound;
}
```

3.3 Descripción

Se ha extraído el bucle que procesa las alertas y los rides en un nuevo método llamado processAlerts. Esto reduce el número de branch points en el método principal y mejora la legibilidad. El nuevo método processAlerts se encarga de toda la lógica relacionada con el procesamiento de alertas, dejando el método principal más limpio y enfocado solo en la transacción de la base de datos. Este sería un bad smell de tipo Write simple units of code.

Pablo Jurado

4.1 Código Inicial

```
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, eragiketa:"BookDeny", price);
                db.getTransaction().begin();
            }
            booking.setStatus(status:"Rejected");
            db.merge(booking);
        }
        ride.setActive(active:false);
        db.merge(ride);

        db.getTransaction().commit();
    } catch (Exception e) {
        if (db.getTransaction().isActive()) {
            db.getTransaction().rollback();
        }
        e.printStackTrace();
    }
}
```

4.2 Código refactorizado

```java
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();
        handleBookingsCancellation(ride);
        deactivateRide(ride);
        db.getTransaction().commit();
    } catch (Exception e) {
        if (db.getTransaction().isActive()) {
            db.getTransaction().rollback();
        }
        e.printStackTrace();
    }
}

private void handleBookingsCancellation(Ride ride) throws Exception {
    for (Booking booking : ride.getBookings()) {
        if (isCancelable(booking)) {
            processTravelerRefund(booking);
        }
        booking.setStatus(status:"Rejected");
        db.merge(booking);
    }
}

private boolean isCancelable(Booking booking) {
    return booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined");
}
```

```java
private void processTravelerRefund(Booking booking) throws Exception {    Define and throw a dedicated exception instead of using a gener
    double price = booking.prezioaKalkulatu();
    Traveler traveler = booking.getTraveler();
    updateTravelerFrozenMoney(traveler, price);
    updateTravelerBalance(traveler, price);
    db.merge(traveler);
    db.getTransaction().commit();
    addMovement(traveler, eragiketa:"BookDeny", price);
    db.getTransaction().begin();
}

private void updateTravelerFrozenMoney(Traveler traveler, double price) {
    double frozenMoney = traveler.getIzoztatutakoDirua();
    traveler.setIzoztatutakoDirua(frozenMoney - price);
}

private void updateTravelerBalance(Traveler traveler, double price) {
    double money = traveler.getMoney();
    traveler.setMoney(money + price);
}

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

4.3 Descripción

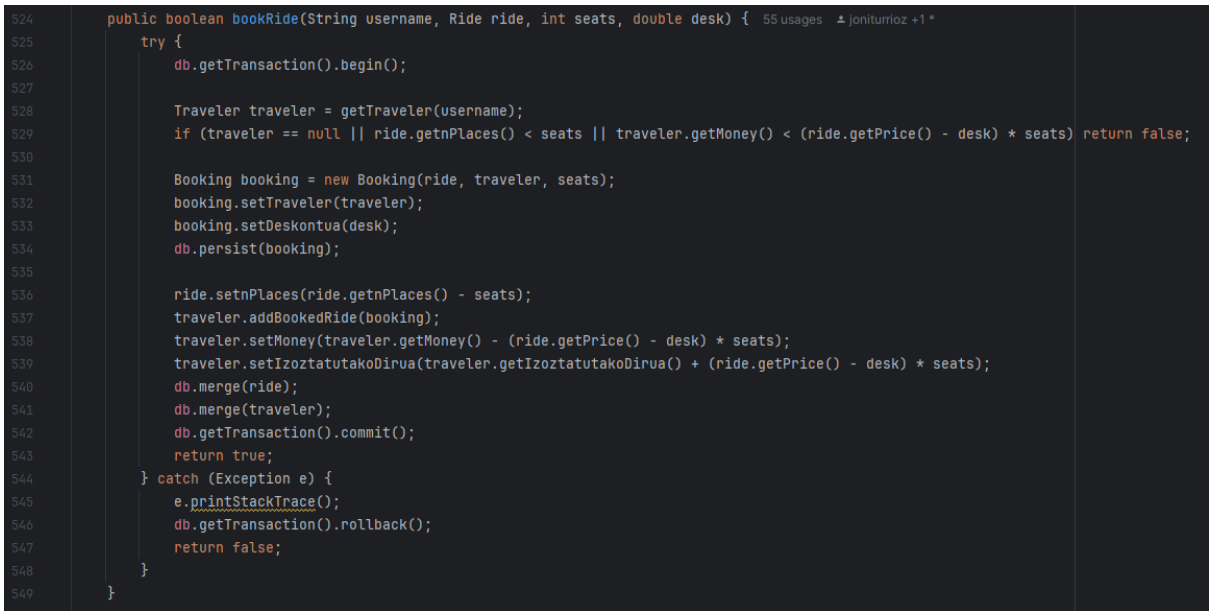
Este método se encargaba de varias tareas: realizar la cancelación, gestionar el estado de las reservas, manipular los fondos congelados, actualizar el estado de las reservas, entre otras. El método tenía más de 15 líneas, lo que hace que sea más difícil de entender y mantener.

Pablo Jurado

5.1 Código inicial

```
530     public boolean bookRide(String username, Ride ride, int seats, double desk) {  55 usages  joniturrioz
531         try {
532             db.getTransaction().begin();
533
534             Traveler traveler = getTraveler(username);
535             if (traveler == null) {
536                 return false;
537             }
538
539             if (ride.getnPlaces() < seats) {
540                 return false;
541             }
542
543             double ridePriceDesk = (ride.getPrice() - desk) * seats;
544             double availableBalance = traveler.getMoney();
545             if (availableBalance < ridePriceDesk) {
546                 return false;
547             }
548
549             Booking booking = new Booking(ride, traveler, seats);
550             booking.setTraveler(traveler);
551             booking.setDeskontua(desk);
552             db.persist(booking);
553
554             ride.setnPlaces(ride.getnPlaces() - seats);
555             traveler.addBookedRide(booking);
556             traveler.setMoney(availableBalance - ridePriceDesk);
557             traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
558             db.merge(ride);
559             db.merge(traveler);
560             db.getTransaction().commit();
561             return true;
562         } catch (Exception e) {
563             e.printStackTrace();
564             db.getTransaction().rollback();
565             return false;
566         }
567     }
```

5.2 Código refactorizado



```
524     public boolean bookRide(String username, Ride ride, int seats, double desk) {  55 usages  joniturrioz +1 *
525         try {
526             db.getTransaction().begin();
527
528             Traveler traveler = getTraveler(username);
529             if (traveler == null || ride.getnPlaces() < seats || traveler.getMoney() < (ride.getPrice() - desk) * seats) return false;
530
531             Booking booking = new Booking(ride, traveler, seats);
532             booking.setTraveler(traveler);
533             booking.setDeskontua(desk);
534             db.persist(booking);
535
536             ride.setnPlaces(ride.getnPlaces() - seats);
537             traveler.addBookedRide(booking);
538             traveler.setMoney(traveler.getMoney() - (ride.getPrice() - desk) * seats);
539             traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + (ride.getPrice() - desk) * seats);
540             db.merge(ride);
541             db.merge(traveler);
542             db.getTransaction().commit();
543             return true;
544         } catch (Exception e) {
545             e.printStackTrace();
546             db.getTransaction().rollback();
547             return false;
548         }
549     }
```

5.3 Descripción

El código inicial tiene un total de 4 branch points. Aunque es verdad que no sobrepasa el límite de 4 branch points que se recomienda, he considerado que sería útil hacer la refactorización ya que además de reducir el número de branch points a 2, el código se reduce bastante y se evita repetir “return false” varias veces en pocas líneas. La refactorización que he hecho ha sido mover las condiciones de varios if a uno solo.

AUTOR: Iñigo Alzugaray

6.1 Código inicial

```
524     public boolean bookRide(String username, Ride ride, int seats, double desk) {  55 usages  joniturrioz +1 *
525         try {
526             db.getTransaction().begin();
527
528             Traveler traveler = getTraveler(username);
529             if (traveler == null || ride.getnPlaces() < seats || traveler.getMoney() < (ride.getPrice() - desk) * seats) return false;
530
531             Booking booking = new Booking(ride, traveler, seats);
532             booking.setTraveler(traveler);
533             booking.setDeskontua(desk);
534             db.persist(booking);
535
536             ride.setnPlaces(ride.getnPlaces() - seats);
537             traveler.addBookedRide(booking);
538             traveler.setMoney(traveler.getMoney() - (ride.getPrice() - desk) * seats);
539             traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + (ride.getPrice() - desk) * seats);
540             db.merge(ride);
541             db.merge(traveler);
542             db.getTransaction().commit();
543             return true;
544         } catch (Exception e) {
545             e.printStackTrace();
546             db.getTransaction().rollback();
547             return false;
548         }
549     }
```

6.2 Código final

```
524     public boolean bookRide(String username, Ride ride, int seats, double desk) {  55 usages  iaz07 +1 *
525         try {
526             db.getTransaction().begin();
527             Traveler traveler = getTraveler(username);
528             if (traveler == null || ride.getnPlaces() < seats || traveler.getMoney() < (ride.getPrice() - desk) * seats) return false;
529             saveBooking(ride, seats, desk, traveler);
530             return true;
531         } catch (Exception e) {
532             e.printStackTrace();
533             db.getTransaction().rollback();
534             return false;
535         }
536     }
537
538     private void saveBooking(Ride ride, int seats, double desk, Traveler traveler)  1 usage  joniturrioz +1
539     {
540         Booking booking = new Booking(ride, traveler, seats);
541         booking.setTraveler(traveler);
542         booking.setDeskontua(desk);
543         db.persist(booking);
544
545         ride.setnPlaces(ride.getnPlaces() - seats);
546         traveler.addBookedRide(booking);
547         traveler.setMoney(traveler.getMoney() - (ride.getPrice() - desk) * seats);
548         traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + (ride.getPrice() - desk) * seats);
549         db.merge(ride);
550         db.merge(traveler);
551         db.getTransaction().commit();
552     }
```

6.3 Descripción

He hecho otra refactorización en el método bookRide. En este caso he reducido el número de líneas de 25 a 12 extrayendo el código encargado de guardar la reserva en la base de datos a un método nuevo. Tras esta refactorización y la anterior, los test de bookRide siguen ejecutándose con los mismos resultados, por lo que las refactorizaciones no han provocado ningún problema aparente.

AUTOR: Iñigo Alzugaray

7.1 Código inicial

```java
public boolean isRegistered(String erab, String passwd) {  1 usage  joniturrioz
    TypedQuery<Long> travelerQuery = db.createQuery(
            s: "SELECT COUNT(t) FROM Traveler t WHERE t.username = :username AND t.passwd = :passwd", Long.class);
    travelerQuery.setParameter( s: "username", erab);
    travelerQuery.setParameter( s: "passwd", passwd);
    Long travelerCount = travelerQuery.getSingleResult();

    TypedQuery<Long> driverQuery = db.createQuery(
            s: "SELECT COUNT(d) FROM Driver d WHERE d.username = :username AND d.passwd = :passwd", Long.class);
    driverQuery.setParameter( s: "username", erab);
    driverQuery.setParameter( s: "passwd", passwd);
    Long driverCount = driverQuery.getSingleResult();

    /*TypedQuery<Long> adminQuery = db.createQuery(
            "SELECT COUNT(a) FROM Admin a WHERE a.username = :username AND a.passwd = :passwd", Long.class);
    adminQuery.setParameter("username", erab);
    adminQuery.setParameter("passwd", passwd);
    Long adminCount = adminQuery.getSingleResult();*/

    boolean isAdmin=((erab.compareTo("admin")==0) && (passwd.compareTo(adminPass)==0));
    return travelerCount > 0 || driverCount > 0 || isAdmin;
}
```

7.2 Código final

```java
private Long isRegisteredAs(String username, String password, String type)  2 usages  iaz07
{
    TypedQuery<Long> travelerQuery = db.createQuery( s: "SELECT COUNT(u) FROM " + type + " u WHERE u.username = :username AND u.passwd = :passwd", Long.class);
    travelerQuery.setParameter( s: "username", username);
    travelerQuery.setParameter( s: "passwd", password);
    return travelerQuery.getSingleResult();
}

public boolean isRegistered(String erab, String passwd) {  1 usage  joniturrioz +1
    Long travelerCount = isRegisteredAs(erab, passwd,  type: "Traveler");
    Long driverCount = isRegisteredAs(erab, passwd,  type: "Driver");

    /*TypedQuery<Long> adminQuery = db.createQuery(
            "SELECT COUNT(a) FROM Admin a WHERE a.username = :username AND a.passwd = :passwd", Long.class);
    adminQuery.setParameter("username", erab);
    adminQuery.setParameter("passwd", passwd);
    Long adminCount = adminQuery.getSingleResult();*/

    boolean isAdmin=((erab.compareTo("admin")==0) && (passwd.compareTo(adminPass)==0));
    return travelerCount > 0 || driverCount > 0 || isAdmin;
}
```

7.3 Descripción

En el código inicial se hace una consulta por cada tipo de usuario en el que podría estar registrado el usuario. La mayoría del código de estas consultas se repite, y además, como puede verse en el comentario que hay en el código, es probable que en un futuro se quieran añadir nuevos tipos de usuario y esto supondría repetir más código. Por tanto, para evitar que el código se repita, he movido la consulta a un método nuevo.

AUTOR: Iñigo Alzugaray

8.1 Código inicial

```
234     public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)  31 usages  joniturrioz +2
235             throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
236         logger.info(
237                 msg: ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
238         if (driverName==null) return null;
239         try {
240             if (new Date().compareTo(date) > 0) {
241                 logger.info( msg: "ppppp");
242                 throw new RideMustBeLaterThanTodayException(
243                         ResourceBundle.getBundle( baseName: "Etiquetas").getString( key: "CreateRideGUI.ErrorRideMustBeLaterThanToday"));
244             }
245
246             db.getTransaction().begin();
247             Driver driver = db.find(Driver.class, driverName);
248             if (driver.doesRideExists(from, to, date)) {
249                 db.getTransaction().commit();
250                 throw new RideAlreadyExistException(
251                         ResourceBundle.getBundle( baseName: "Etiquetas").getString( key: "DataAccess.RideAlreadyExist"));
252             }
253             Ride ride = driver.addRide(from, to, date, nPlaces, price);
254             // next instruction can be obviated
255             db.persist(driver);
256             db.getTransaction().commit();
257
258             return ride;
259         } catch (NullPointerException e) {
260             // TODO Auto-generated catch block
261             return null;
262         }
```

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

8.2 Código refactorizado

```
229 @     public Ride createRide(RideData rideData)  31 usages  joniturrioz +2
230               throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
231           logger.info(
232                   msg: ">> DataAccess: createRide=> from= " + rideData.getFrom() + " to= " + rideData.getTo() + " driver=" + rideData.getDriverName() + " date " + rideData.getDate());
233           if (rideData.getDriverName() ==null) return null;
234           try {
235               if (new Date().compareTo(rideData.getDate()) > 0) {
236                   logger.info( msg: "ppppp");
237                   throw new RideMustBeLaterThanTodayException(
238                           ResourceBundle.getBundle( baseName: "Etiquetas").getString( key: "CreateRideGUI.ErrorRideMustBeLaterThanToday"));
239               }
240
241               db.getTransaction().begin();
242               Driver driver = db.find(Driver.class, rideData.getDriverName());
243               if (driver.doesRideExists(rideData.getFrom(), rideData.getTo(), rideData.getDate())) {
244                   db.getTransaction().commit();
245                   throw new RideAlreadyExistException(
246                           ResourceBundle.getBundle( baseName: "Etiquetas").getString( key: "DataAccess.RideAlreadyExist"));
247               }
248               Ride ride = driver.addRide(rideData.getFrom(), rideData.getTo(), rideData.getDate(), rideData.getnPlaces(), rideData.getPrice());
249               // next instruction can be obviated
250               db.persist(driver);
251               db.getTransaction().commit();
252
253               return ride;
254           } catch (NullPointerException e) {
255               // TODO Auto-generated catch block
256               return null;
257           }
```

```
public class RideData {  37 usages   iaz07
    private final String from;  2 usages
    private final String to;  2 usages
    private final Date date;  2 usages
    private final int nPlaces;  2 usages
    private final float price;  2 usages
    private final String driverName;  2 usages
```

```
Ride ride = dbManager.createRide(new RideData(from, to, date, nPlaces, price, driverName));
```

8.3 Descripción

El método createRide tiene 6 parámetros, que supera el límite recomendado de 4. Para solucionarlo, he extraído los parámetros a un objeto de tipo RideData.

AUTOR: Iñigo Alzugaray

9.1 Código inicial

```
public boolean addTraveler(String username, String password) {
    try {
        db.getTransaction().begin();

        Driver existingDriver = getDriver(username);
        Traveler existingTraveler = getTraveler(username);
        if (existingDriver != null || existingTraveler != null) {
            return false;
        }

        Traveler traveler = new Traveler(username, password);
        db.persist(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

9.2 Código refactorizado

```
public boolean addTraveler(String username, String password) {
    try {
        db.getTransaction().begin();
        Driver existingDriver = getDriver(username);
        Traveler existingTraveler = getTraveler(username);
        if (existingDriver != null || existingTraveler != null) return false;
        Traveler traveler = new Traveler(username, password);
        db.persist(traveler);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

9.3 Descripción

En este caso se ha refactorizado el método addTraveler. Se ha reducido el número de líneas, ha pasado de 19 a 15, modificando el código original, para que quede todo de forma más compacta (por ejemplo, devolviendo el posible resultado del if en la misma línea).

AUTOR: Aitor Gil

10.1 Código inicial

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

10.2 Código final

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

private double calculateAmount(double currentMoney, double amount,boolean deposit) {
    if (deposit) {
        return currentMoney + amount;
    } else {
        if ((currentMoney - amount) < 0)
            return 0;
        else
            return currentMoney - amount;
    }
}
```

10.3 Descripción

En este caso se ha refactorizado el método gauzatuEragiketa. El código inicial tiene un total de 4 branch points. Aunque es verdad que no sobrepasa el límite de 4 branch points que se recomienda, se ha considerado que sería útil hacer la refactorización ya que se puede reducir el número de branch points a 2. La refactorización se ha realizado creando el método clalculateAmunt, el cual cumple la función de obtener la cantidad de dinero que se modificara (lo cual se realizaba antes directamente en el código). Se han aprovechado los tests de gauzatuEragiketa para comprobar que, a pesar de la modificación, el método sigue funcionando perfectamente, y así es.

AUTOR: Aitor Gil

11.1 Código inicial

```
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;
    }
}
```

11.2 Código final

```
public boolean erreklamazioaBidali(ErreklamazioData erreklamazioData) {
    try {
        db.getTransaction().begin();

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

```java
package dataAccess;

import java.util.Date;

public class ErreklamazioData {
    public String nor;
    public String nori;
    public Date gaur;
    public Booking booking;
    public String textua;
    public boolean aurk;

    public ErreklamazioData(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;
    }
}
```

11.3 Descripción

El método erreklamazioaBidali tiene 6 parámetros, lo cual supera el límite recomendado de 4. Para solucionarlo, se han extraído los parámetros a un objeto de tipo ErreklamazioData (como se ve en la segunda imagen del código final) para reducir el número de parámetros del método.

AUTOR: Aitor Gil

12.1 Código inicial

```java
                    } catch (Exception e1) {

                        e1.printStackTrace();
                    }
                    tableRides.getColumnModel().getColumn(0).setPreferredWidth(50);
                    tableRides.getColumnModel().getColumn(1).setPreferredWidth(50);
                    tableRides.getColumnModel().getColumn(2).setPreferredWidth(100);
                    tableRides.getColumnModel().getColumn(3).setPreferredWidth(50);
                    tableRides.getColumnModel().removeColumn(tableRides.getColumnModel().getColumn(4));

                    tableRides.getTableHeader().setReorderingAllowed(false);
                    tableRides.setColumnSelectionAllowed(false);
                    tableRides.setRowSelectionAllowed(true);
                    tableRides.setDefaultEditor(Object.class, null);
                }
            }

        });
```

```
        this.getContentPane().add(jCalendar1, null);

        scrollPaneEvents.setBounds(new Rectangle(172, 257, 346, 150));

        scrollPaneEvents.setViewportView(tableRides);
        tableModelRides = new DefaultTableModel(null, columnNamesRides);

        tableRides.setModel(tableModelRides);

        tableModelRides.setDataVector(null, columnNamesRides);
        tableModelRides.setColumnCount(5); // another column added to allocate ride objects

        tableRides.getColumnModel().getColumn(0).setPreferredWidth(50);
        tableRides.getColumnModel().getColumn(1).setPreferredWidth(50);
        tableRides.getColumnModel().getColumn(2).setPreferredWidth(100);
        tableRides.getColumnModel().getColumn(3).setPreferredWidth(50);

        tableRides.getTableHeader().setReorderingAllowed(false);
        tableRides.setColumnSelectionAllowed(false);
        tableRides.setRowSelectionAllowed(true);
        tableRides.setDefaultEditor(Object.class, null);
```

12.2 Código refactorizado

```
private void setUpColumns(){
    tableRides.getColumnModel().getColumn(0).setPreferredWidth(50);
    tableRides.getColumnModel().getColumn(1).setPreferredWidth(50);
    tableRides.getColumnModel().getColumn(2).setPreferredWidth(100);
    tableRides.getColumnModel().getColumn(3).setPreferredWidth(50);

    tableRides.getColumnModel().removeColumn(tableRides.getColumnModel().getColumn(4));

    tableRides.getTableHeader().setReorderingAllowed(false);
    tableRides.setColumnSelectionAllowed(false);
    tableRides.setRowSelectionAllowed(true);
    tableRides.setDefaultEditor(Object.class, null);
}
```

```
                for (domain.Ride ride : rides) {
                    Vector<Object> row = new Vector<Object>();
                    row.add(ride.getDriver().getUsername());
                    row.add(ride.getnPlaces());
                    row.add(ride.getPrice());
                    row.add(decimalFormat
                            .format(ride.getDriver().getBalorazioa() / ride.getDriver().getBalkop()));
                    row.add(ride); // ev object added in order to obtain it with
                                   // tableModelEvents.getValueAt(i,3)
                    tableModelRides.addRow(row);
                }
                datesWithRidesCurrentMonth = facade.getThisMonthDatesWithRides(
                        (String) jComboBoxOrigin.getSelectedItem(),
                        (String) jComboBoxDestination.getSelectedItem(), jCalendar1.getDate());
                paintDaysWithEvents(jCalendar1, datesWithRidesCurrentMonth, Color.CYAN);

            } catch (Exception e1) {

                e1.printStackTrace();
            }
            setUpColumns();
        }
    }

});

  this.getContentPane().add(jCalendar1, null);

  scrollPaneEvents.setBounds(new Rectangle(172, 257, 346, 150));

  scrollPaneEvents.setViewportView(tableRides);
  tableModelRides = new DefaultTableModel(null, columnNamesRides);

  tableRides.setModel(tableModelRides);

  tableModelRides.setDataVector(null, columnNamesRides);
  tableModelRides.setColumnCount(5); // another column added to allocate ride objects

  setUpColumns();

  this.getContentPane().add(scrollPaneEvents, null);
  datesWithRidesCurrentMonth = facade.getThisMonthDatesWithRides((String) jComboBoxOrigin.getSelectedItem(),
          (String) jComboBoxDestination.getSelectedItem(), jCalendar1.getDate());
  paintDaysWithEvents(jCalendar1, datesWithRidesCurrentMonth, Color.CYAN);
```

12.3 Descripción

Se ha refactorizado parte del código de la clase de FindRdiesGUI, ya que en el mismo se repetían algunas líneas de código relacionadas con tableRides, para evitar esto se ha creado un método llamado setUpColumns, el cual hace la función de estas líneas de código, para reducir el código principal.

AUTOR: Aitor Gil