

LAPORAN PRAKTIKUM PERTEMUAN-10

Diajukan untuk memenuhi salah satu tugas praktikum Mata kuliah Pemrograman Berorientasi Objek



**Disusun Oleh:
Daiva Raditya Pradipa (231511039)**

**Jurusan Teknik Komputer dan Informatika
Program Studi D-3 Teknik Informatika
Politeknik Negeri Bandung
2024**

1. Listing 20.1

a. Screenshot Hasil Program

```
package com.example;

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;
    }
}
```

b. Penjelasan

Pada Listing 20.1 ini kita hanya membuat class Passaner yang memiliki atribut name bertipe string dan Vip bertipe Boolean yang masing masing memiliki modifie private (menerapkan encapsulation). Kemudian terdapat constructor untuk menginstansikan objek dari class ini, lalu terakhir terdapat method getter untuk atribut name dan vip.

c. Masalah

d. Solusi

e. Teman yang membantu

2. Listing 20.2

a. Screenshot hasil program

```
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);
        }
    }
}
```

b. Penjelasan

Pada listing 20.2, terdapat class Flight yang memiliki atribut id, List passenger (composite kepada passenger), dan FlightType. Terdapat pula constructor untuk menginstansikan class ini menjadi object sekaligus untuk inisialisasi nilai masing

masing atribut. Kemudian terdapat method getter untuk masing-masing atribut dan method addPassanger serta removePassanger.

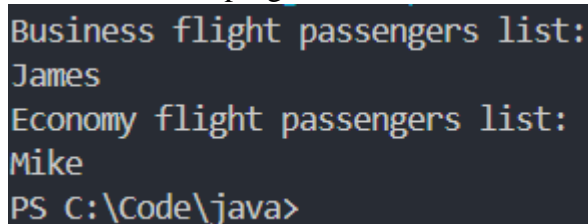
Method ini berfungsi untuk menentukan apakah passanger yang ditambahkan pada penerbangan ini diperbolehkan dengan melihat status vip passanger. Apabila penerbangan bertipe "Economy", maka semua passanger dengan status vip apapun (true/false) diperbolehkan untuk mengikuti penerbangan. Sedangkan untuk penerbangan bertipe "Business" hanya memperbolehkan passanger dengan status vip true untuk mengikuti penerbangan, passanger dengan tipe status vip false tidak diperbolehkan mengiktui penerbangan dengan ditandai method ini akan mereturn nilai false.

Kemudian terdapat method removePassanger yang digunakan untuk menghapus passanger dari list passanger pada suatu penerbangan. Method ini memastikan bahwa hanya passanger dengan penerbangan Economy dan dengan status vip false yang hanya boleh dihapus dari list passanger penerbangan.

- c. Masalah
- d. Solusi
- e. Teman yang membantu

3. Listing 20.3

- a. Screenshot hasil program



```
Business flight passengers list:
James
Economy flight passengers list:
Mike
PS C:\Code\java>
```

- b. Penjelasan

Pada listing 20.3, kita membuat main file untuk menginstansikan dan membuat program untuk menampilkan list passanger pada setiap jenis penerbangan.



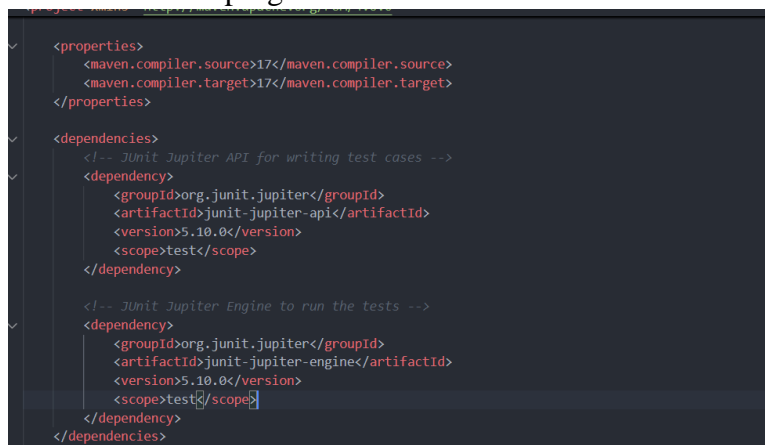
```
public static void main(String[] args) {
    Flight economyFlight = new Flight(id:"1", flightType:"Economy");
    Flight businessFlight = new Flight(id:"2", flightType:"Business");
    Passenger james = new Passenger(name:"James", VIP:true);
    Passenger mike = new Passenger(name:"Mike", VIP:false);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

Bisa dilihat disini kita membuat 2 penerbangan yaoti economy dan business dengan id 1 dan 2. Kemudian kita membuat object passanger dengan nama james dan mike serta status vip true dan false. Terdapat pu;a pemanggilan method add passanger untuk james ke penerbangan bisnis kemudian mencoba menghapusnya dengan method removePassenger. Namun pada hasil program terlihat james masih terdaftar pada penerbangan business, hal dikarenakan james memiliki tipe status vip true. Kemudian terdapat penambahan passanger mike ke penerbangan business dan ecomony. Pada hasil akhir program, terlihat bahwa mike terdaftar pada penerbangan economy, namun pada penerbangan business tidak. Hal ini dikarenakan mike memiliki status vip false.

- c. Masalah
- d. Solusi
- e. Teman yang membantu

4. Listing 20.4

- a. Screenshot hasil program



```

<properties>
  <maven.compiler.source>17</maven.compiler.source>
  <maven.compiler.target>17</maven.compiler.target>
</properties>

<dependencies>
  <!-- JUnit Jupiter API for writing test cases -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>

  <!-- JUnit Jupiter Engine to run the tests -->
  <dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
  </dependency>
</dependencies>

```

- b. Penjelasan

Pada listing 20.4, kita akan mencoba mengadd dependencies pada project java maven kita menggunakan tag dependencies dan dependency pada file pom.xml yang memungkinkan kita untuk add dependency atau library pihak ketiga pada project maven kita (hal ini serupa dengan jar pada project java basic, namun perbedaannya pada maven kita hanya perlu add dependency apa yang kita inginkan dan nanti maven akan secara otomatis mendownloadkan dependency yang kita inginkan pada project java kita).

- c. Masalah
- d. Solusi
- e. Teman yang membantu

5. Listing 20.5

- a. Screenshot hasil program

```

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running AirportTest
[INFO] Running AirportTest$EconomyFlightTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.076 s -- in AirportTest$EconomyFlightTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.109 s -- in AirportTest
[INFO] Results:
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO] -----

```

b. Penjelasan

Pada listing 20.5, kita akan membuat sebuah pengujian menggunakan junit yang telah kita install sebelumnya pada listing 20.4. Disini terdapat beberapa anotasi yaitu

a) @BeforeEach

```

@BeforeEach
void setUp() {
    economyFlight = new Flight(id:"1", flightType:"Economy");
}

```

Anotasi Before Each ini menandakan bahwa function setup() ini akan dijalankan terlebih dahulu sebelum function pengujian dijalankan.

b) @DisplayName

```

@DisplayName("Given there is an economy flight")
@Nested
class EconomyFlightTest {

```

Anotasi DisplayName digunakan untuk memberikan nama pengujian custom untuk setiap kelas atau method pengujian. Hal ini membuat pengujian mudah dipahami maksud atau tujuan dari kelas atau method pengujian ini.

c) @Nested

Anotasi ini digunakan untuk menggrouping pengujian (biasanya class pengujian) sehingga setiap kelas pengujian dapat fokus pada skenario pengujian tertentu atau bagian spesifik dari fitur yang diuji. Selain itu nested membuat file pengujian lebih terorganisir dan terstruktur

d) @Test

```

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

```

Anotasi ini digunakan untuk menandakan bahwa method ini merupakan method pengujian yang memiliki implementasi pengujian((assert) untuk menguji scenario tertentu.

Pada listing 20.5 ini juga merupakan implementasi dari unit testing pada java menggunakan assert dari library junit. Sebagai contoh pada gambar diatas terdapat method pengujian EconomyFlightRegularPassenger dimana terdapat beberapa

pengujian seperti menggunakan assertEquals pada id dengan membandingkan hasil expected yaitu 1 dan hasil actual yaitu menggunakan method getId pada object economyFlight yang telah kita instansikan pada method setup. Selain itu terdapat pengujian dengan syntax “assertEquals(true, economyFlight.addPassenger(mike));” untuk memastikan Passenger mike ini bisa terdaftar pada penerbangan economy, dengan ekspektasi hasil true, karena memang penerbangan dengan tipe economy ini memperbolehkan semua passenger apapun untuk mengikuti penerbangan.

c. Masalah

Terdapat masalah yang saya hadapi pada listing 20.5 ini yaitu library junit yang tidak terbaca pada saat akan diimport pada file class AirportTest.java ini.

d. Solusi

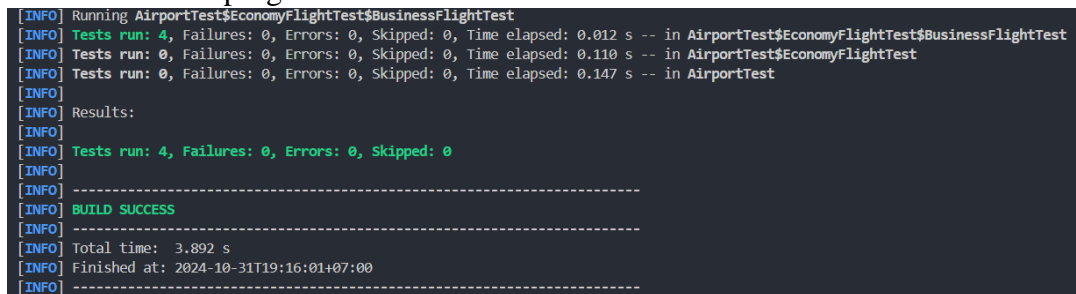
Untuk mengatasi masalah ini saya menginstall maven pada website apache, kemudian mengadd path bin maven pada system variabel OS. Kemudian menginstall dependency yang terdaftar pada file pom.xml dengan menjalankan perintah “mvn clean install” pada direktori project maven.

Selain itu juga saya memindahkan file AirportTest.java yang asalnya berada pada direktori ‘src/com/example’ ke direktori ‘test/java’.

e. Teman yang membantu

6. Listing 20.6

a. Screenshot hasil program



```
[INFO] Running AirportTest$EconomyFlightTest$BusinessFlightTest
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.012 s -- in AirportTest$EconomyFlightTest$BusinessFlightTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.110 s -- in AirportTest$EconomyFlightTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.147 s -- in AirportTest
[INFO] Results:
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO] BUILD SUCCESS
[INFO] Total time: 3.892 s
[INFO] Finished at: 2024-10-31T19:16:01+07:00
[INFO]
```

b. Penjelasan

Pada listing 20.6, terdapat penambahan unit test untuk penerbangan business. Untuk pengujiannya sendiri identic dengan unit test untuk penerbangan economy dimana terdapat dua scenario case yaitu pendaftaran penerbangan bisnis untuk passenger vip dan non vip. Implementasi junit yang digunakan juga sama dengan penerbangan economy yaitu menggunakan assertEquals untuk membandingkan output pada program dan ekspektasi output yang diinginkan.

```

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

```

Contoh pada test case untuk passenger non vip terdapat pengujian “assertEquals(false, businessFlight.addPassenger(mike));” dengan ekspektasi hasil false dan hasil actual program dengan menggunakan object businessFlight dengan method addPassenger dengan parameter object Passenger mike yang memiliki status vip false. Test case ini bertujuan untuk memastikan passenger non vip tidak bisa terdaftar pada penerbangan business.

- c. Masalah
- d. Solusi
- e. Teman yang membantu

7. Listing 20.7

- a. Screenshot hasil program

```

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);
}

```

- b. Penjelasan

Pada listing 20.7, terdapat modifikasi code untuk class Flight yang sebelumnya merupakan non-abstract class menjadi abstract class serta perubahan non abstract method addPassenger dan removePassenger menjadi abstract method. Kemudian terdapat penghapusan atribut flightType karena nantinya atribut ini akan dijadikan subclass dari class flight yaitu economyFlight dan businessFlight.

- c. Masalah

- d. Solusi
- e. Teman yang membantu

8. Listing 20.8

- a. Screenshot hasil program

```
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;  
    }  
}
```

- b. Penjelasan

Pada listing 20.8, terdapat penambahan class yang merupakan subclass dari abstract class Flight yang bernama EconomyFlight yang memiliki constructor dengan atribut yang merujuk pada class flight (id) serta terdapat implementasi abstract method addPassenger dan removePassenger dari class Flight yang ditandai dengan anotasi '@override'.

- c. Masalah
- d. Solusi
- e. Teman yang membantu

9. Listing 20.9

- a. Screenshot hasil program


```

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;
    }
}

```

b. Penjelasan

Pada listing 20.9, terdapat penambahan class yang merupakan subclass dari abstract class Flight yang bernama BusinessFlight yang memiliki constructor dengan atribut yang merujuk pada class flight (id) serta terdapat implementasi abstract method addPassenger dan removePassenger dari class Flight yang ditandai dengan anotasi '@override' (Implementasi dari method addPassenger dan removePassenger pada dua subclass ini identik dengan implementasi pada class flight sebelum dijadikan abstract class, hanya saja tidak ada pengecekan tipe flightType karena pada subclass ini sudah spesifik implementasinya untuk setiap jenis penerbangan).

c. Masalah

d. Solusi

e. Teman yang membantu

10. Listing 20.9

a. Screenshot hasil program

```

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running AirportTest
[INFO] Running AirportTest$BusinessFlightTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.077 s -- in AirportTest$BusinessFlightTest
[INFO] Running AirportTest$EconomyFlightTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.007 s -- in AirportTest$EconomyFlightTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.120 s -- in AirportTest
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 3.922 s
[INFO] Finished at: 2024-10-31T19:38:22+07:00
[INFO]

```

b. Penjelasan

Pada listing 20.10, disini kita hanya mengupdate method setup untuk masing masing class pengujian karena Sekarang class flight merupakan abstract class, sehingga class ini tidak bisa lagi diinstansikan. Maka dari itu pada setiap method setup pada masing masing kelas pengujian kita update dengan menggunakan instansi objek dari subclass economyFlight dan businessFlight.

- c. Masalah
- d. Solusi
- e. Teman yang membantu

Link github: [Tugas-PBO/week-10 at main · RaditZX/Tugas-PBO](https://github.com/RaditZX/Tugas-PBO/tree/main/Tugas-PBO/week-10)