Refactorización

Ingeniería del Software II

13/10/2024

Autores:
HaiYong Zhan
Roman Mardakhaev

Índice

HaiYong Zhan

Métodos a refactorizar (DataAccess.java)

public void cancelRide(Ride ride)

1. Longitud de línea a 15 líneas de código
2. Limita el número de puntos de ramificación por unidad a 4.

public void deleteUser(User us)

1. Longitud de línea a 15 líneas de código
2. Limita el número de puntos de ramificación por unidad a 4.
3. Código duplicado

Método cancelRide(Ride ride)

El método cancelRide tiene más de 15 líneas, lo que dificulta su comprensión y mantenibilidad. No cumple con la guía de "Escribir unidades simples de código" del capítulo 3, que sugiere limitar el número de puntos de ramificación a 4. Observamos que alcanzó complejidad ciclomática 5, esto se debe a las condiciones de estados(try-catch, for, if).

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();
            bookingCancellation(ride);

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


    private void bookingCancellation(Ride ride) {
        for (Booking booking : ride.getBookings()) {
            if (booking.getStatus().equals("Accepted") ||
            booking.getStatus().equals("NotDefined")) {
                double price = booking.prezioaKalkulatu();
                Traveler traveler = booking.getTraveler();
                traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() - price);
                traveler.setMoney(traveler.getMoney() + price);
                db.merge(traveler);
                addMovement(traveler, "BookDeny", price);
            }
            booking.setStatus("Rejected");
            db.merge(booking);
        }
    }
```

Es recomendable mantener una única transacción para todo el bloque de operaciones, en lugar de hacer múltiples commits (db.getTransaction().commit() y db.getTransaction().begin()) Todo el proceso está envuelto en una única transacción que comienza al inicio de cancelRide y se confirma al final, lo que garantiza que todas las operaciones se completen o se reviertan juntas.

Responsabilidad más clara, bookingCancellation se encarga solo de cancelar reservas, mientras que cancelRide gestiona la transacción.
Al separar la lógica de cancelación, reducen la complejidad ciclomática del método.

Método deleteUser(User us)

El método no cumple con la guía de "Escribir unidades cortas de código" del capítulo 2, que recomienda limitar la longitud de las unidades de código a 15 líneas. A su vez, vemos que tiene varios estados que superan el límite de ramificación 4 con considerable cantidad de condicional if y bucle for.

Código inicial:

```
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();
            }
}
```

Código refactorizado:

Para comenzar, observamos que la llamada "us.getUsername()" se realiza cuatro veces dividida en distintas responsabilidades. Entonces para evitar repetición, vamos a introducir un variable local para realizar la llamada solo una vez.

Además, dividiremos la responsabilidad del método deleteUser en varios métodos más pequeños. Los métodos deleteDriver y deleteTraveler verifican el tipo de User que reciben como parámetro antes de realizar las llamadas a los métodos correspondientes: deleteAllAlertsForUser, rejectAllBookingsForUser, deleteDriver, deleteAllCarsForDriverl.

```
    public void deleteUser(User us) {
        try {
            String username = us.getUsername();
            deleteDriver(us, username);
            deleteTraveler(us, username);
            db.getTransaction().begin();
            us = db.merge(us);
            db.remove(us);
            db.getTransaction().commit();
        } catch (Exception e) {
```

```
                e.printStackTrace();
        }
}

private void deleteDriver(User us, String username) {
        if (us.getMota().equals("Driver")) {
                cancelAllRidesForDriver(username);
                deleteAllCarsForDriver(username);
        }
}

private void cancelAllRidesForDriver(String username) {
        List<Ride> rl = getRidesByDriver(username);
        if (rl != null) {
                for (Ride ri : rl) {
                        cancelRide(ri);
                }
        }
}

private void deleteAllCarsForDriver(String username) {
        Driver d = getDriver(username);
        List<Car> cl = d.getCars();
        if (cl != null) {
                for (int i = cl.size() - 1; i >= 0; i--) {
                        Car ci = cl.get(i);
                        deleteCar(ci);
                }
        }
}

private void deleteTraveler(User us, String username) {
        if (us.getMota().equals("Traveler")) {
                rejectAllBookingsForUser(username);
                deleteAllAlertsForUser(username);
        }
}

private void rejectAllBookingsForUser(String username) {
        List<Booking> lb = getBookedRides(username);
        if (lb != null) {
                for (Booking li : lb) {
```

```
                    li.setStatus("Rejected");
                    li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
                }
            }
        }

        private void deleteAllAlertsForUser(String username) {
            List<Alert> la = getAlertsByUsername(username);
            if (la != null) {
                for (Alert lx : la) {
                    deleteAlert(lx.getAlertNumber());
                }
            }
        }
```

Ahora, los métodos son más concisos y cumplen con la recomendación de mantener cada unidad de código por debajo de 15 líneas. Al dividir la lógica en métodos separados, se reducen los puntos de ramificación en cada uno de ellos. Además, se han implementado otras mejoras, como la eliminación de la duplicidad de código, el uso de parámetros claros y una clara asignación de responsabilidades.

Roman Mardakhaev

Métodos a refactorizar (BezeroGUI.java, DataAccess.java)

1. publicBezeroGUI(String Username) - **Duplicación del Código**
2. public boolean isRegistered(String erab, String passwd) - **Escribir las funciones más cortas (15 líneas)**
3. public boolean bookRide(String username, Ride ride, int seats, double desk) - **Limite el número de puntos de ramificación a 4 o menos (complejidad ciclomatica)**
4. public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua) - **Mantener las interfaces de parámetros más simples (menos que 4)**

Método public BezeroGUI(String Username) BezeroGUI.java

En este método nos encontramos ante un problema descrito en el capítulo 4 “Duplicate Code” que nos indica utilizar variable local en debe de repetir el mismo código. Por lo tanto, utilizando ya

definido quinto método de refactorización (: Creación de variable local por repetición de código).definiremos una variable local llamada compl de tipo String que tendrá el mismo valor y la vamos a reutilizar en nuestra refactorización del código.

Código inicial:

```
public BezeroGUI(String username) {
		setBussinessLogic(DriverGUI.getBusinessLogic());
		this.setSize(new Dimension(600, 537));
		this.setResizable(false);
		getContentPane().setLayout(new BorderLayout(0, 0));
		// Lista
		taula = new JTable();
		List<Booking> TravelsList = appFacadeInterface.getBookingFromDriver(username);
		List<Booking> BezeroLista = new ArrayList<>();
		scrollPane = new JScrollPane(taula);
		getContentPane().add(scrollPane, BorderLayout.NORTH);
		// Etiketak
		String[] columnNames = {
ResourceBundle.getBundle("Etiquetas").getString("EgoeraGUI.BookingNumber"),
					ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.RideDate"),
					ResourceBundle.getBundle("Etiquetas").getString("BezeroGUI.Bezeroa"),
					ResourceBundle.getBundle("Etiquetas").getString("EgoeraGUI.Egoera"),
					ResourceBundle.getBundle("Etiquetas").getString("BezeroGUI.Zergatia") };
		DefaultTableModel model = new DefaultTableModel(columnNames, 0);
		SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy/MM/dd");
		if (TravelsList != null) {
			for (Booking bo : TravelsList) {

				String status;
				switch (bo.getStatus()) {
				case "Completed":
					status = ResourceBundle.getBundle("Etiquetas").getString("Completed");
					break;
				case "Accepted":
					status = ResourceBundle.getBundle("Etiquetas").getString("Accepted");
					break;
				case "Rejected":
					status = ResourceBundle.getBundle("Etiquetas").getString("Rejected");
					break;
				case "NotCompleted":
					status = ResourceBundle.getBundle("Etiquetas").getString("NotCompleted");
					break;
				case "Complained":
					status = ResourceBundle.getBundle("Etiquetas").getString("Complained");
					break;

				case "Valued":

					status = ResourceBundle.getBundle("Etiquetas").getString("Valued");
```

```
                                        break;
                                default:
                                        status = ResourceBundle.getBundle("Etiquetas").getString("NotDefined");
                                        break;
                                }
```

(linea 114)

```
else if (bo.getStatus().equals("Completed") || bo.getStatus().equals("Valued")

                                                || bo.getStatus().equals("Complained"))
```

(linea 182)

```
booking.setStatus("Complained");
                                                        appFacadeInterface.updateBooking(booking);
```

(linea 198)

```
model.setValueAt(ResourceBundle.getBundle("Etiquetas").getString("Complained"), pos, 3);
                                                        model.setValueAt("", pos, 4);
```

(linea 204)

```
booking.setStatus("Complained");
                                                        appFacadeInterface.updateBooking(booking);
```

(linea 218)

```
model.setValueAt(ResourceBundle.getBundle("Etiquetas").getString("Complained"), pos, 3);
                                                        model.setValueAt("", pos, 4);
```

(linea 226)

```
.equals(ResourceBundle.getBundle("Etiquetas").getString("Complained"))) {
                                                lblErrorea.setForeground(Color.RED);
```

Código refactorizado:

```
public class BezeroGUI extends JFrame {
        private static final long serialVersionUID = 1L;
        private static BLFacade appFacadeInterface;
        private JTable taula;
        private JButton jButtonBaloratu;
        private JButton jButtonErreklamatu;
        private JButton jButtonClose;
        private JScrollPane scrollPane;
        private String compl="Complained";
```

```
case "Complained":

                                        status = ResourceBundle.getBundle("Etiquetas").getString(this.compl);
                                        break;
```

(linea 114)

```
                                } else if (bo.getStatus().equals("Completed") || bo.getStatus().equals("Valued")

                                        || bo.getStatus().equals(this.compl)) {
```

(linea 182)

```
                                                booking.setStatus(compl);
                                                appFacadeInterface.updateBooking(booking);
```

(linea 198)

```
model.setValueAt(ResourceBundle.getBundle("Etiquetas").getString(compl), pos, 3);
                                                model.setValueAt("", pos, 4);
```

(linea 204)

```
                                                booking.setStatus(compl);
                                                appFacadeInterface.updateBooking(booking);
```

(linea 218)

```
model.setValueAt(ResourceBundle.getBundle("Etiquetas").getString(compl), pos, 3);
                                                model.setValueAt("", pos, 4);
```

(linea 226)

```
.equals(ResourceBundle.getBundle("Etiquetas").getString(compl))) {
                                                lblErrorea.setForeground(Color.RED);
```

Método public boolean isRegistered(String erab, String passwd) DataAccess.java

Código inicial:

En este caso, nos encontramos ante el problema del capítulo 2, es decir, de “Write short units of code”. En nuestro caso el código original del método ocupa 22 líneas de código, lo cual vamos a reducir hasta que sea menos de 15 líneas. Para ello, voy a utilizar la cuarta herramienta de refactorización (Extracción de un método) y voy a crear dos métodos de exterior que nos devolverán con una llamada ya los resultados de TypedQuery de Traveler y de Driver.

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

```
                driverQuery.setParameter("username", erab);
                driverQuery.setParameter("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;
        }
```

Código refactorizado:

```
public Long getTravelerCount(String username, String passwd){
                TypedQuery<Long> travelerQuery = db.createQuery(
                                "SELECT COUNT(t) FROM Traveler t WHERE t.username = :username AND
t.passwd = :passwd", Long.class);
                travelerQuery.setParameter("username", username);
                travelerQuery.setParameter("passwd", passwd);
                return travelerQuery.getSingleResult();
        }

        public Long getDriverCount(String username, String passwd){
                TypedQuery<Long> driverQuery = db.createQuery(
                                "SELECT COUNT(d) FROM Driver d WHERE d.username = :username AND
d.passwd = :passwd", Long.class);
                driverQuery.setParameter("username", username);
                driverQuery.setParameter("passwd", passwd);
                return driverQuery.getSingleResult();

        }


        public boolean isRegistered(String erab, String passwd) {
                Long travelerCount = getTravelerCount(erab,passwd);
                Long driverCount = getDriverCount(erab,passwd);
                /*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;
        }
```

Por lo tanto, he podido reducir el código original de 22 líneas a 10 haciendo la extracción. Los métodos nuevos tampoco ocupan más de 15 líneas.

Método boolean bookRide(String username, Ride ride, int seats, double desk)

El método bookRide contiene muchos condicionales de tipo **if()** por lo que su complejidad ciclomática se aumentó hasta 5. Voy a intentar reducir la complejidad hasta que sea 4, o, incluso menos, si es posible, quitando los if's y dejando solo aquellas condiciones que sean necesarias.

Código inicial:

```
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;
		}
	}
```

Código refactorizado:

```
public boolean bookRide(String username, Ride ride, int seats, double desk) {
		try {
			db.getTransaction().begin();
			Traveler traveler = getTraveler(username);
			double ridePriceDesk = (ride.getPrice() - desk) * seats;
			double availableBalance = traveler.getMoney();
			if ( ride.getnPlaces() < seats || traveler.getMoney() < 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;
		}
	}
```

Para nuestro caso, es suficiente que juntemos todas las condiciones dentro de solo un if(). Podemos quitar la condición de traveler==null dado que todo el bloque de condición está dentro de try{}, por lo que si habrá algún error surgido por parte de transacción de db de búsqueda de traveler, el método se podrá procesar dicha excepción lanzada y devolver false. Así, se puede simplificar la complejidad ciclomática de 5 a solo 3. (try + if + 1)

Método public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua)

Dicho método viola la regla del capítulo 5 "Keep unit interfaces small", por lo que, vamos a utilizar la tercera herramienta de refactorización (Extracción de una interface). Dado que ya disponemos de la clase Complain creada y definida dentro del proyecto la vamos a utilizar como parámetro directamente en el método en debe de pasarle muchos parámetros y pedir que se crea. En las líneas de código donde se invoca dicho parámetro he añadido la inicialización y definición de un objeto “complaint” de tipo Complaint para pasarlo al método.

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;
                }
        }
```

Las lineas que lo utilizan:

```
if (!textua.isEmpty()) {
                                                appFacadeInterface.erreklamazioaBidali(nork, nori, gaur, booking, textua,
true);
```

(ErreklamazioakGUI, linea 101)

```
                        public void actionPerformed(java.awt.event.ActionEvent e) {
                                appFacadeInterface.erreklamazioaBidali(nori, nork, gaur, booking, "Ez da
agertu",false);
                                jButtonClose_actionPerformed(e);
                        }
```

(ArrazoaGUI, linea 54)

Código refactorizado:

```
public boolean erreklamazioaBidali(Complaint erreklamazioa) {
                try {
                        db.getTransaction().begin();
                        db.persist(erreklamazioa);
                        db.getTransaction().commit();
```

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

(ErreklamazioakGUI, linea 101)

```
                        if (!textua.isEmpty()) {
                            Complaint complaint = new Complaint(nork, nori, gaur, booking, textua, true);
                            appFacadeInterface.erreklamazioaBidali(complaint);
                            jButtonClose_actionPerformed(e);
```

(ArrazoaGUI, linea 54)

```
            jButtonJun.addActionListener(new java.awt.event.ActionListener() {
                public void actionPerformed(java.awt.event.ActionEvent e) {
                    Complaint complaint = new Complaint(nori, nork, gaur, booking, "Ez da agertu",false);
                    appFacadeInterface.erreklamazioaBidali(complaint);
                    jButtonClose_actionPerformed(e);
```