ERREFAKTORIZAZIOA (RIDES24)

Egileak: **Ander Mena, Ekhi Ugarte, Haritz Eizagirre**
Data: **2025/10/12**
Irakaslea: Jon Iturrioz
Irakasgaia: Software Ingeniaritza II

AURKIBIDEA

1. TALDEKIDEA: Haritz Eizaguirre

1. Aldagaien izendatze pobrea

- Hasierako kodea:

```
ConfigXML c =ConfigXML.getInstance();
MainGUI a=new MainGUI();
DataAccess da= new DataAccess();
```

- Errefaktorizatutako kodea:

```
ConfigXML config =ConfigXML.getInstance();
MainGUI mainWindow =new MainGUI();
DataAccess dataAccess = new DataAccess();
```

“c”, “a” eta “da” aldagaiek ez dute argi adierazten zer motako aldagaiak diren eta zertarako erabiltzen diren. Hori dela eta, hobe da izen esanguratsuak jartzea kodea irakurtzen duen pertsonak (kodearen egilea edo lehen aldiz irakurtzen duena) jakin dezan zer den aldagai hori.

2. getDriverByEmail, getTravelerByEmail eta getAdminByEmail metodoak ia berdinak

- Hasierako kodea:

```
        public Driver getDriverByEmail(String email, String password) throws
UserDoesNotExistException, PasswordDoesNotMatchException{
                db.getTransaction().begin();
                Driver d = db.find(Driver.class, email);
                db.getTransaction().commit();
                if(d==null) {
                        this.close();
                        throw new
UserDoesNotExistException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.UserD
oesNotExist"));
                }
                if(!d.getPassword().equals(password)) {
                        this.close();
                        throw new
PasswordDoesNotMatchException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.
PasswordDoesNotMatch"));
                }
                return d;
        }
```

```
	public Traveler getTravelerByEmail(String email, String password) throws
UserDoesNotExistException, PasswordDoesNotMatchException{
		db.getTransaction().begin();
		Traveler t = db.find(Traveler.class, email);
		db.getTransaction().commit();
		if(t==null) {
			this.close();
			throw new
UserDoesNotExistException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.UserD
oesNotExist"));
		}
		if(!t.getPassword().equals(password)) {
			this.close();
			throw new
PasswordDoesNotMatchException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.
PasswordDoesNotMatch"));
		}
		return t;
	}


	public Admin getAdminByEmail(String email, String password) throws
UserDoesNotExistException, PasswordDoesNotMatchException{
		db.getTransaction().begin();
		Admin a = db.find(Admin.class, email);
		db.getTransaction().commit();
		if(a==null) {
			this.close();
			throw new
UserDoesNotExistException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.UserD
oesNotExist"));
		}
		if(!a.getPassword().equals(password)) {
			this.close();
			throw new
PasswordDoesNotMatchException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.
PasswordDoesNotMatch"));
		}
		return a;
	}
```

- Errefaktorizatutako kodea:

```
package domain;


public interface User {
   String getEmail();
   String getPassword();
   String getName();
}
```

```
  private <T extends User> T getUserByEmail(String email, String password, Class<T>
userClass)
        throws UserDoesNotExistException, PasswordDoesNotMatchException {

    db.getTransaction().begin();
    T user = db.find(userClass, email);
    db.getTransaction().commit();

    if (user == null) {
       this.close();
       throw new UserDoesNotExistException(
          ResourceBundle.getBundle("Etiquetas").getString("DataAccess.UserDoesNotExist"));
    }

    if (!user.getPassword().equals(password)) {
       this.close();
       throw new PasswordDoesNotMatchException(

ResourceBundle.getBundle("Etiquetas").getString("DataAccess.PasswordDoesNotMatch"));
    }

    return user;
  }


  public Driver getDriverByEmail(String email, String password)
        throws UserDoesNotExistException, PasswordDoesNotMatchException {
    return getUserByEmail(email, password, Driver.class);
  }


  public Traveler getTravelerByEmail(String email, String password)
        throws UserDoesNotExistException, PasswordDoesNotMatchException {
    return getUserByEmail(email, password, Traveler.class);
  }


  public Admin getAdminByEmail(String email, String password)
        throws UserDoesNotExistException, PasswordDoesNotMatchException {
    return getUserByEmail(email, password, Admin.class);
  }
```

getDriverByEmail, getTravelerByEmail eta getAdminByEmail metodoak ia berdinak dira eta kode duplikatua ekiditeko metodo guztiak bateratzeko getUserByEmail metodoa sortu dugu, non deitzen den Driver, Traveler edo Adminen metodo guztietan metodo bera exekutatzen da eta horrela hiru metodoen logika bateratzen da.

Gainera hori egin ahal izateko User interfaze bat sortu dugu eta Driver, Traveler eta Admin klaseek interfaze horretatik hartzen dute getEmail, getPassword eta getName.

Errefaktorizazio hau egitean, aurretik esan bezala, oso antzekoak diren metodo batzuen logika bateratzen dugu metodo bakar batean. Horrela, metodo honen logika aldatu beharra dagoenean metodo bakarra aldatu behar da eta ez 3 metodo hasieran zegoenean bezala.

3. Hirien izenak errepikatuta

- Hasierako kodea:

```
//Create rides
driver1.addRide("Donostia", "Bilbo", UtilDate.newDate(year,month,15), 7, car1);
driver1.addRide("Donostia", "Gazteiz", UtilDate.newDate(year,month,6), 8, car1);
driver1.addRide("Bilbo", "Donostia", UtilDate.newDate(year,month,25), 4, car5);
driver1.addRide("Donostia", "Iruña", UtilDate.newDate(year,month,7), 8, car5);

driver2.addRide("Donostia", "Bilbo", UtilDate.newDate(year,month,15), 3, car2);
driver2.addRide("Bilbo", "Donostia", UtilDate.newDate(year,month,25), 5, car4);
driver2.addRide("Eibar", "Gasteiz", UtilDate.newDate(year,month,6), 5, car2);
driver3.addRide("Bilbo", "Donostia", UtilDate.newDate(year,month,14), 3, car3);
```

- Errefaktorizatutako kodea:

```
private static final String BILBO = "Bilbo";
private static final String DONOSTIA = "Donostia";
private static final String GASTEIZ = "Gasteiz";
private static final String IRUÑA = "Iruña";
private static final String EIBAR = "Eibar";

//Create rides
driver1.addRide(DONOSTIA, BILBO, UtilDate.newDate(year,month,15), 7, car1);
driver1.addRide(DONOSTIA, GASTEIZ, UtilDate.newDate(year,month,6), 8, car1);
driver1.addRide(BILBO, DONOSTIA, UtilDate.newDate(year,month,25), 4, car5);
driver1.addRide(DONOSTIA, IRUÑA, UtilDate.newDate(year,month,7), 8, car5);

driver2.addRide(DONOSTIA, BILBO, UtilDate.newDate(year,month,15), 3, car2);
driver2.addRide(BILBO, DONOSTIA, UtilDate.newDate(year,month,25), 5, car4);
driver2.addRide(EIBAR, GASTEIZ, UtilDate.newDate(year,month,6), 5, car2);
driver3.addRide(BILBO, DONOSTIA, UtilDate.newDate(year,month,14), 3, car3);
```

Lehenengo kodean hirien izenak string literalen bidez jartzen dira eta gainera horietako batzuk askotan errepikatzen dira, horretarako konstanteak sortu ditugu string literalen ordez jartzeko.
Konstanteak erabiltzeak (private static final), strings literalen ordez, akats tipografikoak eta balioen bikoizketa saihesten ditu. Gainera mantentzea errazten du, izen bat aldatzen bada, leku batean bakarrik aldatzen baita.

4. addRatingToDriver eta addRatingToTraveler ia berdinak

- Hasierako kodea:

```
	public void addRatingToTraveler(String e, int z, Integer resCode) {
		db.getTransaction().begin();
		Reservation r = db.find(Reservation.class, resCode);
		Traveler t = db.find(Traveler.class, e);
		r.setRatedD(true);
		t.addRating(z);
		db.persist(t);
		db.persist(r);
		db.getTransaction().commit();
	}

	public void addRatingToDriver(String email, int z, Integer resCode){
		db.getTransaction().begin();
		Driver d = db.find(Driver.class, email);
		Reservation r = db.find(Reservation.class, resCode);
		d.addRating(z);
		r.setRatedT(true);
		db.persist(d);
		db.persist(r);
		db.getTransaction().commit();
	}
```

- Errefaktorizatutako kodea:

```
	private void addRating(String email, int rating, Integer reservationCode, Class<?>
userClass, boolean isDriver) {
		db.getTransaction().begin();
		Reservation reservation = db.find(Reservation.class, reservationCode);
		Object user = db.find(userClass, email);
		if (isDriver) {
			((Driver) user).addRating(rating);
			reservation.setRatedT(true);
		} else {
			((Traveler) user).addRating(rating);
			reservation.setRatedD(true);
		}
		db.persist(user);
		db.persist(reservation);
		db.getTransaction().commit();
	}
	public void addRatingToDriver(String email, int rating, Integer reservationCode) {
		addRating(email, rating, reservationCode, Driver.class, true);
	}
	public void addRatingToTraveler(String email, int rating, Integer reservationCode) {
		addRating(email, rating, reservationCode, Traveler.class, false);
	}
```

Hasierako kodean ikusi daiteke bi metodoak ia identikoak direla eta beraz metodoen logika metodo batean bateratu daitekela. Horretarako sortu dugu *addRating* metodo berria beste bi metodoek deituko dutena eta horrela kode bikoiztua ekidituko da. Gainera bertsio berrian mantentzea errazten du, aldaketaren bat egin behar denean metodo bakar batean aldatuta nahikoa izango delako eta horretaz aparte,kodearen portaeraren sendotasuna hobetzen dugu.

2. TALDEKIDEA: Ander Mena

1. Aldagaia berrizendatu

- Hasierako kodea:

```
private String complaint;
```

- Errefaktorizatutako kodea:

```
private String description;
```

Complaint klaseko “complain” izendun String motako propietatea “description”-era aldatu da, izan ere, ez da praktika ona klase batek bere barnean aldagai bat klasearen izen berarekin izatea.

2. Metodoa sinplifikatu

- Hasierako kodea:

```
	public void pay(Reservation res) throws NotEnoughMoneyException{
		db.getTransaction().begin();
		try {
			Traveler t = db.find(Traveler.class, res.getTraveler().getEmail());
			Driver d = db.find(Driver.class, res.getDriver().getEmail());
			float price = res.getHmTravelers()*res.getRide().getPrice();
			if(t.getMoney()-price<0){
				db.getTransaction().commit();
				throw new NotEnoughMoneyException();
			}
			Reservation r = db.find(Reservation.class,
res.getReservationCode());
			r.setPayed(true);
			t.setMoney(t.getMoney()-price);
			d.setMoney(d.getMoney()+price);
```

```
                        Transaction tr = new Transaction(price, d, t);
                        d.addTransaction(tr);
                        t.addTransaction(tr);
                        db.persist(r);
                        db.persist(tr);
                        db.persist(d);
                        db.persist(t);
                        db.getTransaction().commit();
                }catch(NullPointerException e) {
                        db.getTransaction().commit();
                }
        }
```

- Errefaktorizatutako kodea:

```
        public void pay(Reservation res) throws NotEnoughMoneyException{
                db.getTransaction().begin();
                try {
                        Traveler traveler = db.find(Traveler.class,
res.getTraveler().getEmail());
                        Driver driver = db.find(Driver.class, res.getDriver().getEmail());
                        float price = calculateReservationPrice(res);

                        validateTravelerHasSufficientFunds(traveler, price);

                        Reservation reservation = db.find(Reservation.class,
res.getReservationCode());
                        processPayment(reservation, traveler, driver, price);

                        db.getTransaction().commit();
                }catch(NullPointerException e) {
                        db.getTransaction().commit();
                }
        }

        private float calculateReservationPrice(Reservation res) {
                return res.getnTravelers() * res.getRide().getPrice();
        }

        private void validateTravelerHasSufficientFunds(Traveler traveler, float price)
throws NotEnoughMoneyException {
                if(traveler.getMoney() - price < 0) {
                        db.getTransaction().commit();
                        throw new NotEnoughMoneyException();
                }
        }

        private void processPayment(Reservation reservation, Traveler traveler, Driver
driver, float price) {
                reservation.setPayed(true);
                transferMoneyBetweenUsers(traveler, driver, price);
```

```
            Transaction transaction = new Transaction(price, driver, traveler);
            recordTransaction(reservation, transaction, traveler, driver);
        }
```

DataAccess klaseko “pay” metodoa laburtu da, bere barne logika kanpo-metodo laguntzaileetan sartuz. Metodo laguntzaile hauei izen esanguratsuak eman zaizkie egiten dutena ulerterraza izan dadin.

3. Erabili gabeko aldagaiak ezabatu

```
// Admin GUI klasean, rideId ezabatu, ez da erabiltzen
Integer rideId = res.getRide().getRideNumber();

//Application Launcher klasean, driver ezabatu, ez da erabiltzen
Driver driver=new Driver("driver3@gmail.com","Test Driver", "123");
```

4. Metodoa sinplifikatu

- Hasierako kodea:

```
        public void acceptComplaint(Complaint co) {
            db.getTransaction().begin();
            Complaint complaint = db.find(Complaint.class, co.getId());
            Driver d = db.find(Driver.class, complaint.getDriverEmail());
            Traveler t = db.find(Traveler.class, complaint.getTravelerEmail());
            Reservation res = db.find(Reservation.class,
co.getRes().getReservationCode());
            float price = res.getHmTravelers()*res.getRide().getPrice();
            d.setMoney(d.getMoney()-price);
            t.setMoney(t.getMoney()+price);
            d.removeComplaint(complaint);
            t.removeComplaint(complaint);
            Transaction tra = new Transaction(price, t, d);
            d.addTransaction(tra);
            t.addTransaction(tra);
            db.persist(tra);
            db.persist(t);
            db.persist(d);
            db.remove(complaint);
            db.getTransaction().commit();
        }
```

- Errefaktorizatutako kodea:

```
	public void acceptComplaint(Complaint co) {
		db.getTransaction().begin();
		Complaint complaint = db.find(Complaint.class, co.getId());
		Driver driver = db.find(Driver.class, complaint.getDriverEmail());
		Traveler traveler = db.find(Traveler.class, complaint.getTravelerEmail());
		Reservation reservation = db.find(Reservation.class,
co.getRes().getReservationCode());

		float refundAmount = calculateReservationPrice(reservation);
		processComplaintRefund(driver, traveler, complaint, refundAmount);

		db.getTransaction().commit();
	}


	private void processComplaintRefund(Driver driver, Traveler traveler, Complaint
complaint, float refundAmount) {
		refundMoneyToTraveler(driver, traveler, refundAmount);
		removeComplaintFromUsers(driver, traveler, complaint);
		Transaction refundTransaction = new Transaction(refundAmount, traveler,
driver);
		recordComplaintRefund(refundTransaction, driver, traveler, complaint);
	}

	private void refundMoneyToTraveler(Driver driver, Traveler traveler, float amount) {
		driver.setMoney(driver.getMoney() - amount);
		traveler.setMoney(traveler.getMoney() + amount);
	}

	private void removeComplaintFromUsers(Driver driver, Traveler traveler, Complaint
complaint) {
		driver.removeComplaint(complaint);
		traveler.removeComplaint(complaint);
	}
```

DataAccess klaseko “acceptComplaint” metodoa errefaktorizatu da bere barne-logika metodo laguntzailetara ateraz. Lehenago sortu dugun calculateReservationPrice erabilita eta refundMoneyToTraveler zein removeComplaintFromUsers berrien bitartez, jatorrizko metodoa sinplifikatu da, berriro ere izen esanguratsuak ezarrita.

3. TALDEKIDEA: Ekhi Ugarte

1. Aldagaia berrizendatu

- Hasierako kodea:

```
private int hmTravelers;
```

- Errefaktorizatutako kodea:

```
private int nTravelers;
```

hmTravelers aldagaiak traveler kopurua erreferentziatzen du, notazioa ez da oso intuituboa eta nTravelers era aldatzean, n kopuruarekin lotzen dugulako, intuitiboagoa da.

2. Metodo bat ezabatu

Bi metodo daude Rides en berdina egiten dutena, setNPlaces eta setnPlaces

```
public void setNPlaces(int nPlaces) {
                this.nPlaces = nPlaces;
        }
.
.
.
public void setnPlaces(int nPlaces) {
                this.nPlaces = nPlaces;
        }
```

Hau ezabatuz konfusio posibleak desagertuko dira metodo hauekiko eta notazio ona duena mantenduko dugu.

3. Aldagai bat sortu textua duplikatu ordez

“Etiquetas” string a 8 aldiz erabiltzen da, kodea garbiagoa izan dadin, aldagai bat sortzea horretarako praktika ona da

```
private static final String ETIQUETAS = "Etiquetas";
jLabelSelectOption = new
JLabel(ResourceBundle.getBundle(ETIQUETAS).getString("DriverAndTravelerGUI.Welcome")+"
"+admin.getName());

jLabelSelectOption = new
JLabel(ResourceBundle.getBundle("Etiquetas").getString("DriverAndTravelerGUI.Welcome")+"
"+admin.getName());
```

4. Kode errepikakorra saiesteko metodo bat

```
private ActionListener createLocaleActionListener(String localeCode) {
		return new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				Locale.setDefault(new Locale(localeCode));
				System.out.println("Locale: " + Locale.getDefault());
				paintAgain();
			}
		};
	}


	rdbtnNewRadioButton.addActionListener(new ActionListener() {
			public void actionPerformed(ActionEvent e) {
				Locale.setDefault(new Locale("en"));
				System.out.println("Locale: "+Locale.getDefault());
				paintAgain();	}
		});

rdbtnNewRadioButton.addActionListener(createLocaleActionListener("en"));
```

Kodea asko errepikatzen zenez, metodo laguntzaile bat sortu dut errepikapen hauek saiesteko.