**­­LAPORAN PRAKTIKUM**

**TEKNIK PEMROGRAMAN**

PERTEMUAN KE-10



*Diajukan untuk memenuhi mata kuliah Teknik Pemrograman*

Dosen:

Zulkifli Arsyad, S.Kom., M.T

­­­­­­Link Github: <https://github.com/ElLukman/Penugasan-Tekpro>

Disusun Oleh :

LUKMAN AHMAD

241524050

1B

DAFTAR ISI

Contents

[Laporan Praktikum Clean Code Menggunakan SonarQube Dengan Docker Sebagai Perantara 3](#_Toc197356750)

[Pendahuluan 3](#_Toc197356751)

[Tujuan 4](#_Toc197356752)

[Analisis Masalah 4](#_Toc197356753)

[Sebelum Melakukan Refactor 4](#_Toc197356754)

[Code Before 4](#_Toc197356755)

[Setelah Melakukan Refactor 11](#_Toc197356756)

[Code After 11](#_Toc197356757)

# Laporan Praktikum Clean Code Menggunakan SonarQube Dengan Docker Sebagai Perantara

## Pendahuluan

Clean code adalah konsep dalam pengembangan perangkat lunak yang menekankan pentingnya menulis kode yang mudah dibaca, dipahami, dan dikelola. Kode yang bersih (clean code) tidak hanya berfungsi dengan benar, tetapi juga memiliki struktur yang jelas, nama variabel dan fungsi yang deskriptif, serta menghindari kompleksitas yang tidak perlu. Bagi mahasiswa Teknik Informatika, memahami clean code sangat penting karena ini adalah fondasi untuk membangun aplikasi yang scalable dan mudah dikembangkan.

Dalam kasus ini, saya menggunakan SonarQube dan Docker sebagai alat untuk menganalisis kualitas kode. SonarQube adalah platform yang membantu mengidentifikasi masalah seperti bug, kerentanan, dan "code smells" (tanda-tanda kode yang buruk). Docker digunakan untuk mengemas aplikasi dan dependensinya dalam container, memastikan konsistensi lingkungan pengembangan.

**Masalah yang Ditangani:**

1. Keterbacaan Kode:

Kode yang ditulis harus mudah dipahami oleh developer lain. Contohnya, penggunaan nama kelas seperti EconomyFlight dan BusinessFlight yang jelas menggambarkan fungsinya.

1. Prinsip SOLID:

Kode mengikuti prinsip SOLID, seperti Single Responsibility Principle (SRP) di mana setiap kelas memiliki satu tanggung jawab. Misalnya, kelas Passenger hanya menangani data penumpang, sedangkan kelas Flight menangani operasi penerbangan.

1. Pengujian (Testing):

Pengujian dengan JUnit (seperti di AirportTest.java) memastikan bahwa setiap komponen berfungsi sesuai harapan. Ini adalah bagian dari clean code untuk meminimalkan bug.

1. Konsistensi dan Standar:

Penggunaan alat seperti SonarQube membantu memastikan kode memenuhi standar kualitas, seperti menghindari duplikasi kode atau metode yang terlalu panjang.

**Manfaat Clean Code:**

1. Memudahkan kolaborasi dalam tim.
2. Mempercepat proses debugging dan pengembangan fitur baru.
3. Mengurangi biaya maintenance dalam jangka panjang.

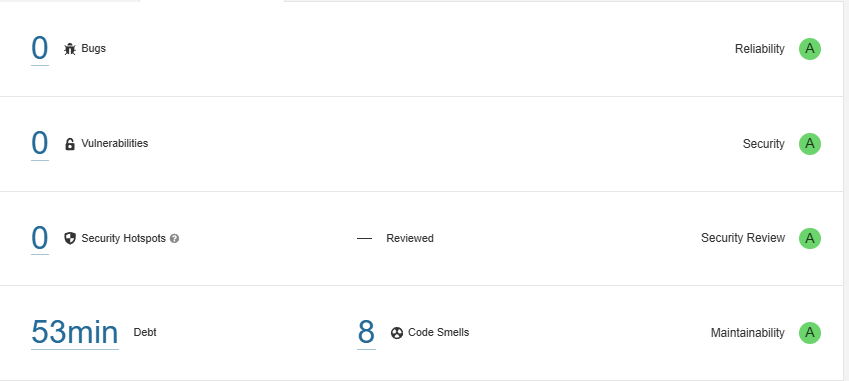
Dengan memahami dan menerapkan clean code, mahasiswa dapat membangun kebiasaan baik yang akan sangat berguna dalam karir pemrograman mereka. Tools seperti SonarQube dan Docker membantu memvalidasi praktik ini secara otomatis.

## Tujuan

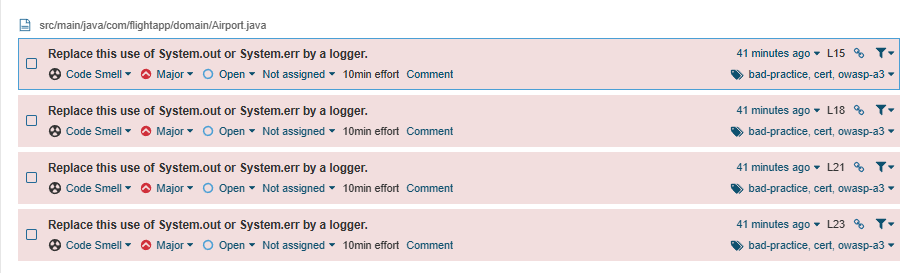
1. Memahami konsep clean code.
2. Menggunakan SonarQube untuk analisis kualitas kode.
3. Memanfaatkan Docker untuk konsistensi lingkungan pengembangan.
4. Mengembangkan program agar sesuai dengan prinsip clean code.
5. Mendokumentasikan perubahan kode yang sudah dianalisis melalui SonarQube.

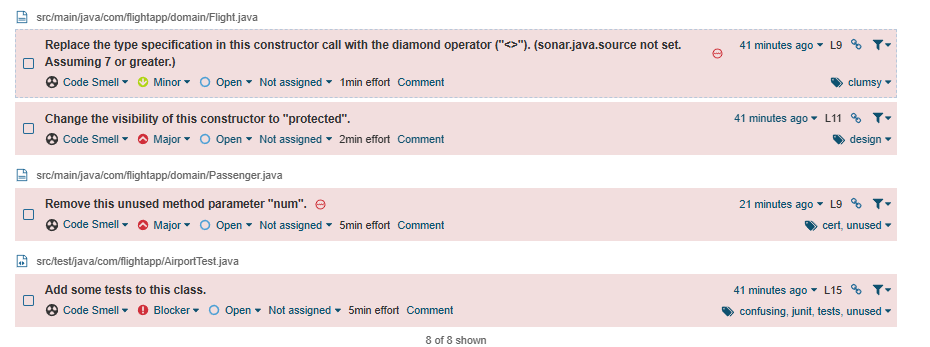
## Analisis Masalah

### Sebelum Melakukan Refactor



**List Code Smells**





### Code Before

File main\java\com\flightapp\domain

Airport.java

|  |
| --- |
| package com.flightapp.domain;  public class Airport {      public static *void* main(*String*[] *args*) {  *Flight* economyFlight = new EconomyFlight("1");  *Flight* businessFlight = new BusinessFlight("2");  *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());          }      }  } |

BussinesFlight.java

|  |
| --- |
| package com.flightapp.domain;  public class BusinessFlight extends *Flight* {      public BusinessFlight(*String* *id*) {          super(id);      }      @*Override*      public *boolean* addPassenger(*Passenger* *passenger*) {          if (passenger.isVip()) {              return getPassengers().add(passenger);          }          return false;      }      @*Override*      public *boolean* removePassenger(*Passenger* *passenger*) {          return false;      }  } |

EconomyFlight.java

|  |
| --- |
| package com.flightapp.domain;  public class EconomyFlight extends *Flight* {      public EconomyFlight(*String* *id*) {          super(id);      }      @*Override*      public *boolean* addPassenger(*Passenger* *passenger*) {          return getPassengers().add(passenger);      }      @*Override*      public *boolean* removePassenger(*Passenger* *passenger*) {          if (!passenger.isVip()) {              return getPassengers().remove(passenger);          }          return false;      }  } |

Flight.java

|  |
| --- |
| package com.flightapp.domain;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public abstract class Flight {      private *String* id;      private *List*<*Passenger*> passengers = new *ArrayList*<*Passenger*>();      public Flight(*String* *id*) {          this.id = id;      }      public *String* getId() {          return id;      }      protected *List*<*Passenger*> getPassengers() {          return passengers;      }        public *List*<*Passenger*> getPassengersList() {          return Collections.unmodifiableList(passengers);      }      public abstract *boolean* addPassenger(*Passenger* *passenger*);      public abstract *boolean* removePassenger(*Passenger* *passenger*);  } |

Passenger.java

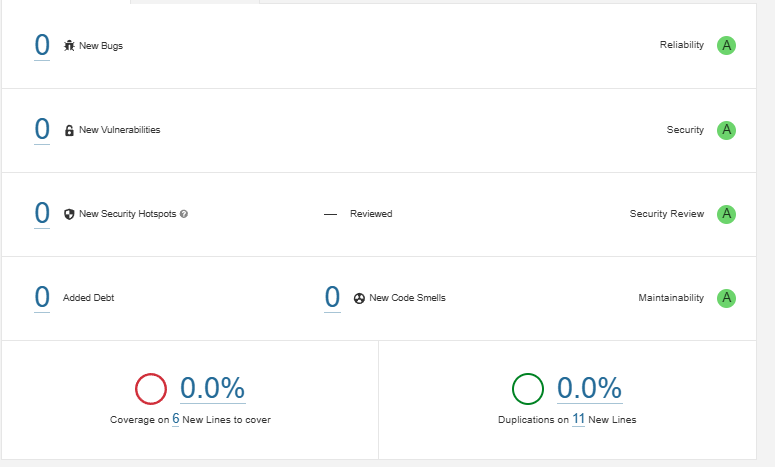
|  |
| --- |
| package com.flightapp.domain;  public class Passenger {      private *String* name;      private *boolean* vip;      private *int* num;      private *int* d;      public Passenger (*String* *name*, *boolean* *vip*, *int* *num*)      {          this.name = name;          this.vip = vip;      }      public *String* getName()      {          return name;      }      public *boolean* isVip()      {          return vip;      }      public *int* getNum()      {          return num;      }      public *int* getD()      {          return d;      }  } |

test\java\com\flightapp

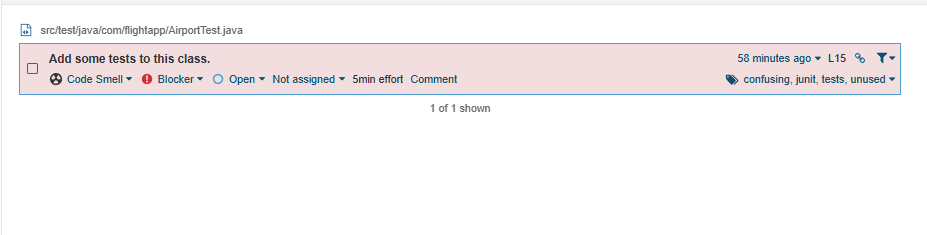
AirportTest.java

|  |
| --- |
| package com.flightapp;  import static org.junit.jupiter.api.Assertions.assertEquals;  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 com.flightapp.domain.BusinessFlight;  import com.flightapp.domain.EconomyFlight;  import com.flightapp.domain.Flight;  import com.flightapp.domain.Passenger;  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());          }      }      class BusinessFlightTest {          private *Flight* businessFlight;          @*BeforeEach*  *void* setUp() {              businessFlight = new BusinessFlight("1");          }          @*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());          }      }  } |

### Setelah Melakukan Refactor



**List Code Smells**

****

SonarQube hanya memberikan rekomendasi untuk menambahkan test sebagai bagian dari praktik code quality yang baik, namun tidak mewajibkannya karena tidak memengaruhi fungsi program secara langsung. Peringatan tersebut bersifat informatif dan bertujuan mendorong developer untuk memastikan setiap kelas, terutama kelas pengujian, benar-benar memuat pengujian yang relevan.

### Code After

File main\java\com\flightapp\domain

Airport.java

|  |
| --- |
| package com.flightapp.domain;  import java.util.logging.Logger;  public class Airport {      private static final *Logger* logger = Logger.getLogger(Airport.class.getName());      public static *void* main(*String*[] *args*) {  *Flight* economyFlight = new EconomyFlight("1");  *Flight* businessFlight = new BusinessFlight("2");  *Passenger* james = new Passenger("James", true);  *Passenger* mike = new Passenger("Mike", false);          businessFlight.addPassenger(james);          businessFlight.removePassenger(james);          businessFlight.addPassenger(mike);          economyFlight.addPassenger(mike);          logger.info("Business flight passengers list:");          for (*Passenger* passenger : businessFlight.getPassengersList()) {              logger.info(passenger.getName());          }          logger.info("Economy flight passengers list:");          for (*Passenger* passenger : economyFlight.getPassengersList()) {              logger.info(passenger.getName());          }      }  } |

BussinesFlight.java

|  |
| --- |
| package com.flightapp.domain;  public class BusinessFlight extends *Flight* {      public BusinessFlight(*String* *id*) {          super(id);      }      @*Override*      public *boolean* addPassenger(*Passenger* *passenger*) {          if (passenger.isVip()) {              return getPassengers().add(passenger);          }          return false;      }      @*Override*      public *boolean* removePassenger(*Passenger* *passenger*) {          return false;      }  } |

EconomyFlight.java

|  |
| --- |
| package com.flightapp.domain;  public class EconomyFlight extends *Flight* {      public EconomyFlight(*String* *id*) {          super(id);      }      @*Override*      public *boolean* addPassenger(*Passenger* *passenger*) {          return getPassengers().add(passenger);      }      @*Override*      public *boolean* removePassenger(*Passenger* *passenger*) {          if (!passenger.isVip()) {              return getPassengers().remove(passenger);          }          return false;      }  } |

Flight.java

|  |
| --- |
| package com.flightapp.domain;  import java.util.ArrayList;  import java.util.Collections;  import java.util.List;  public abstract class Flight {      private *String* id;      private *List*<*Passenger*> passengers = new *ArrayList*<>();      protected Flight(*String* *id*) {          this.id = id;      }      public *String* getId() {          return id;      }      protected *List*<*Passenger*> getPassengers() {          return passengers;      }        public *List*<*Passenger*> getPassengersList() {          return Collections.unmodifiableList(passengers);      }      public abstract *boolean* addPassenger(*Passenger* *passenger*);      public abstract *boolean* removePassenger(*Passenger* *passenger*);  } |

Passenger.java

|  |
| --- |
| package com.flightapp.domain;  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;      }  } |

test\java\com\flightapp

AirportTest.java

|  |
| --- |
| package com.flightapp;  import static org.junit.jupiter.api.Assertions.assertEquals;  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 com.flightapp.domain.BusinessFlight;  import com.flightapp.domain.EconomyFlight;  import com.flightapp.domain.Flight;  import com.flightapp.domain.Passenger;  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());          }      }      class BusinessFlightTest {          private *Flight* businessFlight;          @*BeforeEach*  *void* setUp() {              businessFlight = new BusinessFlight("1");          }          @*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());          }      }  } |