Praktek Pertemuan 9

Disusun untuk memenuhi tugas mata kuliah Pemrograman Berorientasi Objek (Teori)

Disusun Oleh:
Radja Restu Arsita (231511061) 2B – D3

Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

2024

DAFTAR ISI

PEMBAHASAN

1.1 Passanger class

a. Code

```java
package org.example;

public class Passenger {  no usages
    private String name; // 1  2 usages
    private boolean vip; // 2  2 usages

    public Passenger(String name, boolean vip) { // 3  no usages
        this.name = name;
        this.vip = vip;
    }

    public String getName() { // 4  no usages
        return name;
    }

    public boolean isVip() { // 5  no usages
        return vip;
    }
}
```

b. Penjelasan

Mendeklarasikan class Passanger dengan 2 instance variable yaitu name dan vip yang dimana ada constructor untuk kedua instace variable tersebut dan juga ada method untuk get name juga untuk mengecek apakah passanger itu vip atau tidak

1.2 Flight Class

a. Code

```java
public class Flight {  5 usages  2 inheritors
    private String id;  2 usages
    private List<Passenger> passengers = new ArrayList<Passenger>();  5 usages
    private String flightType;  6 usages

    public Flight(String id, String flightType) {  2 usages
        this.id = id;
        this.flightType = flightType;
    }

    public String getId() {  1 usage
        return id;
    }

    public List<Passenger> getPassengersList() {  3 usages
        return Collections.unmodifiableList(passengers);
    }

    public String getFlightType() {  no usages
        return flightType;
    }
```

```java
public boolean addPassenger(Passenger passenger) {  1 usage  2 overrides
    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);
    }
}
```

```java
public boolean removePassenger(Passenger passenger) {  1 usage  2 overrides
    switch (flightType) {
        case "Economy":
            if (passenger.isVip()) {
                return passengers.remove(passenger);
            }
            return false;
        case "Business":
            return passengers.remove(passenger);
        default:
            throw new RuntimeException("Unknown type: " + flightType);
    }
}
```

b. Penjelasaan

Class Flight memiliki 3 instance variable yaitu id , passanggers , flight type Yang mana ada constructor untuk id dan Flight type lalu ada getter untuk id, passangers, flightType lalu ada 2 method tambahan yaitu add passanger yang dimana passanger akan dicek flightType nya ekonomi maka akan dilakukan add ke economy jika si passanger itu VIP dan business maka dia kana di add ke dengan flight type bussines dan juga ada 1 method untuk melakukan remove passanger yang dimana ada dua kondisi yaitu ketika flight type economy dan passanger itu vip maka dia akan di remove dan jika flight type nya business maka dia akan langsung di remove tanpa melakukan cek vip

1.3 Throwing exception

a. Code

```
package org.example;

public class Airport {
    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("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());
        }
    }
}
```

b. Output

```
Business flight passengers list:
James
Economy flight passengers list:
Mike
```

c. Penjelasan

Output yang dikeluarkan dari main program Airport sesuai dengan yang di harapkan dimana passenger yang termasuk VIP akan di masukan ke penerbangan bertipe business sedangkan passenger yang tidak termasuk ke dalam VIP akan dimasukan ke penerbangan bertipe economy

1.4 Junit Dependecies added to the pom.xml file

a. Code

```
<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>
```

b. Penjelasan

Diatas merupakan script untuk menambahka dependency dengan memanfaatkan fitur yang diberikan maven yang dimana kita bisa melakukan testing terhadap method atau program yang kita miliki.

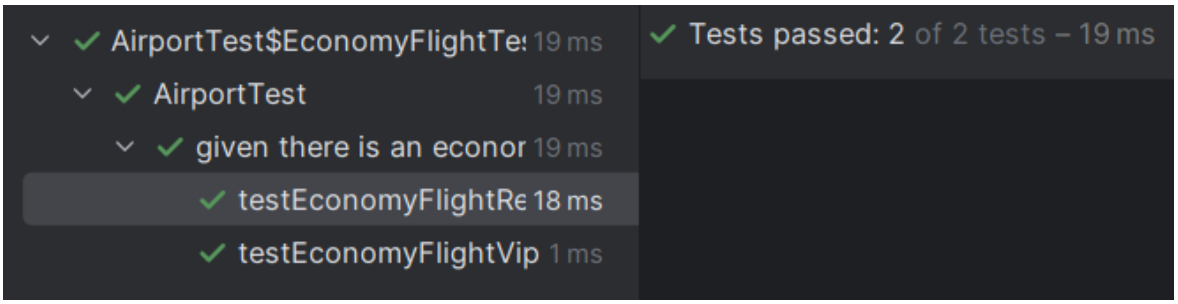
1.5 testing the business logic for an economy flight

a. Code

```java
package org.example;

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

public class AirportTest {

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

        private Flight economyFlight;  13 usages

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

        @Test
        public void testEconomyFlightRegularPassenger() {
            Passenger mike = new Passenger( name: "Mike",  vip: false);

            assertEquals( expected: "1", economyFlight.getId());
            assertEquals( expected: true, economyFlight.addPassenger(mike));
            assertEquals( expected: 1, economyFlight.getPassengersList().size());
            assertEquals( expected: "Mike", economyFlight.getPassengersList().get(0).getName());

            assertEquals( expected: true, economyFlight.removePassenger(mike));
            assertEquals( expected: 0, economyFlight.getPassengersList().size());
        }
```

```java
        @Test
        public void testEconomyFlightVipPassenger() {
            Passenger james = new Passenger( name: "James",  vip: true);

            assertEquals( expected: "1", economyFlight.getId());
            assertEquals( expected: true, economyFlight.addPassenger(james));
            assertEquals( expected: 1, economyFlight.getPassengersList().size());
            assertEquals( expected: "James", economyFlight.getPassengersList().get(0).getName());

            assertEquals( expected: false, economyFlight.removePassenger(james));
            assertEquals( expected: 1, economyFlight.getPassengersList().size());
        }
    }
}
```

b. Output



c. Penjelasan

Diatas merupakan contoh testing dari method yang sudah dibuat sebelum nya menggunakan assertEquals yang mana akan mereturn true ketika nilai dari expected dan actual sama dan false ketika nilai dari expected dan actual berbeda ketika dilakukan perbandingan oleh fungsi tersebut.

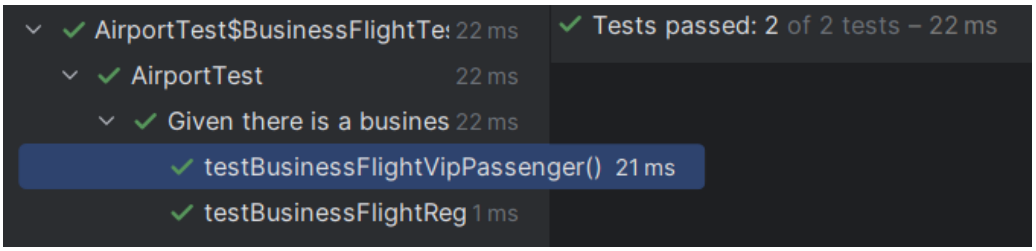
1.6 testing the business logic of the business flight

a. Code

```
@DisplayName("Given there is a business flight")
@Nested
class BusinessFlightTest {

    private Flight businessFlight;  10 usages

    @BeforeEach
    void setUp() {
        businessFlight = new Flight( id: "2",  flightType: "Business");
    }

    @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());
    }
```

```
@Test
public void testBusinessFlightVipPassenger() {
    Passenger james = new Passenger( name: "James",  vip: true);

    assertEquals( expected: true, businessFlight.addPassenger(james));
    assertEquals( expected: 1, businessFlight.getPassengersList().size());
    assertEquals( expected: "James", businessFlight.getPassengersList().get(0).getName());

    assertEquals( expected: false, businessFlight.removePassenger(james));
    assertEquals( expected: 1, businessFlight.getPassengersList().size());
}
```

b. Output



c. Penjelasan

Sama seperti sebelum nya dilakukan test menggunakan assertEquals yang dimana ada dua kasus yang diuji yaitu ketika tidak ada penumpang VIP dan ketika ada ada penumpang VIP yang dimana kasus tersebut berhasil lulus test tanpa ada bug.

1.7 Abstract flight class , the basis of hierarc

a. Code

```
package org.example;

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

public abstract class Flight {  8 usages  6 related problems
    private String id;  2 usages
    List<Passenger> passengers = new ArrayList<>();  1 usage

    public Flight(String id) {  4 usages  6 related problems
        this.id = id;
    }

    public String getId() {  2 usages  2 related problems
        return id;
    }

    public List<Passenger> getPassengersList() {  13 usages  13 related
        return Collections.unmodifiableList(passengers);
    }

    public abstract boolean addPassenger(Passenger passenger);  7 u

    public abstract boolean removePassenger(Passenger passenger);
}
```

b. Penjelasan

Mengubah flight menjadi abstract class karena ingin merubah dari yang sebelum nya decision menjadi polymorphism yang dimana nanti akan ada business flight, economy flight dan premium flight sebagai class

1.8 EconomyFlight class , extending the abstract Flight class

a. Code

```java
package org.example;

public class EconomyFlight extends Flight {  no usages

    public EconomyFlight(String id) {  no usages
        super(id);
    }

    @Override  7 usages
    public boolean addPassenger(Passenger passenger) {
        return passengers.add(passenger);
    }

    @Override  5 usages
    public boolean removePassenger(Passenger passenger) {
        if (!passenger.isVip()) {
            return passengers.remove(passenger);
        }
        return false;
    }
}
```

b. Penjelasan

Membuat class untuk economyFlight yang mana melakukan override method dari Flight karena melakukan extend terhadap flight, yang dimana ketika passanger bukan VIP maka akan di remove.

1.9 BusinessFlight class, extending the abstract Flight class

a. Code

```java
package org.example;

public class BusinessFlight extends Flight {  no usages

    public BusinessFlight(String id) {  no usages
        super(id);
    }

    @Override  7 usages
    public boolean addPassenger(Passenger passenger) {
        if (passenger.isVip()) {
            return passengers.add(passenger);
        }
        return false;
    }

    @Override  5 usages
    public boolean removePassenger(Passenger passenger) {
        return false;
    }
}
```

b. Penjelasan

Membuat class Bussines flight yang mana melakukan extends terhadap flight dan melakukan override terhadap method nya yang dimana addPassanger bisa dilakukan ketika passanger nya itu merupakan VIP.

1.10 Refactoring propagation into the AirportTest class

a. Code

```
public class AirportTest {

    @DisplayName("Given there is an economy flight")
    @Nested
    class EconomyFlightTest {
        private Flight economyFlight;  11 usages

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

        @Test
        public void testEconomyFlightRegularPassenger() {
            Passenger mike = new Passenger( name: "Mike",  vip: false); // Regular passenger (non-VIP)

            // Regular passenger should be able to be added in EconomyFlight
            assertEquals( expected: true, economyFlight.addPassenger(mike));
            assertEquals( expected: 1, economyFlight.getPassengersList().size());
            assertEquals( expected: "Mike", economyFlight.getPassengersList().get(0).getName());

            // Regular passenger should be able to be removed in EconomyFlight
            assertEquals( expected: true, economyFlight.removePassenger(mike));
            assertEquals( expected: 0, economyFlight.getPassengersList().size());
        }
```

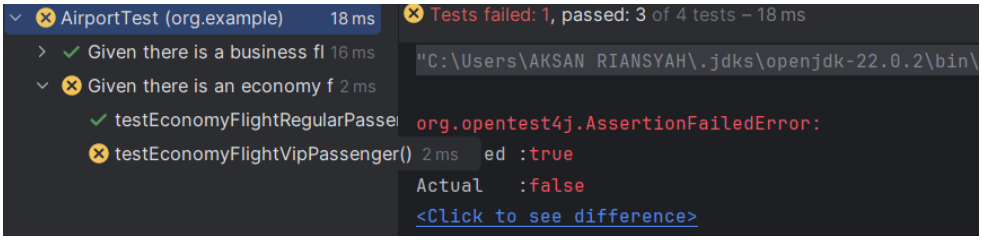
```
    @Test
    public void testEconomyFlightVipPassenger() {
        Passenger james = new Passenger( name: "James",  vip: true); // VIP passenger

        // VIP passenger should be able to be added in EconomyFlight
        assertEquals( expected: true, economyFlight.addPassenger(james));
        assertEquals( expected: 1, economyFlight.getPassengersList().size());
        assertEquals( expected: "James", economyFlight.getPassengersList().get(0).getName());

        // VIP passenger should be able to be removed in EconomyFlight
        assertEquals( expected: true, economyFlight.removePassenger(james));
        assertEquals( expected: 0, economyFlight.getPassengersList().size());
    }
```

```java
@DisplayName("Given there is a business flight")
@Nested
class BusinessFlightTest {
    private Flight businessFlight;  10 usages

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

    @Test
    public void testBusinessFlightRegularPassenger() {
        Passenger mike = new Passenger( name: "Mike",  vip: false); // Regular passenger (non-VIP)

        // Regular passenger should NOT be added in BusinessFlight
        assertEquals( expected: false, businessFlight.addPassenger(mike));
        assertEquals( expected: 0, businessFlight.getPassengersList().size());

        // Attempt to remove should also be false, as he was never added
        assertEquals( expected: false, businessFlight.removePassenger(mike));
        assertEquals( expected: 0, businessFlight.getPassengersList().size());
    }
```

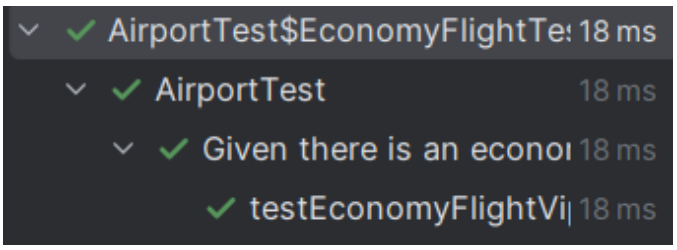
```java
@Test
public void testBusinessFlightVipPassenger() {
    Passenger james = new Passenger( name: "James",  vip: true); // VIP passenger

    // VIP passenger should be able to be added in BusinessFlight
    assertEquals( expected: true, businessFlight.addPassenger(james));
    assertEquals( expected: 1, businessFlight.getPassengersList().size());
    assertEquals( expected: "James", businessFlight.getPassengersList().get(0).getName());

    // VIP passenger should NOT be able to be removed in BusinessFlight
    assertEquals( expected: false, businessFlight.removePassenger(james));
    assertEquals( expected: 1, businessFlight.getPassengersList().size());
}
```

b. Output

Percobaan expected != actual



Percobaan expected == actual



c. Penjelasan

Pada testing kali ini diberikan 2 kasus yaitu ketika ada passenger regular atau ketika ada passenger VIP lalu dilakukan pengujian assertEquals dengan kasus (expected != actual) yang mana terjadi error ketika melakukan test pada economyFlight yang dimana ketika ada passenger VIP dicoba untuk di remove itu hasil nya gagal karena expected nya adalah true sedangkan pada method remove passenger di class economyFlight ketika ada passanger VIP respon yang diberikan adalah passenger tersebut tidak akan di remove ketika data expected diperbaiki sesuai dengan data actual akan kembali success.

LESSON LEARNED

Dari tugas praktek PBO week 10 ini saya mempelajari bagaimana untuk melakukan testing menggunakan maven pada intelij saya diberikan kasus untuk menangani program airport atau penerbangan yang dimana ada kasus seorang passenger itu memiliki data di dalam diri nya yaitu VIP atau bukan VIP yang menentukan flightType yang akan dia gunakan apakah itu economy ataupun business

Selain itu saya belajar menggunakan assertEquals yang membandingkan data expected dan actual yang akan mengembalikan nilai true ketika value nya sama dan juga sebalik nya ketika expected dan actual tidak sama maka akan mereturn nilai false lalu ketika mereturn nilai false sistem akan memberi tahukan mana yang menyebabkan error yang diberikan tampilan mengenai value dari expected dan value dari actual yang berbeda dan juga memberika path pada bagian mana assertEqual mereturn nilai false.

REFERENSI

CodeGym. (n.d.). *Pengujian unit di Java dengan JUnit*. Diakses pada 31 Oktober 2024, dari https://codegym.cc/id/groups/posts/id.191.pengujian-unit-di-java-dengan-junit

Code Intelligence. (2023). *How to Do Unit Testing in Java*. Diakses pada 31 Oktober 2024, dari https://www.code-intelligence.com/blog/how-to-do-unit-testing-in-java

Baeldung. (n.d.). *Java Unit Testing Best Practices*. Diakses pada 31 Oktober 2024, dari https://www.baeldung.com/java-unit-testing-best-practices

Baeldung. (n.d.). *Java Assert Equality Without Using the Equals Method*. Diakses pada 31 Oktober 2024, dari https://www.baeldung.com/java-assert-equality-no-equals

JUnit. (n.d.). *Assert (JUnit API)*. Diakses pada 31 Oktober 2024, dari https://junit.org/junit4/javadoc/4.8/org/junit/Assert.html

LAMPIRAN

Link github :

https://github.com/Radja-Restu-A/Object-orianted-programming/tree/main/Pertemuan%209%20(P)