

LAPORAN PRAKTIKUM

Pemrograman Berorientasi Objek

Diajukan untuk memenuhi salah satu tugas praktikum mata kuliah Pemrograman berorientasi objek



Disusun Oleh:

Hasbi Andi Muttaqin (231511049)

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

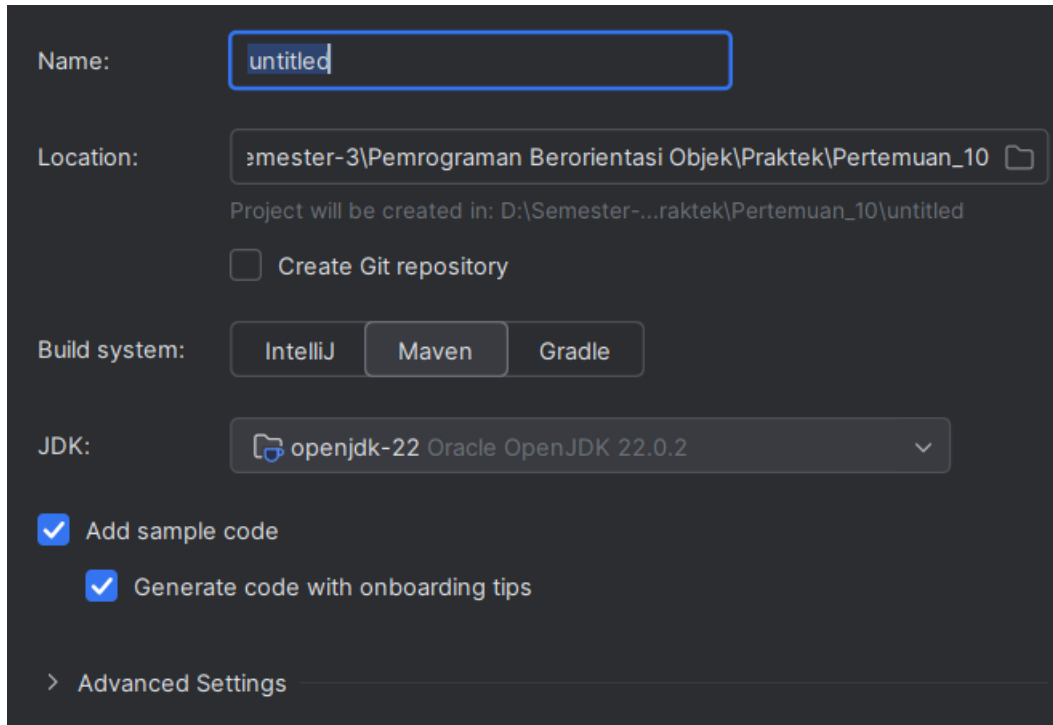
DAFTAR ISI

Pemrograman Berorientasi Objek.....	1
1. Listing 20.1 Passanger Class.....	3
2. Listing 20.2 Flight Class.....	3
3. Listing 20.3 Airport Class including the Main Method	4
4. Listing 20.4 Junit 5 Dependencies added to the pom.xml	5
5. Class AirportTest.....	6
6. Listing 20.5 Testing the business logic for an economic flight.....	7
7. Listing 20.6 Testing the business logic for an business flight	8
8. Listing 20.7 Abstract Flight class, the basis of the hierarchy	8
9. Listing 20.8 EconomyFlight class, extending the abstract Flight class	9
10. Listing 20.9 BusinessFlight class, extending the abstract Flight class	9
11. Listing 20.10 Refactoring propagation into the AirportTest class	9
LINK GITHUB	11


Pertemuan 10 Test Driven DevelopmentJunit5

Konfigurasi

Pada saat membuat project baru pilih Build Up Maven




Name:

Location: 

Project will be created in: D:\Semester-...raktek\Pertemuan_10\untitled

☐ Create Git repository

Build system:

JDK: 

☒ Add sample code

☒ Generate code with onboarding tips

[> Advanced Settings](#)

1. Listing 20.1 Passanger Class

Source Code :

```
package org.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;
    }
}
```

class ini terdapat tulisan “Package prg.example” yang menunjukkan bahwa file ini terdapat pada package org dengan tujuan kelasnya untuk membuat data dari penumpang(passenger) yang terdiri atas name dengan tipe data string dan status vip dengan tipe data boolean

2. Listing 20.2 Flight Class

```
package org.example;

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

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);
    }
}
}

```

Terdapat beberapa import dari package java.util yaitu :

1. ArrayList, yang berfungsi dalam pembuatan dan pengelolaan list array dan digunakan untuk menyimpan list objek dengan ukuran yang dinamis
2. Collection, sebagai pemanggilan metode seperti list dan set
3. List untuk mengurutkan daftar item

Kelas ini berfungsi untuk proses menambah dan mengkategorikan penumpang

3. Listing 20.3 Airport Class including the Main Method

```

package org.example;

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());
        }
    }
}

```

Kelas ini menunjukan bila Seseorang memiliki value vip true maka akan masuk ke fungsi businessflight sedangkan bila false akan masuk ke Economy flight

4. Listing 20.4 Junit 5 Dependencies added to the pom.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>org.example</groupId>
    <artifactId>W10-JUnit5</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>22</maven.compiler.source>
        <maven.compiler.target>22</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.10.0</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

</project>

```

1. Maven

Maven adalah alat manajemen proyek dan build automation, untuk mengatur depedensi, mengelola versi dan mengotomasi proses penyusunan kode

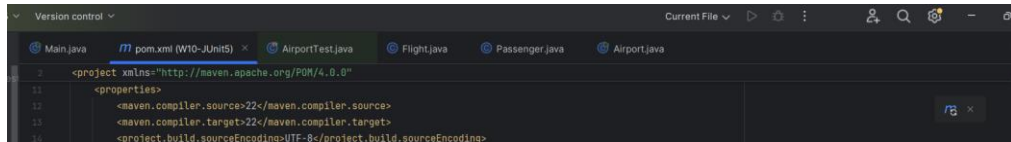
2. XML

Extensible Markup Language adalah format data berbasis teks yang digunakan untuk menyimpan dan mengatur data dengan struktur yang mudah dibaca oleh manusia dan mesin, pada program ini xml digunakan untuk Menyusun konfigurasi maven yang disebut POM atau Project Object Model dengan ekstensi .xml

POM ini dapat berisi model versi yang digunakan, dalam program diatas menggunakan versi 5.10.0 yang menandakan menggunakan Junit 5

3. Kendala & Temuan

1. Pastikan Maven sudah di download, pada intelij sendiri, maven secara otomatis di download, dan bila ingin memperbaruinya, kita hanya perlu mengetikan versi berapa yang kita ingin kan, bila merah dapat mengklik Logo m berwarna biru dengan tanda refresh di dalamnya



2. Pastikan pada saat build up POM file ini tidak terdapat deklarasi versi ganda, dikarenakan akan menyebabkan konflik pada saat memanggilnya.

5. Class AirportTest

```
import org.example.Flight;
import org.example.Passenger;
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;

        @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());
        }
    }

    @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());
}
}

```

Class ini berfungsi untuk melakukan testing pada beberapa fungsi yang telah di buat

1. Import fungsi dari Junit

Terdapat beberapa fungsi yang diimport dari Junit seperti:

- import org.junit.jupiter.api.BeforeEach;
Untuk menginisialisasi kondisi atau objek yang diperlukan dalam setiap pengujian, di notasikan dengan @BeforeEach dan diletakan sebelum metode@Test
- import org.junit.jupiter.api.DisplayName;
di notasikan dengan @Display Name yang digunakan untuk memberikan nama deksriptig yang lebih informatik pada pengujian dan akan muncul dalam laporan hasil pengujian yang memudahkan dalam mengenali maksud pengujiannya
- import org.junit.jupiter.api.Nested
@Nested figunakan untuk membuatk nestedclass atau kelas dalam kelas, untuk mengelompokan pengujian berdasrkan kategori atau skenario berbeda
- import org.junit.jupiter.api.Test;
@Test menandai metode sebagai metode pengujian, dan akan di eksekusi oleh Junit saat pengujian dijalankan, dan setiap metode ini menguji suatu aspek spesifik pada kode
- import static org.junit.jupiter.api.Assertions.assertEquals;
Merupakan metode statis dari kelas Assertions yang digunakan untuk memverifikasi ouput yang diharapkan (expected) dengan output yang sebenarnya(actual), jika nilai tidak cocok maka program akan menandai nya dengan gagal/error

6. Listing 20.5 Testing the business logic for an economic flight

Hasil Test:

The screenshot displays the source code of the `AirportTest` class in an IDE. The code includes two test methods:

```

public class AirportTest {
    class EconomyFlightTest {
        @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());
        }

        @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());
        }
    }
}

```

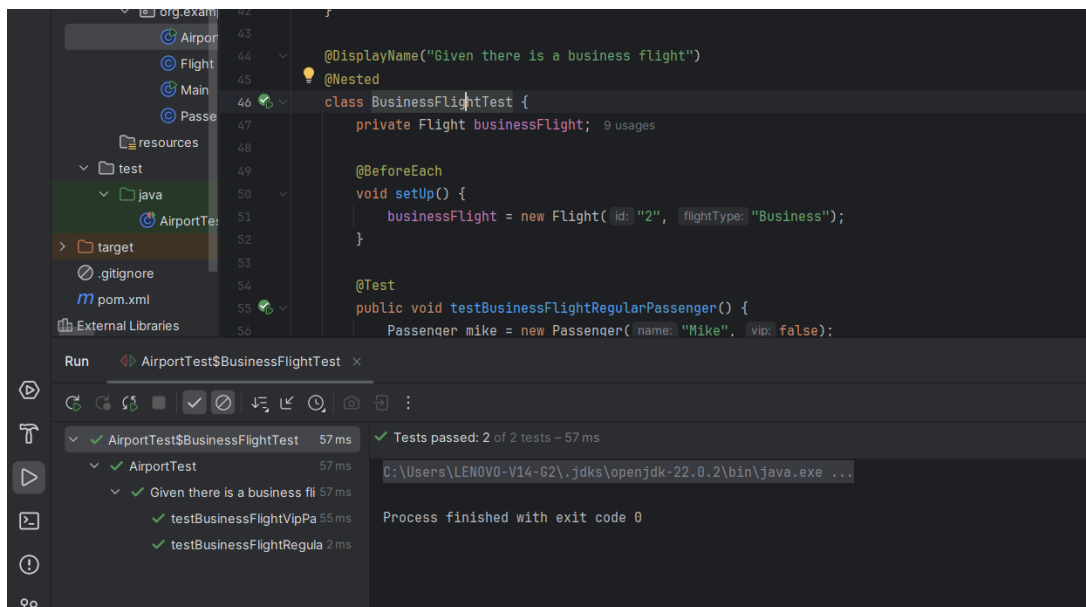
Below the code, the test results are shown in the Run tab. The tests passed successfully:

```

Run: AirportTest$EconomyFlightTest.testEconomyFlightRegularPass...
Tests passed: 1 of 1 test - 38 ms
AirportTest$EconomyFlightTest 38ms
AirportTest 38ms
Given there is an economy 38ms
testEconomyFlightRegu 38ms
Process finished with exit code 0

```

7. Listing 20.6 Testing the business logic for an business flight



8. Listing 20.7 Abstract Flight class, the basis of the hierarchy

Disini saya membedakan proyek antara sebelum menggunakan metode abstraksi dan sesudahnya.

```
package org.example;

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

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);
}
```


9. Listing 20.8 EconomyFlight class, extending the abstract Flight class

```
package org.example;

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;
    }
}
```

10. Listing 20.9 BusinessFlight class, extending the abstract Flight class

```
package org.example;

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;
    }
}
```

11. Listing 20.10 Refactoring propagation into the AirportTest class

```
import org.example.BusinessFlight;
import org.example.EconomyFlight;
import org.example.Flight;
import org.example.Passenger;
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;
import static org.junit.jupiter.api.Assertions.assertAll;

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

        @BeforeEach
        void setUp() {
```

```

        economyFlight = new EconomyFlight("1");
        mike = new Passenger("Mike", false);
        james = new Passenger("James", true);
    }

    @Nested
    @DisplayName("When we have a regular passenger")
    class RegularPassenger {
        @Test
        @DisplayName("Then you can add and remove him from an economy flight")
        public void testEconomyFlightRegularPassenger() {
            assertAll("Verify all conditions for a regular passenger and an economy flight",
                () -> 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())
            );
        }
    }

    @Nested
    @DisplayName("When we have a VIP passenger")
    class VipPassenger {
        @Test
        @DisplayName("Then you can add him but cannot remove him from an economy flight")
        public void testEconomyFlightVipPassenger() {
            assertAll("Verify all conditions for a VIP passenger and an economy flight",
                () -> 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;
        private Passenger mike;
        private Passenger james;

        @BeforeEach
        void setUp() {
            businessFlight = new BusinessFlight("2");
            mike = new Passenger("Mike", false);
            james = new Passenger("James", true);
        }

        @Nested
        @DisplayName("When we have a regular passenger")
        class RegularPassenger {
            @Test
            @DisplayName("Then you cannot add or remove him from a business flight")
            public void testBusinessFlightRegularPassenger() {
                assertAll("Verify all conditions for a regular passenger and a business
flight",
                    () -> assertEquals(false, businessFlight.addPassenger(mike)),
                    () -> assertEquals(0, businessFlight.getPassengersList().size()),
                    () -> assertEquals(false, businessFlight.removePassenger(mike)),
                    () -> assertEquals(0, businessFlight.getPassengersList().size())
                );
            }
        }

        @Nested
        @DisplayName("When we have a VIP passenger")
        class VipPassenger {

```

```
@Test
@DisplayName("Then you can add him but cannot remove him from a business flight")
public void testBusinessFlightVipPassenger() {
    assertAll("Verify all conditions for a VIP passenger and a business flight",
        () -> assertEquals(true, businessFlight.addPassenger(james)),
        () -> assertEquals(1, businessFlight.getPassengersList().size()),
        () -> assertEquals(false, businessFlight.removePassenger(james)),
        () -> assertEquals(1, businessFlight.getPassengersList().size())
    );
}
```

Kendala: Masih terdapat Bug pada IDE yang dipakai, dimana terdapat kondisi bahwa error sudah di fix namun line masih dinyatakan error, atau program tiba tiba tidak dapat di running

LINK GITHUB

<https://github.com/Yorubreak/Pemrograman-Berorientasi-Objek.git>

