Entregable trabajo Refactorización

Asignatura: Ingeniería del Software II
Profesor: Juan Ignacio Iturrioz Sanchez

Miembros del equipo:
Araujo Galan Maximiliano
Mohammed Yassine Britel

Fecha: 14 de Octubre del 2024

Repositorio de GitHub:
https://github.com/bob239999999/Rides24Complete.git
Proyecto SonarCloud:
https://sonarcloud.io/project/overview?id=bob239999999_Rides24Complete

Introducción :

La Refactorización es el cambio realizado en la estructura interna de algún componente software para hacerlo más fácil de entender y más barato para modificar, sin modificar el comportamiento observable del software. Cambiando su estructura interna sin modificar su comportamiento externo.

"Write short units of code"

Objetivo: Limitar las funciones a un máximo de 15 líneas de código.

Código inicial:

```
public void cancelRide(Ride ride) {
    try {
        db.getTransaction().begin();

        for (Booking booking : ride.getBookings()) {
            if (booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined")) {
                double price = booking.prezioaKalkulatu();
                Traveler traveler = booking.getTraveler();
                double frozenMoney = traveler.getIzoztatutakoDirua();
                traveler.setIzoztatutakoDirua(frozenMoney - price);

                double money = traveler.getMoney();
                traveler.setMoney(money + price);
                db.merge(traveler);
                db.getTransaction().commit();
                addMovement(traveler, "BookDeny", price);
                db.getTransaction().begin();
            }
            booking.setStatus("Rejected");
            db.merge(booking);
        }
        ride.setActive(false);
        db.merge(ride);

        db.getTransaction().commit();
    } catch (Exception e) {
        if (db.getTransaction().isActive()) {
            db.getTransaction().rollback();
        }
        e.printStackTrace();
    }
}
```

Código refactorizado:

```java
public void cancelRide(Ride ride) {
    db.getTransaction().begin();
    rejectRideBookings(ride);
    deactivateRide(ride);
    db.getTransaction().commit();
}

private void rejectRideBookings(Ride ride) {
    for (Booking booking : ride.getBookings()) {
        if (isBookingEligibleForRefund(booking)) {
            processRefund(booking);
        }
        booking.setStatus("Rejected");
        db.merge(booking);
    }
}

private boolean isBookingEligibleForRefund(Booking booking) {
    return booking.getStatus().equals("Accepted") || booking.getStatus().equals("NotDefined");
}

private void processRefund(Booking booking) {
    double price = booking.prezioaKalkulatu();
    Traveler traveler = booking.getTraveler();
    traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() - price);
    traveler.setMoney(traveler.getMoney() + price);
    db.merge(traveler);
    addMovement(traveler, "BookDeny", price);
}

private void deactivateRide(Ride ride) {
    ride.setActive(false);
    db.merge(ride);
}
```

Descripción : En la versión refactorizar se ha dividido en partes más pequeñas. Mejorando la durabilidad y la adhesión con el principio de escribir pequeñas y simples bloques de código.
El código funciona así :
cancelRide(Ride ride) : Se encarga de inicializar la transacción and rejection book y desactivar el viaje.
rejectRide(Ride ride) : Itera sobre todas las reservas del viaje y llama a rejectBook para cada una.
rejectBooking(Booking booking) : Mira si la reserva es apta a un reembolso si lo es llama a process Refund.
isBookingEligibleForRefund : Mira si la reserva es elegible a un reembolso.
processRefund(Booking booking) : Realiza el rembolso.

"Write simple units of code" (Capítulo 3)

Objetivo: Reducir la cantidad de puntos de ramificación (branch points) en cada función, limitándose a un máximo de 4.

Código inicial :

```
  private void rejectRideBookings(Ride ride) {
      for (Booking booking : ride.getBookings()) {
          if (isBookingEligibleForRefund(booking)) {
              processRefund(booking);
          }
          booking.setStatus("Rejected");
          db.merge(booking);
      }
  }
```

Código refactorizado :

```
  private void rejectRideBookings(Ride ride) {
      for (Booking booking : ride.getBookings()) {
          rejectBooking(booking);
      }
  }

  private void rejectBooking(Booking booking) {
      if (isBookingEligibleForRefund(booking)) {
          processRefund(booking);
      }
      booking.setStatus("Rejected");
      db.merge(booking);
  }
```

Descripción : Se ha refactorizado un método rejectRideBookings(Ride ride) que ahora itera sobre las reservas y llama al nuevo método rejectBooking para procesar cada uno.
Se ha dividido la lógica compleja en unidades más simples y fáciles de usar.

"Duplicate code" (Capítulo 4)

Objetivo: Eliminar la duplicación de código.

Código inicial (duplicado en varios lugares):

```
  public Driver getDriver(String erab) {
      TypedQuery<Driver> query = db.createQuery("SELECT d FROM Driver d WHERE d.username = :username", Driver.class);
      query.setParameter("username", erab);
      List<Driver> resultList = query.getResultList();
      if (resultList.isEmpty()) {
          return null;
      } else {
          return resultList.get(0);
      }
  }
```

```
  public Car getKotxeByMatrikula(String matrikula) {
      TypedQuery<Car> query = db.createQuery("SELECT k FROM Car k WHERE k.matrikula = :matrikula", Car.class);
      query.setParameter("matrikula", matrikula);
      List<Car> resultList = query.getResultList();
      if (resultList.isEmpty()) {
          return null;
      } else {
          return resultList.get(0);
      }
  }
```

Código refactorizado:

```
public <T> T getEntityByField(Class<T> entityClass, String fieldName, String fieldValue) {
    TypedQuery<T> query = db.createQuery(
        "SELECT e FROM " + entityClass.getSimpleName() + " e WHERE e." + fieldName + " = :" + fieldName, entityClass);
    query.setParameter(fieldName, fieldValue);
    return query.getSingleResult();
}

public Car getKotxeByMatrikula(String matrikula) {
    return getEntityByField(Car.class, "matrikula", matrikula);
}

public Driver getDriverByUsername(String username) {
    return getEntityByField(Driver.class, "username", username);
}
```

Descripción : Este método lo que hace es sacar el tipo por la clase que se especifica, nombre y el valor. Reduciendo la duplicación del código centralizando la lógica de la función y tratandolo en el método getEntityByField().

"Keep unit interfaces small" (Capítulo 5)

Objetivo: Limitar el número de parámetros por función a un máximo de 4.

```
public void deleteCar(Car car) {
    try {
        db.getTransaction().begin();

        Car managedCar = db.merge(car);
        db.remove(managedCar);
        Driver driver = managedCar.getDriver();
        driver.removeCar(managedCar);
        db.merge(driver);

        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
    }
}
```

Código refactorizado:

```java
public void deleteCar(Car car) {
    db.getTransaction().begin();
    try {
        Car managedCar = db.merge(car);
        removeCarFromDriver(managedCar);
        db.remove(managedCar);
        db.getTransaction().commit();
    } catch (Exception e) {
        e.printStackTrace();
        if (db.getTransaction().isActive()) {
            db.getTransaction().rollback();
        }
    }
}

private void removeCarFromDriver(Car car) {
    Driver driver = car.getDriver();
    if (driver != null) {
        driver.removeCar(car);
        db.merge(driver);
    }
}
```

Descripción : Buscamos métodos anidados ya que todos los métodos no pasan de 4 parámetros.
El método deleteCar() utiliza otras tres variables dentro del método para facilitar la eliminación y la gestión de la relación entre el coche y el conductor.
Para reducir estas variables hemos decidido partiendo la función en dos unidades reduciendo el número de parámetros que se tratan, realizando el proceso de eliminación del coche fuera de la función original.

Maximiliano Araujo Galán

1. "Write short units of code" (limitar a 15 líneas)
   Código Inicial:

```java
public void initializeDB() {
        db.getTransaction().begin();
        try {
                Driver driver1 = new Driver("Urtzi", "123");
                driver1.setMoney(15);
                driver1.setBalorazioa(14);
                driver1.setBalkop(3);
                Driver driver2 = new Driver("Zuri", "456");
                driver2.setBalorazioa(10);
                driver2.setBalkop(3);
                db.persist(driver1);
                db.persist(driver2);
                Traveler traveler1 = new Traveler("Unax", "789");
                traveler1.setIzoztatutakoDirua(68);
                traveler1.setMoney(100);
                traveler1.setBalorazioa(14);
                traveler1.setBalkop(4);
                Traveler traveler2 = new Traveler("Luken", "abc");
                traveler2.setBalorazioa(4);
```

```
                traveler2.setBalkop(3);
                db.persist(traveler1);
                db.persist(traveler2);
                Calendar cal = Calendar.getInstance();
                cal.set(2024, Calendar.MAY, 20);
                Date date1 = UtilDate.trim(cal.getTime());
                cal.set(2024, Calendar.MAY, 30);
                Date date2 = UtilDate.trim(cal.getTime());
                cal.set(2024, Calendar.MAY, 10);
                Date date3 = UtilDate.trim(cal.getTime());
                cal.set(2024, Calendar.APRIL, 20);
                Date date4 = UtilDate.trim(cal.getTime());
                driver1.addRide("Donostia", DESTINATION_MADRID, date2, 5, 20); //ride1
                driver1.addRide("Irun", "Donostia", date2, 5, 2); //ride2
                driver1.addRide(DESTINATION_MADRID, "Donostia", date3, 5, 5); //ride3
                driver1.addRide("Barcelona", DESTINATION_MADRID, date4, 0, 10); //ride4
                driver2.addRide("Donostia", "Hondarribi", date1, 5, 3); //ride5
                Ride ride1 = driver1.getCreatedRides().get(0);
                Ride ride2 = driver1.getCreatedRides().get(1);
                Ride ride3 = driver1.getCreatedRides().get(2);
                Ride ride4 = driver1.getCreatedRides().get(3);
                Ride ride5 = driver2.getCreatedRides().get(0);
                Booking book1 = new Booking(ride4, traveler1, 2);
                Booking book2 = new Booking(ride1, traveler1, 2);
                Booking book4 = new Booking(ride3, traveler1, 1);
                Booking book3 = new Booking(ride2, traveler2, 2);
                Booking book5 = new Booking(ride5, traveler1, 1);
                book1.setStatus("Accepted");
                book2.setStatus("Rejected");
                book3.setStatus("Accepted");
                book4.setStatus("Accepted");
                book5.setStatus("Accepted");
                db.persist(book1);
                db.persist(book2);
                db.persist(book3);
                db.persist(book4);
                db.persist(book5);
                Movement m1 = new Movement(traveler1, "BookFreeze", 20);
                Movement m2 = new Movement(traveler1, "BookFreeze", 40);
                Movement m3 = new Movement(traveler1, "BookFreeze", 5);
                Movement m4 = new Movement(traveler2, "BookFreeze", 4);
                Movement m5 = new Movement(traveler1, "BookFreeze", 3);
                Movement m6 = new Movement(driver1, "Deposit", 15);
                Movement m7 = new Movement(traveler1, "Deposit", 168);

                db.persist(m6);
                db.persist(m7);
                db.persist(m1);
                db.persist(m2);
                db.persist(m3);
                db.persist(m4);
                db.persist(m5);

                traveler1.addBookedRide(book1);
                traveler1.addBookedRide(book2);
                traveler2.addBookedRide(book3);
                traveler1.addBookedRide(book4);
                traveler1.addBookedRide(book5);
                db.merge(traveler1);
                Car c1 = new Car("1234ABC", "Renault", 5);
                Car c2 = new Car("5678DEF", "Citroen", 3);
                Car c3 = new Car("9101GHI", "Audi", 5);
                driver1.addCar(c1);
                driver1.addCar(c2);
                driver2.addCar(c3);
                db.persist(c1);
```

```
                db.persist(c2);
                db.persist(c3);
                //Admin a1 = new Admin("Jon", "111");
                //db.persist(a1);
                Discount dis = new Discount("Uda24", 0.2, true);
                db.persist(dis);
                db.getTransaction().commit();
                System.out.println("Db initialized");
            } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
            }
        }
```

Código refactorizado:

```
public void initializeDB() {
        db.getTransaction().begin();
        try {
            initializeUsers();
            initializeRidesAndBookings();
            initializeMovementsAndCars();
            initializeDiscount();

            db.getTransaction().commit();
            System.out.println("Db initialized");
        } catch (Exception e) {
            e.printStackTrace();
            db.getTransaction().rollback();
        }
    }
    private void initializeUsers() {
        Driver driver1 = new Driver("Urtzi", "123");
        driver1.setMoney(15);
        driver1.setBalorazioa(14);
        driver1.setBalkop(3);
        Driver driver2 = new Driver("Zuri", "456");
        driver2.setBalorazioa(10);
        driver2.setBalkop(3);
        db.persist(driver1);
        db.persist(driver2);
        Traveler traveler1 = new Traveler("Unax", "789");
        traveler1.setIzoztatutakoDirua(68);
        traveler1.setMoney(100);
        traveler1.setBalorazioa(14);
        traveler1.setBalkop(4);
        Traveler traveler2 = new Traveler("Luken", "abc");
        traveler2.setBalorazioa(4);
        traveler2.setBalkop(3);
        db.persist(traveler1);
        db.persist(traveler2);
    }
    private void initializeRidesAndBookings() {
        Driver driver1 = db.find(Driver.class, "Urtzi");
        Driver driver2 = db.find(Driver.class, "Zuri");
        Traveler traveler1 = db.find(Traveler.class, "Unax");
        Traveler traveler2 = db.find(Traveler.class, "Luken");
        Calendar cal = Calendar.getInstance();
        cal.set(2024, Calendar.MAY, 20);
        Date date1 = UtilDate.trim(cal.getTime());
```

```
            cal.set(2024, Calendar.MAY, 30);
            Date date2 = UtilDate.trim(cal.getTime());
            cal.set(2024, Calendar.MAY, 10);
            Date date3 = UtilDate.trim(cal.getTime());
            cal.set(2024, Calendar.APRIL, 20);
            Date date4 = UtilDate.trim(cal.getTime());
            driver1.addRide("Donostia", "Madrid", date2, 5, 20); //ride1
            driver1.addRide("Irun", "Donostia", date2, 5, 2); //ride2
            driver1.addRide("Madrid", "Donostia", date3, 5, 5); //ride3
            driver1.addRide("Barcelona", "Madrid", date4, 0, 10); //ride4
            driver2.addRide("Donostia", "Hondarribi", date1, 5, 3); //ride5
            Ride ride1 = driver1.getCreatedRides().get(0);
            Ride ride2 = driver1.getCreatedRides().get(1);
            Ride ride3 = driver1.getCreatedRides().get(2);
            Ride ride4 = driver1.getCreatedRides().get(3);
            Ride ride5 = driver2.getCreatedRides().get(0);
            Booking book1 = new Booking(ride4, traveler1, 2);
            Booking book2 = new Booking(ride1, traveler1, 2);
            Booking book4 = new Booking(ride3, traveler1, 1);
            Booking book3 = new Booking(ride2, traveler2, 2);
            Booking book5 = new Booking(ride5, traveler1, 1);
            book1.setStatus("Accepted");
            book2.setStatus("Rejected");
            book3.setStatus("Accepted");
            book4.setStatus("Accepted");
            book5.setStatus("Accepted");
            db.persist(book1);
            db.persist(book2);
            db.persist(book3);
            db.persist(book4);
            db.persist(book5);
            traveler1.addBookedRide(book1);
            traveler1.addBookedRide(book2);
            traveler2.addBookedRide(book3);
            traveler1.addBookedRide(book4);
            traveler1.addBookedRide(book5);
            db.merge(traveler1);
        }
        private void initializeMovementsAndCars() {
            Traveler traveler1 = db.find(Traveler.class, "Unax");
            Traveler traveler2 = db.find(Traveler.class, "Luken");
            Driver driver1 = db.find(Driver.class, "Urtzi");
            Driver driver2 = db.find(Driver.class, "Zuri");
            Movement m1 = new Movement(traveler1, "BookFreeze", 20);
            Movement m2 = new Movement(traveler1, "BookFreeze", 40);
            Movement m3 = new Movement(traveler1, "BookFreeze", 5);
            Movement m4 = new Movement(traveler2, "BookFreeze", 4);
            Movement m5 = new Movement(traveler1, "BookFreeze", 3);
            Movement m6 = new Movement(driver1, "Deposit", 15);
            Movement m7 = new Movement(traveler1, "Deposit", 168);

            db.persist(m1);
            db.persist(m2);
            db.persist(m3);
            db.persist(m4);
            db.persist(m5);
            db.persist(m6);
            db.persist(m7);
```

```
        Car c1 = new Car("1234ABC", "Renault", 5);
        Car c2 = new Car("5678DEF", "Citroen", 3);
        Car c3 = new Car("9101GHI", "Audi", 5);
        driver1.addCar(c1);
        driver1.addCar(c2);
        driver2.addCar(c3);
        db.persist(c1);
        db.persist(c2);
        db.persist(c3);
    }
    private void initializeDiscount() {
        Discount dis = new Discount("Uda24", 0.2, true);
        db.persist(dis);
    }
```

Descripción: El método `initializeDB()` era demasiado largo (más de 100 líneas), lo que lo hacía difícil de leer y mantener. Se refactorizó extrayendo partes del código en métodos más pequeños y específicos. Esto mejora la legibilidad, facilita el mantenimiento y permite una mejor organización del código.

2. "Write simple units of code" (limitar puntos de ramificación a 4)

   Código Inicial:

```
{
    public boolean bookRide(String username, Ride ride, int seats, double desk)
try {
                db.getTransaction().begin();
                Traveler traveler = getTraveler(username);
                if (traveler == null) {
                        return false;
                }
                if (ride.getnPlaces() < seats) {
                        return false;
                }
                double ridePriceDesk = (ride.getPrice() - desk) * seats;
                double availableBalance = traveler.getMoney();
                if (availableBalance < ridePriceDesk) {
                        return false;
                }
                Booking booking = new Booking(ride, traveler, seats);
                booking.setTraveler(traveler);
                booking.setDeskontua(desk);
                db.persist(booking);
                ride.setnPlaces(ride.getnPlaces() - seats);
                traveler.addBookedRide(booking);
                traveler.setMoney(availableBalance - ridePriceDesk);
                traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
                db.merge(ride);
                db.merge(traveler);
                db.getTransaction().commit();
                return true;
        } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
                return false;
        }
```

```
        }
```

Código refactorizado:

```
public boolean bookRide(String username, Ride ride, int seats, double desk) {
        try {
            db.getTransaction().begin();
            if (!isValidBooking(username, ride, seats, desk)) {
                db.getTransaction().rollback();
                return false;
            }
            createBooking(username, ride, seats, desk);
            db.getTransaction().commit();
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            db.getTransaction().rollback();
            return false;
        }
    }
    private boolean isValidBooking(String username, Ride ride, int seats, double desk) {
        Traveler traveler = getTraveler(username);
        if (traveler == null || ride.getnPlaces() < seats) {
            return false;
        }
        double ridePriceDesk = (ride.getPrice() - desk) * seats;
        return traveler.getMoney() >= ridePriceDesk;
    }
    private void createBooking(String username, Ride ride, int seats, double desk) {
        Traveler traveler = getTraveler(username);
        double ridePriceDesk = (ride.getPrice() - desk) * seats;

        Booking booking = new Booking(ride, traveler, seats);
        booking.setDeskontua(desk);
        db.persist(booking);
        ride.setnPlaces(ride.getnPlaces() - seats);
        traveler.addBookedRide(booking);
        traveler.setMoney(traveler.getMoney() - ridePriceDesk);
        traveler.setIzoztatutakoDirua(traveler.getIzoztatutakoDirua() + ridePriceDesk);
        db.merge(ride);
        db.merge(traveler);
    }
```

Descripción: El método `bookRide()` original tenía múltiples puntos de decisión y varias responsabilidades. Se refactorizó dividiéndolo en métodos más pequeños y específicos: `isValidBooking()` para verificar las condiciones y `createBooking()` para realizar la reserva.

3. "Duplicate code" (código duplicado)

   Codigo Inicial:

```
public void updateTraveler(Traveler traveler) {
                try {
                        db.getTransaction().begin();
                        db.merge(traveler);
```

```
                db.getTransaction().commit();
            } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
            }
        }

public void updateDriver(Driver driver) {
try {
                db.getTransaction().begin();
                db.merge(driver);
                db.getTransaction().commit();
            } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
            }
        }
        public void updateUser(User user) {
            try {
                db.getTransaction().begin();
                db.merge(user);
                db.getTransaction().commit();
            } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
            }
        }
```

Código refactorizado:

```
        private <T> void updateEntity(T entity) {
            try {
                db.getTransaction().begin();
                db.merge(entity);
                db.getTransaction().commit();
            } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
            }
        }
        public void updateTraveler(Traveler traveler) {
            updateEntity(traveler);
        }
        public void updateDriver(Driver driver) {
            updateEntity(driver);
        }
        public void updateUser(User user) {
            updateEntity(user);
        }
```

Descripción:

Los métodos `updateTraveler()`, `updateDriver()` y `updateUser()` contenían código duplicado para actualizar entidades en la base de datos. Se creó un método genérico `updateEntity()` que puede ser utilizado para actualizar cualquier tipo de entidad. Esto

elimina la duplicación de código, mejora la reutilización y facilita la adición de métodos de actualización similares en el futuro.

4. "Keep unit interfaces small" (limitar parámetros a 4)

   Código Inicial:

```java
public boolean erreklamazioaBidali(String nor, String nori, Date gaur, Booking booking, String textua,
                boolean aurk) {
        try {
                db.getTransaction().begin();
                Complaint erreklamazioa = new Complaint(nor, nori, gaur, booking, textua, aurk);
                db.persist(erreklamazioa);
                db.getTransaction().commit();
                return true;
        } catch (Exception e) {
                e.printStackTrace();
                db.getTransaction().rollback();
                return false;
        }
}
```

   Código refactorizado:

```java
package dataAccess;

import java.util.Date;

public class ErreklamazioaBidaliParameter {
    public String nor;
    public String nori;
    public Date gaur;
    public Booking booking;
    public String textua;
    public boolean aurk;

    public ErreklamazioaBidaliParameter(String nor, String nori, Date gaur, Booking booking, String textua,
            boolean aurk) {
        this.nor = nor;
        this.nori = nori;
        this.gaur = gaur;
        this.booking = booking;
        this.textua = textua;
        this.aurk = aurk;
    }
}
```

```java
public boolean erreklamazioaBidali(ErreklamazioaBidaliParameter parameterObject) {
    try {
        db.getTransaction().begin();

        Complaint erreklamazioa = new Complaint(parameterObject.nor, parameterObject.nori,
                parameterObject.gaur, parameterObject.booking, parameterObject.textua, parameterObject.aurk);
        db.persist(erreklamazioa);
        db.getTransaction().commit();
        return true;
    } catch (Exception e) {
        e.printStackTrace();
        db.getTransaction().rollback();
        return false;
    }
}
```

Descripción:

Esta refactorización aplica el principio "Keep unit interfaces small" al reducir el número de parámetros de 6 a 1. Se ha creado una nueva clase `ErreklamazioaBidaliParameter` que encapsula todos los parámetros.