Laporan Praktikum

"Praktikum 9 Test-Driven Development JUnit"

Mata Kuliah: Teknik Pemrograman

Disusun oleh:

Nama : Arkan Ramadhan Nugraha

NIM : 241524033

Kelas : 1B-TI4

POLITEKNIK NEGERI BANDUNG

2025

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR LAMPIRAN

BAB I PENDAHULUAN

I.1. Latar Belakang

Test-driven development adalah pemrograman praktis yang menggunakan siklus pengembangan pendek yang berulang di mana persyaratan diubah menjadi kasus uji, dan kemudian program dimodifikasi untuk membuat tes

I.2. Tujuan

Mempelajari penerapan Test-Driven Development (TDD) dalam pengembangan perangkat lunak menggunakan JUnit di Java.

BAB II PEMBAHASAN

II.1. Kasus 1

II.1.1. Screenshot Hasil Akhir Program

my-app/src/main/java/com/mycompany/app/Passenger.java

```java
package com.mycompany.app;

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 this.name;
    }

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

Gambar II.1: Class Passenger

my-app/src/main/java/com/mycompany/app/Flight.java › public abstract class Flight

```java
package com.mycompany.app;

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

public abstract class Flight {

    private String id;
    protected List<Passenger> passengers = new ArrayList<>();

    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);
    //     return flightType;      Arney1, 1 hour ago
    // }
```

Gambar II.2: Class Flight

my-app/src/main/java/com/mycompany/app/Airport.java › public class Airport › public static main( )

```java
package com.mycompany.app;

import com.mycompany.app.flights.BusinessFlight;
import com.mycompany.app.flights.EconomyFlight;

public class Airport {

    public static void main(String[] args) {
        Flight economyFlight = new EconomyFlight("1");
        Flight businessFlight = new BusinessFlight("2");

        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());
        }
    }
}
```

Gambar II.3: Class Airport (Main)

my-app/src/main/java/com/mycompany/app/flights/BusinessFlight.java

```java
package com.mycompany.app.flights;

import com.mycompany.app.Flight;
import com.mycompany.app.Passenger;

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;
    }
}
```

Gambar II.4: Class BusinessFlight

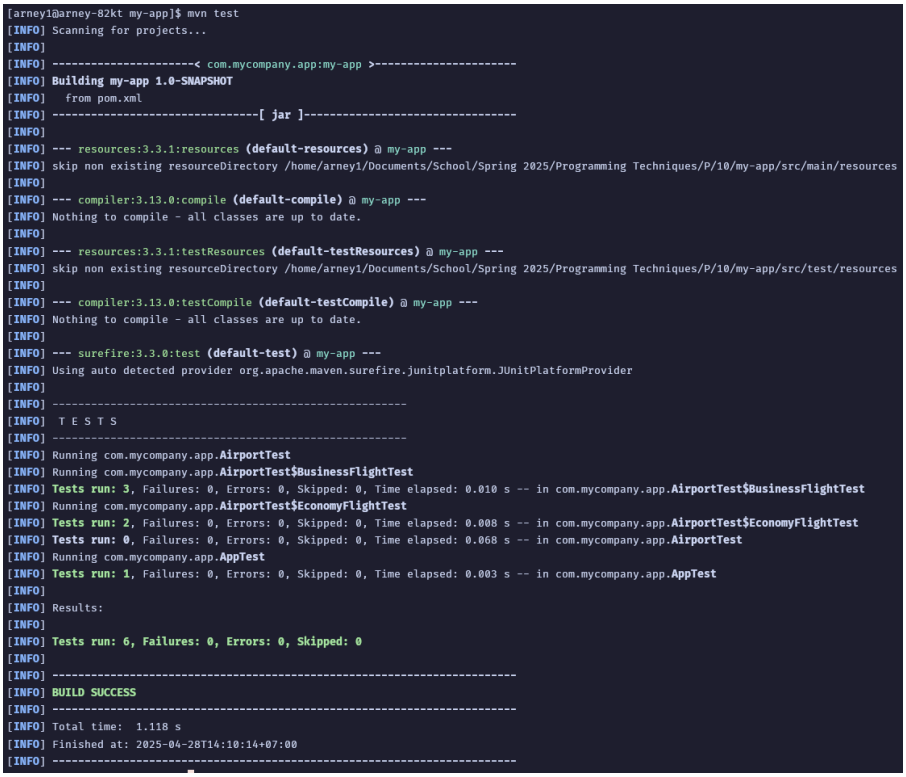
my-app/src/main/java/com/mycompany/app/flights/EconomyFlight.java

```java
package com.mycompany.app.flights;

import com.mycompany.app.Flight;
import com.mycompany.app.Passenger;

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;
    }
}
```

Gambar II.5: Class EconomyFlight

Gambar II.6: Class EconomyFlightTest di Class AirportTest

Gambar II.7: Class BusinessFlightTest di Class AirportTest



Gambar II.8: Output “mvn test” dengan hasil Success

II.1.2. Penjelasan

1. Deskripsi Kasus

   Sistem manajemen penerbangan, dapat mengelola penumpang dalam dua jenis penerbangan, dan mengatur aturan siapa saja yang boleh naik/turun berdasarkan tipe penumpang.

2. Penjelasan AirportTest

   1. Testing EconomyFlight

      1. Membuat grup tes khusus untuk EconomyFlight.

         1. `@Nested` digunakan untuk membuat pengelompokan yang lebih rapi.

         2. `@DisplayName` hanya untuk memberi label saat test dijalankan, supaya output di laporan lebih deskriptif.

      2. `@BeforeEach`

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

1. Ini akan selalu dijalankan sebelum setiap tes dalam kelas `EconomyFlightTest`.
2. Membuat instance baru `economyFlight` setiap kali supaya tes tidak saling memengaruhi.

3. Test 1: Regular Passenger di EconomyFlight
    1. Membuat penumpang biasa (false berarti dia bukan VIP).
    2. Menguji:
        1. addPassenger(mike) berhasil (true).
        2. Penumpang masuk ke daftar (passengersList bertambah 1).
        3. Nama penumpang betul ("Mike").
        4. removePassenger(mike) berhasil (true), lalu daftar jadi kosong lagi.
    3. Jadi, Penumpang biasa boleh naik EconomyFlight dan boleh turun.
4. Test 2: VIP Passenger di EconomyFlight
    1. Membuat penumpang VIP (`true` berarti VIP).
    2. Menguji:
        1. `addPassenger(james)` berhasil (`true`).
        2. Penumpang tercatat di daftar.
        3. Tapi tidak boleh `removePassenger(james)` (hasil `false`).
        4. Penumpang tetap ada di daftar.
    3. Jadi, VIP boleh naik EconomyFlight, tapi tidak bisa turun.

2. Testing BusinessFlight
    1. @BeforeEach
        1. Membuat objek BusinessFlight varu sebelum setiap tes
    2. Test 1: Regular Passenger di BusinessFlight
        1. Penumpang biasa tidak boleh naik ke BusinessFlight (`addPassenger` `false`).
        2. Tidak boleh menghapus (karena tidak pernah berhasil masuk).
        3. `passengersList` tetap 0.
    3. Test 2: VIP Passenger di BusinessFlight

1. VIP boleh naik BusinessFlight.
2. Tapi, tidak bisa turun (`removePassenger false`).
3. Tetap ada di daftar.

BAB III HASIL DAN ANALISIS

III.1. Hasil

Unit test yang dibuat berhasil memvalidasi bahwa penanganan penumpang biasa dan VIP pada EconomyFlight dan BusinessFlight sesuai dengan aturan yang ditentukan, dengan seluruh skenario pengujian berjalan sukses tanpa error.

III.2. Lesson Learned

1. Menulis tes terlebih dahulu (sebelum membuat fitur) membantu:
    1. Menentukan spesifikasi dan aturan program lebih jelas sejak awal.
    2. Menghindari kesalahan logika karena semuanya diuji dari awal.
2. Jika ada perubahan di kode, kita langsung tahu jika ada yang rusak karena test akan gagal.
3. Dengan TDD, design class menjadi lebih rapi, modular, dan fokus pada kebutuhan.

BAB IV KESIMPULAN DAN SARAN

IV.1. Kesimpulan

Penerapan Test-Driven Development (TDD) membantu memastikan aplikasi dikembangkan dengan lebih terarah, mengurangi risiko kesalahan, dan mempercepat deteksi bug, sehingga menghasilkan kode yang lebih stabil dan mudah dirawat.

LAMPIRAN

Lampiran 1 - Source Code .zip

Telampir di Google Classroom

Lampiran 2 - Link Repository GitHub

https://github.com/Arney1/Test-Driven-Development-Java