LAPORAN PRAKTIKUM PEMROGRAMAN
TES DRIVEN DEVELOPMENT JUNIT

Oleh:

Ihsan Fauzi 241524048

PROGRAM STUDI D4-TEKNIK INFORMATIKA
JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA
POLITEKNIK NEGERI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI i
BAB I PENDAHULUAN 1
BAB II JAWABAN TDD 2
BAB III ANALISIS TDD 3
BAB IV KESIMPULAN 13

BAB I
PENDAHULUAN

Test-Driven Development (TDD) adalah metode pengembangan perangkat lunak yang menekankan siklus pengembangan yang dimulai dengan menulis tes gagal sebelum menulis kode baru atau bisa dikenal dengan istilah tes->coding->refactoring repeat. Pendekatan ini bertujuan untuk menghasilkan desain perangkat lunak yang lebih clean, mengurangi bug, dan mempercepat pengembangan.

JUnit adalah framework testing unit untuk bahasa Java yang mendukung pendekatan TDD. Dengan JUnit, kita bisa dengan mudah menulis dan menjalankan tes otomatis untuk memverifikasi fungsionalitas dari potongan kode yang dibuat. JUnit 5 memperkenalkan banyak fitur modern yang memudahkan penerapan TDD, seperti nested tests, parameterized tests, dan lainnya.

BAB II
JAWABAN TDD

1. Tujuan menggunakan Junit

   Tujuan utama menggunakan JUnit adalah untuk menguji kelayakan sebuah fungsi dalam aplikasi. Dengan menulis tes menggunakan JUnit, kita dapat memastikan bahwa setiap fungsi bekerja sesuai spesifikasi yang diinginkan sebelum digunakan dalam skala yang lebih luas.

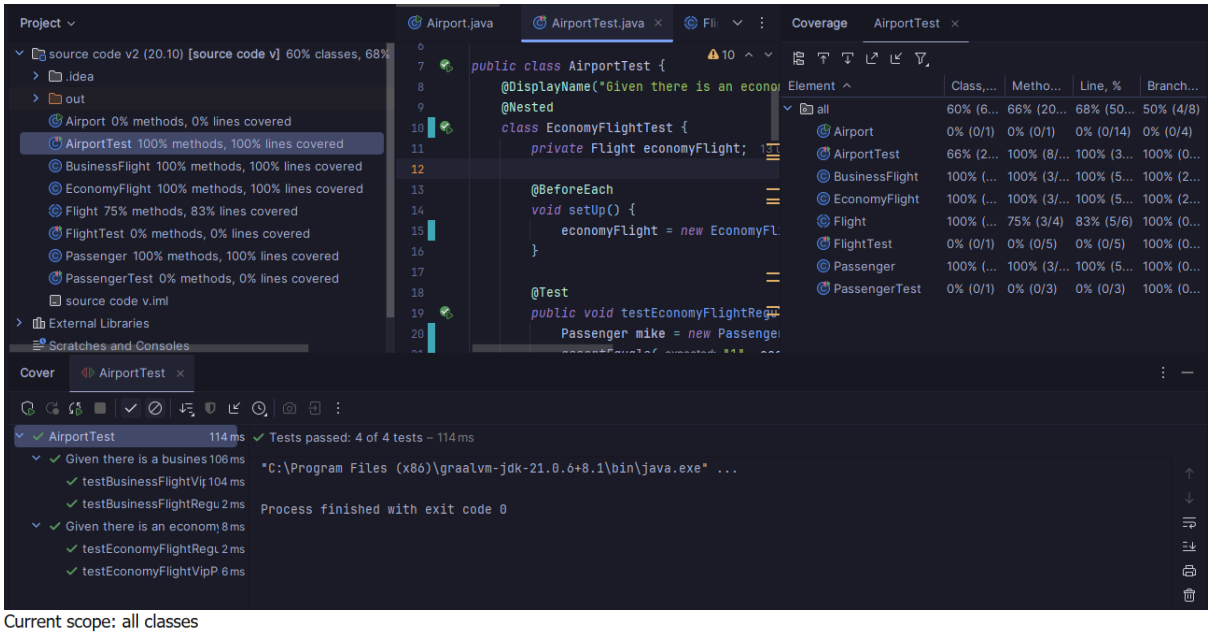
2. Penerapan Utama: Test, Code, Refactor, Repeat

   Dalam TDD, siklus pengembangan mengikuti langkah:

   a. Menulis tes yang gagal
   b. Menulis kode sekecil mungkin untuk melewati tes tersebut
   c. Refactor kode agar lebih bersih
   d. Repeat

   Sebelum menaikkan suatu fungsi menjadi bagian dari logika aplikasi yang lebih besar, kita harus memastikan melalui tes bahwa fungsi tersebut telah layak dan memenuhi syarat.

3. Jawaban



Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	70% (7/10)	61.8% (21/34)	50% (4/8)	66.2% (51/77)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
<empty package name>	70% (7/10)	61.8% (21/34)	50% (4/8)	66.2% (51/77)

generated on 2025-04-28 15:58

Dari implementasi pada kelas AirportTest, semua tes berhasil sukses. Ini menunjukkan bahwa fungsi-fungsi dalam program Flight atau Airport telah berfungsi sesuai business rule yang diinginkan.

.

BAB III
ANALISIS TDD

1. Listing 20.1 - Passenger class

```
public class Passenger {
    private String name;
    private boolean vip;

    public Passenger(String name, boolean vip) {
        this.name = name;
        this.vip = vip;
    }

    public String getName() {
        return name;
    }

    public boolean isVip() {
        return vip;
    }
}
```

Apa yang dilakukan: Membuat entitas Passenger dengan nama dan status VIP.

JUnitnya di mana: Belum ada JUnit di sini, ini adalah class model.

Apa yang dites: Nantinya yang akan dites adalah status VIP dan nama Passenger.

Test sukses/failure: (Akan diujikan pada AirportTest)

2. Listing 20.2 - Flight class

```java
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;

public class Flight {
    private String id;
    private List<Passenger> passengers = new ArrayList<Passenger>();
    private String flightType;

    public Flight(String id, String flightType) {
        this.id = id;
        this.flightType = flightType;
    }
    public String getId() {
        return id;
    }

    public List<Passenger> getPassengersList() {
        return Collections.unmodifiableList(passengers);
    }
    public String getFlightType() {
        return flightType;
    }

    public boolean addPassenger(Passenger passenger) {
        switch (flightType) {
            case "Economy":
                return passengers.add(passenger);
            case "Business":
                if (passenger.isVip()) {
                    return passengers.add(passenger);
                }
                return false;
            default:
                throw new RuntimeException("Unknown type: " + flightType);
        }
    }
    public boolean removePassenger(Passenger passenger) {
        switch (flightType) {
            case "Economy":
                if (!passenger.isVip()) {
                    return passengers.remove(passenger);
                }
                return false;
            case "Business":
                return false;
            default:
                throw new RuntimeException("Unknown type: " + flightType);
        }
    }
}
```

Apa yang dilakukan: Mengelola penerbangan dan daftar penumpang berdasarkan tipe flight.

JUnitnya di mana: Belum ada JUnit di sini, ini adalah class bisnis utama.

Apa yang dites: Tambah dan hapus penumpang berdasarkan aturan tipe flight.

Test sukses/failure: Akan diuji pada AirportTest.

3. Listing 20.3 - Airport class

```
public class Airport {
    public static void main(String[] args) {
        Flight economyFlight = new Flight("1", "Economy");
        Flight businessFlight = new Flight("2", "Business");
        Passenger james = new Passenger("James", true);
        Passenger mike = new Passenger("Mike", false);
        businessFlight.addPassenger(james);
        businessFlight.removePassenger(james);
        businessFlight.addPassenger(mike);
        economyFlight.addPassenger(mike);
        System.out.println("Business flight passengers list:");
        for (Passenger passenger: businessFlight.getPassengersList()) {
            System.out.println(passenger.getName());
        }
        System.out.println("Economy flight passengers list:");
        for (Passenger passenger: economyFlight.getPassengersList()) {
            System.out.println(passenger.getName());
        }
    }
}
```

Apa yang dilakukan: Menyimulasikan eksekusi program dengan Flight dan Passenger.

JUnitnya di mana: Tidak ada, ini hanya main method.

Apa yang dites: Secara manual, bukan melalui JUnit.

Test sukses/failure: Tidak berbasis JUnit.

4. Listing 20.4 - Dependencies Maven

```
<dependencies>
   <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.6.0</version>
      <scope>test</scope>
   </dependency>
   <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-engine</artifactId>
      <version>5.6.0</version>
      <scope>test</scope>
   </dependency>
 </dependencies>
```

Apa yang dilakukan: Menambahkan dependency JUnit 5 di Maven pom.xml.

JUnitnya di mana: Menyiapkan project untuk menggunakan JUnit.

Apa yang dites: Menyediakan dasar untuk testing.

5. Listing 20.5 - AirportTest (EconomyFlightTest)

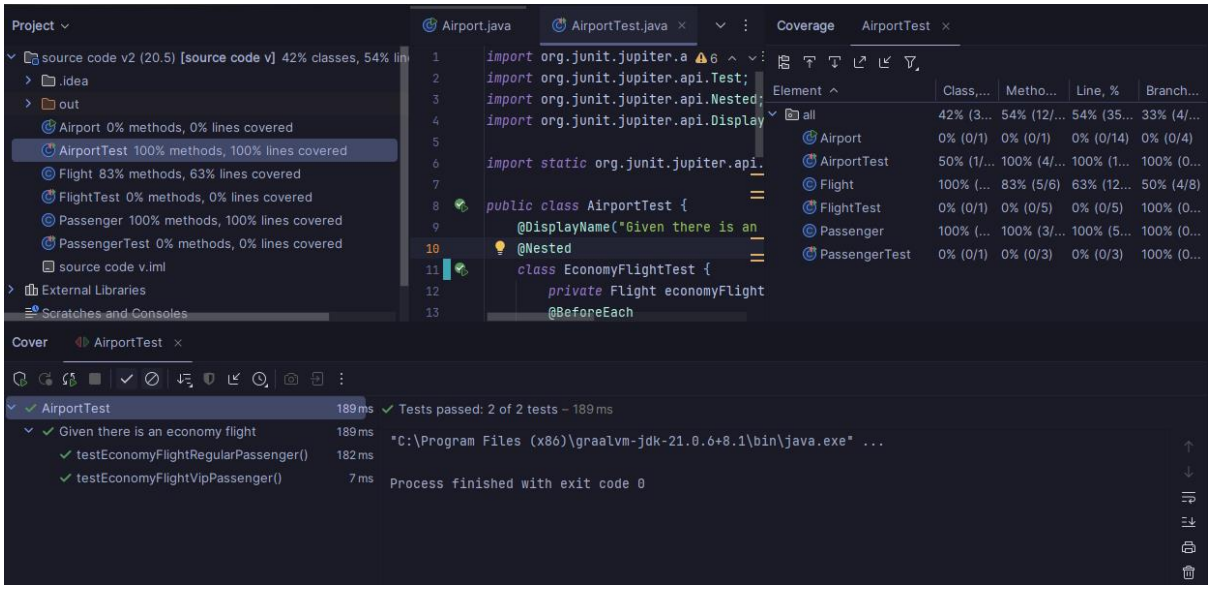
```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.DisplayName;

import static org.junit.jupiter.api.Assertions.*;

public class AirportTest {
    @DisplayName("Given there is an economy flight")
    @Nested
    class EconomyFlightTest {
        private Flight economyFlight;
        @BeforeEach
        void setUp() {
            economyFlight = new Flight("1", "Economy");
        }
        @Test
        public void testEconomyFlightRegularPassenger() {
            Passenger mike = new Passenger("Mike", false);
            assertEquals("1", economyFlight.getId());
            assertEquals(true, economyFlight.addPassenger(mike));
            assertEquals(1, economyFlight.getPassengersList().size());
            assertEquals("Mike",
                    economyFlight.getPassengersList().get(0).getName());
            assertEquals(true, economyFlight.removePassenger(mike));
            assertEquals(0, economyFlight.getPassengersList().size());
        }
        @Test
        public void testEconomyFlightVipPassenger() {
            Passenger james = new Passenger("James", true);
            assertEquals("1", economyFlight.getId());
            assertEquals(true, economyFlight.addPassenger(james));
            assertEquals(1, economyFlight.getPassengersList().size());
            assertEquals("James",
                    economyFlight.getPassengersList().get(0).getName());
            assertEquals(false, economyFlight.removePassenger(james));
            assertEquals(1, economyFlight.getPassengersList().size());
        }
    }
}
```

Apa yang dilakukan: Mengetes EconomyFlight dengan Passenger reguler dan VIP.

JUnitnya di mana: Menggunakan @Nested, @Test, dan assertEquals.

Apa yang dites: Menambahkan dan menghapus penumpang di EconomyFlight.

Test sukses/failure: Semua tes berhasil sukses.


Project
source code v2 (20.5) [source code v] 42% classes, 54% lin
.idea
out
Airport 0% methods, 0% lines covered
AirportTest 100% methods, 100% lines covered
Flight 83% methods, 63% lines covered
FlightTest 0% methods, 0% lines covered
Passenger 100% methods, 100% lines covered
PassengerTest 0% methods, 0% lines covered
source code v.iml
External Libraries
Scratches and Consoles
Airport.java
AirportTest.java
import org.junit.jupiter.a
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Display
import static org.junit.jupiter.api.
public class AirportTest {
@DisplayName("Given there is an
@Nested
class EconomyFlightTest {
private Flight economyFlight
@BeforeEach
Coverage
AirportTest
Element
Class,...
Metho...
Line, %
Branch...
all 42% (3... 54% (12/... 54% (35... 33% (4/...
Airport 0% (0/1) 0% (0/1) 0% (0/14) 0% (0/4)
AirportTest 50% (1/... 100% (4/... 100% (1... 100% (0...
Flight 100% (... 83% (5/6) 63% (12... 50% (4/8)
FlightTest 0% (0/1) 0% (0/5) 0% (0/5) 100% (0...
Passenger 100% (... 100% (3/... 100% (5... 100% (0...
PassengerTest 0% (0/1) 0% (0/3) 0% (0/3) 100% (0...
Cover
AirportTest
AirportTest 189 ms
Given there is an economy flight 189 ms
testEconomyFlightRegularPassenger() 182 ms
testEconomyFlightVipPassenger() 7 ms
Tests passed: 2 of 2 tests – 189 ms
"C:\Program Files (x86)\graalvm-jdk-21.0.6+8.1\bin\java.exe" ...
Process finished with exit code 0

6. Listing 20.6 - AirportTest (BusinessFlightTest)

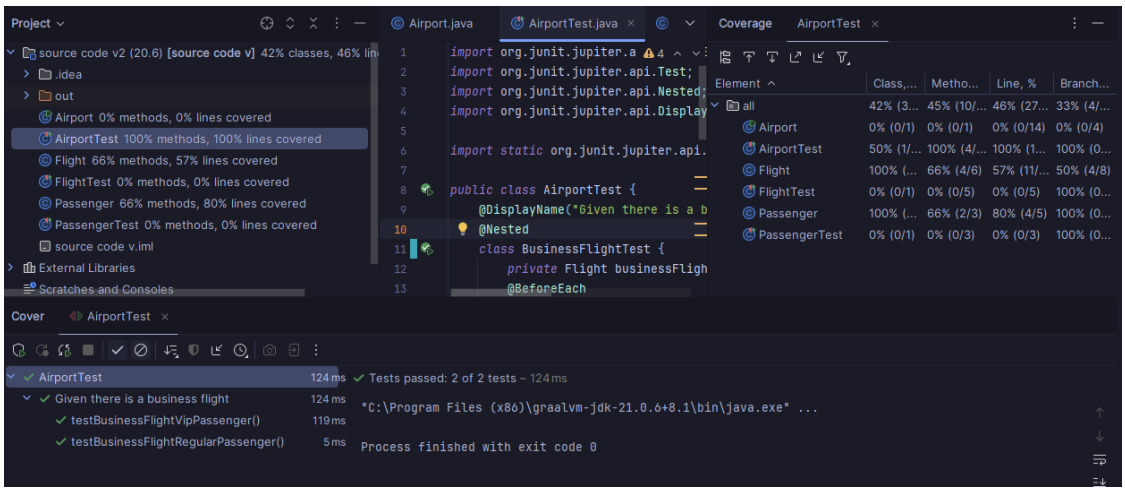
```java
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.DisplayName;

import static org.junit.jupiter.api.Assertions.*;

public class AirportTest {
    @DisplayName("Given there is a business flight")
    @Nested
    class BusinessFlightTest {
        private Flight businessFlight;
        @BeforeEach
        void setUp() {
            businessFlight = new Flight("2", "Business");
        }
        @Test
        public void testBusinessFlightRegularPassenger() {
            Passenger mike = new Passenger("Mike", false);
            assertEquals(false, businessFlight.addPassenger(mike));
            assertEquals(0, businessFlight.getPassengersList().size());
            assertEquals(false, businessFlight.removePassenger(mike));
            assertEquals(0, businessFlight.getPassengersList().size());
        }
        @Test
        public void testBusinessFlightVipPassenger() {
            Passenger james = new Passenger("James", true);
            assertEquals(true, businessFlight.addPassenger(james));
            assertEquals(1, businessFlight.getPassengersList().size());
            assertEquals(false, businessFlight.removePassenger(james));
            assertEquals(1, businessFlight.getPassengersList().size());
        }
    }
}
```

Apa yang dilakukan: Mengetes BusinessFlight dengan Passenger reguler dan VIP.

JUnitnya di mana: Sama, menggunakan @Nested, @Test, dan assertEquals.

Apa yang dites: Penambahan dan penghapusan penumpang di BusinessFlight.

Test sukses/failure: Semua tes berhasil sukses.



7. Listing 20.7 - Abstract Flight class

```
import java.util.Collections;
import java.util.List;
import java.util.ArrayList;

public abstract class Flight {
    private String id;
    List<Passenger> passengers = new ArrayList<Passenger>();
    public Flight(String id) {
        this.id = id;
    }
    public String getId() {
        return id;
    }
    public List<Passenger> getPassengersList() {
        return Collections.unmodifiableList(passengers);
    }
    public abstract boolean addPassenger(Passenger passenger);
    public abstract boolean removePassenger(Passenger passenger);
}
```

Apa yang dilakukan: Refactor Flight menjadi abstract class.

JUnitnya di mana: Tidak langsung, tetapi mendukung struktur testing.

Apa yang dites: Akan diuji melalui subclass EconomyFlight dan BusinessFlight.

8. Listing 20.8 - EconomyFlight class

```
public class EconomyFlight extends Flight {
    public EconomyFlight(String id) {
        super(id);
    }
    @Override
    public boolean addPassenger(Passenger passenger) {
        return passengers.add(passenger);
    }
    @Override
    public boolean removePassenger(Passenger passenger) {
        if (!passenger.isVip()) {
            return passengers.remove(passenger);
        }
        return false;
    }
}
```

Apa yang dilakukan: Mengimplementasikan aturan EconomyFlight.

JUnitnya di mana: Diuji melalui AirportTest.

Apa yang dites: Penambahan dan penghapusan sesuai aturan EconomyFlight.

Test sukses/failure: Berhasil sukses.

9. Listing 20.9 - BusinessFlight class

```
public class BusinessFlight extends Flight {
    public BusinessFlight(String id) {
        super(id);
    }
    @Override
    public boolean addPassenger(Passenger passenger) {
        if (passenger.isVip()) {
            return passengers.add(passenger);
        }
        return false;
    }
    @Override
    public boolean removePassenger(Passenger passenger) {
        return false;
    }
}
```

Apa yang dilakukan: Mengimplementasikan aturan BusinessFlight.

JUnitnya di mana: Diuji melalui AirportTest.

Apa yang dites: Penambahan dan penghapusan sesuai aturan BusinessFlight.

Test sukses/failure: Berhasil sukses.

10. Listing 20.10 - Refactoring AirportTest

```
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.DisplayName;

public class AirportTest {
    @DisplayName("Given there is an economy flight")
    @Nested
    class EconomyFlightTest {
        private Flight economyFlight;

        @BeforeEach
        void setUp() {
            economyFlight = new EconomyFlight("1");
        }

        @Test
        public void testEconomyFlightRegularPassenger() {
            Passenger mike = new Passenger("Mike", false);
            assertEquals("1", economyFlight.getId());
            assertEquals(true, economyFlight.addPassenger(mike));
            assertEquals(1, economyFlight.getPassengersList().size());
            assertEquals("Mike",
                    economyFlight.getPassengersList().get(0).getName());
            assertEquals(true, economyFlight.removePassenger(mike));
            assertEquals(0, economyFlight.getPassengersList().size());
        }

        @Test
        public void testEconomyFlightVipPassenger() {
            Passenger james = new Passenger("James", true);
            assertEquals("1", economyFlight.getId());
            assertEquals(true, economyFlight.addPassenger(james));
            assertEquals(1, economyFlight.getPassengersList().size());
            assertEquals("James",
                    economyFlight.getPassengersList().get(0).getName());
            assertEquals(false, economyFlight.removePassenger(james));
            assertEquals(1, economyFlight.getPassengersList().size());
        }
    }

    @DisplayName("Given there is a business flight")
    @Nested
    class BusinessFlightTest {
        private Flight businessFlight;

        @BeforeEach
        void setUp() {
            businessFlight = new BusinessFlight("2");

        }

        @Test
        public void testBusinessFlightRegularPassenger() {
            Passenger mike = new Passenger("Mike", false);
            assertEquals(false, businessFlight.addPassenger(mike));
            assertEquals(0, businessFlight.getPassengersList().size());
            assertEquals(false, businessFlight.removePassenger(mike));
```

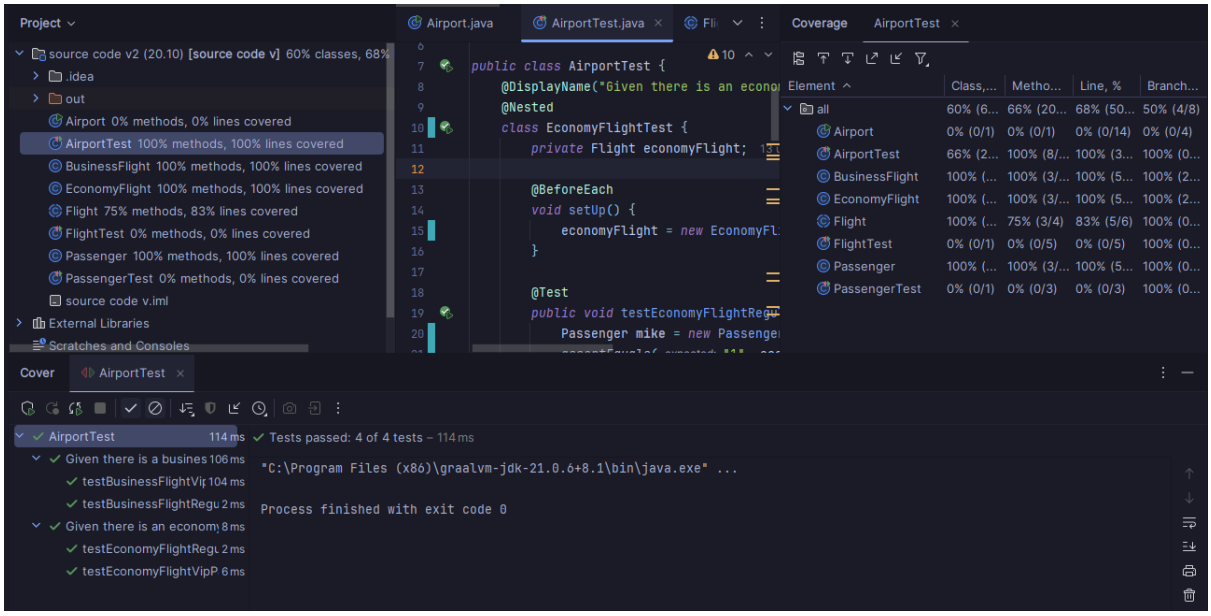
```
            assertEquals(0, businessFlight.getPassengersList().size());
        }
        @Test
        public void testBusinessFlightVipPassenger() {
            Passenger james = new Passenger("James", true);

            assertEquals(true, businessFlight.addPassenger(james));
            assertEquals(1, businessFlight.getPassengersList().size());
            assertEquals(false, businessFlight.removePassenger(james));
            assertEquals(1, businessFlight.getPassengersList().size());
        }
    }
}
```

Apa yang dilakukan: Mengganti penggunaan Flight menjadi EconomyFlight dan BusinessFlight.

JUnitnya di mana: Menyesuaikan class ujiannya.

Apa yang dites: Sesuai dengan refactor Flight class.

Test sukses/failure: Semua tes sukses, 100% coverage.



Current scope: all classes

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	70% (7/10)	61.8% (21/34)	50% (4/8)	66.2% (51/77)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
<empty package name>	70% (7/10)	61.8% (21/34)	50% (4/8)	66.2% (51/77)

generated on 2025-04-28 15:58

BAB IV
KESIMPULAN

Dari proses belajar Test-Driven Development (TDD) menggunakan JUnit, dapat disimpulkan bahwa:

- TDD membantu menjaga fokus pengembangan agar selalu berdasarkan kebutuhan.
- JUnit sangat mempermudah pembuatan dan verifikasi tes unit.
- Proses "test, code, refactor, repeat" membuat kode lebih bersih dan minim bug.
- Semua fungsi dalam Flight telah berjalan sesuai business rule.