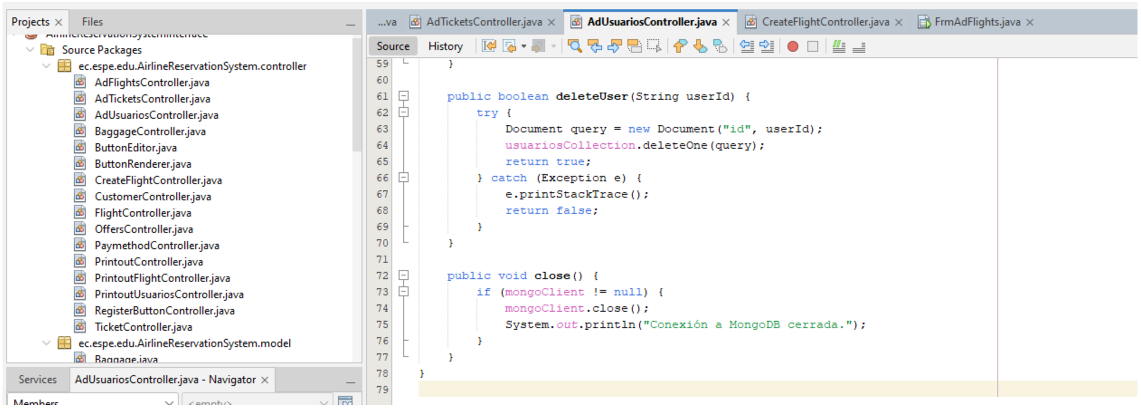
HW23 – SOLID Principles

1. Liskov Substitution Principle



```java
    }

    public boolean deleteUser(String userId) {
        try {
            Document query = new Document("id", userId);
            usuariosCollection.deleteOne(query);
            return true;
        } catch (Exception e) {
            e.printStackTrace();
            return false;
        }
    }

    public void close() {
        if (mongoClient != null) {
            mongoClient.close();
            System.out.println("Conexión a MongoDB cerrada.");
        }
    }
}
```

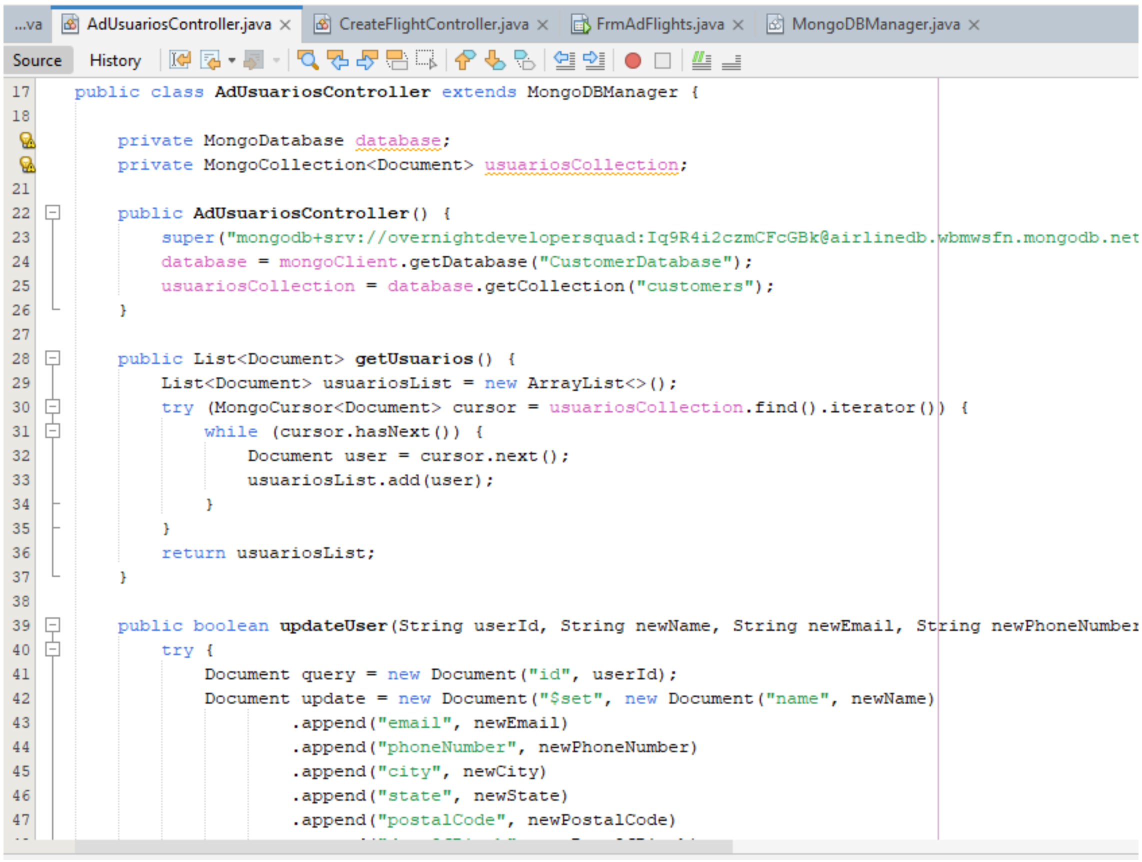
Explanation:

In this class and in two others we repeat code to close the connection to the database which does not make it reusable.

Possible solution:

Create an abstract class with the method that is repeated and implement it in each of the classes adding the code that you needed applying the Liskov Substitution principle.

2. Liskov Substitution Principle



```java
public class AdUsuariosController extends MongoDBManager {

    private MongoDatabase database;
    private MongoCollection<Document> usuariosCollection;

    public AdUsuariosController() {
        super("mongodb+srv://overnightdevelopersquad:Iq9R4i2czmCFcGBk@airlinedb.wbmwsfn.mongodb.net
        database = mongoClient.getDatabase("CustomerDatabase");
        usuariosCollection = database.getCollection("customers");
    }

    public List<Document> getUsuarios() {
        List<Document> usuariosList = new ArrayList<>();
        try (MongoCursor<Document> cursor = usuariosCollection.find().iterator()) {
            while (cursor.hasNext()) {
                Document user = cursor.next();
                usuariosList.add(user);
            }
        }
        return usuariosList;
    }

    public boolean updateUser(String userId, String newName, String newEmail, String newPhoneNumber
        try {
            Document query = new Document("id", userId);
            Document update = new Document("$set", new Document("name", newName)
                    .append("email", newEmail)
                    .append("phoneNumber", newPhoneNumber)
                    .append("city", newCity)
                    .append("state", newState)
                    .append("postalCode", newPostalCode)
```

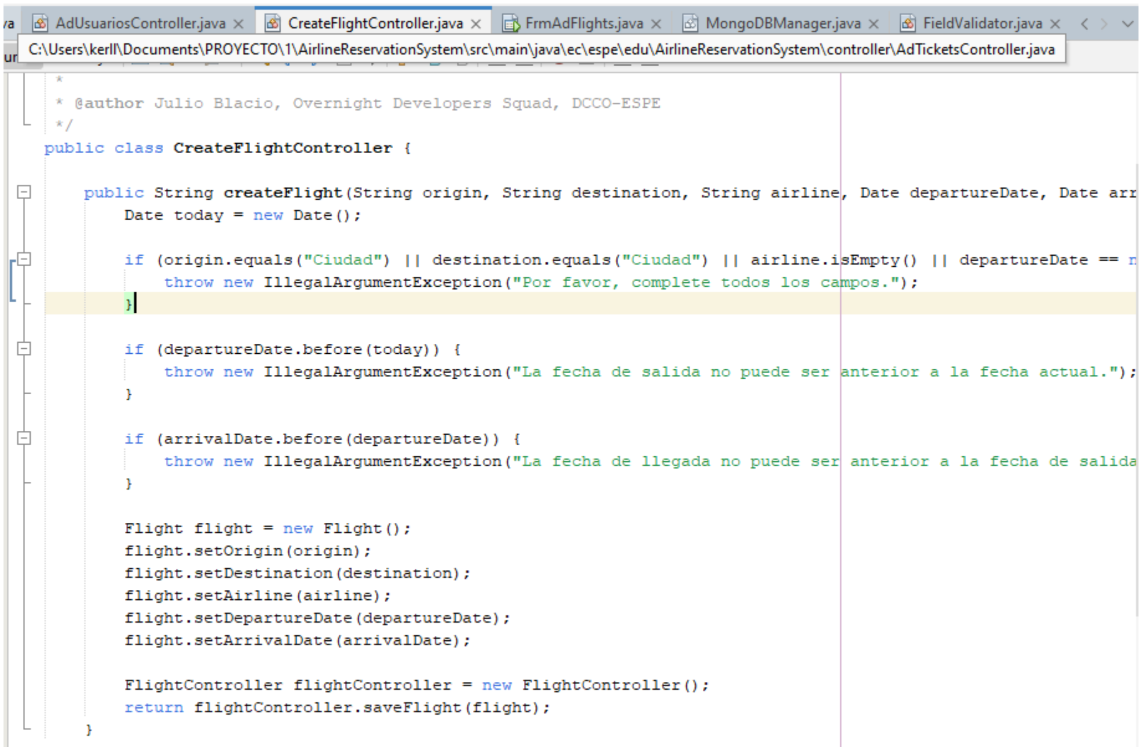
Explanation:

In this class, and like the other two previous ones, code was repeated that in the end could not be reused.

Possible solution:

In the class I created previously, the methods for updating and deleting documents could be increased, which would help to better structure the classes.

3. Single Responsability Principle



AdUsuariosController.java
CreateFlightController.java
FrmAdFlights.java
MongoDBManager.java
FieldValidator.java
C:\Users\kerll\Documents\PROYECTO\1\AirlineReservationSystem\src\main\java\ec\espe\edu\AirlineReservationSystem\controller\AdTicketsController.java

```
 *
 * @author Julio Blacio, Overnight Developers Squad, DCCO-ESPE
 */
public class CreateFlightController {

    public String createFlight(String origin, String destination, String airline, Date departureDate, Date arr
        Date today = new Date();

        if (origin.equals("Ciudad") || destination.equals("Ciudad") || airline.isEmpty() || departureDate == n
            throw new IllegalArgumentException("Por favor, complete todos los campos.");
        }

        if (departureDate.before(today)) {
            throw new IllegalArgumentException("La fecha de salida no puede ser anterior a la fecha actual.");
        }

        if (arrivalDate.before(departureDate)) {
            throw new IllegalArgumentException("La fecha de llegada no puede ser anterior a la fecha de salida
        }

        Flight flight = new Flight();
        flight.setOrigin(origin);
        flight.setDestination(destination);
        flight.setAirline(airline);
        flight.setDepartureDate(departureDate);
        flight.setArrivalDate(arrivalDate);

        FlightController flightController = new FlightController();
        return flightController.saveFlight(flight);
    }
```

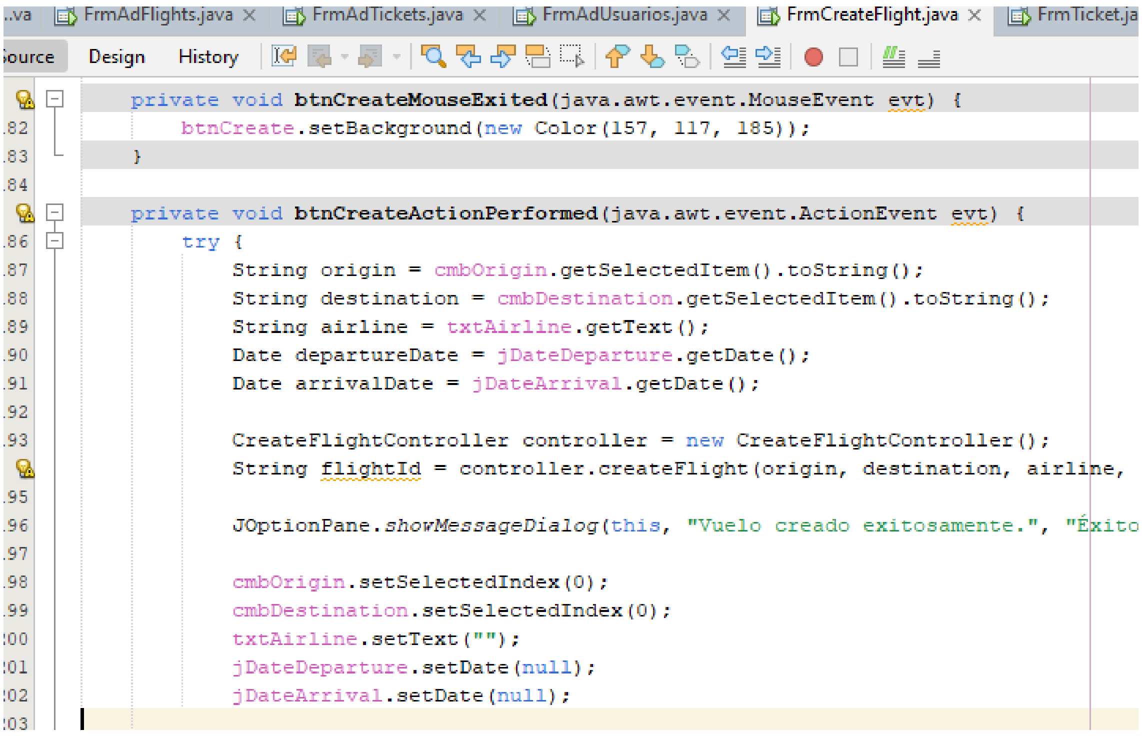
Explanation:

This code violates the principle of single responsibility since this function should only create the flight but not validate its fields.

Possible solution:

Move the field validation code to a class in utils to only call it in this class and thus keep this function clean.

4. Single Responsability Principle



```java
    private void btnCreateMouseExited(java.awt.event.MouseEvent evt) {
        btnCreate.setBackground(new Color(157, 117, 185));
    }

    private void btnCreateActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            String origin = cmbOrigin.getSelectedItem().toString();
            String destination = cmbDestination.getSelectedItem().toString();
            String airline = txtAirline.getText();
            Date departureDate = jDateDeparture.getDate();
            Date arrivalDate = jDateArrival.getDate();

            CreateFlightController controller = new CreateFlightController();
            String flightId = controller.createFlight(origin, destination, airline,

            JOptionPane.showMessageDialog(this, "Vuelo creado exitosamente.", "Éxito

            cmbOrigin.setSelectedIndex(0);
            cmbDestination.setSelectedIndex(0);
            txtAirline.setText("");
            jDateDeparture.setDate(null);
            jDateArrival.setDate(null);
```

Explanation:

In this code we give the button more actions than it should have and we overload it.

Possible solution:

Create a class with a function that clears the fields and only makes a call to the button.