

VERSI 0.1
JANUARI 2025



PEMROGRAMAN LANJUT

MODUL 1 - PROGRAM CORRECTNESS

DISUSUN OLEH:

Ir. Wildan Suharso, M.Kom.
Muhammad Ega Faiz Fadlillah
M. Ramadhan Titan Dwi .C

TIM LABORATORIUM INFORMATIKA
UNIVERSITAS MUHAMMADIYAH MALANG

PENDAHULUAN

TUJUAN

1. Mahasiswa mampu menganalisis dan menyusun spesifikasi formal untuk sebuah program.
2. Mahasiswa mampu mengidentifikasi serta membedakan berbagai jenis error dalam kode.
3. Mahasiswa dapat menerapkan teknik code review secara sistematis untuk menemukan potensi masalah dan meningkatkan kualitas kode.
4. Mahasiswa dapat mengimplementasikan strategi defensive programming.

TARGET MODUL

1. Mahasiswa dapat mengembangkan program yang tidak hanya berjalan sesuai fungsionalitas yang diharapkan, tetapi juga memiliki tingkat correctness yang tinggi.
2. Mahasiswa mampu menulis kode yang bersih, mudah dibaca, dan kokoh (robust) dengan menerapkan spesifikasi yang jelas, melakukan peninjauan kode yang efektif, serta mengintegrasikan mekanisme pertahanan terhadap error.

PERSIAPAN

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

KEYWORDS

Program Correctness, Type of Errors, Code Review, Defensive Programming

TABLE OF CONTENTS

PENDAHULUAN	2
TUJUAN.....	2
TARGET MODUL.....	2
PERSIAPAN.....	2
KEYWORDS.....	2
TABLE OF CONTENTS.....	2
TEORI	4
1. Types Of Errors.....	4
a. Syntax Error.....	4
b. Logic Error.....	5
c. Runtime Error.....	6
d. Resource Error.....	8
e. Time Limit Exceeded (TLE) Error.....	10
2. The Concept Of Specification.....	12



3. Defensive Programming.....	14
a. Tujuan Defensive Programming.....	14
b. Teknik-Teknik Implementasi.....	14
4. Code Review.....	16
5. Unit Testing.....	18
a. Keuntungan dari Unit Testing.....	18
b. Unit Testing dalam Siklus Pengembangan.....	18
c. Best Practice dalam Unit Testing.....	19
d. Membuat Unit Testing dengan JUnit di IntelliJ IDEA.....	20
REFERENSI.....	23
CODELAB.....	24
CODELAB 1.....	24
CODELAB 2.....	24
TUGAS.....	25
TUGAS 1.....	25
TUGAS 2.....	25
TUGAS 3.....	25
TUGAS 4.....	26
KRITERIA & DETAIL PENILAIAN.....	27



TEORI

1. Types Of Errors

a. Syntax Error

Syntax Error adalah jenis error yang sering terjadi karena kesalahan dalam aturan tata bahasa (syntax) dari bahasa pemrograman yang digunakan. Error ini biasanya sangat mudah dideteksi oleh compiler atau interpreter karena mereka akan memberi tahu lokasi baris yang menyebabkan error sebelum program dijalankan. Contoh umum syntax error adalah ketika ada tanda kurung atau tanda baca yang hilang.

Contoh Syntax Error :

```

1 package Modul1;
2
3 public class SyntaxErrorException {
4     public static void main(String[] args) {
5         System.out.println("Hello World")
6     }
7 }

```

Pada program di atas, terdapat syntax error di baris ke-5 karena kurangnya tanda titik koma (;) setelah perintah `System.out.println("Hello, World!")`. Jika program ini dijalankan, compiler akan memberikan pesan error yang menunjukkan bahwa tanda titik koma diperlukan di akhir pernyataan tersebut. Dan terminal menunjukkan error: `Syntax error on token ";", ; expected.`

Untuk memperbaikinya, tambahkan tanda titik koma (;) di akhir baris ke-5 seperti pada gambar dibawah. Setelah perbaikan tersebut, program dapat dijalankan dengan benar.

```

1 package Modul1;
2
3 public class SyntaxErrorException {
4     public static void main(String[] args) {
5         System.out.println("Hello World");
6     }
7 }

```



b. Logic Error

Logic Error adalah jenis error yang lebih sulit untuk dideteksi dibandingkan dengan syntax error atau runtime error. Kesalahan ini terjadi ketika algoritma atau logika yang digunakan dalam program tidak sesuai dengan yang diinginkan, meskipun program berjalan tanpa adanya kesalahan sintaksis atau runtime. Kesalahan logika biasanya terjadi pada pemrogram yang salah dalam merancang algoritma atau menggunakan operator yang tidak tepat.

Contoh Logic Error :

```
package Modul1;

public class LogicErrorExample {
    public static void main(String[] args) {
        int[] angka = {1, 2, 3, 4, 5, 6};
        int jumlahBilanganGenap = hitungBilanganGenap(angka);
        System.out.println("Jumlah dari bilangan yang bernilai genap : " + jumlahBilanganGenap);
    }

    public static int hitungBilanganGenap(int[] angka) {
        int jumlah = 0;
        for (int i = 0; i < angka.length; i++) {
            if (angka[i] % 2 == 1) { // Logic error : harusnya angka[i] % 2 == 0
                jumlah += angka[i];
            }
        }
        return jumlah;
    }
}
```

Pada contoh di atas, terdapat kesalahan logika pada kondisi if ($\text{angka}[i] \% 2 == 1$). Logika yang benar seharusnya adalah $\text{angka}[i] \% 2 == 0$ untuk menghitung bilangan genap, namun program ini malah menghitung bilangan ganjil.

Untuk memperbaikinya, kita harus mengganti operator modulus pada kondisi if sehingga menjadi $\text{angka}[i] \% 2 == 0$ untuk memastikan hanya bilangan genap yang dihitung. Setelah perbaikan, program ini akan memberikan output yang sesuai.

```
package Modul1;

public class LogicErrorExample {
    public static void main(String[] args) {
        int[] angka = {1, 2, 3, 4, 5, 6};
        int jumlahBilanganGenap = hitungBilanganGenap(angka);
        System.out.println("Jumlah dari bilangan yang bernilai genap : " + jumlahBilanganGenap);
    }

    public static int hitungBilanganGenap(int[] angka) {
        int jumlah = 0;
        for (int i = 0; i < angka.length; i++) {
            if (angka[i] % 2 == 0) { // Memperbaiki logic error
                jumlah += angka[i];
            }
        }
        return jumlah;
    }
}
```



c. Runtime Error

Runtime Error adalah jenis error yang terjadi saat program dijalankan, dan sering kali tidak dapat dideteksi oleh compiler pada saat kompilasi. Error ini biasanya terjadi ketika program menerima input yang tidak sesuai, mencoba mengakses memori yang tidak dialokasikan, atau melakukan pembagian dengan nol. Kesalahan ini hanya terdeteksi pada saat eksekusi dan dapat menyebabkan crash atau membuat aplikasi menjadi tidak responsif. Berikut adalah contoh Runtime Error :

1. NumberFormatException

NumberFormatException adalah kesalahan yang muncul saat mencoba mengubah teks (string) yang tidak sesuai formatnya menjadi angka.

Contoh NumberFormatException :

```
package Modul1;

public class RuntimeErrorExample {
    public static void main(String[] args){
        String invalidNumber = "abc123";

        int number = Integer.parseInt(invalidNumber);
        System.out.println("Converted Number : " + number);
    }
}
```

```
C:\Users\Titan\.jdk\openjdk-24\bin\java.exe ...
Exception in thread "main" java.lang.NumberFormatException: Create breakpoint : For input string: "abc123"
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:564)
    at java.base/java.lang.Integer.parseInt(Integer.java:661)
    at Modul1.RuntimeErrorExample.main(RuntimeErrorExample.java:7)
```

Misalnya, pada contoh diatas terdapat string "abc123", kita tidak bisa mengubahnya menjadi angka karena ada huruf di dalamnya. Kesalahan ini terjadi saat program dijalankan, bukan saat kita menulis kode, sehingga disebut sebagai Runtime Exception.



2. `ArrayIndexOutOfBoundsException`

Error ini terjadi ketika program mencoba mengakses indeks array yang berada di luar batas yang valid.

Contoh `ArrayIndexOutOfBoundsException` :

```
package Modul1;

public class ArrayIndexOutOfBoundsExceptionExample {
    public static void main(String[] args) {
        int[] arr = {1,2,3};

        System.out.println("Elemen dari indeks ke-3 : " + arr[3]);
    }
}
```

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
at Modul1.ArrayIndexOutOfBoundsExceptionExample.main(ArrayIndexOutOfBoundsExceptionExample.java:7)
```

Program di atas adalah contoh runtime error dalam Java yang disebabkan oleh akses indeks array di luar batas. Ketika mencoba mencetak `arr[3]`, program akan melemparkan `ArrayIndexOutOfBoundsException`, karena hanya ada tiga elemen dalam array (indeks 0, 1, dan 2).

3. `NullPointerException`

Error ini terjadi ketika program mencoba mengakses atau memanggil metode pada objek yang belum diinisialisasi (null).

Contoh `NullPointerException` :

```
package Modul1;

public class NullPointerExceptionExample {
    public static void main(String[] args) {

        String str = null;
        System.out.println("Panjang dari String str : " + str.length());
    }
}
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot invoke "String.length()" because "str" is null
at Modul1.NullPointerExceptionExample.main(NullPointerExceptionExample.java:7)
```



Program di atas menunjukkan contoh runtime error dalam Java yang terjadi akibat referensi null. Saat mencoba mengakses metode `length()` pada objek `str` yang bernilai null, program akan melemparkan `NullPointerException`. Kesalahan ini menekankan pentingnya memastikan bahwa objek telah diinisialisasi sebelum mengakses metode atau atributnya, untuk mencegah kesalahan saat runtime

d. Resource Error

Resource Error terjadi ketika program menggunakan lebih banyak memori daripada yang tersedia atau ketika program gagal dalam mengelola alokasi memori. Hal ini dapat menyebabkan program crash atau kehabisan sumber daya. Resource Error dapat disebabkan oleh :

- Gagal melepaskan memori yang dialokasikan setelah tidak digunakan lagi, menyebabkan kebocoran memori.
- Pengelolaan resource yang tidak efisien, seperti membuka terlalu banyak file pada saat yang bersamaan.
- Kesalahan dalam penulisan loop, yang dapat menyebabkan kondisi yang salah pada loop.

Contoh Resource Error :

```
package Modul1;

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

public class ResourceErrorExample {
    public static void main(String[] args) {
        List<int[]> list = new ArrayList<>();
        int arraySize = 10_000_000; // 10 juta elemen per array

        try {
            while (true) { // Loop terus-menerus sampai memori habis
                list.add(new int[arraySize]);
                System.out.println("Menambahkan array besar. Total list: " + list.size());
            }
        } catch (OutOfMemoryError e) {
            System.err.println("Kehabisan memori! Terjadi OutOfMemoryError.");
        }

        System.out.println("Program selesai dijalankan.");
    }
}
```

Pada program di atas, sebuah array besar terus ditambahkan ke dalam list tanpa batasan, yang menyebabkan program akhirnya kehabisan memori dan menyebabkan `OutOfMemoryError`.

Output jika program dijalankan : Program terus menambahkan data ke dalam list sampai akhirnya sistem kehabisan memori.




```

Run ResourceErrorExample x
Menambahkan array besar. Total list: 80
Menambahkan array besar. Total list: 81
Menambahkan array besar. Total list: 82
Menambahkan array besar. Total list: 83
Menambahkan array besar. Total list: 84
Menambahkan array besar. Total list: 85
Menambahkan array besar. Total list: 86
Menambahkan array besar. Total list: 87
Menambahkan array besar. Total list: 88
Menambahkan array besar. Total list: 89
Menambahkan array besar. Total list: 90
Menambahkan array besar. Total list: 91
Menambahkan array besar. Total list: 92
Menambahkan array besar. Total list: 93
Menambahkan array besar. Total list: 94
Menambahkan array besar. Total list: 95
Menambahkan array besar. Total list: 96
Menambahkan array besar. Total list: 97
Program selesai dijalankan.
Kehabisan memori! Terjadi OutOfMemoryError.

Process finished with exit code 0

```

Solusi :

Untuk menghindari resource error, kita dapat menetapkan batas maksimal ukuran list, atau menggunakan algoritma yang lebih efisien dalam mengelola memori seperti pada gambar dibawah.

```

package Modul1;

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

public class ResourceErrorExample {
    public static void main(String[] args) {
        List<int[]> list = new ArrayList<>();
        int arraySize = 10_000_000; // 10 juta elemen per array
        int maxSize = 50; // Batasi jumlah array yang ditambahkan

        try {
            for (int i = 0; i < maxSize; i++) {
                list.add(new int[arraySize]);
                System.out.println("Menambahkan array besar ke-" + (i + 1));
            }
        } catch (OutOfMemoryError e) {
            System.err.println("Kehabisan memori! Terjadi OutOfMemoryError.");
        }

        System.out.println("Program selesai dijalankan.");
    }
}

```



e. Time Limit Exceeded (TLE) Error

Time Limit Exceeded (TLE) Error terjadi ketika program melebihi batas waktu yang diperbolehkan untuk menjalankan eksekusi. Biasanya kesalahan ini terjadi pada program yang menggunakan algoritma tidak efisien atau ukuran input yang besar. Untuk menghindari TLE, kita perlu mengoptimalkan algoritma agar lebih efisien, terutama pada perhitungan yang membutuhkan banyak iterasi.

Penyebab Time Limit Exceeded (TLE) Error :

- Penggunaan algoritma yang tidak efisien, terutama pada skala input yang besar.
- Proses yang memakan waktu lama dalam operasi tertentu, seperti perhitungan atau pemrosesan data

Ciri - Ciri Time Limit Exceeded (TLE) Error :

- Program membutuhkan waktu yang sangat lama untuk menyelesaikan tugas.
- Muncul pesan error yang menunjukkan bahwa waktu eksekusi melebihi batas.

Contoh Time Limit Exceeded (TLE) Error :

```
package Modul1;

public class TLEError {
    public static void main(String[] args) {
        int n = 1000000; // Mencari jumlah bilangan prima hingga 1000000
        int count = countPrimes(n);
        System.out.println("Jumlah bilangan prima hingga " + n + " adalah: " + count);
    }

    public static int countPrimes(int n) {
        int count = 0;
        for (int i = 2; i <= n; i++) {
            if (isPrime(i)) {
                count++;
            }
        }
        return count;
    }

    public static boolean isPrime(int num) {
        for (int i = 2; i < num; i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }
}
```

Output :

```
Jumlah bilangan prima hingga 1000000 adalah: (hitungannya memakan waktu terlalu lama)
```



Pada program di atas, fungsi `isPrime()` memeriksa apakah sebuah angka adalah bilangan prima dengan cara memeriksa pembagi dari angka tersebut satu per satu. Pendekatan ini tidak efisien, sehingga ketika digunakan untuk mencari bilangan prima hingga 1000000, program memerlukan waktu yang sangat lama dan melebihi batas waktu yang diperbolehkan, menghasilkan TLE error. Untuk mengatasi masalah TLE, kita dapat menggunakan Algoritma Sieve of Eratosthenes, yang jauh lebih efisien dalam menghitung bilangan prima dalam rentang besar.

Contoh Program :

```
package Modul1;

public class TLError {
    public static void main(String[] args) {
        int n = 1000000; // Mencari jumlah bilangan prima hingga 1000000
        int count = countPrimes(n);
        System.out.println("Jumlah bilangan prima hingga " + n + " adalah: " + count);
    }

    public static int countPrimes(int n) {
        boolean[] isPrime = new boolean[n + 1];
        for (int i = 0; i <= n; i++) {
            isPrime[i] = true;
        }

        for (int i = 2; i * i <= n; i++) {
            if (isPrime[i]) {
                for (int j = i * i; j <= n; j += i) {
                    isPrime[j] = false;
                }
            }
        }

        int count = 0;
        for (int i = 2; i <= n; i++) {
            if (isPrime[i]) {
                count++;
            }
        }

        return count;
    }
}
```

Output :

```
Jumlah bilangan prima hingga 1000000 adalah: 78498
```

Algoritma Sieve of Eratosthenes menggunakan pendekatan yang lebih efisien untuk menemukan semua bilangan prima hingga suatu angka n . Algoritma ini menandai kelipatan setiap angka yang ditemukan sebagai bukan bilangan prima, mengurangi waktu eksekusi secara signifikan dibandingkan dengan pendekatan brute force.



2. The Concept Of Specification

Spesifikasi program adalah penjelasan rinci tentang bagaimana suatu program harus berfungsi. Tujuan dari spesifikasi ini adalah untuk memberikan pemahaman yang jelas mengenai input yang diharapkan, proses yang terjadi di dalam program, serta output yang akan dihasilkan. Spesifikasi program juga membantu pengembang untuk menulis kode yang sesuai dengan persyaratan yang telah ditetapkan. Berikut ini adalah contoh dari program spesifikasi.

Deskripsi Program :

Program ini akan menghitung luas lingkaran dengan jari-jari yang diberikan oleh pengguna. Luas lingkaran dihitung dengan rumus: $\pi * r^2$, dimana r adalah jari-jari lingkaran.

a. Spesifikasi Formal

$$\{I P\} \text{ C } \{I Q\}$$

$$\{r \geq 0\} \text{ C } \{\text{Luas} = \pi * r^2\}$$

Penjelasan :

1. P (Precondition): Kondisi awal atau predikat yang harus dipenuhi sebelum program dijalankan. Dalam kasus ini, program mengharapkan input berupa jari-jari lingkaran yang lebih besar atau sama dengan 0 ($r \geq 0$).
2. C (Command): Program atau perintah yang dieksekusi. Di sini, program akan mengalikan nilai π dengan kuadrat dari nilai input jari-jari lingkaran (r^2).
3. Q (Postcondition): Kondisi yang harus dipenuhi setelah program dijalankan. Pada contoh ini, program harus menghasilkan luas lingkaran yang dihitung dengan rumus $\pi * r^2$.



b. Spesifikasi Informal

Parameter : Sebuah bilangan real positif r yang merupakan jari-jari lingkaran.

Returns : Luas lingkaran dengan jari-jari r , dihitung menggunakan rumus $\pi * r^2$.

Contoh Program :

```
import java.util.Scanner;

public class CircleArea {
    public static void main(String[] args) {
        // Input: jari-jari lingkaran
        Scanner scanner = new Scanner(System.in);
        System.out.print("Masukkan jari-jari lingkaran: ");
        double radius = scanner.nextDouble();

        // Validasi input
        if (radius < 0) {
            System.out.println("Jari-jari tidak bisa negatif. Masukkan nilai yang valid.");
        } else {
            // Proses: menghitung luas lingkaran
            double area = Math.PI * Math.pow(radius, 2);

            // Output: hasil luas lingkaran
            System.out.println("Luas lingkaran dengan jari-jari " + radius + " adalah: " + area);
        }
    }
}
```

Output :

```
Masukkan jari-jari lingkaran: 5
Luas lingkaran dengan jari-jari 5.0 adalah: 78.53981633974483
```

1. Precondition: Program meminta input jari-jari lingkaran dan memverifikasi apakah nilai tersebut valid (lebih besar atau sama dengan 0).
2. Command: Program kemudian menghitung luas lingkaran menggunakan rumus $\pi * r^2$ dan menampilkan hasilnya.
3. Postcondition: Program memastikan hasil keluaran adalah luas lingkaran yang sesuai dengan rumus matematika.



3. Defensive Programming

Defensive Programming adalah teknik pengembangan perangkat lunak yang memastikan perangkat lunak dapat menangani keadaan yang tak terduga dengan baik. Hal ini bertujuan untuk mencegah program dari kegagalan atau kesalahan yang dapat merusak fungsionalitas aplikasi. Teknik ini mengutamakan pengelolaan kesalahan (error handling) dan validasi input untuk memastikan bahwa program berfungsi dengan benar meskipun diberikan input yang tidak sesuai harapan.

a. Tujuan Defensive Programming

1. Meningkatkan Kualitas Umum : Dengan mengurangi jumlah bug dan masalah yang mungkin muncul pada kode.
2. Mengembangkan Kode yang Dapat Dipahami : Kode yang ditulis dengan teknik defensive programming akan lebih mudah dibaca dan dipahami, serta memudahkan dalam pemeriksaan kode.
3. Menghindari Program Crash : Dengan menangani kesalahan sebelum menyebabkan program berhenti, perangkat lunak akan tetap berjalan meskipun terjadi masalah.

b. Teknik-Teknik Implementasi

1. Try - Catch : Menggunakan blok try-catch untuk menangani kesalahan yang terjadi selama eksekusi program. Jika terjadi kesalahan, maka program akan melanjutkan dengan cara yang lebih aman.
2. Periksa Argumen Fungsi : Memastikan bahwa argumen yang diterima oleh fungsi adalah valid dan sesuai dengan yang diharapkan sebelum melanjutkan eksekusi.
3. Gunakan Assert : Fungsi assert digunakan untuk memastikan bahwa kondisi yang diharapkan benar-benar terpenuhi.

Contoh Program :

Di bawah ini adalah contoh program yang meminta pengguna untuk memasukkan beberapa angka, lalu menghitung rata-rata. Program ini juga dilengkapi dengan pengecekan apakah input yang diberikan adalah angka positif.



```
import java.util.Scanner;

public class DefensiveExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Meminta pengguna memasukkan jumlah angka yang akan dihitung rata-ratanya
        System.out.print("Masukkan jumlah angka yang akan dihitung rata-ratanya: ");
        int count = scanner.nextInt();

        // Validasi jumlah angka harus lebih dari 0
        stopIfNot(count > 0, "Jumlah angka harus lebih besar dari 0!");

        int[] numbers = new int[count];

        System.out.println("Masukkan " + count + " angka:");
        for (int i = 0; i < count; i++) {
            System.out.print("Angka ke-" + (i + 1) + ": ");
            numbers[i] = scanner.nextInt();
        }

        int sum = 0;
        for (int number : numbers) {
            sum += number;
        }

        double average = (double) sum / count;
        System.out.println("Rata-rata dari angka-angka yang dimasukkan adalah: " + average);

        scanner.close();
    }

    // Fungsi untuk memeriksa kondisi dan membuang exception jika kondisi tidak terpenuhi
    public static void stopIfNot(boolean condition, String errorMessage) {
        if (!condition) {
            throw new IllegalArgumentException(errorMessage);
        }
    }
}
```

Output jika input benar :

```
Masukkan jumlah angka yang akan dihitung rata-ratanya: 4
Masukkan 4 angka:
Angka ke-1: 10
Angka ke-2: 20
Angka ke-3: 30
Angka ke-4: 40
Rata-rata dari angka-angka yang dimasukkan adalah: 25.0
```

Output jika input salah :

```
Masukkan jumlah angka yang akan dihitung rata-ratanya: -5
Exception in thread "main" java.lang.IllegalArgumentException: Jumlah angka harus lebih besar dari 0!
    at Modul11.DefensiveExample.stopIfNot(DefensiveExample.java:38)
    at Modul11.DefensiveExample.main(DefensiveExample.java:14)
```

1. **Precondition** : Sebelum meminta input dari pengguna, program memeriksa apakah jumlah angka yang dimasukkan lebih besar dari 0. Jika tidak, program akan membuang `IllegalArgumentException`.
2. **Command** : Program kemudian meminta pengguna untuk memasukkan angka-angka dan menghitung rata-rata dari angka tersebut.
3. **Postcondition** : Setelah input dan perhitungan selesai, program mengeluarkan hasil rata-rata dari angka yang dimasukkan.



4. Code Review

Code reviews adalah proses sistematis untuk mengevaluasi kode yang telah dibuat dengan tujuan untuk mengidentifikasi bug, meningkatkan kualitas kode, dan membantu pengembang untuk memahami serta belajar dari kode yang telah dibuat.

Berikut adalah langkah-langkah dalam melakukan code reviews :

1. **Preparation** : Pengembang yang telah menyelesaikan kode mulai peninjauan kode dengan membuat permintaan review kepada satu atau lebih anggota tim yang ditugaskan sebagai reviewer. Reviewer harus memiliki keahlian yang relevan untuk menilai kode.
2. **Tools** : Review kode dapat dilakukan menggunakan alat khusus atau IDE yang memungkinkan peninjauan melihat perubahan kode dan memberikan umpan balik. Contohnya: GitHub, GitLab, Visual Studio Code, dll.
3. **Code Review Checklist** : Reviewer dapat merujuk pada daftar Periksa atau standar untuk memastikan bahwa kode tersebut sesuai dengan pedoman yang ditetapkan.
4. **Inspeksi Kode** : Reviewer memeriksa kode secara menyeluruh, mencari apakah dalam kode ditemukan kesalahan sintaksis, kesalahan logika, hambatan kerja, kerentanannya, konsistensi standar pengkodean, skalabilitas, dan pemeliharaan.
5. **Feedbacks and Discussion** : Reviewer memberikan komentar dalam tools peninjauan kode, menjelaskan masalah apa yang mereka temukan. Penulis kode serta reviewer terlibat dalam diskusi tentang perubahan kode.
6. **Revisions** : Penulis kode membuat kode berdasarkan umpan balik yang diterima.
7. **Approval and Integration** : Setelah reviewer setuju dengan perubahan kode dan semua permasalahan diselesaikan, perubahan kode yang disetujui diintegrasikan ke dalam basis kode utama, biasanya menggunakan sistem kontrol versi.
8. **Follow-Up** : Setelah diintegrasikan, kode pada lingkungan produksi dipantau untuk memastikan bahwa perubahan berfungsi sesuai harapan dan tidak menimbulkan masalah baru.
9. **Documentation and Closure** : Proses peninjauan kode didokumentasikan untuk referensi yang akan datang, dokumentasi berupa komentar tinjauan, keputusan yang diambil, dan tindakan apapun. Setelah itu, peninjauan dapat ditutup secara resmi.



Contoh Program :

Program berikut adalah kalkulator sederhana yang melakukan operasi pembagian dan pengurangan. Program ini akan direview untuk memastikan bahwa kode tersebut efektif dan efisien, serta tidak ada kesalahan logika.

```
package Modul1;

public class SimpleCalculator {

    // Method untuk pembagian
    public static int division(int a, int b) {
        return a / b;
    }

    // Method untuk pengurangan
    public static int subtraction(int a, int b) {
        return a - b;
    }

    // Main program untuk menguji kalkulator
    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 5;

        // Operasi pembagian
        System.out.println("Hasil Pembagian: " + division(num1, num2));

        // Operasi pengurangan
        System.out.println("Hasil Pengurangan: " + subtraction(num1, num2));
    }
}
```

Feedback dari Code Review :

1. Inspeksi Kode :
 - a. Masalah Potensial: Program ini tidak memeriksa pembagian dengan nol, yang dapat menyebabkan kesalahan `ArithmeticException` saat `b` bernilai 0.
 - b. Saran: Menambahkan pengecekan sebelum melakukan operasi pembagian untuk memastikan bahwa pembagi tidak nol
2. Revisi :
 - a. Menambahkan pengecekan di method `division` untuk memastikan pembagi tidak nol.

```
// Method untuk pembagian
public static int division(int a, int b) {
    if (b == 0){
        throw new IllegalArgumentException("Pembagi tidak boleh nol");
    }
    return a/b;
}
```



3. Approval and Integration : Setelah revisi dilakukan, perubahan tersebut diterima dan digabungkan dengan kode utama.

Proses code review dapat membantu memastikan bahwa kode yang ditulis tidak hanya berfungsi dengan benar, tetapi juga efisien dan mudah dipahami oleh pengembang lain. Selain itu, peninjauan kode juga membantu mencegah kesalahan yang dapat merusak fungsionalitas aplikasi.

5. Unit Testing

Unit Testing adalah teknik pengujian perangkat lunak di mana masing-masing komponen atau unit aplikasi diuji secara terpisah untuk memastikan bahwa kinerjanya sesuai dengan yang diharapkan. Tujuannya adalah untuk mengidentifikasi kesalahan atau bug di awal proses pengembangan sehingga dapat mempercepat proses pengembangan dan memperbaiki kode lebih efisien.

Unit Testing sangat penting untuk memastikan bahwa bagian-bagian kecil dari perangkat lunak (seperti fungsi, metode, atau prosedur) berfungsi dengan benar, sebelum digabungkan menjadi sistem yang lebih besar. Dengan melakukan unit testing, pengembang dapat lebih cepat mendeteksi dan memperbaiki kesalahan pada bagian-bagian tertentu dari aplikasi.

a. Keuntungan dari Unit Testing

- Mempercepat Pengembangan : Unit Testing memungkinkan pengembang untuk menangani masalah lebih awal dalam siklus pengembangan, yang mempercepat proses pengembangan secara keseluruhan.
- Meningkatkan Kepercayaan terhadap Kode : Setelah unit diuji dengan baik, pengembang memiliki keyakinan yang lebih tinggi bahwa kode tersebut berfungsi dengan baik.
- Mempermudah Pemeliharaan Kode : Jika ada perubahan atau penambahan fitur baru, unit tests membantu untuk memastikan bahwa perubahan tersebut tidak merusak fungsi yang sudah ada.
- Mencegah Regresi : Unit Testing juga membantu mencegah regresi, yaitu kegagalan fungsi yang sebelumnya telah bekerja dengan baik setelah perubahan atau pembaruan kode.

b. Unit Testing dalam Siklus Pengembangan

Unit Testing dilakukan pada tahap awal pengembangan, ketika unit-unit kode yang lebih kecil masih terpisah, sebelum kode digabungkan dengan sistem yang lebih besar. Dengan memisahkan pengujian ke dalam unit-unit kecil, pengembang dapat fokus untuk memastikan bahwa setiap bagian berfungsi sebagaimana mestinya.



Langkah-langkah dalam Melakukan Unit Testing :

1. Merencanakan dan Menyiapkan Lingkungan

Pengembang akan merencanakan unit-unit mana yang perlu diuji dan bagaimana cara menyiapkan lingkungan untuk pengujian tersebut. Misalnya, pengembang perlu memastikan bahwa kode untuk unit tersebut dapat berjalan secara terpisah dari sistem lainnya.

2. Menulis Test Case dan Skrip

Pengembang menulis test case yang berfungsi untuk menguji unit tersebut. Test case ini dapat mencakup berbagai kemungkinan input dan output yang diharapkan untuk unit yang diuji. Test case ini ditulis dalam bentuk skrip otomatis yang akan dijalankan untuk memastikan fungsi bekerja sesuai dengan yang diharapkan.

3. Mengeksekusi Unit Test

Setelah test case ditulis, pengembang menjalankan unit tests untuk menguji fungsi dari unit tersebut. Hasil dari tes ini akan menunjukkan apakah kode bekerja dengan benar sesuai dengan yang diinginkan.

4. Analisis Hasil

Setelah menjalankan tes, pengembang menganalisis hasilnya untuk menemukan masalah atau kesalahan dalam kode. Jika tes gagal, pengembang akan memeriksa bagian mana yang menyebabkan kesalahan dan memperbaikinya.

c. Best Practice dalam Unit Testing

1. **Write Readable Tests** : Pengujian yang mudah dibaca membantu pengembang lain memahami cara kerja kode dan apa yang diharapkan. Jika pengujian gagal, pengembang lain dapat dengan cepat mengidentifikasi masalah.
2. **Write Deterministic Tests** : Pengujian yang deterministik akan selalu memberikan hasil yang sama setiap kali kode dijalankan, baik berhasil ataupun gagal. Ini membantu dalam pengujian yang stabil dan dapat diprediksi.
3. **Don't Use Multiple Asserts in a Single Unit Test** : Sebaiknya setiap unit test hanya memiliki satu assertion (pernyataan pengujian). Dengan cara ini, jika tes gagal, pengembang dapat lebih mudah mengetahui penyebab kesalahan tanpa kebingungan.



d. Membuat Unit Testing dengan JUnit di IntelliJ IDEA

1. Buka IntelliJ IDEA dan pastikan struktur proyek menggunakan Maven atau gradle. Hal ini memudahkan pengelolaan dependensi dan menjalankan pengujian. Jika belum ada anda bisa membuat project baru.
2. Buat fungsi atau method yang ingin diuji, atau ikuti contoh seperti yang dibawah.

```
package org.example;

public class NumberUtils {


    // Mengecek apakah bilangan genap
    public static boolean isEven(int n) {
        return n % 2 == 0;
    }

    // Mengecek apakah bilangan ganjil
    public static boolean isOdd(int n) {
        return n % 2 != 0;
    }

    // Mengecek apakah bilangan prima
    public static boolean isPrime(int n) {
        if (n <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(n); i++) {
            if (n % i == 0) {
                return false;
            }
        }
        return true;
    }

    // Mengecek apakah bilangan desimal
    public static boolean isDecimal(double n) {
        return n % 1 != 0;
    }
}
```

3. Klik kanan pada class yang berisi method yang ingin diuji, pilih Generate, kemudian Test



```
public class NumberUtils { no usages

    // Men
    public
    re
}

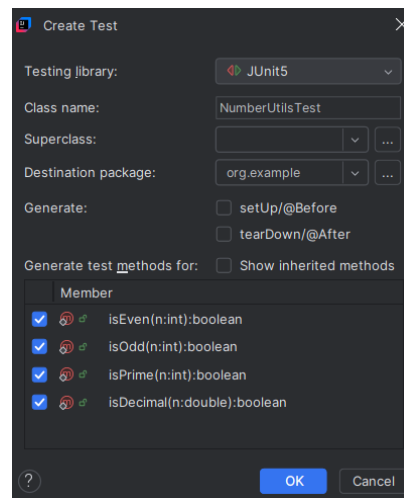
// Men
public static boolean isOdd(int n) { no u
    return n % 2 != 0;
```

Generate

- Constructor
- toString()
- Override Methods... Ctrl+O
- Test...**
- Copyright



4. Lalu pilih method apa saja yang ingin dilakukan testing, pada contoh ini kita akan memilih semua method untuk di testing



5. Buka file testing yang baru dibuat oleh IntelliJ IDEA (**biasanya terdapat di dalam folder src/test**) dan tambahkan import yang diperlukan jika IntelliJ belum otomatis menambahkan paket yang dibutuhkan.
6. Buat assert statement untuk setiap fungsi atau method yang ingin diuji, agar setiap unit kode dapat diverifikasi secara tepat. Berikut contohnya :

```
package org.example;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

class NumberUtilsTest {

    @Test
    void isEven() {
        assertTrue(NumberUtils.isEven(2));
    }

    @Test
    void isOdd() {
        assertTrue(NumberUtils.isOdd(3));
    }

    @Test
    void isPrime() {
        assertTrue(NumberUtils.isPrime(13));
    }

    @Test
    void isDecimal() {
        assertTrue(NumberUtils.isDecimal(3.2));
    }
}
```



Output :

⌵	✖ NumberUtilsTest (org.example)	20 ms
✔	isEven()	17 ms
✖	isOdd()	3 ms
✔	isDecimal()	
✔	isPrime()	

Untuk testing yang bertanda ✔ menandakan bahwa proses testing berhasil dan sesuai dengan yang diinginkan oleh program. Sedangkan jika bertanda ✖ menunjukkan hasil yang diharapkan tidak sesuai dengan yang diinginkan program, contohnya pada testing testIsOdd.

Pada metode **isOdd()** terdapat kesalahan logika. Metode tersebut menggunakan ekspresi $n \% 2 == 0$, yang artinya akan mengembalikan true jika angka habis dibagi 2 padahal itu justru definisi dari angka genap, bukan ganjil. Akibatnya, setiap kali memanggil isOdd dengan angka ganjil, metode ini akan mengembalikan false. Untuk memperbaikinya, kondisi dalam metode harus diubah menjadi $n \% 2 != 0$, karena angka ganjil adalah angka yang tidak habis dibagi 2.



REFERENSI

[10 Common Programming Errors and How to Avoid Them | Codacy](#)

[Types of Issues and Errors in Programming/Coding | geeksforgeeks](#)

[Defensive programming in R | geeksforgeeks](#)

[Program specification - Stanford Reference](#)

[What is a code review? | GitLab](#)

[What is Software Testing? | geeksforgeeks](#)

[What is Unit Testing? | geeksforgeeks](#)



CODELAB

CODELAB 1

Buatlah program untuk mencari nama terpanjang dalam sebuah array nama. Program ini memiliki kesalahan dalam cara mengakses array dan memproses string. Identifikasi kesalahan dan perbaiki agar program dapat berjalan sesuai dengan yang diharapkan.

```
public class CodeLab1 {  
    public static void main(String[] args) {  
        String[] nama = {"Adi", "Budi", "Cahyo", "Diana", "Eva"};  
        String namaTerpanjang = cariNamaTerpanjang(nama);  
        System.out.println("Nama terpanjang adalah: " + namaTerpanjang);  
    }  
  
    public static String cariNamaTerpanjang(String[] array) {  
        String namaMax = array[0];  
        for (String nama : array) {  
            if (nama.length() > namaMax.length()) {  
                namaMax = nama;  
            }  
        }  
        return namaMax;  
    }  
}
```

CODELAB 2

Buatlah sebuah program yang meminta pengguna untuk memasukkan usia mereka. Program ini harus memastikan bahwa usia yang dimasukkan valid (lebih besar dari 0 dan kurang dari 120). Jika usia tidak valid, lemparkan custom exception **InvalidAgeException** dengan pesan kesalahan yang sesuai.



TUGAS

TUGAS 1

Program berikut dapat menghitung rata-rata dari sejumlah angka yang dimasukkan. Namun, terdapat kesalahan dalam cara perhitungan dan penanganan input. Temukan dan perbaiki kesalahan pada program ini.

```
import java.util.Scanner;

public class AverageCalculator {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Masukkan jumlah angka: ");
        int n = scanner.nextInt();

        int[] angka = new int[n];
        System.out.println("Masukkan angka-angka:");
        for (int i = 0; i < n; i++) {
            angka[i] = scanner.nextInt();
        }

        int total = 0;
        for (int i = 1; i < n; i++) { // Perhatikan kesalahan pada index perulangan
            total += angka[i];
        }

        double rataRata = (double) total / n;
        System.out.println("Rata-rata adalah: " + rataRata);

        scanner.close();
    }
}
```

TUGAS 2

Buatlah program untuk memvalidasi apakah input yang diberikan adalah angka positif. Jika tidak, lemparkan custom exception **InvalidNumberException** dengan pesan kesalahan yang sesuai. Program harus meminta pengguna untuk memasukkan angka dan memeriksa validitasnya.



TUGAS 3

Seorang guru ingin membuat sebuah program yang dapat menerima data mahasiswa berupa **nama** dan **nilai ujian akhir**. Program kemudian menentukan apakah mahasiswa tersebut **Lulus** atau **Tidak Lulus** dengan ketentuan berikut:

- **Lulus** jika nilai ujian akhir ≥ 60
- **Tidak Lulus** jika nilai ujian akhir < 60

Spesifikasi yang harus dipenuhi:

- **Input:** Nama mahasiswa dan nilai ujian akhir
- **Output:** Nama mahasiswa beserta status kelulusannya

Tugas :

1. Buatlah **deskripsi formal dan informal** dari program tersebut !
2. Buatlah program sesuai dengan deskripsi di atas !

TUGAS 4

Buat sebuah fungsi `findMin()` yang menerima tiga bilangan bulat sebagai parameter dan mengembalikan nilai minimum dari ketiganya.

```
public class MinFinder {
    public static int findMin(int a, int b, int c) {
        if (a <= b && a <= c) {
            return a;
        } else if (b <= a && b <= c) {
            return b;
        } else {
            return c;
        }
    }
}
```

Buatlah unit test untuk fungsi `findMin()` dengan mempertimbangkan beberapa skenario, berikut :

- Skenario 1 : Pengujian angka 3 dari nilai a, b, dan c yang bernilai 1, 2, dan 3
- Skenario 2 : Pengujian angka -1 dari nilai a, b, dan c yang bernilai -1, -2, dan -3
- Skenario 3 : Pengujian angka 0 dari nilai a, b, dan c yang bernilai 0, 0, dan 1



KRITERIA & DETAIL PENILAIAN

KRITERIA PENILAIAN	POIN
CODELAB 1	5
CODELAB 2	5
TUGAS 1	10
TUGAS 2	15
TUGAS 3	15
TUGAS 4	15
PEMAHAMAN	35
TOTAL	100%

