SI2

TALDEKO LAN PRAKTIKOA

Ivan Priala, Maialen Velazquez eta Maitane Rodriguez

Aurkibidea:

1. createBalorazio():
   1.1 Hasierako kodea
   1. 2 Write short units of code
   1. 3 Write simple units of code
   1. 4 Duplicate code
   1. 5 Keep unit interfaces small
   1. 6 Amaierako kodea
   1.7 Egilea

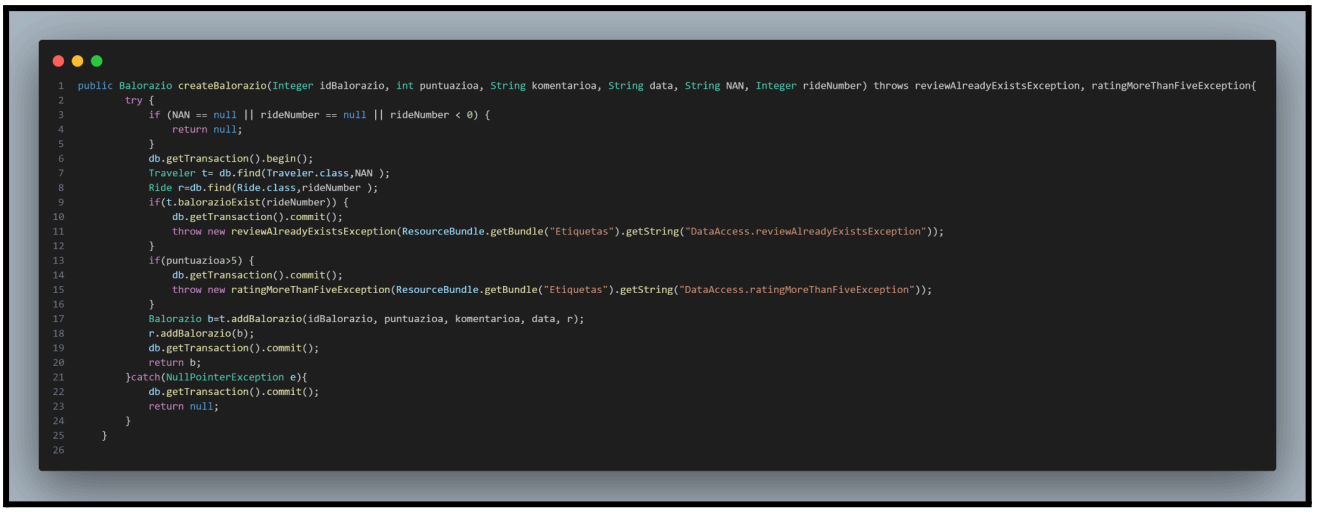
2. createAlert():
   2.1 Hasierako kodea
   2. 2 Write short units of code
   2. 3 Write simple units of code
   2. 4 Duplicate code
   2. 5 Keep unit interfaces small
   2.6 Amaierako kodea
   2.7 Egilea

3. createBook():
   3. 1 Hasierako kodea
   3. 2 Write short units of code
   3. 3 Write simple units of code
   3. 4 Duplicate code
   3. 5 Keep unit interfaces small
   3.6 Amaierako kodea
   3.7 Egilea

4. Github kodea

createBalorazio():

1.1 Hasierako kodea:



```
public Balorazio createBalorazio(Integer idBalorazio, int puntuazioa, String komentarioa, String data, String NAN, Integer rideNumber) throws reviewAlreadyExistsException, ratingMoreThanFiveException{
        try {
            if (NAN == null || rideNumber == null || rideNumber < 0) {
                return null;
            }
            db.getTransaction().begin();
            Traveler t= db.find(Traveler.class,NAN );
            Ride r=db.find(Ride.class,rideNumber );
            if(t.balorazioExist(rideNumber)) {
                db.getTransaction().commit();
                throw new reviewAlreadyExistsException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.reviewAlreadyExistsException"));
            }
            if(puntuazioa>5) {
                db.getTransaction().commit();
                throw new ratingMoreThanFiveException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.ratingMoreThanFiveException"));
            }
            Balorazio b=t.addBalorazio(idBalorazio, puntuazioa, komentarioa, data, r);
            r.addBalorazio(b);
            db.getTransaction().commit();
            return b;
        }catch(NullPointerException e){
            db.getTransaction().commit();
            return null;
        }
    }
```

1. 2 Write short units of code:

Metodoak 15 lerro baino gehiago ditu eta hori, “Bad Smell” da, izan ere, metodo bakar batean informazio eta funtzionalitate gehiegi pilatzeak kodea zailagoa egiten du ulertzeko, mantentzeko eta testeazteko.

Arazo hau errefaktorizazio bidez konpon daiteke: kodea hainbat azpimetododetan banatuko dugu, bakoitzak funtzio zehatz eta txiki bat betetzeko. Horrela, kodea ulergarriagoa bihurtuko da.

Egindako errefaktorizazioaren deskribapena Extract Method erabilita: Kodea azpimetodo txiki eta espezifikoetan banatu da, funtzionalitate bakoitzari zeregin bakarra emanez.

- **checkBalorazioDoesExist(Traveler t, Integer rideNumber):** Lehenengo if-ak, balorazioa dagoeneko existitzen den egiaztatzen du, hala bada, salbuespena jaurtitzen du. Ondoren, metodo nagusistik deituko dugu kodea ordezkatuz.
- **checkPuntuazioa(int puntuazioa):** Metodoa nagusiaren bigarren if-ak puntuazioa 5 baino handiagoa den egiaztatzen du. puntuazioa > 5 bada, salbuespena jaurtitzen du. Azpimetodo batean sortu dugu, ondoren metodo nagusitik deia egin dugu.
- **createAndLinkBalorazioa(Ride r, Traveler t, int puntuazioa, String komentarioa, String data):** Azpimetodo hau, balorazioa berri bat sortzeko erabiltzen da. Behin

balorazioa sortu dela, Traveler eta Ride objetuekin eralazionatzen du. Azpimetodoa behar bezala sortzean, ezinbestekoa da createBalorazio(...) metodo nagusitik deitzea.

Egindako errefaktorizazioaren deskribapena Extract Local Variable: Errepikatatutako kodea dago, horretarako, errepikapenak saihesteko, kodea Local Variable batean gordetzea pentsatu dugun arren, ezinezkoa da void bat delako:

- **var transakzio = db.getTransaction().commit();**

Eta horrez gain, **Extract Method errefaktorizazioak** aplikatu ondoren, kode errepikapena murriztu egin da. Ondorioz, errefaktorizazioa hauei esker instrukzioa askoz kontrolatuagoa da. Beraz, ez da beharrezkoa hasieran planteatutako Local Variable hau sortzea.

Errefaktorizatutako kodea:

```
public Balorazio createBalorazio(Integer idBalorazio, int puntuazioa, String komentarioa, String data, String NAN, Integer rideNumber) throws reviewAlreadyExistsException, ratingMoreThanFiveException{
        try {
            if (NAN == null || rideNumber == null || rideNumber < 0) {
                return null;
            }
            db.getTransaction().begin();
            Traveler t= db.find(Traveler.class,NAN );
            Ride r=db.find(Ride.class,rideNumber );
            checkBalorazioDoesExist(rideNumber, t);
            checkPuntuazioa(puntuazioa);
            return createAndLinkBalorazioa(idBalorazio, puntuazioa, komentarioa, data, t, r);
        }catch(NullPointerException e){
            db.getTransaction().commit();
            return null;
        }
    }

    private Balorazio createAndLinkBalorazioa(Integer idBalorazio, int puntuazioa, String komentarioa, String data, Traveler t,
            Ride r) {
        Balorazio b=t.addBalorazio(idBalorazio, puntuazioa, komentarioa, data, r);
        r.addBalorazio(b);
        db.getTransaction().commit();
        return b;
    }

    private void checkPuntuazioa(int puntuazioa) throws ratingMoreThanFiveException {
        if(puntuazioa>5) {
            db.getTransaction().commit();
            throw new ratingMoreThanFiveException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.ratingMoreThanFiveException"));
        }
    }

    private void checkBalorazioDoesExist(Integer rideNumber, Traveler t) throws reviewAlreadyExistsException {
        if(t.balorazioExist(rideNumber)) {
            db.getTransaction().commit();
            throw new reviewAlreadyExistsException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.reviewAlreadyExistsException"));
        }
    }
```

Errefaktorizazioa aplikatu ondoren, createBalorazio(...) metodoa laburtzea lortu dugu eta 15 lerrotik beherako kode bat geratu da. Orain, azpimetodo bakoitzak ardura bakarra da eta modu honetan, ulergarriagoa eta testagarriagoa da.

1. 3 Write simple units of code:

Jatorrizko createBalorazio(...) metodoak konplexutasun handiegia zuen, izan ere, hainbat if, try, catch eta return instrukzio zituen.

Horrek metodoaren konplexutasun ziklomatikoa handitzen zuen, eta ondorioz, kodea zailagoa egiten zuen ulertzeko, mantentzeko eta testak garatzeko.
Konplexutasun ziklomatikoa errefaktorizazioa aplikatu aurretik: V(G)=4.

Errefaktorizazioa aplikatu ondoren, metodo nagusiko adarkapen logikoak azpimetodo berezietan banatu dira, bakoitzan zeregin bakarra izan dezan.
Horrela, createBalorazio(...) metodoaren Konplexutasun ziklomatikoa V(G)=4-tik V(G)=2-ra jaistea lortu dugu (try/catch + return).
Konplexutasun ziklomatikoa errefaktorizazioa aplikatu ondoren: V(G)=2.
Ondorioz, metodoa irakurgarriagoa eta ulergarriagoa izatea lortu dugu. Gainera, balidazioa bakoitza independentea denez, test sinple eta zehatzagoak aplikatzeko aukera izango dugu.

1. 4 Duplicate code:

- **Hasierako arazoa:** Jatorrizko createBalorazio(...) metodoan, kode errepikatua zegoen. Horretarako Local Variable bat sortzea pentsatu genuen, **db.getTransaction().commit()** instrukzioarekin. Instrukzio hori lau aldiz agertzen zen: baldintzetan, salbuespenetan eta metodoaren amaieran. Horrek, kodearen mantengarritasuna zailtzen du, izan ere, kodea kopiatu egiten denean, akatsak hainbat lekutan zuzendu behar izaten dira edo balio bat eguneratu behar denean, eskuz eguneratu beharko da dena eta hori ez da eraginkorra.

- **Egindako errefaktorizazioa:** Esan bezala, errepikatutako kodea ezabatzeko, gomendagarria izango litzateke instrukzio hori Local Variable batean gordetzea, baina, kontuan hartuta void bat dela eta azpimetodoen errefaktorizazioa egitean **db.getTransaction().commit()** instrukzioaren errepikapena murriztu egin dela, honen kudeaketa askoz kontrolatuagoa da, eta ez da beharrezkoa Local Variable hau sortzea, kodea ez baita errepikatzen.

- **Ondorioa:** Aldaketa hauek aplikatuz, kodeak Duplicate code araua betetzea lortu dugu.

1. 5 Keep unit interfaces small:

- **Hasierako arazoa:** createBalorazio(...) metodoak sei parametro jasotzen ditu: **createBalorazio(Integer idBalorazio, int puntuazioa, String data, String NAN, Integer rideNumber).** Horrek, **Keep unit interfaces small** araua hausten du, izan ere, arauak esaten duenez, metodo batek gehienez lau parametro izan beharko lituzke.

  Parametro kopuru handiak, metodoaren erabilera eta ulermena zailtzen du, baita testak egitea ere, kodearen konplexutasuna handituz eta erabilgarritasuna murriztuz.

- **Egindako errefaktorizazioa:** Arazo hau konpontzeko, gomendagarria da parametro guztiak datu objektu batean kapsulatzea. Kasu honetan, sartzen ziren parametro guztiak, Balorazio bat osatzen zuten. Beraz, parametroak aldatu ditugu, Balorazioa sartuz. Ondoren, beharrezko datuak getters eta setters-ekin lortuko ditugu. Horrela sinplifikatu dugu metodoa:
  - **createBalorazioa(Balorazio balorazioa, String NAN, Integer rideNumber)**

  Modu honetan, metodoaren interfazea askoz txikiagoa eta argiagoa izatea lortu dugu, 6 parametroetatik 3-ra murriztu dugu. Horrez gain, datuen kudeaketa objektu bakar batean zentralizau dugu.

  Aldaketa hauek gehitzean, ezinbestekoa izan da hasierako errefaktorizazio batean aldaketak egitea:

  - **createAndLinkBalorazioa(Ride r, Traveler t, int puntuazioa, String komentarioa, String data)**
  - **createAndLinkBalorazioa(Balorazio balorazioa, Ride r, Traveler t)**

- **Ondorioa:** Aldaketa hauek aplikatuz, kodeak Keep unit interfaces small araua betetzea lortu dugu.

1. 6 Amaierako kodea:

Beharrezko errefaktorizazio guztiak aplikatu ondoren, hau da lortutako kodea:

```java
public Balorazio createBalorazio(Balorazio balorazioa, String NAN, Integer rideNumber) throws reviewAlreadyExistsException, ratingMoreThanFiveException{
    try {
        if (NAN == null || rideNumber == null || rideNumber < 0) {
            return null;
        }
        db.getTransaction().begin();
        Traveler t= db.find(Traveler.class,NAN );
        Ride r=db.find(Ride.class,rideNumber );
        checkBalorazioDoesExist(rideNumber, t);
        checkPuntuazioa(balorazioa.getPuntuazioa());
        return createAndLinkBalorazioa(balorazioa, t, r);
    }catch(NullPointerException e){
        db.getTransaction().commit();
        return null;
    }
}

private Balorazio createAndLinkBalorazioa(Balorazio balorazioa, Traveler t, Ride r) {
    Balorazio b=t.addBalorazio(balorazioa.getId(), balorazioa.getPuntuazioa(), balorazioa.getKomentarioa(), balorazioa.getData(), r);
    r.addBalorazio(b);
    db.getTransaction().commit();
    return b;
}

private void checkPuntuazioa(int puntuazioa) throws ratingMoreThanFiveException {
    if(puntuazioa>5) {
        db.getTransaction().commit();
        throw new ratingMoreThanFiveException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.ratingMoreThanFiveException"));
    }
}

private void checkBalorazioDoesExist(Integer rideNumber, Traveler t) throws reviewAlreadyExistsException {
    if(t.balorazioExist(rideNumber)) {
        db.getTransaction().commit();
        throw new reviewAlreadyExistsException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.reviewAlreadyExistsException"));
    }
}
```

1. 7 Egilea: Maitane Rodriguez

createAlert():

2.1 Hasierako kodea:

```
public Alerta createAlert(String nondik, String nora, Date data, String NAN) throws alertAlreadyExists, RideMustBeLaterThanTodayException{
        try {
            if(new Date().compareTo(data)>0) {
                throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
            }
            db.getTransaction().begin();
            Traveler t= db.find(Traveler.class,NAN );
            if (t.doesAlertExist(nondik,nora, data)) {
                db.getTransaction().commit();
                throw new alertAlreadyExists(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.alertAlreadyExists"));
            }
            Alerta alerta=t.addAlert(nondik, nora, data);
            //db.persist(t); -> ez da beharrezkoa, dagoeneko commit egiten delako
            db.getTransaction().commit();
            return alerta;
        } catch (NullPointerException e) {
            db.getTransaction().commit();
            return null;
        }
    }

```

2. 2 Write short units of code:

Metodo honek 15 lerro baino gehiago ditu. Horren ondorioz, metodo bakar batean logika gehiegi pilatzen da, eta ulertzea eta testeatzea zailtzen du. Hori konpontzeko, errefaktorizazioaren bidez, azpimetodoetan banatuko dugu gauza bakoitza metodo batean egin dadin.

Egindako errefaktorizazioaren deskribapena Extract Method erabilita:

Metodoak kode asko zuenez, 4 metodo txiki desberdinetan banatzea errazagoa da ulermenerako eta testeatzeko.

- **checkDateIsValid(Date data):** Lehenengo if-ak, data gaurkoa baina beranduagokoa den edo ez konprobatzen du, eta lehenagokoa bada, salbuespena jaurtitzen du. Horretarako, metodo hau sortu dugu eta bertan balioztatu dugu data, ondoren createAlert() metodotik honi deituz.

- **findTraveler(String NAN):** Bidaiari bat sortzen da pasatako NAN-arekin, baina horretarako, bidaiaria datu basean bilatu behar da. findTraveler() metodoa sortu dugu hori metodo honetatik kanpo kudeatzeko, ondoren deia eginez.

- **checkAlertDoesNotExist(String nondik, String nora, Date data, Traveler t):** Alerta bat sortu behar dugu, baina lehenengo, sortu dugun bidaiariarentzat alerta hori jada existitzen den ala ez egiaztatu behar dugu, sortuta badago ez

salbuespen bat jaurituz. checkAlertDoesNotExist() metodoa sortu dugu horretarako, createAlert() metodotik honi deituz.

- **saveAlert(String nondik, String nora, Date data, Traveler t):** Konprobazio guztiak egin ondoren, alerta hau gordeko diogu bidaiariari, horretarako sortu dugun saveAlert() metodoari deituz.

Egindako errefaktorizazioaren deskribapena Extract Local Variable: Errepikatatutako kodea dago, horretarako, errepikapenak saihesteko, kodea Local Variable batean gordetzea pentsatu dugun arren, ezinezkoa da void bat delako:

- **var transakzio = db.getTransaction().commit();**

Eta horrez gain, **Extract Method errefaktorizazioak** aplikatu ondoren, kode errepikapena murriztu egin da. Ondorioz, errefaktorizazioa hauei esker instrukzioa askoz kontrolatuagoa da. Beraz, ez da beharrezkoa hasieran planteatutako Local Variable hau sortzea.

Errefaktorizatutako kodea:

```
private Alerta saveAlert(String nondik, String nora, Date data, Traveler t) throws alertAlreadyExists {
    if (t.doesAlertExist(nondik,nora, data)) {
        db.getTransaction().commit();
        throw new alertAlreadyExists(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.alertAlreadyExists"));
    }
    Alerta alerta=t.addAlert(nondik, nora, data);
    //db.persist(t); -> ez da beharrezkoa, dagoeneko commit egiten delako
    db.getTransaction().commit();
    return alerta;
}
private void checkAlertDoesNotExist(String nondik, String nora, Date data, Traveler t) throws alertAlreadyExists {
    if (t.doesAlertExist(nondik,nora, data)) {
        db.getTransaction().commit();
        throw new alertAlreadyExists(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.alertAlreadyExists"));
    }
}
private Traveler findTraveler(String NAN) {
    Traveler t= db.find(Traveler.class,NAN );
    return t;
}
private void checkDateIsValid(Date data) throws RideMustBeLaterThanTodayException {
    if(new Date().compareTo(data)>0) {
        throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
    }
}
```

2.3 Write simple units of code:

Jatorrizko createAlert(...) konprobazio eta try catch ugari zituenez, nahiko konplexutasun ziklomatiko handia zuen, eta hori jaistea da gure helburua.

Izan ere, horrela kodea ulertzeko zailagoa da, eta ahalik eta ulergarrien egin nahi dugu.

> Konplexutasun ziklomatikoa errefaktorizazioa aplikatu aurretik: V(G)=4.

Errefaktorizazioa aplikatu ondoren, metodo nagusiko adarkapen logikoak azpimetodo berezietan banatu dira, bakoitzak zeregin bakarra izan dezan.

Horrela, createaAlert(...) metodoaren **Konplexutasun ziklomatikoa** V(G)=4-tik V(G)=2-ra jaistea lortu dugu (try/catch + return).

> Konplexutasun ziklomatikoa errefaktorizazioa aplikatu ondoren: V(G)=2.

Modu honetan, metodoa ulergarriago egitea lortu dugu, azpimetodotan banatuz eta bakoitza bere aldetik eginez.

2. 4 Duplicate code:

- **Hasierako arazoa:** Jatorrizko createAlert(...) metodoan, kode errepikatua zegoen. Horretarako Local Variable bat sortzea pentsatu dugu, **db.getTransaction().commit();** instrukzioarekin. Instrukzio hori hiru aldiz agertzen da: baldintzetan, salbuespenetan eta metodoaren amaieran. Honek arazoak ekar ditzake kodea eguneratzean edo akatsak zuzentzean. Local Variable bat sortuz, kodearen mantengarritasuna hobetzea lortu dezakegu.

- **Egindako errefaktorizazioa:** Aurretik azaldutakoarekin onuragarria dirudien arren, kasu honetan ez da eraginkorra aipatutako Local Variablea sortzea. Hau void bat da, eta gainera Extract Method egitean, **db.getTransaction().commit()** instrukzioaren errepikapena asko murrizten da. Honekin iada lortu dugu gure helburua, hau oso onuragarria izango baita.

- **Ondorioa:** Aldaketa hauek aplikatuz, kodeak Duplicate code araua betetzea lortu dugu.

2. 5 Keep unit interfaces small:

createAlert(...) metodoak lau parametro jasotzen ditu: **createAlert(String nondik, String nora, Date data, String NAN).** Kasu honetan, ez da **Keep unit interfaces small** araua hausten, izan ere, arauak esaten duenez, metodo batek gehienez lau parametro izan beharko lituzke, parametro kopuru handiak metodoaren erabilpena eta ulermena zailtzen baitu.

Kasu honetan, ez da errefaktorizaziorik egin behar izan, metodoak lau parametro izanik, erabilpen eta ulermen egokia egin daitekeelako.

2. 6 Amaierako kodea:

Beharrezko errefaktorizazio guztiak aplikatu ondoren, hau da lortutako kodea:

```java
private Alerta saveAlert(String nondik, String nora, Date data, Traveler t) throws alertAlreadyExists {
    if (t.doesAlertExist(nondik,nora, data)) {
        db.getTransaction().commit();
        throw new alertAlreadyExists(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.alertAlreadyExists"));
    }
    Alerta alerta=t.addAlert(nondik, nora, data);
    //db.persist(t); -> ez da beharrezkoa, dagoeneko commit egiten delako
    db.getTransaction().commit();
    return alerta;
}
private void checkAlertDoesNotExist(String nondik, String nora, Date data, Traveler t) throws alertAlreadyExists {
    if (t.doesAlertExist(nondik,nora, data)) {
        db.getTransaction().commit();
        throw new alertAlreadyExists(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.alertAlreadyExists"));
    }
}
private Traveler findTraveler(String NAN) {
    Traveler t= db.find(Traveler.class,NAN );
    return t;
}
private void checkDateIsValid(Date data) throws RideMustBeLaterThanTodayException {
    if(new Date().compareTo(data)>0) {
        throw new RideMustBeLaterThanTodayException(ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
    }
}
```

2.7 Egilea: Maialen Velazquez

createBook()

3.1 Hasierako kodea:

```java
public Book createBook(String NAN, Integer rideNumber, int seats)throws bookAlreadyExistException, NoCashException,NotEnoughSeatsException {
        try {
            db.getTransaction().begin();
            Traveler t=db.find(Traveler.class,NAN);
            Ride r=db.find(Ride.class, rideNumber);

            if(t.existBook(r)) {
                db.getTransaction().commit();
                throw new bookAlreadyExistException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.BookAlreadyExist"));
            }

            if(r.getnPlaces()<seats) {
                db.getTransaction().commit();
                throw new NotEnoughSeatsException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.NotEnoughSeats"));
            }
            if(!t.hasCash(r, seats)) {
                db.getTransaction().commit();
                throw new NoCashException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.NoCash"));
            }
            //float prezioa=r.getPrice()*seats;
            float price=r.calculatePrice(seats);
            Book book=t.addBook(r, seats);
            book.setDiruIzoztuta(price);
            float d= t.diruaAtera(price);
            //t.setDirua(t.getDirua()-prezioa);
            //db.persist(book);
            r.addBook(book);
            //r.setnPlaces((int)r.getnPlaces() - seats);
            int nPlaces=r.updatenPlaces(seats);
            db.persist(t);
            db.getTransaction().commit();
            return book;
        }catch (NullPointerException e){
            db.getTransaction().commit();
            return null;
        }
    }
```

3. 2 Write short units of code:

Metodoak 15 lerro baino gehiago ditu, eta hori “Bad Smell” edo kodearen usain txarra da. Izan ere, metodo bakar batean informazio eta funtzionalitate gehiegi pilatzeak kodea ulertzea, mantentzea eta testatzea zailtzen du.

Arazo hau errefaktorizazioaren bidez konpon daiteke: kodea hainbat azpimetodotan banatzea proposatzen da, bakoitzak funtzio zehatz eta mugatua betetzeko. Horrela, kodea ulergarriagoa eta kudeagarriagoa bihurtzen da.

Egindako errefaktorizazioaren deskribapena, Extract Method teknikaren bidez: kodea azpimetodo txiki eta espezifikoetan banatu da, funtzionalitate bakoitzari zeregin bakarra esleituz.

- **createBook(String NAN, Integer rideNumber, int seats):** Funtzio nagusia da, eta erreserba bat sortzeko prozesu osoa koordinatzen du. Bidaiaria eta bidaia eskuratzen ditu, balidazioak egiten ditu, erreserba prozesatzen du eta

transakzioa kudeatzen du. Metodo honek ez du xehetasunetan sakontzen, baizik eta beste metodoetan delegatzen du funtzionalitatea.

- **validateBooking(Traveler traveler, Ride ride, int seats):** Erreserba sortu aurretik, hainbat baldintza konprobatzen dituzten metodoak edukiko ditu.
- **checkIfAlreadyBooked(Traveler traveler, Ride ride):** Bidaiariak bidaia horretarako erreserba bat dagoeneko egina duen egiaztatzen du. Hala bada, salbuespen bat jaurtiko du.
- **checkSeatAvailability(Ride ride, int seats):** metodo honen bitartez, erreserbatu nahi den bidaia eserleku nahikoa dituen konprobatzen da.
- **checkTravelerHasCash(Traveler traveler, Ride ride, int seats):** metodo honen bitartez, bidaia bat erreserbatu nahi duen bidaiariak dirua duen konprobatzen da.
- **processBooking(Traveler traveler, Ride ride, int seats, float price):** metodo honek, erreserba sortzeaz arduratuko diren metodoak edukiko ditu.
- **createBookEntry(Traveler traveler, Ride ride, int seats, float price):** erreserba sortuko du.
- **updateTravelerCash(Traveler traveler, float price):** metodo honek, bidaiariaren dirua eguneratzeaz arduratuko da.
- **updateRideBooking(Ride ride, Book book, int seats):** metodo honek, erreserba bidaiaren erreserba listan gehituko du eta eserleku kopurua eguneratuko du.

Egindako errefaktorizazioaren deskribapena Extract Local Variable: Errepikatatutako kodea dago, horretarako, errepikapenak saihesteko, kodea Local Variable batean gordetzea pentsatu dugun arren, ezinezkoa da void bat delako:

- **var transakzio = db.getTransaction().commit();**

Eta horrez gain, **Extract Method errefaktorizazioak** aplikatu ondoren, kode errepikapena murriztu egin da. Ondorioz, errefaktorizazioa hauei esker instrukzioa askoz kontrolatuagoa da. Beraz, ez da beharrezkoa hasieran planteatutako Local Variable hau sortzea.

Errefaktorizatutako kodea:

```java
public Book createBook(String NAN, Integer rideNumber, int seats)
            throws bookAlreadyExistException, NoCashException, NotEnoughSeatsException {
        try {
            db.getTransaction().begin();

            Traveler traveler = db.find(Traveler.class, NAN);
            Ride ride = db.find(Ride.class, rideNumber);

            validateBooking(traveler, ride, seats);

            float price = ride.calculatePrice(seats);
            Book book = processBooking(traveler, ride, seats, price);

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

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

    private void validateBooking(Traveler traveler, Ride ride, int seats)
            throws bookAlreadyExistException, NotEnoughSeatsException, NoCashException {
        checkIfAlreadyBooked(traveler, ride);
        checkSeatAvailability(ride, seats);
        checkTravelerHasCash(traveler, ride, seats);
    }

    private void checkIfAlreadyBooked(Traveler traveler, Ride ride) throws bookAlreadyExistException {
        if (traveler.existBook(ride)) {
            throw new bookAlreadyExistException(
                    ResourceBundle.getBundle("Etiquetas").getString("DataAccess.BookAlreadyExist"));
        }
    }

    private void checkSeatAvailability(Ride ride, int seats) throws NotEnoughSeatsException {
        if (ride.getnPlaces() < seats) {
            throw new NotEnoughSeatsException(
                    ResourceBundle.getBundle("Etiquetas").getString("DataAccess.NotEnoughSeats"));
        }
    }

    private void checkTravelerHasCash(Traveler traveler, Ride ride, int seats) throws NoCashException {
        if (!traveler.hasCash(ride, seats)) {
            throw new NoCashException(ResourceBundle.getBundle("Etiquetas").getString("DataAccess.NoCash"));
        }
    }

    private Book processBooking(Traveler traveler, Ride ride, int seats, float price) {
        Book book = createBookEntry(traveler, ride, seats, price);
        updateTravelerCash(traveler, price);
        updateRideBooking(ride, book, seats);
        return book;
    }

    private Book createBookEntry(Traveler traveler, Ride ride, int seats, float price) {
        Book book = traveler.addBook(ride, seats);
        book.setDiruIzoztuta(price);
        return book;
    }

    private void updateTravelerCash(Traveler traveler, float price) {
        traveler.diruaAtera(price);
    }

    private void updateRideBooking(Ride ride, Book book, int seats) {
        ride.addBook(book);
        ride.updatenPlaces(seats);
    }
```

3. 3 Write simple units of code:

Hasieran, createBook() metodoaren konplexutasun ziklomatikoa V(G)=5 zen. Errefaktorizazioa aplikatu ondoren, createBook() metodoaren konplexutasun ziklomatikoa V(G)=2-ra jaitsi da.

3. 4 Duplicate code:

- **Hasierako arazoa:** createBook() metodoa errefaktorizatu ondoren, oraindik, metodo batzuetan, kode bikoizketak zeuden. Arazo hori salbuespenak tratatzen dituzten metodoetan dago, izan ere, denetan“ResourceBundle.getBundle("Etiquetas").getString("DataAccess.")” erabiltzen da.
- **Egindako errefaktorizazioa:** goian azaldutako arazoari konponbide bat emateko, beste metodo bat sortzea pentsatu dugu. Metodo honek, salbuespenen bat altxatzen denean, mezuak bidaltzeaz arduratuko da.

```
1  private String getMessage(String key) {
2          return ResourceBundle.getBundle("Etiquetas").getString(key);
3      }
```

“Key” parametroa jasoko du, bertan adieraziko da nondik eta zer mezu eskuratu nahi dugun.

3. 5 Keep unit interfaces small: metodo honek 3 parametro ditu, hortaz arau hau betetzen du.

3. 6 Amaierako kodea:

```java
public Book createBook(String NAN, Integer rideNumber, int seats)
            throws bookAlreadyExistException, NoCashException, NotEnoughSeatsException {
        try {
            db.getTransaction().begin();

            Traveler traveler = db.find(Traveler.class, NAN);
            Ride ride = db.find(Ride.class, rideNumber);

            validateBooking(traveler, ride, seats);

            float price = ride.calculatePrice(seats);
            Book book = processBooking(traveler, ride, seats, price);

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

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

    private void validateBooking(Traveler traveler, Ride ride, int seats)
            throws bookAlreadyExistException, NotEnoughSeatsException, NoCashException {
        checkIfAlreadyBooked(traveler, ride);
        checkSeatAvailability(ride, seats);
        checkTravelerHasCash(traveler, ride, seats);
    }

    private void checkIfAlreadyBooked(Traveler traveler, Ride ride) throws bookAlreadyExistException {
        if (traveler.existBook(ride)) {
            throw new bookAlreadyExistException(getMessage("DataAccess.BookAlreadyExist"));
        }
    }

    private void checkSeatAvailability(Ride ride, int seats) throws NotEnoughSeatsException {
        if (ride.getnPlaces() < seats) {
            throw new NotEnoughSeatsException(getMessage("DataAccess.NotEnoughSeats"));
        }
    }

    private void checkTravelerHasCash(Traveler traveler, Ride ride, int seats) throws NoCashException {
        if (!traveler.hasCash(ride, seats)) {
            throw new NoCashException(getMessage("DataAccess.NoCash"));
        }
    }

    private Book processBooking(Traveler traveler, Ride ride, int seats, float price) {
        Book book = createBookEntry(traveler, ride, seats, price);
        updateTravelerCash(traveler, price);
        updateRideBooking(ride, book, seats);
        return book;
    }

    private Book createBookEntry(Traveler traveler, Ride ride, int seats, float price) {
        Book book = traveler.addBook(ride, seats);
        book.setDiruIzoztuta(price);
        return book;
    }

    private void updateTravelerCash(Traveler traveler, float price) {
        traveler.diruaAtera(price);
    }

    private void updateRideBooking(Ride ride, Book book, int seats) {
        ride.addBook(book);
        ride.updatenPlaces(seats);
    }

    private String getMessage(String key) {
        return ResourceBundle.getBundle("Etiquetas").getString(key);
    }
```

3.7 Egilea: Ivan Priala

4. Github kodea: https://github.com/maialenvelazquez/Rides24-PROJECT.git