**LAPORAN PRAKTIKUM**

**(Junit)**

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



**Disusun Oleh:**

**Nalendra Praja Bredtyoapti Yudo (231511056)**

**Jurusan Teknik Komputer dan Informatika**

**Program Studi D-3 Teknik Informatika**

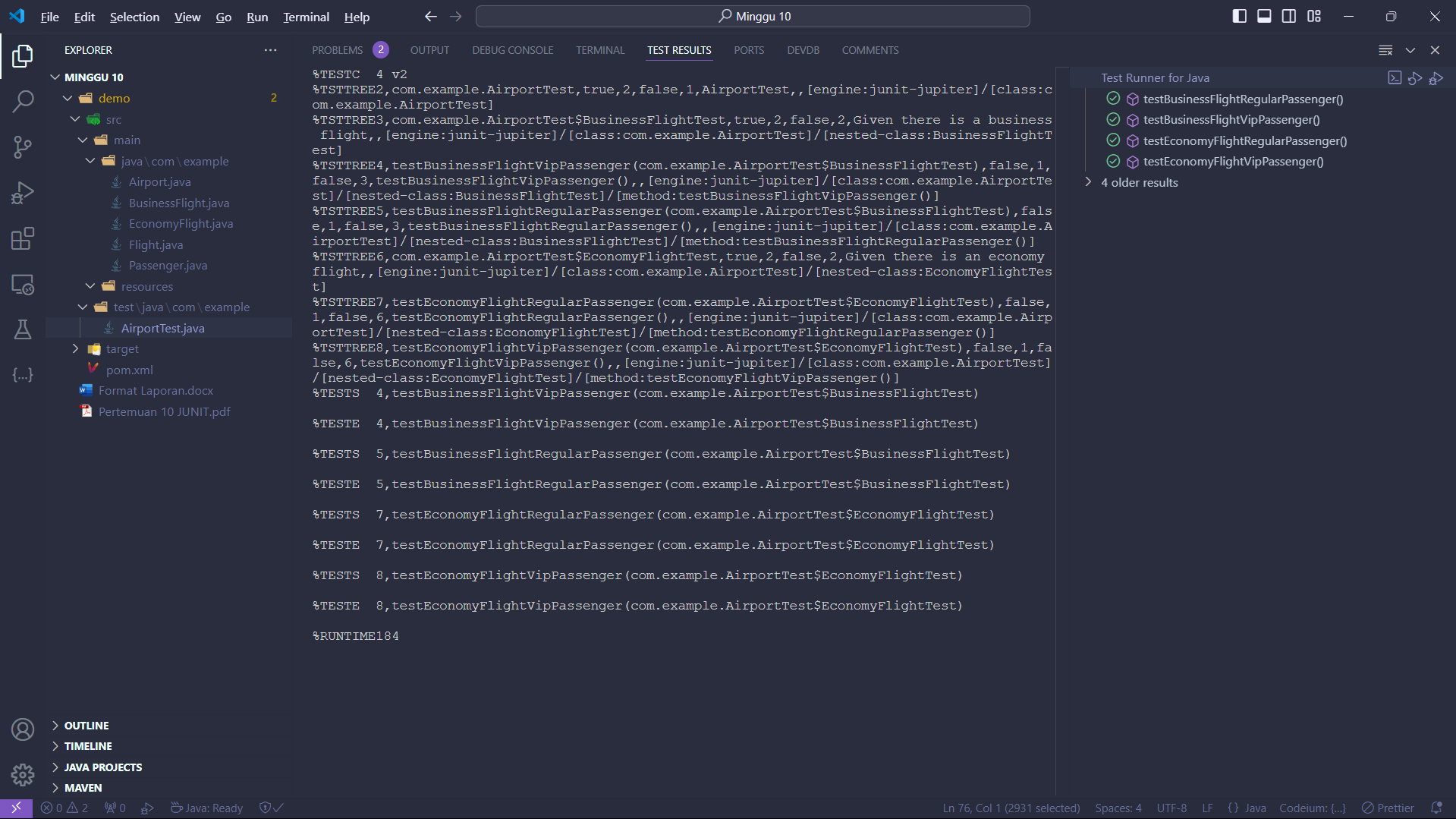
**Politeknik Negeri Bandung**

**2024**

1. Source Code Program

|  |  |
| --- | --- |
| Airport.java | package com.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());          }      }  } |
| BusinessFlight.java | package com.example;  public class BusinessFlight extends Flight {      public BusinessFlight(String id) {          super(id, "Business");      }      @Override      public boolean addPassenger(Passenger passenger) {  *if* (passenger.isVip()) {  *return* super.addPassenger(passenger);          }  *return* false;      }      @Override      public boolean removePassenger(Passenger passenger) {  *return* false;      }  } |
| EconomyFlight.java | package com.example;  public class EconomyFlight extends Flight {      public EconomyFlight(String id) {          super(id, "Economy");      }      @Override      public boolean addPassenger(Passenger passenger) {  *return* passengers.add(passenger);      }      @Override      public boolean removePassenger(Passenger passenger) {  *if* (!passenger.isVip()) {  *return* passengers.remove(passenger); // *Pastikan daftar 'passengers' digunakan*          }  *return* false;      }  } |
| Flight.java | package com.example;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public class Flight {      private String id;      protected List<Passenger> passengers = *new* ArrayList<>();      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);          }      }  } |
| Pasengger.java | 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;      }  } |
| AirportTest.java | package com.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;          @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("2", businessFlight.getId());              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("2", businessFlight.getId());              assertEquals(true, businessFlight.addPassenger(james));              assertEquals(1, businessFlight.getPassengersList().size());              assertEquals("James", businessFlight.getPassengersList().get(0).getName());              assertEquals(false, businessFlight.removePassenger(james));              assertEquals(1, businessFlight.getPassengersList().size());          }      }  } |
| 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>com.example</groupId>      <artifactId>demo</artifactId>      <version>1.0-SNAPSHOT</version>      <properties>          <maven.compiler.source>17</maven.compiler.source>          <maven.compiler.target>17</maven.compiler.target>      </properties>      <dependencies>          <!-- *Dependency untuk JUnit Jupiter API* -->          <dependency>              <groupId>org.junit.jupiter</groupId>              <artifactId>junit-jupiter-api</artifactId>              <version>5.10.0</version>              <scope>test</scope>          </dependency>          <!-- *Dependency untuk JUnit Jupiter Engine* -->          <dependency>              <groupId>org.junit.jupiter</groupId>              <artifactId>junit-jupiter-engine</artifactId>              <version>5.10.0</version>              <scope>test</scope>          </dependency>      </dependencies>      <build>          <plugins>              <plugin>                  <groupId>org.apache.maven.plugins</groupId>                  <artifactId>maven-surefire-plugin</artifactId>                  <version>2.22.2</version>              </plugin>          </plugins>      </build>  </project> |

1. Hasil Implementasi



A screenshot of a computer

Description automatically generated

1. Penjelasan Program

Pada program ini terdapat kelas penerbangan utama, yang berfungsi sebagai kerangka dasar untuk penerbangan, dan kelas turunannya, EconomyFlight dan BusinessFlight. Kelas penerbangan menyimpan data penumpang dan mengatur penambahan atau penghapusan penumpang. Setiap penerbangan memiliki jenis kebijakan penumpang tertentu, seperti kemampuan untuk mendaftarkan atau menghapus penumpang.

Kelas penumpang menampilkan data tentang penumpang, termasuk nama dan status VIP-nya. Penumpang VIP mungkin memiliki keistimewaan tertentu, seperti kemampuan untuk ditambahkan ke penerbangan bisnis tetapi tidak dapat dihapus dari penerbangan tersebut. Ini terlihat dari penggunaan addPassenger dan removePassenger dalam BusinessFlight, yang hanya mengizinkan penumpang VIP untuk ditambahkan tetapi tidak dapat dihapus.

Aturan EconomyFlight lebih fleksibel, memungkinkan penumpang VIP dan reguler. Hanya penumpang non-VIP yang dapat dihapus dari penerbangan. Ini menunjukkan bahwa ada perbedaan dalam cara penumpang ditangani berdasarkan kelas penerbangan. Dibandingkan dengan BusinessFlight, yang memiliki batasan yang lebih ketat untuk menghapus penumpang, metode menambahkan penumpang dan menghapus penumpang di kelas ini lebih mudah dilakukan.

Program ini diuji coba menggunakan framework JUnit. Kelas AirportTest menguji berbagai skenario yang mungkin terjadi pada penerbangan ekonomi dan bisnis, seperti menambah dan menghapus penumpang VIP dan non-VIP. Kode seperti @BeforeEach memastikan bahwa objek penerbangan diinisialisasi sebelum setiap metode uji dijalankan, sedangkan @Test menentukan metode pengujian individu.

Struktur proyek ini didukung oleh file pom.xml untuk mengelola dependensi, dan framework pengujian JUnit tersedia. Metode ini memungkinkan pengembang untuk memastikan bahwa logika bisnis yang digunakan dalam program diuji secara menyeluruh untuk memastikan keakuratan fungsionalitasnya. Penggunaan maven-surefire-plugin dalam pom.xml memudahkan integrasi dan pengembangan berkelanjutan dan membantu menjalankan tes secara otomatis.