IS-2 / LABORATORIO

REFACTORIZACIÓN

PABLO ALCOLEA, IKER GARCÍA, UNAI LEÓN

REFACTORIZACIÓN 1.

- **BAD SMELL**: “Write short units of code”
- **AUTOR:** Unai

El método cancelRide es demasiado largo, tiene 23 líneas. Esto se debe a que antes de cancelar el viaje, borra todas las reservas realizadas para ese viaje en un bucle bastante largo. La solución es extraer dicho bucle a un nuevo método que solo elimine las reservas para un viaje concreto (utilizando la función Extract Method de Eclipse), y llamarlo desde el método para cancelar el viaje. Obtendremos dos métodos más cortos.

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, "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();
    }
}
```

Código refactorizado

```
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();

        rejectBookings(ride);
        ride.setActive(false);
        db.merge(ride);

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

public void rejectBookings(Ride ride) {
    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);
    }
}
```

REFACTORIZACIÓN 2.

- **BAD SMELL**: “Write simple units of code”
- **AUTOR:** Unai

El método updateAlertaAurkituak tiene una complejidad ciclomática de 6, ya que para cada alerta busca entre todos los viajes uno que coincida con la alerta. En vez de realizar todas las operaciones en un mismo método, extraeremos un método que busque si existe algún viaje para una alerta (utilizando Extract Method de Eclipse) y lo utilizaremos desde el método principal. Así conseguiremos dos métodos de menor complejidad cada uno. Además, también simplificaremos ciertas condiciones redundantes que pueden expresarse en una sola línea y sin necesidad del bloque condicional, reduciendo más la complejidad del segundo método.

Código inicial

```
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;
    }
}
```

Código refactorizado

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

            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();

            boolean alertFound = findAlerts(alerts, rides);

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

    public boolean findAlerts(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(true);
                    found = true;
                    alertFound = alert.isActive();
                    break;
                }
            }
            alert.setFound(found);
            db.merge(alert);
        }
        return alertFound;
    }
```

REFACTORIZACIÓN 3.

- **BAD SMELL**: “Duplicate code”
- **AUTOR:** Unai

En la constructora de la clase AlertaAurkituakGUI (en el paquete gui), se utiliza un String literal “Etiquetas” que se repite cada vez que se quiere obtener las etiquetas para la ventana que se construye. En vez de copiar el mismo String tantas veces, lo pondremos en una variable aparte. De esta forma, podrá cambiarse de forma mucho más sencilla en un único lugar. Utilizaremos el método Extract Local Variable de Eclipse.

Código inicial

```
public AlertaAurkituakGUI(String username) {

    setBussinessLogic(TravelerGUI.getBusinessLogic());
    Duplication
    this.setTitle(ResourceBundle.getBundle( 1 "Etiquetas").getString("AlertGUI.Alert"));
    setSize(new Dimension(600, 400));
    setResizable(false);
    getContentPane().setLayout(new BorderLayout());

    List<Alert> alertList = appFacadeInterface.getAlertsByUsername(username);
    DefaultTableModel model = new DefaultTableModel(
            Duplication
            new Object[] { ResourceBundle.getBundle( 2 "Etiquetas").getString("AlertGUI.Zenbakia"),
                    Duplication
                    ResourceBundle.getBundle( 3 "Etiquetas").getString("CreateRideGUI.LeavingFrom"),
                    Duplication
                    ResourceBundle.getBundle( 4 "Etiquetas").getString("CreateRideGUI.GoingTo"),
                    Duplication
                    ResourceBundle.getBundle( 5 "Etiquetas").getString("CreateRideGUI.RideDate"),
                    Duplication
                    ResourceBundle.getBundle( 6 "Etiquetas").getString("AlertGUI.Aurkitua"),
                    Duplication
                    ResourceBundle.getBundle( 7 "Etiquetas").getString("AlertGUI.Aktibo") },
            0);
    table = new JTable(model);
    JScrollPane scrollPane = new JScrollPane(table);
    getContentPane().add(scrollPane, BorderLayout.CENTER);

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

    SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
```

Código refactorizado

```
public AlertaAurkituakGUI(String username) {

    setBussinessLogic(TravelerGUI.getBusinessLogic());
    String bundleEtiquetas = "Etiquetas";
    Duplication
    this.setTitle(ResourceBundle.getBundle(bundleEtiquetas).getString("AlertGUI.Alert"));
    setSize(new Dimension(600, 400));
    setResizable(false);
    getContentPane().setLayout(new BorderLayout());

    List<Alert> alertList = appFacadeInterface.getAlertsByUsername(username);
    DefaultTableModel model = new DefaultTableModel(
            Duplication
            new Object[] { ResourceBundle.getBundle(bundleEtiquetas).getString("AlertGUI.Zenbakia"),
                    Duplication
                    ResourceBundle.getBundle(bundleEtiquetas).getString("CreateRideGUI.LeavingFrom"),
                    Duplication
                    ResourceBundle.getBundle(bundleEtiquetas).getString("CreateRideGUI.GoingTo"),
                    Duplication
                    ResourceBundle.getBundle(bundleEtiquetas).getString("CreateRideGUI.RideDate"),
                    Duplication
                    ResourceBundle.getBundle(bundleEtiquetas).getString("AlertGUI.Aurkitua"),
                    Duplication
                    ResourceBundle.getBundle(bundleEtiquetas).getString("AlertGUI.Aktibo") },
            0);
    table = new JTable(model);
    JScrollPane scrollPane = new JScrollPane(table);
    getContentPane().add(scrollPane, BorderLayout.CENTER);

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

REFACTORIZACIÓN 4.

- **BAD SMELL**: “Keep unit interfaces small”
- **AUTOR:** Unai

El método ErreklamazioaBidali tiene 6 parámetros con todos los datos de la reclamación a enviar. En lugar de pasar los 6 parámetros a la función, es mucho más comprensible construir el objeto de reclamación antes y utilizar ese único objeto como parámetro de la función. Para cambiar la signatura del método, utilizaremos Change Method Signature de Eclipse. Una vez cambiada, vemos que solo se hace una llamada a este método desde otro método de la clase BLFacadeImplementation, por lo que lo corregimos a mano para que el sistema siga funcionando de la misma forma. Construimos la reclamación con todos los datos, y después pasamos dicho objeto al método. Podría mejorarse extendiendo la refactorización a la interfaz BLFacade y construyendo el objeto antes de las llamadas en los puntos de la aplicación en los que se envíe la reclamación.

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;
    }
}
```

Código refactorizado

```
public boolean erreklamazioaBidali(Complaint complaint) {
    try {
        db.getTransaction().begin();
        db.persist(complaint);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

REFACTORIZACIÓN 5.

- **BAD SMELL**: “Write short units of code”
- **AUTOR:** Pablo

El método getBookingFromDriver es demasiado extenso, tiene 21 líneas. El método tiene un for que tiene como objetivo para cada “ride” del driver si es un ride activo, añadir todas las reservas de ese “ride” al array “bookings”. Como solución, extraer el for en un método llamado “bookingRides” simplificando el código.

Código inicial

```
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;
    }
}
```

Código refactorizado

```
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);
        List<Booking> bookings = bookingRides(query);
        db.getTransaction().commit();
        return bookings;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return null;
    }
}

private List<Booking> bookingRides(TypedQuery<Driver> query) {
    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());
        }
    }
    return bookings;
}
```

REFACTORIZACIÓN 6.

- **BAD SMELL**: “Write simple units of code”
- **AUTOR:** Pablo

El método createRide es un método con una complejidad diplomática de 5. Se ha extraido el método if de comprobar la fecha a un método externo al propio, y de igual manera a la hora de comprobar si existe un driver con un ride determinado. Su complejidad diplómatica ahora es de 3.

Código inicial

```
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;
    }

}
```


Código refactorizado

```
private Driver existeDriver(String from, String to, Date date, String driverName) throws RideAlreadyExistException {
    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"));
    }
    return driver;
}

private void comprobarDate(Date date) throws RideMustBeLaterThanTodayException {
    if (new Date().compareTo(date) > 0) {
        System.out.println("ppppp");
        throw new RideMustBeLaterThanTodayException(
                ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
    }
}
```

```
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 {
        comprobarDate(date);
        db.getTransaction().begin();
        Driver driver = existeDriver(from, to, date, driverName);
        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;
    }

}
```

REFACTORIZACIÓN 7.

- **BAD SMELL**: “Duplicate code”
- **AUTOR:** Pablo

La constructora BusinessLogicServer duplica código en la instancia “CancelButton”.Se ha refactorizado el string “Cancel” a una variable llamada “Cancelar”. Utilizando “extract local variable”.

Código inicial

```
public BusinessLogicServer() {
    addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosed(WindowEvent arg0) {
            System.exit(1);
        }
    });
    setTitle("BusinessLogicServer: running the business logic");
    setBounds(100, 100, 486, 209);
    getContentPane().setLayout(new BorderLayout());
    contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
    getContentPane().add(contentPanel, BorderLayout.CENTER);
    contentPanel.setLayout(new BorderLayout(0, 0));
    {
        textArea = new JTextArea();
        contentPanel.add(textArea);
    }
    {
        JPanel buttonPane = new JPanel();
        buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
        getContentPane().add(buttonPane, BorderLayout.SOUTH);
        {
            JButton okButton = new JButton("OK");
            okButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent e) {
                    textArea.append("\n\n\nClosing the server... ");

                    // server.close();

                    System.exit(1);
                }
            });
            okButton.setActionCommand("OK");
            buttonPane.add(okButton);
            getRootPane().setDefaultButton(okButton);
        }
        {
            JButton cancelButton = new JButton("Cancel");
            cancelButton.setActionCommand("Cancel");
            buttonPane.add(cancelButton);
        }
```

Código refactorizado

```
            okButton.setActionCommand("OK");
            buttonPane.add(okButton);
            getRootPane().setDefaultButton(okButton);
        }
        {
            String Cancelar = "Cancel";
            JButton cancelButton = new JButton(Cancelar);
            cancelButton.setActionCommand(Cancelar);
            buttonPane.add(cancelButton);
        }
    }
```

REFACTORIZACIÓN 8.

- **BAD SMELL**: “Keep unit interfaces small”
- **AUTOR:** Pablo

El método addRide tiene como entrada 5 parámetros. Esto hace que la compresión del código sea más compleja. Como solución, cambiar la signatura del método y que entre por parámetro un objeto de tipo Ride.

Código inicial

```
public Ride addRide(String from, String to, Date date, int nPlaces, float price) {
    Ride ride = new Ride(from, to, date, nPlaces, price, this);
    createdRides.add(ride);
    return ride;
}
```

Código Refactorizado

```
public Ride addRide(Ride r) {

    createdRides.add(r);
    return r;
}
```

REFACTORIZACIÓN 9.

- **BAD SMELL**: “Write short units of code”
- **AUTOR:** Iker

El método gauzatuEragiketa contiene 21 líneas de código, es demasiado extenso. Para simplificarlo, tras comprobar que el usuario no tenga valor nulo, vamos a extraer el if que actualiza el dinero que dispone dicho usuario.

Código inicial

```
507    public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
508        try {
509            db.getTransaction().begin();
510            User user = getUser(username);
511            if (user != null) {
512                double currentMoney = user.getMoney();
513                if (deposit) {
514                    user.setMoney(currentMoney + amount);
515                } else {
516                    if ((currentMoney - amount) < 0)
517                        user.setMoney(0);
518                    else
519                        user.setMoney(currentMoney - amount);
520                }
521                db.merge(user);
522                db.getTransaction().commit();
523                return true;
524            }
525            db.getTransaction().commit();
526            return false;
527        } catch (Exception e) {
528            e.printStackTrace();
529            db.getTransaction().rollback();
530            return false;
531        }
532    }
```

Código refactorizado

```
507    public boolean gauzatuEragiketa(String username, double amount, boolean deposit) {
508        try {
509            db.getTransaction().begin();
510            User user = getUser(username);
511            if (user != null) {
512                double currentMoney = user.getMoney();
513                actualizarDinero(amount, deposit, user, currentMoney);
514                db.merge(user);
515                db.getTransaction().commit();
516                return true;
517            }
518            db.getTransaction().commit();
519            return false;
520        } catch (Exception e) {
521            e.printStackTrace();
522            db.getTransaction().rollback();
523            return false;
524        }
525    }
526

527    private void actualizarDinero(double amount, boolean deposit, User user, double currentMoney) {
528        if (deposit) {
529            user.setMoney(currentMoney + amount);
530        } else {
531            if ((currentMoney - amount) < 0)
532                user.setMoney(0);
533            else
534                user.setMoney(currentMoney - amount);
535        }
536    }
```

REFACTORIZACIÓN 10.

- **BAD SMELL**: “Write simple units of code”
- **AUTOR:** Iker

El método bookRide tiene una complejidad ciclomática de 4 (al no haber más métodos con comp. cicl. > 4, refactorizaremos ese). Por ende, extraeremos los dos primeros if’s del

método, con la intención de filtrar de forma más rápida si el viajero tiene valor nulo y si el viaje elegido tiene asientos suficientes.

Código inicial

```
551    public boolean bookRide(String username, Ride ride, int seats, double desk) {
552        try {
553            db.getTransaction().begin();
554
555            Traveler traveler = getTraveler(username);
556            if (traveler == null) {
557                return false;
558            }
559
560            if (ride.getnPlaces() < seats) {
561                return false;
562            }
563
564            double ridePriceDesk = (ride.getPrice() - desk) * seats;
565            double availableBalance = traveler.getMoney();
566            if (availableBalance < ridePriceDesk) {
567                return false;
568            }
569
570            Booking booking = new Booking(ride, traveler, seats);
571            booking.setTraveler(traveler);
572            booking.setDeskontua(desk);
573            db.persist(booking);
574
575            ride.setnPlaces(ride.getnPlaces() - seats);
576            traveler.addBookedRide(booking);
577            traveler.setMoney(availableBalance - ridePriceDesk);
578            traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
579            db.merge(ride);
580            db.merge(traveler);
581            db.getTransaction().commit();
582            return true;
583        } catch (Exception e) {
584            e.printStackTrace();
585            db.getTransaction().rollback();
586            return false;
587        }
588    }
```

Código refactorizado

```
551    public boolean bookRide(String username, Ride ride, int seats, double desk) {
552        try {
553            db.getTransaction().begin();
554
555            Traveler traveler = getTraveler(username);
556
557            if (!existeTravelerYAsientosSuficientes(ride, seats, traveler)) return false;
558
559            double ridePriceDesk = (ride.getPrice() - desk) * seats;
560            double availableBalance = traveler.getMoney();
561            if (availableBalance < ridePriceDesk) {
562                return false;
563            }
564
565            Booking booking = new Booking(ride, traveler, seats);
566            booking.setTraveler(traveler);
567            booking.setDeskontua(desk);
568            db.persist(booking);
569
570            ride.setnPlaces(ride.getnPlaces() - seats);
571            traveler.addBookedRide(booking);
572            traveler.setMoney(availableBalance - ridePriceDesk);
573            traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
574            db.merge(ride);
575            db.merge(traveler);
576            db.getTransaction().commit();
577            return true;
578        } catch (Exception e) {
579            e.printStackTrace();
580            db.getTransaction().rollback();
581            return false;
582        }
583    }
```

```
585    private boolean existeTravelerYAsientosSuficientes(Ride ride, int seats, Traveler traveler) {
586        if (traveler == null || ride.getnPlaces() < seats) {
587            return false;
588        }
589        return true;
590    }
```

REFACTORIZACIÓN 11.

- **BAD SMELL**: “Duplicate code”
- **AUTOR:** Iker

En el método initializeDB se repite el String “BookFreeze” 5 veces. Por tanto, se ha creado una variable String bookFreeze con el valor “BookFreeze” para evitar repeticiones innecesarias en el código.

Código inicial

```
131
132             db.persist(book1);
133             db.persist(book2);
134             db.persist(book3);
135             db.persist(book4);
136             db.persist(book5);
137
138             Movement m1 = new Movement(traveler1, "BookFreeze", 20);
139             Movement m2 = new Movement(traveler1, "BookFreeze", 40);
140             Movement m3 = new Movement(traveler1, "BookFreeze", 5);
141             Movement m4 = new Movement(traveler2, "BookFreeze", 4);
142             Movement m5 = new Movement(traveler1, "BookFreeze", 3);
143             Movement m6 = new Movement(driver1, "Deposit", 15);
144             Movement m7 = new Movement(traveler1, "Deposit", 168);
145
```

Código refactorizado

```
137
138             String bookFreeze = "BookFreeze";
139             Movement m1 = new Movement(traveler1, bookFreeze, 20);
140             Movement m2 = new Movement(traveler1, bookFreeze, 40);
141             Movement m3 = new Movement(traveler1, bookFreeze, 5);
142             Movement m4 = new Movement(traveler2, bookFreeze, 4);
143             Movement m5 = new Movement(traveler1, bookFreeze, 3);
144             Movement m6 = new Movement(driver1, "Deposit", 15);
145             Movement m7 = new Movement(traveler1, "Deposit", 168);
146
```

REFACTORIZACIÓN 12.

- **BAD SMELL**: “Keep unit interfaces small”
- **AUTOR:** Iker

El método createRide tiene 6 parámetros en su cabecera. Vamos a reducirlo a uno para evitar usar parámetros de más.

Código inicial

```
228    public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
229            throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
230        System.out.println(
231                ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
232        if (driverName == null)
233            return null;
234        try {
235            comprobarDate(date);
236            db.getTransaction().begin();
237            Driver driver = existeDriver(from, to, date, driverName);
238            Ride ride = driver.addRide(from, to, date, nPlaces, price);
239            db.persist(driver);
240            db.getTransaction().commit();
241            return ride;
242        } catch (NullPointerException e) {
243            // TODO Auto-generated catch block
244            return null;
245        }
246
247    }
```

Código refactorizado

```
228    public Ride createRide(Ride r) throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
229        System.out.println(">> DataAccess: createRide=> from= " + r.getFrom() + " to= " + r.getTo() + " driver="
230                + r.getDriver().getUsername() + " date " + r.getDate());
231        if (r.getDriver().getUsername() == null)
232            return null;
233        try {
234            comprobarDate(r.getDate());
235            db.getTransaction().begin();
236            Driver driver = existeDriver(r.getFrom(), r.getTo(), r.getDate(), r.getDriver().getUsername());
237            Ride ride = driver.addRide(r.getFrom(), r.getTo(), r.getDate(), r.getnPlaces(), (float) r.getPrice());
238            db.persist(driver);
239            db.getTransaction().commit();
240            return ride;
241        } catch (NullPointerException e) {
242            // TODO Auto-generated catch block
243            return null;
244        }
245
246    }
```