VERSION 1.1

SEPTEMBER 13, 2023

PEMROGRAMAN LANJUT

MODUL 1 – PROGRAM CORRECTNESS

TIM PENYUSUN:

- WILDAN SUHARSO, S.KOM., M.KOM
- LARYNT SAWFA KENANGA
- AHYA NIKA SALSABILA

LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN LANJUT

PERSIAPAN MATERI

Mahasiswa harus memahami konsep OOP yang telah dipelajari sebelumnya.

TUJUAN

1. Mahasiswa dapat mengidentifikasi jenis-jenis error.
2. Mahasiswa mampu mengidentifikasi spesifikasi kode.
3. Mahasiswa dapat mengimplementasikan exception handling.
4. Mahasiswa dapat melakukan proses code review.
5. Mahasiswa mampu melakukan pengujian (testing) menggunakan **Maven**.

TARGET MODUL

Mahasiswa akan mendapatkan pemahaman tentang konsep program correctness dan metode yang diterapkan untuk mencapainya.

PERSIAPAN SOFTWARE/APLIKASI

1. Java Development Kit.
2. Text Editor / IDE (**Preferably** Intellij IDEA, Visual Studio Code, Netbeans, etc).

TEORI

1. Types Of Errors

a. Syntax Error

Error tata bahasa (sintaks) merupakan jenis error yang paling banyak terjadi dalam pembuatan program. Namun error ini paling mudah terdeteksi karena umumnya compiler atau interpreter dari masing-masing bahasa program akan melakukan pengecekan sebelum program dijalankan (saat dikompilasi).Lokasi baris yang menyebabkan error juga biasanya sudah ditunjukkan. Hanya perlu kejelian untuk memperbaikinya. Contoh dalam Java:

```
class SyntaxErrorExample {
    public static void main(String[] args) {
        // Contoh syntax error
        int x = 5;
        System.out.println("Nilai x: " + x);
    }
}
```

Pada contoh di atas, terjadi kesalahan sintaks karena kurangnya tanda semicolon (;) setelah deklarasi variabel x.

b. Logic Error

Jenis error yang satu ini merupakan jenis error yang paling susah dideteksi karena terjadinya bukan karena kesalahan penulisan (sintaks) atau kesalahan proses runtime, namun kesalahan dari sisi programmer, dalam hal ini algoritma yang digunakan. Karena logikanya salah, tentunya output yang dihasilkan juga akan salah. Untuk mendeteksi letak kesalahannya, bukanlah hal yang mudah. Terkadang kita harus merunut algoritma yang digunakan baris per baris (line byline). Contoh dalam Java:

```java
public class LogicErrorExample2 {
    public static void main(String[] args) {
        // Contoh logic error
        int a = 7;
        int b = 2;
        int average = (a + b) / 2; // Rata-rata tidak akan benar jika kita tidak mengonversi hasil ke double.
        System.out.println("Rata-rata: " + average);
    }
}
```

Pada contoh di atas, kita mencoba untuk menghitung rata-rata dari dua bilangan a dan b. Namun, karena operator pembagian (/) digunakan untuk tipe data integer, maka hasil dari (a + b) / 2 akan dibulatkan ke bilangan integer. Sebagai contoh, jika a = 7 dan b = 2, maka hasil dari (a + b) / 2 adalah 4, bukan 4.5. Seharusnya rata-rata dari 7 dan 2 adalah 4.5.

c. Runtime Error

Tingkatan error selanjutnya adalah runtime error. Dimana error ini akan terdeteksi saat program dijalankan (di-running). Penyebabnya beragam, pada umumnya karena terjadi kesalahan dalam proses input, perhitungan dan juga dalam proses output. Sebagai contoh yang banyak terjadi adalah error runtime karena pembagian suatu bilangan dengan nol. Lihat contoh program berikut ini! Secara sintaks tentu tidak terdapat error, namun jika dijalankan, operasi pembagian pada baris ke-6 akan menyebabkan error “division by zero”. Contoh dalam Java:

```java
public class RuntimeExample {
    public static void main(String[] args) {
        // Contoh Runtime error
        int a = 5;
        int b = 0;
        int result = a / b;
        System.out.println("Hasil pembagian: " + result);
    }
}
```

Pada contoh di atas, tidak ada kesalahan sintaks, tetapi terjadi error karena kita mencoba untuk membagi angka dengan nol, yang menyebabkan ArithmeticException (runtime error).

2. The Concept Of a Specification

Spesifikasi merupakan pernyataan yang tepat mengenai efek yang ingin dicapai oleh sebuah program. Spesifikasi program harus secara jelas menyatakan apa yang harus dilakukan oleh program tanpa membuat komitmen apa pun tentang bagaimana hal ini dilakukan. Untuk program yang dimaksudkan untuk diakhiri, spesifikasi program dapat berupa spesifikasi input-output yang menggambarkan pemetaan yang diinginkan dari kumpulan nilai input ke kumpulan nilai output.

3. Defensive Programming (Error Handling)

Defensive Programming adalah pengembangan perangkat lunak komputer yang mempertimbangkan semua keadaan tak terduga yang dapat menimbulkan masalah. Hal ini memungkinkan perangkat lunak untuk berperilaku dengan benar terlepas dari masukan yang diberikan. Teknik pengembangan ini utamanya digunakan ketika mengkodekan perangkat lunak yang harus sangat andal, aman, dan terlindung dari upaya-upaya jahat. Teknik Pemrograman Defensif membantu meningkatkan Perangkat Lunak dan Source code melalui:

- Meningkatkan Kualitas Umum: Meminimalkan jumlah bug dan masalah yang dapat muncul pada kode.
- Mengembangkan Kode yang Dapat Dipahami: Kode sumber yang ditulis dengan teknik pengkodean defensif mudah dibaca dan dipahami. Hal ini membuatnya mudah disetujui dalam audit kode.
- Mengembangkan perangkat lunak yang akan memberikan output yang benar terlepas dari input yang diberikan.

Contoh dalam Java:

```java
public class SafeDivision {
    1 usage
    public static double divide(double numerator, double denominator) throws IllegalArgumentException {
        if (denominator == 0) {
            throw new IllegalArgumentException("Denominator cannot be zero.");
        }
        return numerator / denominator;
    }

    public static void main(String[] args) {
        double numerator = 10;
        double denominator = 0;
        try {
            double result = divide(numerator, denominator);
            System.out.println("Result: " + result);
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

4. Code Reviews

Code reviews, yang juga dikenal sebagai peer reviews, bertindak sebagai jaminan kualitas terhadap code base.

Code review adalah penilaian metodis terhadap kode yang dirancang untuk mengidentifikasi bug, meningkatkan kualitas kode, dan membantu pengembang mempelajari source code.

Setelah pengembang perangkat lunak menyelesaikan pengkodean, code review merupakan langkah penting dalam proses pengembangan perangkat lunak untuk mendapatkan opini kedua tentang solusi dan implementasi sebelum digabungkan ke dalam upstream branch seperti feature branch atau main branch. Reviewer juga dapat bertindak sebagai langkah kedua dalam mengidentifikasi bug, masalah logika, permasalahan yang belum ditemukan, atau masalah lainnya.

Peninjau dapat berasal dari tim atau kelompok mana pun selama mereka adalah ahli domain. Jika baris kode mencakup lebih dari satu domain, dua orang ahli harus meninjau kode tersebut. Contoh dalam Java:

```java
public class SimpleCalculator {
    2 usages
    public int add(int a, int b) {
        return a + b;
    }

    2 usages
    public int subtract(int a, int b) {
        return a - b;
    }

    public static void main(String[] args) {
        SimpleCalculator calculator = new SimpleCalculator();

        // Testing the add() method
        int sum1 = calculator.add( a: 5,  b: 3);
        System.out.println("5 + 3 = " + sum1); // Expected output: 8

        int sum2 = calculator.add( a: -2,  b: 7);
        System.out.println("-2 + 7 = " + sum2); // Expected output: 5

        // Testing the subtract() method
        int diff1 = calculator.subtract( a: 10,  b: 4);
        System.out.println("10 - 4 = " + diff1); // Expected output: 6

        int diff2 = calculator.subtract( a: 3,  b: 8);
        System.out.println("3 - 8 = " + diff2); // Expected output: -5
    }
}
```

5. Testing Fundamentals and Test-Case Generation

Tujuan dari testing adalah untuk menemukan bug dalam sebuah sistem atau aplikasi. Pembuatan test case adalah proses membangun rangkaian uji coba untuk mendeteksi kesalahan sistem. Test suite adalah sekelompok kasus uji yang relevan yang digabungkan menjadi satu. Pembuatan test case adalah proses yang paling penting dan mendasar dalam pengujian perangkat lunak.

Ada beberapa teknik yang tersedia untuk menghasilkan test case:

- Goal-oriented approach - Tujuan dari Goal-oriented approach adalah untuk mencakup bagian, pernyataan, atau fungsi tertentu. Di sini jalur eksekusi tidak penting, tetapi menguji tujuan adalah tujuan utama.

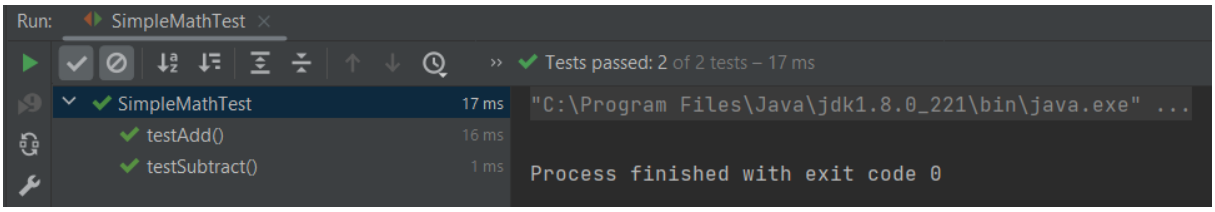
- Random approach - Random approach menghasilkan kasus uji berdasarkan asumsi error dan kesalahan sistem.
- Specification-based technique - Model ini menghasilkan kasus uji berdasarkan spesifikasi kebutuhan formal.
- Source-code-based technique - Pendekatan Source-code-based technique mengikuti jalur aliran kontrol yang akan diuji, dan test case dibuat sesuai dengan itu. Pendekatan ini menguji jalur eksekusi.
- Sketch-diagram-based approach - Jenis pendekatan pembuatan kasus ini mengikuti diagram Unified Modeling Language (UML) untuk merumuskan test case.

  Contoh Dalam Java:

```
public class SimpleMath {
    1 usage
    public int add(int a, int b) {
        return a + b;
    }

    1 usage
    public int subtract(int a, int b) {
        return a - b;
    }
}
```

```java
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

// Test class for Testing Fundamentals and Test-Case Generation
public class SimpleMathTest {
    @Test
    public void testAdd() {
        SimpleMath math = new SimpleMath();
        int result = math.add( a: 3,  b: 5);
        Assertions.assertEquals( expected: 8, result);
    }

    @Test
    public void testSubtract() {
        SimpleMath math = new SimpleMath();
        int result = math.subtract( a: 10,  b: 4);
        Assertions.assertEquals( expected: 6, result);
    }
}
```

Hasil testing:



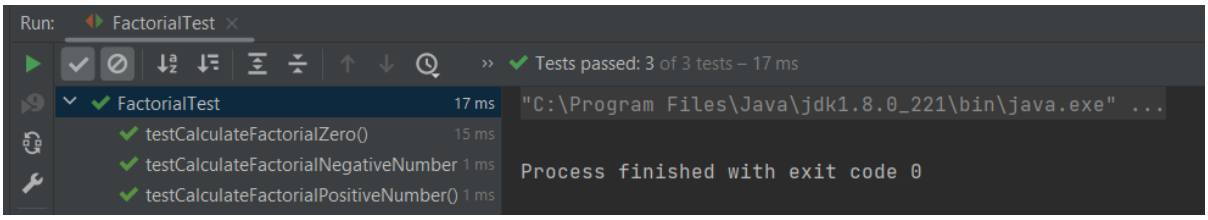
6. Unit Testing

Unit testing adalah proses pengembangan perangkat lunak di mana bagian terkecil yang dapat diuji dari sebuah aplikasi, yang disebut unit, diperiksa secara individual untuk memastikan operasi yang tepat. Pengembang perangkat lunak dan terkadang staf QA menyelesaikan pengujian unit selama proses pengembangan. Tujuan utama unit testing adalah mengisolasi kode yang ditulis untuk diuji dan menentukan apakah kode tersebut berfungsi sebagaimana mestinya. Contoh dalam Java:

```java
public class Factorial {
    3 usages
    public int calculateFactorial(int n) {
        if (n < 0) {
            throw new IllegalArgumentException("Input must be a non-negative integer.");
        }
        if (n == 0 || n == 1) {
            return 1;
        }
        int result = 1;
        for (int i = 2; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

```java
// Test class for Unit Testing
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

public class FactorialTest {
    @Test
    public void testCalculateFactorialPositiveNumber() {
        Factorial factorial = new Factorial();
        int result = factorial.calculateFactorial( n: 5);
        Assertions.assertEquals( expected: 120, result);
    }

    @Test
    public void testCalculateFactorialZero() {
        Factorial factorial = new Factorial();
        int result = factorial.calculateFactorial( n: 0);
        Assertions.assertEquals( expected: 1, result);
    }

    @Test
    public void testCalculateFactorialNegativeNumber() {
        Factorial factorial = new Factorial();
        Assertions.assertThrows(IllegalArgumentException.class, () -> {
            factorial.calculateFactorial( n: -5);
        });
    }
}
```

Hasil testing:



LATIHAN

LATIHAN 1: Logic Error

Berikut terdapat sebuah fungsi yang seharusnya mengembalikan nilai faktorial dari sebuah angka, namun memiliki kesalahan logika. Identifikasi dan perbaiki kesalahan tersebut.

```
public class LogicErrorExample {
    public static void main(String[] args) {
        System.out.println(factorial(5));  // Seharusnya menghasilkan 120
    }

    public static int factorial(int n) {
        int result = 0;
        for (int i = 1; i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

LATIHAN 2: Exception Handling

Buatlah sebuah program sederhana dalam bahasa pemrograman Java yang meminta input dari pengguna. Jika pengguna memasukkan input selain angka, temukan kesalahan tersebut dengan menggunakan exception handling dan tampilkan pesan kesalahan yang sesuai.

```java
import java.util.Scanner;

public class ExceptionHandlingExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Masukkan sebuah angka:");

        try {
            int number = scanner.nextInt();
            System.out.println("Anda memasukkan angka: " + number);
        } catch (Exception e) {
            System.out.println("Input bukan angka!");
        }
    }
}
```

Pada kode di atas, program menggunakan exception handling untuk menangani kesalahan saat pengguna memasukkan input yang bukan angka. Jika pengguna memasukkan input selain angka, blok catch akan dieksekusi dan pesan "Input bukan angka!" akan ditampilkan.

Output:

```
Masukkan sebuah angka:
abc
Input bukan angka!
```

```
Masukkan sebuah angka:
25
Anda memasukkan angka: 25
```

LATIHAN 3: Code Review

Code review dan identifikasi potensi kesalahan atau peningkatan yang dapat dilakukan pada kode dibawah ini

```java
public class ReviewExample {
    public static void main(String[] args) {
        System.out.println(add(5, 3))
    }

    public static int add(int a, int b) {
        return a + b;
    }
}
```

Output :

```
8
```

TUGAS

TUGAS 1: Analisis Kesalahan

Program berikut seharusnya menghitung nilai rata-rata dari sebuah array. Namun, terdapat beberapa kesalahan dalam kode tersebut. Temukan dan perbaiki kesalahan-kesalahan tersebut. Setelah itu, tulislah analisis tentang kesalahan yang ditemukan.

```java
public class AverageCalculation {
    public static void main(String[] args) {
        int[] numbers = {10, 20, 30, 40, 50};
        int sum;

        for (int i = 0; i <= numbers.length; i++) {
            sum += numbers[i];
        }

        double average = sum / numbers.length;
        System.out.println("Rata-rata: " + average);
    }
}
```

TUGAS 2: Complex Exception Handling

Buatlah sebuah program yang mengambil input dari pengguna berupa daftar angka yang dipisahkan oleh spasi. Program akan menghitung rata-rata dari angka-angka tersebut. Namun, program juga harus mengatasi beberapa situasi yang mungkin terjadi, seperti kesalahan input atau division by zero.

TUGAS 3: Palindrome Checker

Buatlah program yang menerima input kata atau kalimat dari pengguna dan menentukan apakah input tersebut adalah palindrome (kata atau kalimat yang dapat dibaca sama dari depan maupun dari belakang) atau tidak. Program harus mengabaikan spasi, huruf kapital atau kecil, dan tanda baca.

TUGAS 4: Testing

Diberikan sebuah kelas yang menghitung luas lingkaran sebagai berikut:

```
public class Lingkaran {
    private static final double PI = 3.14;

    public static double hitungLuas(double radius) {
        if (radius <= 0) {
            throw new IllegalArgumentException("Radius harus bernilai positif");
        }
        return PI * radius * radius;
    }
}
```

Buatlah unit test untuk kelas di atas dengan mempertimbangkan beberapa skenario, termasuk:

- Pengujian dengan radius positif.
- Pengujian dengan radius negatif.
- Pengujian dengan radius bernilai 0.

Berikan penjelasan dari setiap skenario pengujian di atas

REFERENSI

Program specification - Oxford Reference

Defensive Programming Techniques Explained with Examples | GoLinuxCloud

What is a code review? | GitLab

What is Test Case Generation? - Definition from Techopedia

What is Unit Testing? Definition from WhatIs.com (techtarget.com)

KRITERIA & DETAIL PENILAIAN

Kriteria Penilaian		Nilai
Latihan 1		5
Identifikasi Kesalahan Logika	2.5	
Perbaikan Kesalahan Logika	2.5	
Latihan 2		5
Penggunaan Exception Handling	2.5	
Pesan Kesalahan yang Tepat	2.5	
Latihan 3		10
Identifikasi Kesalahan Sintaks	2.5	
Perbaikan Kesalahan Sintaks	2.5	
Kualitas Kode yang Diperbaiki	5	
Tugas 1		15
Identifikasi Kesalahan Kode	5	
Perbaikan Kesalahan Kode	5	
Analisis Kesalahan yang Ditulis	5	
Tugas 2		10
Exception Handling yang Benar	5	
Penanganan Situasi yang Tepat	5	
Tugas 4		10
Pengenalan Palindrome yang Benar	5	
Implementasi Pengecekan yang Akurat	5	
Tugas 5		10
Penggunaan Unit Test yang Tepat	5	
Pengujian pada Skenario yang Disebutkan	5	
Pemahaman		35
Total		**100**