"Write short units of code" (capítulo 2)

• Limita la longitud de las unidades de código a 15 líneas.
• Para ello, evita escribir unidades que superen las 15 líneas desde el principio o divide las unidades largas en varias unidades más pequeñas hasta que cada una tenga un máximo de 15 líneas.
• Esto mejora la mantenibilidad, ya que las unidades pequeñas son fáciles de entender, probar y reutilizar.

1. Refactorización

- Código inicial

```
public Ride bookRide(Integer cd, String emailTraveler) {
    Ride ride = db.find(Ride.class, cd);
    Driver drvr = db.find(Driver.class, ride.getDriverInfo().getEmail());
    Traveler trvlr = db.find(Traveler.class, emailTraveler);

    db.getTransaction().begin();
    ride.setState("reservado");
    ride.setTravelerInfo(trvlr);

    DecimalFormat df = new DecimalFormat("0.00");
    if(trvlr.hasDiscountRide(ride.getFrom(), ride.getTo())) {
        ride.setDiscount();
        trvlr.subCash(Float.parseFloat(df.format((float) (ride.getPrice() - ride.getPrice() * 0.1))));
    }
    else
        trvlr.subCash(Float.parseFloat(df.format(ride.getPrice())));

    trvlr.bookRide(ride);
    drvr.bookRide(ride);
    db.getTransaction().commit();

    System.out.println("Viaje reservado: \n"+ride);
    return ride;
}
```

- Código refactorizado

```
public Ride bookRide(Integer cd, String emailTraveler) {
    Ride ride = db.find(Ride.class, cd);
    Driver drvr = db.find(Driver.class, ride.getDriverInfo().getEmail());
    Traveler trvlr = db.find(Traveler.class, emailTraveler);

    db.getTransaction().begin();
    processRideBooking(ride, drvr, trvlr);
    db.getTransaction().commit();

    System.out.println("Viaje reservado: \n"+ride);
    return ride;
}
```

```java
private void processRideBooking(Ride ride, Driver drvr, Traveler trvlr) {
    ride.setState("reservado");
    ride.setTravelerInfo(trvlr);

    applyPayment(ride, trvlr);

    trvlr.bookRide(ride);
    drvr.bookRide(ride);
}

private void applyPayment(Ride ride, Traveler trvlr) {
    DecimalFormat df = new DecimalFormat("0.00");
    float price = ride.getPrice();

    if(trvlr.hasDiscountRide(ride.getFrom(), ride.getTo())) {
        ride.setDiscount();
        price = (float) (price - price * 0.1);
    }
    trvlr.subCash(Float.parseFloat(df.format(price)));
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

El método bookRide hace demasiadas cosas: Busca entidades (Ride, Driver, Traveler), maneja transacciones, aplica descuentos, cambia estado del viaje y registra reserva.

En el método se pueden extraer funcionalidades que las puede hacer un método a parte. Se puede separar la aplicación del descuento y el cobro: applyPayment(Ride ride, Traveler trvlr)
Y que otro método procese la reserva y los cambios del viaje. processRideBooking(Ride ride, Driver drvr, Traveler trvlr)

- Miembro que ha realizado la refactorización.

Shirley

2. Refactorización

- Código inicial

```java
public void initializeDB(){

                db.getTransaction().begin();
                try {
                  Calendar today = Calendar.getInstance();

                  int month=today.get(Calendar.MONTH);
                  int year=today.get(Calendar.YEAR);
                  if (month==12) { month=1; year+=1;}   … y mucho más (157 linas)
```

- Código refactorizado

```
        public void initializeDB(){
                db.getTransaction().begin();
            try {
                createBaseRides();
                createUsersAndCustomRides();
                db.getTransaction().commit();
                System.out.println("Db initialized");
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
```

```
public void createBaseRides() {
                String[][] routes = {
                        {CITY_DONOSTIA, CITY_MADRID, "Alava", "Logroño", "Soria"},
                        {CITY_DONOSTIA, CITY_BARCELONA, "Pamplona", "Zaragoza",
"Tarragona"},
                        {CITY_BARCELONA, CITY_MADRID, "Tarragona", "Teruel", "Guadalajara"},
                        {CITY_BARCELONA, CITY_VALENCIA, "Tarragona", "Castellón"},
                        {CITY_VALENCIA, CITY_MADRID, "Albacete", "Cuenca"},
                        {CITY_GRANADA, CITY_MADRID, "Jaén", "Toledo"},
                        {CITY_GRANADA, CITY_DONOSTIA, "Jaén", "Toledo", CITY_MADRID,
"Soria"}
                };
                    for (String[] r : routes) {
                        persistRoundTrip(r[0], r[1], Arrays.copyOfRange(r, 2, r.length));
                    }
        }
```

```
private void persistRoundTrip(String from, String to, String... stops) {
            Ride a = new Ride(from, to);
            Ride b = new Ride(to, from);
            for (String s : stops) { a.addStop(s); b.addStop(s); }
            a.setState("ini"); b.setState("ini");
            db.persist(a); db.persist(b);
        }
```

```
private void createUsersAndCustomRides() {
                User userT = createUser("emailT", "T");
            User userD = createUser("emailD", "D");
            float[] prices = {22.19f, 35.13f, 17.75f, 48.22f, 28.19f, 20.19f, 11.75f, 13.95f, 60.42f};
            int[] seats = {2, 3, 1, 4, 2, 2, 1, 1, 5};
            String[][] stops = {
                {"Pamplona"}, {"Pamplona", "Zaragoza"}, {"Pamplona", "Tarragona"}, {},
                {"Pamplona", "Zaragona", "Tarragona"}, {}, {}, {"Tarragona"}, {"Zaragoza"}
            };
            for (int i = 0; i < prices.length; i++)
                addRide(userD, stops[i], seats[i], prices[i]);
            db.persist(userT);
            db.persist(userD);
```

```
        }
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

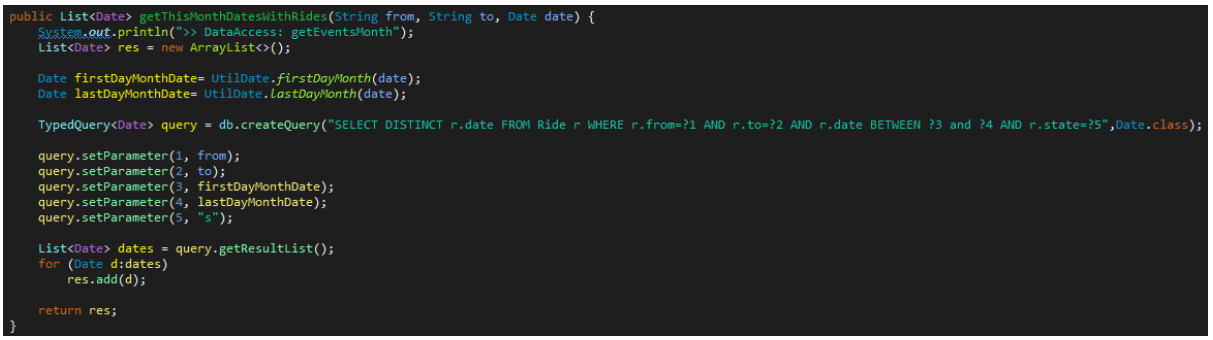
El método tenía exactamente 157 líneas de código. No era difícil de entender ya que como bien el nombre indica, el método inicializa la base de datos. Pero ahora que se ha dividido en 3 métodos el método es muchísimo más legible.

- Miembro que ha realizado la refactorización.

Ander

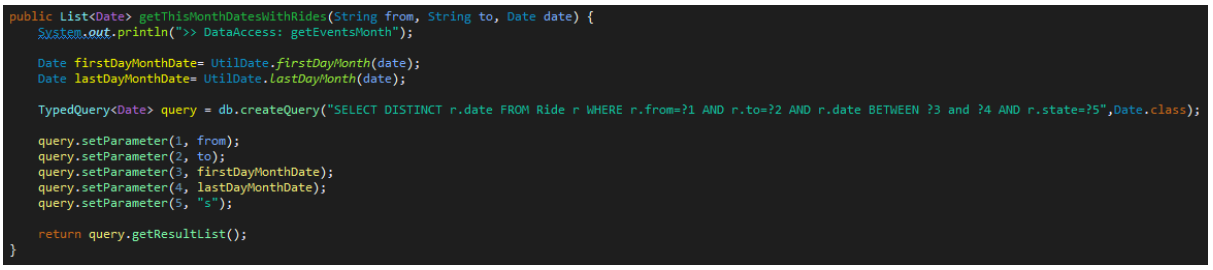
3. Refactorización

- Código inicial



```
public List<Date> getThisMonthDatesWithRides(String from, String to, Date date) {
    System.out.println(">> DataAccess: getEventsMonth");
    List<Date> res = new ArrayList<>();

    Date firstDayMonthDate= UtilDate.firstDayMonth(date);
    Date lastDayMonthDate= UtilDate.lastDayMonth(date);

    TypedQuery<Date> query = db.createQuery("SELECT DISTINCT r.date FROM Ride r WHERE r.from=?1 AND r.to=?2 AND r.date BETWEEN ?3 and ?4 AND r.state=?5",Date.class);

    query.setParameter(1, from);
    query.setParameter(2, to);
    query.setParameter(3, firstDayMonthDate);
    query.setParameter(4, lastDayMonthDate);
    query.setParameter(5, "s");

    List<Date> dates = query.getResultList();
    for (Date d:dates)
        res.add(d);

    return res;
}
```

- Código refactorizado



```
public List<Date> getThisMonthDatesWithRides(String from, String to, Date date) {
    System.out.println(">> DataAccess: getEventsMonth");

    Date firstDayMonthDate= UtilDate.firstDayMonth(date);
    Date lastDayMonthDate= UtilDate.lastDayMonth(date);

    TypedQuery<Date> query = db.createQuery("SELECT DISTINCT r.date FROM Ride r WHERE r.from=?1 AND r.to=?2 AND r.date BETWEEN ?3 and ?4 AND r.state=?5",Date.class);

    query.setParameter(1, from);
    query.setParameter(2, to);
    query.setParameter(3, firstDayMonthDate);
    query.setParameter(4, lastDayMonthDate);
    query.setParameter(5, "s");

    return query.getResultList();
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

en vez de inicializar una lista vacía y recorrer todo el query con un for, se utiliza la propia función query que retorna el resultado la lista de la consulta SQL

- Miembro que ha realizado la refactorización.

Adrian Rodriguez

"Write simple units of code" (capítulo 3)

• Limita el número de puntos de ramificación por unidad a 4.
• Para ello, divide las unidades complejas en unidades más simples y evita las complejas por completo.
• Esto mejora la mantenibilidad, ya que mantener un número bajo de puntos de ramificación facilita la modificación y las pruebas de las unidades.

1. Refactorización

- Código inicial

```
public User registerUser(User user) {
    User u = db.find(User.class, user.getEmail());
    if(u == null) {
        db.getTransaction().begin();
        if(user.getTraveler() != null) {
            user.getTraveler().setUserInfo(user);
            user.getTraveler().setEmail(user.getEmail());
        }
        else if (user.getDriver() != null) {
            user.getDriver().setUserInfo(user);
            user.getDriver().setEmail(user.getEmail());
        }
        db.persist(user);
        db.getTransaction().commit();
    }
    return u;
}
```

- Código refactorizado

```
public User registerUser(User user) {
    User uExists = db.find(User.class, user.getEmail());
    if(uExists != null) return uExists;
    db.getTransaction().begin();
    prepareRoles(user);
    db.persist(user);
    db.getTransaction().commit();

    return user;

}
```

```
private void prepareRoles(User user) {
    if(user.getTraveler() != null) {
        setUpTraveler(user);
    }
    else if (user.getDriver() != null) {
        setUpDriver(user);
    }
}

private void setUpDriver(User user) {
    user.getDriver().setUserInfo(user);
    user.getDriver().setEmail(user.getEmail());
}

private void setUpTraveler(User user) {
    user.getTraveler().setUserInfo(user);
    user.getTraveler().setEmail(user.getEmail());
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

Maneja demasiadas decisiones (if (u == null), if/else if para roles, etc.) Para cada decisión se podría preparar un rol. Si fuese traveler se prepararía el user para que fuese traveler setUpTraveler(User user) y si fuese driver se prepararía el user para que sea driver setUpDriver(User user) y por último estaría el método que dependiendo si es driver o user llamaría al método correspondiente a su rol prepareRoles(User user).

- Miembro que ha realizado la refactorización.

Shirley

2. Refactorización

- Código inicial

```
public String getStopsFromDate(String from, String to, Integer code, Date date, String state) {
                List<Ride> ridesDate = new ArrayList<Ride>();

                if(state.equals("p"))   #1 if
                        ridesDate = this.getCreatedRidesFromDate(from, to, date);
                else if(state.equals("b"))  # 2
                        ridesDate = this.getBookedRidesFromDate(from, to, date);
                else if(state.equals("a")) #3
                        ridesDate = this.getAcceptedRidesFromDate(from, to, date);

                boolean isCode = false;
                for(Ride r: ridesDate) {
                        if(r.getRideNumber().compareTo(code) == 0) { #4
```

```
                                    isCode = true;
                                    if(r.getStops().size() > 0) #5
                                            return r.stopsToString();
                            }
                    }
                    if(isCode == false) return "No se encuentra el viaje"; #6
                    else return "Sin paradas"; #7
            }
```

- Código refactorizado

```
public String getStopsFromDate(String from, String to, Date date, Integer code) {
        List<Ride> ridesDate = new ArrayList<>();

        // Combinar las listas de rides de diferentes estados
        ridesDate.addAll(getCreatedRidesFromDate(from, to, date));
        ridesDate.addAll(getBookedRidesFromDate(from, to, date));
        ridesDate.addAll(getAcceptedRidesFromDate(from, to, date));

        // Buscar el ride por código
        Ride ride = findRideByCode(ridesDate, code);
        if (ride == null) {
            return "No se encuentra el viaje";
        }
        return ride.getStops().isEmpty() ? "Sin paradas" : ride.stopsToString();
    }
    // Método auxiliar para buscar un ride por su código
    private Ride findRideByCode(List<Ride> rides, Integer code) {
        for (Ride r : rides) {
            if (r.getRideNumber().compareTo(code) == 0) {
                return r;
            }
        }
        return null;
    }
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

Code Smell: Too Many Branch Points. El método original tiene 7 puntos de ramificación, lo que lo hace complejo, difícil de probar y mantener. Ahora solo tiene un único punto de ramificación

- Miembro que ha realizado la refactorización.

Ander

3. Refactorización

- Código inicial

```java
public User loginUser(String email, String passw) {
    User user = db.find(User.class, email);
    //Contraseña incorrecta
    if (user != null && !user.getPassword().equals(passw)){
            user.setPassword(null);
    }
    else if(user != null && user.getPassword().equals(passw)) {
        if(user.getDriver() != null) {
            System.out.println(user.getDriver().getCreatedRides());
            System.out.println(user.getDriver().getBookedRides());
            System.out.println(user.getDriver().getAcceptedRides());
        }
        else if(user.getTraveler() != null)
            System.out.println(user.getTraveler().getAcceptedRides());
    }
    return user;
}
```

- Código refactorizado

```java
public User loginUser(String email, String passw) {
    User user = db.find(User.class, email);
    if (user == null || isPasswordValid(user, passw)){
            return null;
    }
    clearPassword(user);
    loadUserRides(user);
    return user;
}

private boolean isPasswordValid(User user, String password) {
    return user.getPassword() != null && user.getPassword().equals(password);
}

private void clearPassword(User user) {
    user.setPassword(null);
}

private void loadUserRides(User user) {
    if (user.getDriver() != null) {
        System.out.println(user.getDriver().getCreatedRides());
        System.out.println(user.getDriver().getBookedRides());
        System.out.println(user.getDriver().getAcceptedRides());
    } else if (user.getTraveler() != null) {
        System.out.println(user.getTraveler().getAcceptedRides());
    }
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

todo se decide dentro del método haciendo varios if / else con condiciones conjuntas haciéndolo complejo de entender.

se refactoriza haciendo tres métodos nuevos donde el propio nombre indica que verifica la contraseña, por seguridad la borra y carga los viajes del usuario dependendiendo si es viajero o conductor

- Miembro que ha realizado la refactorización.

Adrian Rodriguez

“Duplicate code” (capítulo 4).

SonarLint y sonarcloud nos indica dónde hay duplicidad de código.
• No copie el código.
• Para ello, escriba código genérico reutilizable o invoque métodos existentes.
• Esto mejora la mantenibilidad, ya que al copiar código, los errores deben corregirse en varios lugares, lo cual es ineficiente y propenso a errores.

1. Refactorización

- Código inicial

```
public List<Ride> getCreatedRidesFromDate(String from, String to, Date date){
    List<Ride> res = new ArrayList<Ride>();
        for(Ride r: createdRides) {
            if(r.getDate().getTime() == date.getTime() && r.getFrom().equals(from) && r.getTo().equals(to))
                res.add(r);
    }
    System.out.println(res);
    return res;
}

public List<Ride> getBookedRidesFromDate(String from, String to, Date date){
    List<Ride> res = new ArrayList<Ride>();
        for(Ride r: bookedRides) {
            if(r.getDate().getTime() == date.getTime() && r.getFrom().equals(from) && r.getTo().equals(to))
                res.add(r);
    }
    System.out.println(res);
    return res;
}

public List<Ride> getAcceptedRidesFromDate(String from, String to, Date date){
    List<Ride> res = new ArrayList<Ride>();
        for(Ride r: acceptedRides) {
            if(r.getDate().getTime() == date.getTime() && r.getFrom().equals(from) && r.getTo().equals(to))
                res.add(r);
    }
    System.out.println(res);
    return res;
}
```

- Código refactorizado

```java
public List<Ride> getCreatedRidesFromDate(String from, String to, Date date){
    return getRidesFromDate(createdRides, from, to, date);
}

public List<Ride> getBookedRidesFromDate(String from, String to, Date date){
    return getRidesFromDate(bookedRides, from, to, date);
}

public List<Ride> getAcceptedRidesFromDate(String from, String to, Date date){
    return getRidesFromDate(acceptedRides, from, to, date);
}
private List<Ride> getRidesFromDate(List<Ride> rides, String from, String to, Date date) {
    List<Ride> res = new ArrayList<>();
    for (Ride r : rides) {
        if (r.getDate().getTime() == date.getTime() && r.getFrom().equals(from) && r.getTo().equals(to)) {
            res.add(r);
        }
    }
    System.out.println(res);
    return res;
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

Los tres métodos tenían la misma estructura y la misma funcionalidad sólo cambiaba la lista en la que se iteraba. Se podía hacer un método genérico, con la misma funcionalidad solo pasándole en el parámetro la lista a iterar.

- Miembro que ha realizado la refactorización.

Shirley

2. Refactorización

- Código inicial

```java
public void initializeDB(){

                db.getTransaction().begin();
                try {
                  Calendar today = Calendar.getInstance();

                  int month=today.get(Calendar.MONTH);
                  int year=today.get(Calendar.YEAR);
                  if (month==12) { month=1; year+=1;}

                  Ride ride1 = new Ride("Donostia", "Madrid");
                  Ride ride12 = new Ride("Madrid", "Donostia");
                  ride1.addStop("Alava");
                  ride1.addStop("Logroño");
                  ride1.addStop("Soria");
                  ride12.addStop("Alava");
```

```
                    ride12.addStop("Logroño");
                    ride12.addStop("Soria");
                    ride1.setState("ini");
                    ride12.setState("ini");

                    Ride ride2 = new Ride("Donostia", "Barcelona");
                    Ride ride22 = new Ride("Barcelona", "Donostia");
                    ride2.addStop("Pamplona");
                    ride2.addStop("Zaragoza");
                    ride2.addStop("Tarragona");
                    ride22.addStop("Pamplona");
                    ride22.addStop("Zaragoza");
                    ride22.addStop("Tarragona");
                    ride2.setState("ini");
                    ride22.setState("ini");

                    Ride ride3 = new Ride("Barcelona", "Madrid");
                    Ride ride32 = new Ride("Madrid", "Barcelona");
                    ride3.addStop("Tarragona");
                    ride3.addStop("Teruel");
                    ride3.addStop("Guadalajara");
                    ride32.addStop("Tarragona");
                    ride32.addStop("Teruel");
                    ride32.addStop("Guadalajara");
                    ride3.setState("ini");
                    ride32.setState("ini");
```

- Código refactorizado

```
        private static final String CITY_DONOSTIA = "Donostia";
        private static final String CITY_MADRID = "Madrid";
        private static final String CITY_BARCELONA = "Barcelona";
        private static final String CITY_GRANADA = "Granada";
        private static final String CITY_VALENCIA = "Valencia";
```

…

```
public void createBaseRides() {
                String[][] routes = {
                    {CITY_DONOSTIA, CITY_MADRID, "Alava", "Logroño", "Soria"},
                    {CITY_DONOSTIA, CITY_BARCELONA, "Pamplona", "Zaragoza",
"Tarragona"},
                    {CITY_BARCELONA, CITY_MADRID, "Tarragona", "Teruel", "Guadalajara"},
                    {CITY_BARCELONA, CITY_VALENCIA, "Tarragona", "Castellón"},
                    {CITY_VALENCIA, CITY_MADRID, "Albacete", "Cuenca"},
                    {CITY_GRANADA, CITY_MADRID, "Jaén", "Toledo"},
                    {CITY_GRANADA, CITY_DONOSTIA, "Jaén", "Toledo", CITY_MADRID,
"Soria"}
                };
                for (String[] r : routes) {
                    persistRoundTrip(r[0], r[1], Arrays.copyOfRange(r, 2, r.length));
                }
        }
```

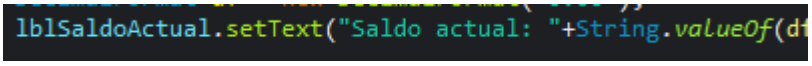
- Descripción del code smell detectado y descripción de la refactorización realizada.

Dentro de la refactorización anteriormente realizada, ya que se repetía muchas veces los Strings de las Ciudades Donostia, Madrid, Barcelona, Valencia y Granada, he hecho variables globales de cada una de ellas.

- Miembro que ha realizado la refactorización.
  Ander

3. Refactorización

- Código inicial



- Código refactorizado

- Descripción del code smell detectado y descripción de la refactorización realizada.

String “Saldo actual: “ duplicado durante toda la clase 3 veces, cambiada por una constante String llamada saldo

- Miembro que ha realizado la refactorización.

Adrian Rodriguez

"Keep unit interfaces small" (capítulo 5).

Limite el número de parámetros por unidad a un máximo de 4.
• Para ello, extraiga los parámetros en objetos.
• Esto mejora la mantenibilidad, ya que mantener un número bajo de parámetros facilita la comprensión y la reutilización de las unidades.

1. Refactorización

- Código inicial

```
public String getRideStopsByCod(String from, String to, Date date, String state, Integer cd) {
    dbManager.open();
    Ride ride = dbManager.getRideStopsByCod(from, to, date, state, cd);
    dbManager.close();

    if(ride == null)
        return "No se encuentra el viaje";
    else if(ride.getStops().size() == 0) {
        return "Sin paradas";
    }
    return ride.stopsToString();
}
```

- Código refactorizado

```
public String getRideStopsByCod(Ride r) {
    dbManager.open();
    Ride ride = dbManager.getRideStopsByCod(r);
    dbManager.close();

    if(ride == null)
        return "No se encuentra el viaje";
    else if(ride.getStops().size() == 0) {
        return "Sin paradas";
    }
    return ride.stopsToString();
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

El método podría utilizar menos parámetros de los que tenía, con un objeto ride es posible obtener todos los parámetros.

- Miembro que ha realizado la refactorización.
  Shirley

2. Refactorización

- Código inicial

```
public String getStopsFromDate(String from, String to, Integer code, Date date, String state) {
        List<Ride> ridesDate = new ArrayList<Ride>();

        if(state.equals("p"))
                ridesDate = this.getCreatedRidesFromDate(from, to, date);
        else if(state.equals("b"))
                ridesDate = this.getBookedRidesFromDate(from, to, date);
        else if(state.equals("a"))
```

```
                ridesDate = this.getAcceptedRidesFromDate(from, to, date);

            boolean isCode = false;
            for(Ride r: ridesDate) {
                if(r.getRideNumber().compareTo(code) == 0) {
                    isCode = true;
                    if(r.getStops().size() > 0)
                        return r.stopsToString();
                }
            }
            if(isCode == false) return "No se encuentra el viaje";
            else return "Sin paradas";
        }
```

- Código refactorizado

```
public String getStopsFromDate(String from, String to, Date date, Integer code) {
        List<Ride> ridesDate = new ArrayList<>();

        // Combinar las listas de rides de diferentes estados
        ridesDate.addAll(getCreatedRidesFromDate(from, to, date));
        ridesDate.addAll(getBookedRidesFromDate(from, to, date));
        ridesDate.addAll(getAcceptedRidesFromDate(from, to, date));

        // Buscar el ride por código
        Ride ride = findRideByCode(ridesDate, code);
        if (ride == null) {
            return "No se encuentra el viaje";
        }
        return ride.getStops().isEmpty() ? "Sin paradas" : ride.stopsToString();
    }
    // Método auxiliar para buscar un ride por su código
    private Ride findRideByCode(List<Ride> rides, Integer code) {
        for (Ride r : rides) {
            if (r.getRideNumber().compareTo(code) == 0) {
                return r;
            }
        }
        return null;
    }
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

El método getStopsFromDate tiene 5 parámetros, lo que excede el límite recomendado de 4. Esto puede dificultar la comprensión de la firma del método, ya que el lector debe procesar múltiples parámetros a la vez y aumentar el riesgo de errores al pasar argumentos en el orden incorrecto.

Refactorización:

- Eliminado state, combinando listas de rides (created, booked, accepted).
- Extraída lógica de búsqueda a findRideByCode (2 parámetros).

- Miembro que ha realizado la refactorización.

Ander Aldanas

3. Refactorización

- Código inicial

```
public Ride getRideStopsByCod(String from, String to, Date date, String state, Integer cd) {
    Ride ride = this.db.find(Ride.class, cd);
    if(ride != null) {
        List<Ride> rides = this.getRides(from, to, date, state);
        if(!rides.contains(ride))
            return null;
    }
    return ride;
}
```

- Código refactorizado

```
public Ride getRideStopsByCod(String state, Ride r) {
    if(r != null) {
        List<Ride> rides = this.getRides(r.getFrom(), r.getTo(), r.getDate(), state);
        if(!rides.contains(r))
            return null;
    }
    return r;
}
```

- Descripción del code smell detectado y descripción de la refactorización realizada.

el metodo utiliza mas parametros de los necesarios teniendo los get y set del propio ride, ahora pasando un propio ride y el estado del propio

- Miembro que ha realizado la refactorización.
    - Adrian Rodriguez