SOFTWARE INGENIARITZA

Egileak:

Braian Porta

Martín Etxeberria

GitHub:

https://github.com/Braian-PC/Rides24Complete

SonarCloud:

https://sonarcloud.io/project/overview?id=braian-pc_rides24complete

"Write short units of code"

“Bad smell” edo “issue” hau zuzentzeko SonarCloud-era joan gaitezke, bertan erroreak DataAccess fitxategi-koak soilik agertzea filtratu, eta behi egiterakoan Adaptavility erroreak aztertu.

Bertan luzeegia den metodo bat agertuko zaigu, deleteUser(User us), honen refaktorizazio lana egite aldera, 15 kode lerro edo gutxiago eduki ditzan, “Refactor->Extract Method” Eclipse-ko tresna erabili dezakegu. Adibidez 906-918 lerroetan (else-aren barne) dagoen kodea metodo berri batetara ateraz.

Cognitive Complexity of methods should not be too high

Adaptability | Not focused Maintainability java:S3776

This rule raises an issue when the code cognitive complexity of a function is above a certain threshold.

```
888    public void deleteUser(User us) {
889        try {
890            if (us.getMota().equals("Driver")) {
891                List<Ride> rl = getRidesByDriver(us.getUsername());
892                if (rl != null) {
893                    for (Ride ri : rl) {
894                        cancelRide(ri);
895                    }
896                }
897                Driver d = getDriver(us.getUsername());
898                List<Car> cl = d.getCars();
899                if (cl != null) {
900                    for (int i = cl.size() - 1; i >= 0; i--) {
901                        Car ci = cl.get(i);
902                        deleteCar(ci);
903                    }
904                }
905            } else {
906                List<Booking> lb = getBookedRides(us.getUsername());
907                if (lb != null) {
908                    for (Booking li : lb) {
909                        li.setStatus(REJECTED);
910                        li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
911                    }
912                }
913                List<Alert> la = getAlertsByUsername(us.getUsername());
914                if (la != null) {
915                    for (Alert lx : la) {
916                        deleteAlert(lx.getAlertNumber());
917                    }
918                }
919            }
920            db.getTransaction().begin();
921            us = db.merge(us);
922            db.remove(us);
923            db.getTransaction().commit();
924        } catch (Exception e) {
925            e.printStackTrace();
926        }
927    }
```

```
888    public void deleteUser(User us) {
889        try {
890            if (us.getMota().equals("Driver")) {
891                List<Ride> rl = getRidesByDriver(us.getUsername());
892                if (rl != null) {
893                    for (Ride ri : rl) {
894                        cancelRide(ri);
895                    }
896                }
897                Driver d = getDriver(us.getUsername());
898                List<Car> cl = d.getCars();
899                if (cl != null) {
900                    for (int i = cl.size() - 1; i >= 0; i--) {
901                        Car ci = cl.get(i);
902                        deleteCar(ci);
903                    }
904                }
905            } else {
906                deleteNonUser(us);
907            }
908            db.getTransaction().begin();
909            us = db.merge(us);
910            db.remove(us);
911            db.getTransaction().commit();
912        } catch (Exception e) {
913            e.printStackTrace();
914        }
915    }
916
917    private void deleteNonUser(User us) {
918        List<Booking> lb = getBookedRides(us.getUsername());
919        if (lb != null) {
920            for (Booking li : lb) {
921                li.setStatus(REJECTED);
922                li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
923            }
924        }
925        List<Alert> la = getAlertsByUsername(us.getUsername());
926        if (la != null) {
927            for (Alert lx : la) {
928                deleteAlert(lx.getAlertNumber());
929            }
930        }
931    }
```

DataAccess klasean luzeegia den beste metodo baten adibidea updateAurkitutakoak(String username) metodoa da(SonarCloud herramientan agertzen ez zaigun arren). Honen luzera “Refactor->Extract Method” tresna bera erabiliz moztu dezakegu. Kasu honetan 374 lerroan dagoen for begizta guztiz metodo berri gisa ateraz.

```java
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;
    }
}
```

```java
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();

        alertFound = UpdateFoundAlert(alertFound, alerts, rides);

        db.getTransaction().commit();
        return alertFound;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
private boolean UpdateFoundAlert(boolean alertFound, List<Alert> alerts, List<Ride> rides) {
    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);
    }
    return alertFound;
}
```

"Write simple units of code"

“Issue” hau SonarLint-en aurkitu ezin dugun arren, aurreko bi metodoek “Bad Smell” hau burutzen dutela antzeman dezakegu; hau da, bi metodoak simplifikatu arren 4 baldintza “nodo” edo gehiago edukitzen jarraitzen dute.

Beraz hau konpontzeko errefaktorizazio tresna bera erabili dezakegu, era honetan metodoen konplexutasuna murriztuz.

```
888    public void deleteUser(User us) {
889        try {
890            if (us.getMota().equals("Driver")) {
891                List<Ride> rl = getRidesByDriver(us.getUsername());
892                if (rl != null) {
893                    for (Ride ri : rl) {
894                        cancelRide(ri);
895                    }
896                }
897                Driver d = getDriver(us.getUsername());
898                List<Car> cl = d.getCars();
899                if (cl != null) {
900                    for (int i = cl.size() - 1; i >= 0; i--) {
901                        Car ci = cl.get(i);
902                        deleteCar(ci);
903                    }
904                }
905            } else {
906                deleteNonUser(us);
907            }
908            db.getTransaction().begin();
909            us = db.merge(us);
910            db.remove(us);
911            db.getTransaction().commit();
912        } catch (Exception e) {
913            e.printStackTrace();
914        }
915    }
916
917    private void deleteNonUser(User us) {
918        List<Booking> lb = getBookedRides(us.getUsername());
919        if (lb != null) {
920            for (Booking li : lb) {
921                li.setStatus(REJECTED);
922                li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
923            }
924        }
925        List<Alert> la = getAlertsByUsername(us.getUsername());
926        if (la != null) {
927            for (Alert lx : la) {
928                deleteAlert(lx.getAlertNumber());
929            }
930        }
931    }
932
```

Era honetan ondorengoa izango da deleteUser metodoaren bertsio errefaktorizatua(11 kode lerro eta 2 baldintza!):

```
888    public void deleteUser(User us) {
889        try {
890            if (us.getMota().equals("Driver")) {
891                exDeleteDriver(us);
892            } else {
893                exDeleteNonUser(us);
894            }
895            db.getTransaction().begin();
896            us = db.merge(us);
897            db.remove(us);
898            db.getTransaction().commit();
899        } catch (Exception e) {
900            e.printStackTrace();
901        }
902    }
903
904    private void exDeleteNonUser(User us) {
905        List<Booking> lb = getBookedRides(us.getUsername());
906        if (lb != null) {
907            for (Booking li : lb) {
908                li.setStatus(REJECTED);
909                li.getRide().setnPlaces(li.getRide().getnPlaces() + li.getSeats());
910            }
911        }
912        List<Alert> la = getAlertsByUsername(us.getUsername());
913        if (la != null) {
914            for (Alert lx : la) {
915                deleteAlert(lx.getAlertNumber());
916            }
917        }
918    }
919
920    private void exDeleteDriver(User us) throws Exception {
921        List<Ride> rl = getRidesByDriver(us.getUsername());
922        if (rl != null) {
923            for (Ride ri : rl) {
924                cancelRide(ri);
925            }
926        }
927        Driver d = getDriver(us.getUsername());
928        List<Car> cl = d.getCars();
929        if (cl != null) {
930            for (int i = cl.size() - 1; i >= 0; i--) {
931                Car ci = cl.get(i);
932                deleteCar(ci);
933            }
934        }
935    }
```

```
960    public boolean updateAlertaAurkituak(String username) {
961        try {
962            db.getTransaction().begin();
963
964            boolean alertFound = false;
965            TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
966                    Alert.class);
967            alertQuery.setParameter("username", username);
968            List<Alert> alerts = alertQuery.getResultList();
969
970            TypedQuery<Ride> rideQuery = db
971                    .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
972            List<Ride> rides = rideQuery.getResultList();
973
974            alertFound = UpdateFoundAlert(alertFound, alerts, rides);
975
976            db.getTransaction().commit();
977            return alertFound;
978        } catch (Exception e) {
979            e.printStackTrace();
980            db.getTransaction().rollback();
981            return false;
982        }
983    }
984    private boolean UpdateFoundAlert(boolean alertFound, List<Alert> alerts, List<Ride> rides) {
985        for (Alert alert : alerts) {
986            boolean found = false;
987            for (Ride ride : rides) {
988                if (UtilDate.datesAreEqualIgnoringTime(ride.getDate(), alert.getDate())
989                        && ride.getFrom().equals(alert.getFrom()) && ride.getTo().equals(alert.getTo())
990                        && ride.getnPlaces() > 0) {
991                    alert.setFound(true);
992                    found = true;
993                    if (alert.isActive())
994                        alertFound = true;
995                    break;
996                }
997            }
998            if (!found) {
999                alert.setFound(false);
000            }
001            db.merge(alert);
002        }
003        return alertFound;
004    }
```

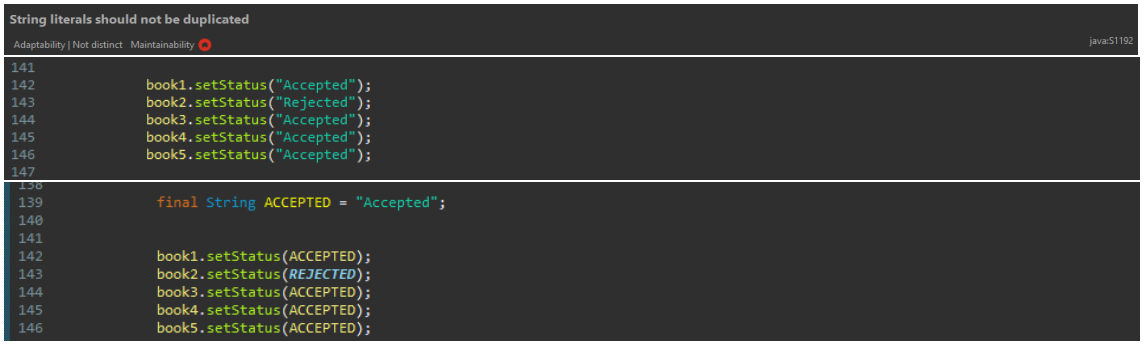
Aurreko ariketan Errefaktorizatutako metodo hau luzera(updateFoundAlert) egokia duen arren, baldintza kopuru handiegia edukitzen jarraitzen du, beraz berriro errefaktorizatu dezakegu, honen konplexutasuna murrizteko.

```
960    public boolean updateAlertaAurkituak(String username) {
961        try {
962            db.getTransaction().begin();
963
964            boolean alertFound = false;
965            TypedQuery<Alert> alertQuery = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
966                    Alert.class);
967            alertQuery.setParameter("username", username);
968            List<Alert> alerts = alertQuery.getResultList();
969
970            TypedQuery<Ride> rideQuery = db
971                    .createQuery("SELECT r FROM Ride r WHERE r.date > CURRENT_DATE AND r.active = true", Ride.class);
972            List<Ride> rides = rideQuery.getResultList();
973
974            alertFound = UpdateFoundAlert(alertFound, alerts, rides);
975
976            db.getTransaction().commit();
977            return alertFound;
978        } catch (Exception e) {
979            e.printStackTrace();
980            db.getTransaction().rollback();
981            return false;
982        }
983    }
984    private boolean UpdateFoundAlert(boolean alertFound, List<Alert> alerts, List<Ride> rides) {
985        for (Alert alert : alerts) {
986            alertFound = findAlert(alertFound, rides, alert);
987            db.merge(alert);
988        }
989        return alertFound;
990    }
991    private boolean findAlert(boolean alertFound, List<Ride> rides, Alert alert) {
992        boolean found = false;
993        for (Ride ride : rides) {
994            if (UtilDate.datesAreEqualIgnoringTime(ride.getDate(), alert.getDate())
995                    && ride.getFrom().equals(alert.getFrom()) && ride.getTo().equals(alert.getTo())
996                    && ride.getnPlaces() > 0) {
997                alert.setFound(true);
998                found = true;
999                if (alert.isActive())
1000                    alertFound = true;
1001                break;
1002            }
1003        }
1004        if (!found) {
1005            alert.setFound(false);
1006        }
1007        return alertFound;
1008    }
```

“Duplicate code”

Kode berdin bat mahiz errepikatua edukitzea oso kaltegarria izan daiteke etorkizunerako kodearen aldaketaren aldera (honen errefaktorizazioan). Kode errepikapenak modu askotan gertatu daitezke, lehen kasu honek String literalen errepikapen baten kiratsaz ohartzen gaitu.

Kodea dagoen moduan utziz gero etorkizunean buruhauste bat izatea bilakatu ahal izango litzateke kodean akats bat egongo balitz, edota karaktere kate guzti hauen balioa aldatu nahi izango bagenu. Beraz, “String” hauen balioa aldagai konstate batean gordetzea onuragarria gerta daiteke arazo hauek ekiditeko.



Aurkitu dugun beste kode errepikapena, “updateAlertaAurkituak” eta “getAlertsAurkituak” metodoetan.

```
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);
```

```
public List<Alert> getAlertsByUsername(String username) {
    try {
        db.getTransaction().begin();

        TypedQuery<Alert> query = db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",
                Alert.class);
```

Haietan, datu-baseari dei egiten die. Hortaz, kodea optimizatzeko eta baldin beste funtzio bat sortzen badugu datu basea (Gauza bera eskatzen dutenak) erabiltzen duela berriro, orduan hurrengo metodoari deia egingo dio:

```
public TypedQuery<Alert> getAlertaQuery() {
    return db.createQuery("SELECT a FROM Alert a WHERE a.traveler.username = :username",Alert.class);
}
```

Horrela, lortuko dugu hasierako bi metodoetan berriro deiak ez egitea datu baseari, baizik eta método bateri:

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

        boolean alertFound = false;
        TypedQuery<Alert> alertQuery = this.getAlertaQuery();
```

```
public List<Alert> getAlertsByUsername(String username) {
    try {
        db.getTransaction().begin();

        TypedQuery<Alert> query = this.getAlertaQuery();
```

"Keep unit interfaces small"

Ariketa hau gauzatu ahal izateko parametro gehiegi jasotzen dituen metodo pare bat identifikatu beharko ditugu. Bestalde, aldaketak egiterako orduan kontuan eduki beharko dugu DataAccess klaseko metodo oro aldatzerakoan hau deitzen duen metodoak ere aldatu beharko direla. Beraz BussinessLogic interfazea, honen inplementazioa, eta metodo hauek erabiltzen dituzten GUI klaseak aldatu beharko ditugula eduki beharko dugu kontuan.

Beraz, erreklamazioa bidali metodoan “keep unit interfaces small” “code smell” arazoa konpontzeko antzeman dezakegu jasotzen dituen parametro guztiak Complaint objetu bat sortzeko erabiltzen direla. Beraz, parametro hauek jaso beharrean metodoak Complaint objetu bat jasotzera aldatuko dugu (eta ondorioz erabiltzen duten klase guztien metodoak ere).

```
749
750     public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua,
751             boolean aurk) {
752         try {
753             db.getTransaction().begin();
754
755             Complaint erreklamazioa = new Complaint(nor, nori, gaur, booking, textua, aurk);
756             db.persist(erreklamazioa);
757             db.getTransaction().commit();
758             return true;
759         } catch (Exception e) {
760             e.printStackTrace();
761             db.getTransaction().rollback();
762             return false;
763         }
764     }
765
```

```
749
750
751     public boolean erreklamazioaBidali(Complaint complaint) {
752         try {
753             db.getTransaction().begin();
754
755             Complaint erreklamazioa = complaint;
756             db.persist(erreklamazioa);
757             db.getTransaction().commit();
758             return true;
759         } catch (Exception e) {
760             e.printStackTrace();
761             db.getTransaction().rollback();
762             return false;
763         }
764     }
765
```

“createRide” metodo honen kasua zertxobait konplexuagoa da, datu baseko “addRide” metodoa erabiltzen duelako jasotako parametroak baliatuz ride objetu bat zuzenean sortzeko eta dagokion gidariari esleitzeko. Hala ere, aurrekoaren era berean zuzendu daiteke arazo hau, parametro kopuru handia Ride objetu parametro bakar batez ordezkatuz.

```
237    public Ride createRide(int nPlaces, float price, Ride rRide)
238            throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
239        String driverName=rRide.getDriver().getUsername();
240        String from= rRide.getFrom();
241        String to= rRide.getTo();
242        Date date= rRide.getDate();
243        logger.info(
244                ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
245        if (driverName==null) return null;
246        try {
247            if (new Date().compareTo(date) > 0) {
248                logger.info("ppppp");
249                throw new RideMustBeLaterThanTodayException(
250                        ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
251            }
252
253            db.getTransaction().begin();
254            Driver driver = db.find(Driver.class, driverName);
255            if (driver.doesRideExists(from, to, date)) {
256                db.getTransaction().commit();
257                throw new RideAlreadyExistException(
258                        ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
259            }
260            Ride ride = driver.addRide(from, to, date, nPlaces, price);
261            // next instruction can be obviated
262            db.persist(driver);
263            db.getTransaction().commit();
264
265            return ride;
266        } catch (NullPointerException e) {
267            return null;
268        }
269
```

```
237    public Ride createRide(String from, String to, Date date, int nPlaces, float price, String driverName)
238            throws RideAlreadyExistException, RideMustBeLaterThanTodayException {
239        logger.info(
240                ">> DataAccess: createRide=> from= " + from + " to= " + to + " driver=" + driverName + " date " + date);
241        if (driverName==null) return null;
242        try {
243            if (new Date().compareTo(date) > 0) {
244                logger.info("ppppp");
245                throw new RideMustBeLaterThanTodayException(
246                        ResourceBundle.getBundle("Etiquetas").getString("CreateRideGUI.ErrorRideMustBeLaterThanToday"));
247            }
248
249            db.getTransaction().begin();
250            Driver driver = db.find(Driver.class, driverName);
251            if (driver.doesRideExists(from, to, date)) {
252                db.getTransaction().commit();
253                throw new RideAlreadyExistException(
254                        ResourceBundle.getBundle("Etiquetas").getString("DataAccess.RideAlreadyExist"));
255            }
256            Ride ride = driver.addRide(from, to, date, nPlaces, price);
257            // next instruction can be obviated
258            db.persist(driver);
259            db.getTransaction().commit();
260
261            return ride;
262        } catch (NullPointerException e) {
263            return null;
264        }
265
266
267    }
268
```