
Jurusan Teknik Komputer dan Informatika

Politeknik Negeri Bandung

# Pertemuan 10 Test Driven Development Junit 5

D3 Kelas 2A/2B

Dosen Pengampu :

Zulkifli Arsyad, Irawan Thamrin

# Introduction

- 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 *pass :*
  - Write a failing test before writing new code.
  - Write the smallest piece of code that will make the new test pass.

# Introduction

- In a classical approach, developing a program means we write code and then do some testing by observing its behavior. So, the conventional development cycle goes something like this: [code, test, (repeat)]
- TDD uses a surprising variation: [test, code, (repeat)]
- In fact, it looks like this:  [test, code, refactor, (repeat)]

- Refactoring adalah proses memodifikasi sistem perangkat lunak dengan cara yang tidak memengaruhi perilaku eksternalnya tetapi meningkatkan struktur internalnya. Untuk memastikan perilaku eksternal tidak terpengaruh, kita perlu mengandalkan tes

# The flight-management application

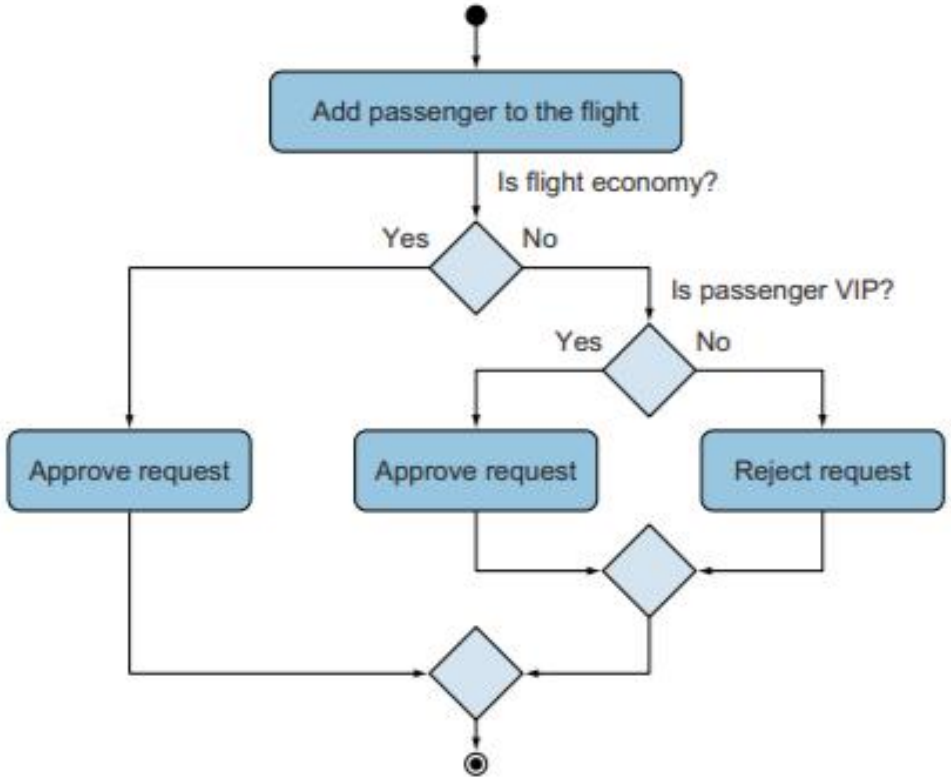


**Figure 20.1 The business logic of adding passengers to a flight: if it is a business flight, only VIP passengers may be added to it. Any passenger can be added to an economy flight.**

# Preparing the flight-management application for TDD

- Apache Maven
  - Apache Maven is a software project management and comprehension tool. Based on the concept of a project object model (POM), Maven can manage a project's build, reporting and documentation from a central piece of information.
- Using dependency **junit-jupiter-api** and **junit-jupiter-engine**

**Listing 20.4 JUnit 5 dependencies added to the pom.xml file**

```
<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>
```

# Annotations JUnit Jupiter

- **@DisplayName** digunakan untuk mendeklarasikan nama kelas pengujian atau metode pengujian beranotasi.
- **@ *BeforeEac*h**
  - Metode yang dianotasi dengan anotasi @ BeforeEach  dijalankan sebelum setiap pengujian. Ini berguna ketika kita ingin mengeksekusi beberapa kode umum  sebelum menjalankan tes.
  - @BeforeEach digunakan untuk memberi sinyal bahwa metode beranotasi harus dieksekusi sebelum setiap metode @Test di kelas pengujian saat ini
- **@test**
  - Menandakan bahwa method tersebut  adalah method pengujian (yang di ujikan)
- **@Nested**
  - Menunjukkan bahwa kelas tersebut adalah kelas pengujian

- https://junit.org/junit5/docs/current/user-guide/

# Example

```
public class AirportTest {
[...]

@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());

    }
}
}
```

1
2
3
4
5
6
7
8

# Bahan Praktikum

- Cobakan Chapter Test-driven development with Junit
  - Listing 20.1 Passanger Class
  - Listing 20.2 Flight Class
  - Listing 20.3 Airport Class, including the Main Method
  - Listing 20.4 Junit 5 Dependencies added to the pom.xml
  - Listing 20.5 Testing the business logic for an economic flight
  - Listing 20.6 Testing the business logic for an business flight
  - Listing 20.7  Abstract Flight class, the basis of the hierarchy
  - Listing 20.8 EconomyFlight class, extending the abstract Flight class
  - Listing 20.9 BusinessFlight class, extending the abstract Flight class
  - Listing 20.10 Refactoring propagation into the AirportTest class