

VERSION 1.0

JULI 8, 2024



# PEMROGRAMAN LANJUT

MODUL 1 – PROGRAM CORRECTNESS

DISUSUN OLEH:

- WILDAN SUHARSO, S.KOM, M.KOM
- LUTHFIA SHOFA DEWI
- RAHMATUN NIKMAH

LAB. INFORMATIKA

UNIVERSITAS MUHAMMADIYAH MALANG

## PEMROGRAMAN LANJUT

---

### PERSIAPAN MATERI

Mahasiswa harus memahami konsep OOP yang telah dipelajari pada semester 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

Syntax Error atau error tata bahasa adalah salah satu jenis error yang paling sering terjadi ketika sedang membuat sebuah program. Error ini terjadi ketika ada kesalahan syntax/rules pada sebuah bahasa program. Jenis error ini sangat mudah terdeteksi karena biasanya compiler atau interpreter pada masing-masing bahasa program melakukan pengecekan terlebih dahulu sebelum program di run, compiler juga biasanya akan menunjukkan lokasi baris yang membuat program error/tidak jalan. Contoh syntax error yang paling sering ditemukan adalah tidak menggunakan titik koma (;) pada akhir suatu proses.

Contoh syntax error:

```

1 package SyntaxError;
2
3 no usages
4 public class SyntaxError {
5     no usages
6     public static void main(String[] args) {
7         System.out.println("Hello, World!");
8     }
9 }

```

```

Build: Build Output x
Program.M1: build failed At 27-Jun-24 20:54 with 1 error
SyntaxError.java src\SyntaxError 1 error
    ')' expected :5

```

pada program diatas terdapat syntax error dimana pada baris ke 5 kekurangan tanda kurung tutup ')' sehingga jika program tidak dapat dijalankan. pada terminal, diperlihatkan jumlah kesalahan syntax error yang terjadi pada program. berikut jika program berhasil dijalankan:

```

1 package SyntaxError;
2
3 no usages
4 public class SyntaxError {
5     no usages
6     public static void main(String[] args) {
7         System.out.println("Hello, World!");
8     }
9 }

```

```

"C:\Program Files\Eclipse Adoptium\j
Hello, World!

Process finished with exit code 0

```

#### b. Logic Error

Logic error merupakan jenis error yang sangat sulit untuk dideteksi karena penyebab terjadinya bukan karena kesalahan penulisan (sintaks) atau kesalahan proses runtime, namun kesalahan dari sisi programmer, dalam hal ini algoritma yang digunakan pada program yang dibuat. Karena logika yang digunakan salah, tentunya output yang dihasilkan juga akan salah. Untuk mendeteksi letak kesalahannya, biasanya harus mencari letak kesalahan logika secara baris per baris (line by line). Kesalahan Logic Error biasanya ditemukan pada baris yang berisi:

- pernyataan atau kondisi perulangan (if-else) yang menyebabkan hasil output salah
- penggunaan operator logika (AND, OR, NOT) yang salah
- Penggunaan algoritma yang salah untuk memecahkan masalah tertentu
- Operasi pada variabel dengan menggunakan tipe data yang tidak cocok
- Penulisan loop yang tidak diakhiri dengan benar

Contoh logic error:

```
Logic Error

public class LogicError {
    public static void main(String[] args) {
        int[] angka = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
        int jumlahGenap = hitungBilanganGenap(angka);
        System.out.println("Jumlah bilangan genap: " + jumlahGenap);
    }

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

Output:

```
Output

Jumlah bilangan genap: 6
```

pada program diatas terdapat logic error dimana program diminta untuk menghitung jumlah bilangan genap dari array 'angka'. Pada kondisi if yang kedua terdapat kesalahan dimana hasil modulus seharusnya sisa 0 untuk bisa menghitung bilangan genap, tetapi pada program tertulis 1. Sehingga, proses operasi yang terjadi pada program tersebut adalah menghitung bilangan ganjil dan output yang diharapkan tidak sesuai. Untuk memperbaikinya, kita bisa merubah nilainya agar output yang diharapkan sesuai. Berikut adalah Program yang sesuai:

```

Logic Error

public class LogicError {
    public static void main(String[] args) {
        int[] angka = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
        int jumlahGenap = hitungBilanganGenap(angka);
        System.out.println("Jumlah bilangan genap: " + jumlahGenap);
    }

    public static int hitungBilanganGenap(int[] angka) {
        int jumlah = 0;
        for (int i = 0; i < angka.length; i++) {
            if (angka[i] % 2 == 0) { // Diperbaiki: 'angka[i] % 2 == 0'
                jumlah++;
            }
        }
        return jumlah;
    }
}

```

Outputnya:

```

Output

Jumlah bilangan genap: 5

```

### c. Runtime Error

Error ini terjadi saat program dijalankan (di-running). Kompiler tidak dapat mendeteksi Error tersebut dan hanya muncul pada saat program dijalankan. Karena itu, error ini biasanya dapat menyebabkan crash, nge-freeze, atau hal yang tidak diinginkan pada saat program berjalan. Error ini terjadi ketika:

- Program menerima inputan yang tidak dapat ditangani, seperti melakukan inputan String akan tetapi tipe data yang dibutuhkan adalah Integer.
- Adanya masalah dengan alokasi memori, misalnya mencoba melakukan akses pada memori yang belum dialokasikan.
- Melakukan Pembagian dengan nol.

```

Runtime Error

public class RuntimeError {
    public static void main(String[] args) {
        String[] words = {"Hello", null, "World"};
        for (int i = 0; i < words.length; i++) {
            System.out.println(words[i].length()); // Kesalahan runtime:
            NullPointerException
        }
    }
}

```

```

Exception in thread "main" java.lang
.NullPointerException Create breakpoint : Cannot invoke "String
.length()" because "words[i]" is null

```

pada program diatas terdapat runtime error dimana array 'words' berisi nilai null di indeks kedua. Ketika program mencoba untuk mengakses 'length' dari elemen null, ini akan menyebabkan NullPointerException, yang merupakan kesalahan runtime. Untuk memperbaikinya, kita bisa menambahkan kondisi jika ada nilai null.

```

Runtime Error

public class RuntimeError {
    public static void main(String[] args) {
        String[] words = {"Hello", null, "World"};
        for (int i = 0; i < words.length; i++) {
            if (words[i] != null) {
                System.out.println(words[i].length());
            } else {
                System.out.println("Elemen adalah null");
            }
        }
    }
}

```

Output:



```

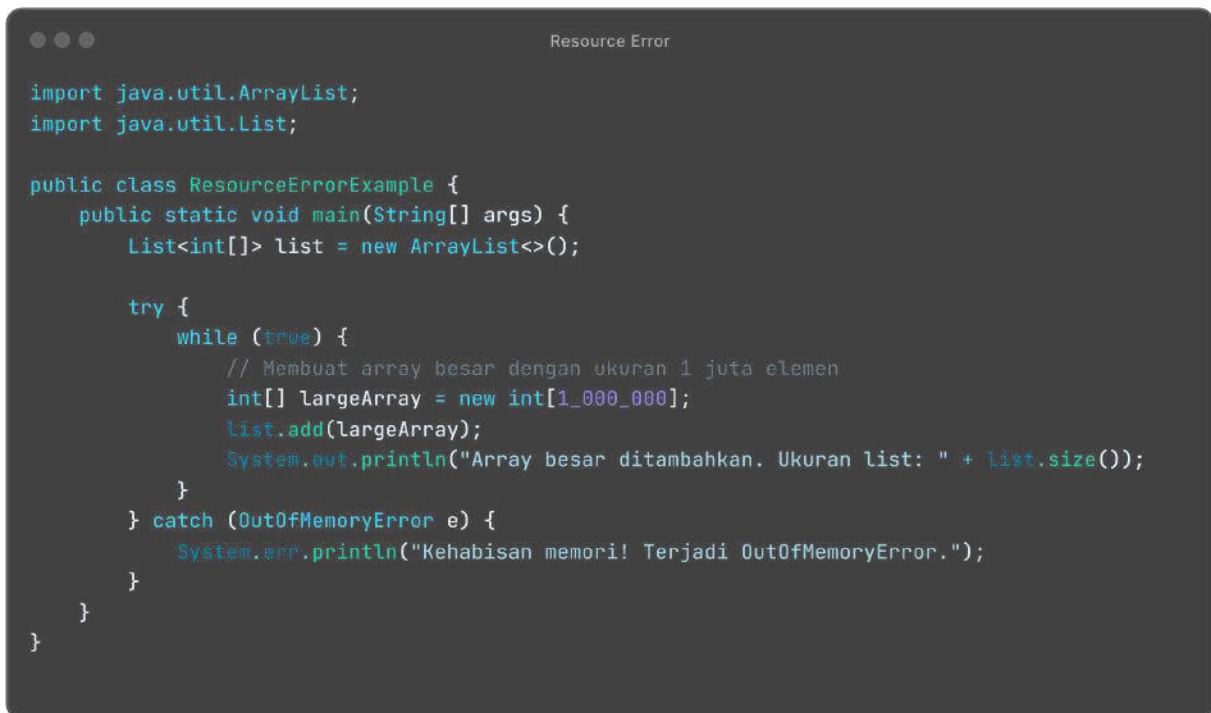
5
Elemen adalah null
5

```

#### d. Resource Error

Ketika suatu program yang sedang di-run dan menggunakan memori lebih banyak dari yang dibutuhkan, kemungkinan program tersebut akan terjadi crash. Hal ini disebut dengan resource error. Kesalahan ini terjadi ketika program kehabisan atau salah mengelola memori yang tersedia. Berikut ini beberapa penyebab terjadinya resource error:

- Gagal melepaskan memori yang dialokasikan secara dinamis setelah tidak dibutuhkan lagi, sehingga menyebabkan kebocoran memori.
- Pengelolaan resource yang tidak efisien, seperti membuka terlalu banyak file secara bersamaan.
- Penulisan looping yang tidak dihentikan, baik dikarenakan kesalahan logika atau kondisi looping yang salah



```

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

public class ResourceErrorExample {
    public static void main(String[] args) {
        List<int[]> list = new ArrayList<>();

        try {
            while (true) {
                // Membuat array besar dengan ukuran 1 juta elemen
                int[] largeArray = new int[1_000_000];
                list.add(largeArray);
                System.out.println("Array besar ditambahkan. Ukuran list: " + list.size());
            }
        } catch (OutOfMemoryError e) {
            System.err.println("Kehabisan memori! Terjadi OutOfMemoryError.");
        }
    }
}

```

Output:

```

Output

Array besar ditambahkan. Ukuran list: 1
Array besar ditambahkan. Ukuran list: 2
...
Kehabisan memori! Terjadi OutOfMemoryError.

```

pada program diatas terdapat runtime error dimana loop while (true) terus membuat array besar berukuran satu juta elemen dan menambahkannya ke dalam ArrayList. Karena memori yang dialokasikan untuk array besar tidak pernah dilepaskan dan loop tidak pernah berhenti, program pada akhirnya akan kehabisan memori, menyebabkan OutOfMemoryError. untuk memperbaikinya, bisa menetapkan batas maksimal ukuran list untuk menghindari kehabisan memori. Dengan membatasi jumlah objek yang dibuat, kita dapat mengelola memori dengan lebih efektif dan menghindari OutOfMemoryError.

```

Resource Error

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

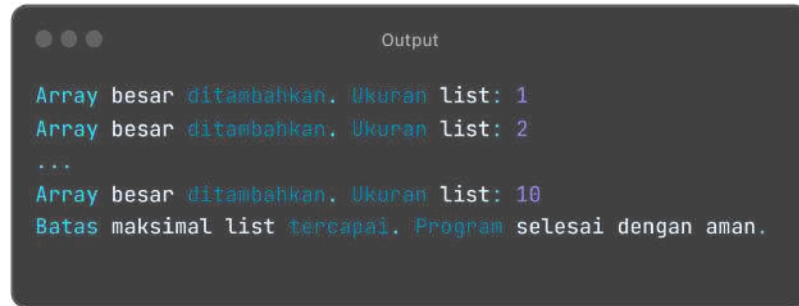
public class ResourceErrorFixed {
    public static void main(String[] args) {
        List<int[]> list = new ArrayList<>();
        int maxSize = 10; // Batasi jumlah objek dalam list untuk menghindari kehabisan memori

        try {
            while (list.size() < maxSize) {
                int[] largeArray = new int[1_000_000];
                list.add(largeArray);
                System.out.println("Array besar ditambahkan. Ukuran list: " + list.size());
            }
            System.out.println("Batas maksimal list tercapai. Program selesai dengan aman.");
        } catch (OutOfMemoryError e) {
            System.err.println("Kehabisan memori! Terjadi OutOfMemoryError.");
        }
    }
}

```

Output:





```
Output
Array besar ditambahkan. Ukuran list: 1
Array besar ditambahkan. Ukuran list: 2
...
Array besar ditambahkan. Ukuran list: 10
Batas maksimal list tercapai. Program selesai dengan aman.
```

e. Time Limit Exceeded (TLE) Error

Error ini terjadi ketika kode memakan waktu terlalu lama untuk di eksekusi dan eksekusi melebihi waktu yang ditentukan pada program yang dijalankan. Kesalahan ini biasanya muncul pada skenario yang memiliki durasi maksimal (waktu batasan) yang diperbolehkan untuk tugas tertentu. Keadaan yang bisa menyebabkan TLE error adalah:

- Ketika suatu program atau operasi membutuhkan waktu yang lebih lama daripada waktu yang diharapkan/diperbolehkan untuk dieksekusi karena beberapa faktor, contohnya karena algoritma yang tidak efisien atau ukuran inputan yang besar, menyebabkan perpanjangan waktu proses.
- ketika terjadi penundaan atau batas waktu dalam pengiriman atau penerimaan data melalui jaringan yang disebabkan oleh kemacetan jaringan atau waktu respon yang lambat dari server.
- Kehabisan resource, seperti kehabisan memori, waktu CPU, atau sistem lainnya yang diperlukan untuk menyelesaikan tugas dalam jangka waktu yang diperlukan

```
TLE Error

public class TLError {
    public static void main(String[] args) {
        int n = 100000; // Menghitung jumlah bilangan prima hingga 100000
        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) {
        if (num <= 1) return false;
        for (int i = 2; i < num; i++) { // Algoritma yang sangat tidak efisien
            if (num % i == 0) return false;
        }
        return true;
    }
}
```

Output:

```
Output

Jumlah bilangan prima hingga 100000 adalah: 9592
```

pada program diatas terdapat TLE error dimana fungsi isPrime menggunakan algoritma yang sangat tidak efisien untuk memeriksa apakah suatu angka adalah bilangan prima. Karena memeriksa semua angka dari 2 hingga num - 1, waktu eksekusi yang dibutuhkan sangat lama untuk angka besar. Untuk menghindari TLE, kita bisa menggunakan algoritma yang lebih efisien untuk menghitung bilangan prima. Salah satu algoritma yang lebih efisien adalah Sieve of Eratosthenes.

```

TLE Error

public class TLError {
    public static void main(String[] args) {
        int n = 100000; // Menghitung jumlah bilangan prima hingga 100000
        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 = 2; 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:

```

Output

Jumlah bilangan prima hingga 100000 adalah: 9592

```

Penjelasan Algoritma Sieve of Eratosthenes:

- 1) Inisialisasi array isPrime dengan semua nilai true untuk menunjukkan bahwa semua angka dari 2 hingga n adalah bilangan prima.
- 2) Mulai dari angka 2, tandai semua kelipatan angka tersebut sebagai bukan bilangan prima.
- 3) Ulangi langkah 2 untuk angka berikutnya yang masih ditandai sebagai bilangan prima.

4) Hitung dan kembalikan jumlah bilangan prima yang tersisa.

## 2. The Concept Of a Specification

Ada beberapa cara atau metode yang digunakan pengguna untuk menyampaikan informasi tentang jenis program yang diinginkan oleh pengguna. Metode inilah yang disebut dengan spesifikasi program. Hal ini bisa melibatkan pemberian contoh input (untuk mengumpulkan data atau meminta keluaran) dan output (untuk mengambil data) dari program yang diinginkan. Spesifikasi program bisa dilakukan secara formal dengan menggunakan kalkulus predikat atau model formal lainnya untuk mendefinisikan perilaku program. Ataupun bisa dengan spesifikasi Informal, yaitu menggunakan deskripsi tekstual yang mudah dipahami. Berikut ini adalah contoh dari program spesifikasi:

### Deskripsi program:

Program ini akan menghitung faktorial dari sebuah bilangan bulat non-negatif 'x'. Faktorial dari 'x', dilambangkan sebagai 'x!'.

#### a. Spesifikasi Formal:

$$\{ |P| \} C \{ |Q| \}$$

$$\{ |n \geq 0 \} C \{ | \text{hasil} = n! \}$$

Penjelasan:

- **P** merupakan Precondition yaitu kondisi awal atau predikat yang harus dipenuhi sebelum mengeksekusi program. P disini memastikan bahwa n yang diinputkan adalah bilangan bulat non-negatif ( $n \geq 0$ )
- **C** adalah Command/program yaitu perintah atau implementasi dari program atau fungsi yang akan dieksekusi
- **Q** adalah Postcondition yaitu kondisi yang harus dipenuhi setelah eksekusi program selesai. Pada program faktorial ini, menjamin bahwa hasil dari eksekusi **C** yaitu hasil yang merupakan 'n!'

#### b. Spesifikasi Informal:

Parameter: Sebuah bilangan bulat non-negatif 'x'

Returns: Faktorial dari 'x' yaitu 'x!'

### 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:

- a. Meningkatkan Kualitas Umum: Meminimalkan jumlah bug dan masalah yang dapat muncul pada kode.
- b. 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.
- c. Mengembangkan perangkat lunak yang akan memberikan output yang benar terlepas dari input yang diberikan.

Defensive Programming dapat diimplementasikan menggunakan:

- a. Try - Catch : Fungsi ini memungkinkan untuk menangani kesalahan dan pengecualian dengan cara yang terkendali. Fungsi ini membutuhkan satu blok kode sebagai argumen lalu melakukan pengeksekusian. Jika terjadi kesalahan atau pengecualian, fungsi ini akan menjalankan penanganan kesalahan yang ditentukan oleh programmer. Hal ini dapat membantu program untuk berhenti dan memungkinkan untuk menangani kesalahan dengan cara yang lebih baik.
- b. Periksa argumen fungsi: Periksa argumen fungsi untuk memastikan bahwa argumen tersebut bertipe benar (TRUE) dan berada dalam rentang yang diharapkan. Hal ini dapat membantu mencegah kesalahan dan hal yang tak terduga pada kode.
- c. Gunakan stopifnot: Fungsi ini memungkinkan untuk menentukan serangkaian kondisi yang harus dipenuhi agar kode dapat terus dijalankan. Jika salah satu kondisi tidak terpenuhi, stopifnot akan menimbulkan kesalahan dan menghentikan program.
- d. Gunakan package assert that: package ini menyediakan serangkaian fungsi yang memungkinkan untuk menentukan pernyataan tentang nilai variabel pada kode. Jika salah satu pernyataan tidak terpenuhi, pernyataan tersebut akan menimbulkan kesalahan dan menghentikan program.

```

Defensive Program (Error Handling)

import java.util.Scanner;

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

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

        // Memastikan count adalah bilangan positif
        stopIfNot(count > 0, "Jumlah angka harus lebih besar dari 0");

        int[] numbers = new int[count];
        int sum = 0;

        System.out.println("Masukkan " + count + " angka:");

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

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

        scanner.close();
    }

    // Metode khusus untuk melakukan pengecekan kondisi
    public static void stopIfNot(boolean condition, String errorMessage) {
        if (!condition) {
            throw new IllegalArgumentException(errorMessage);
        }
    }
}

```

Output jika inputan benar:

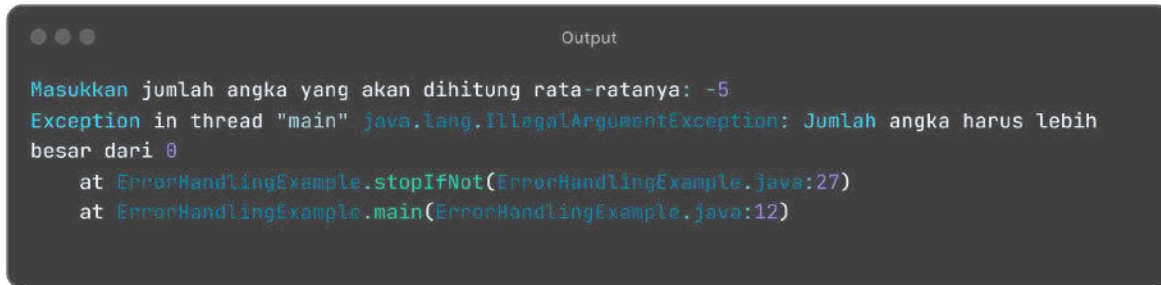
```

Output

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 inputan salah:



```

Output

Masukkan jumlah angka yang akan dihitung rata-ratanya: -5
Exception in thread "main" java.lang.IllegalArgumentException: Jumlah angka harus lebih
besar dari 0
    at ErrorHandlingExample.stopIfNot(ErrorHandlingExample.java:27)
    at ErrorHandlingExample.main(ErrorHandlingExample.java:12)
  
```

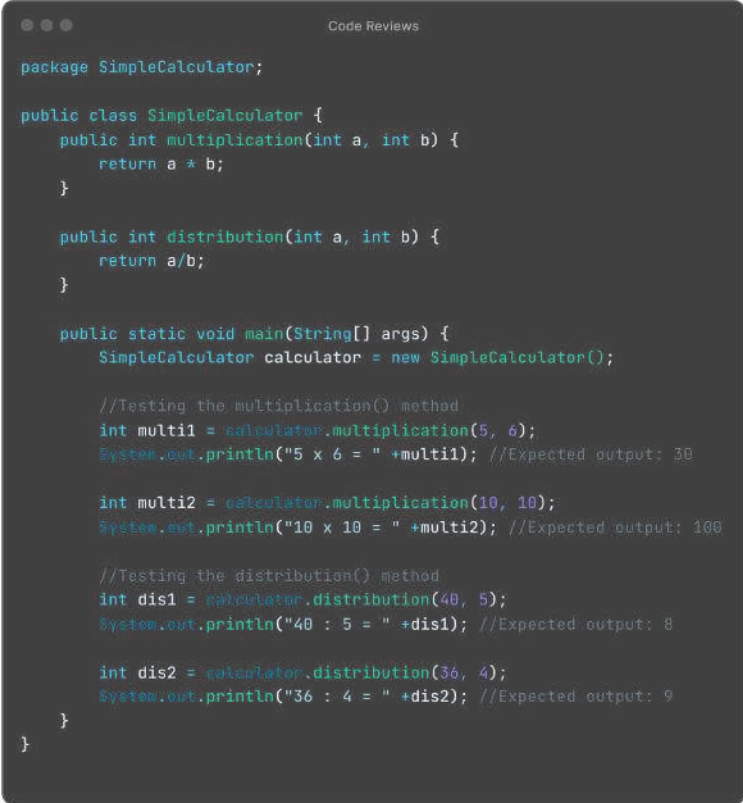
pada gambar diatas merupakan program sederhana yang menghitung nilai rata-rata dari sejumlah angka yang dimasukkan pengguna. Program ini akan menangani kasus di mana pengguna tidak memasukkan angka yang valid. Program akan meminta pengguna untuk memasukkan jumlah angka yang akan dihitung rata-ratanya. lalu inputan akan diperiksa oleh method 'stopIfNot' untuk memastikan bahwa nilai count (jumlah angka) yang dimasukkan oleh pengguna adalah bilangan positif. lalu program akan menjalankan perintah selanjutnya, jika inputan benar maka program akan meminta untuk memasukkan count angka, dan jika inputan benar maka Exception akan terjadi.

#### 4. Code Reviews

Code reviews adalah proses sistematis untuk mengevaluasi kode yang telah dibuat dengan tujuan mengidentifikasi bug, meningkatkan kualitas kode, dan membantu pengembang memahami serta belajar dari kode yang telah dibuat. Proses ini berulang kali terbukti membantu mempercepat dan menyederhanakan proses pengembangan perangkat. Berikut ini langkah-langkah melakukan code reviews:

- a. **Preparation:** Pengembang yang telah menyelesaikan kode memulai 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.
- b. **Tools:** Mereview code dapat dilakukan menggunakan alat khusus atau IDE yang memungkinkan peninjauan melihat perubahan kode dan memberikan umpan balik. Contohnya: GitHub, GitLab, Visual Studio Code, dll.
- c. **Code Review Checklist:** Reviewer dapat merujuk pada daftar periksa atau standar untuk memastikan bahwa kode tersebut sesuai dengan pedoman yang ditetapkan

- d. **Inspeksi Code:** Reviewer memeriksa kode secara menyeluruh, mencari apakah dalam kode ditemukan kesalahan sintaksis, kesalahan logika, hambatan kerja, kerentanan keamanan, konsistensi standar pengkodean, skalabilitas, dan pemeliharaan.
- e. **Feedbacks and Discussion:** Reviewer memberikan komentar dalam tools peninjauan kode, menjelaskan masalah apa yang mereka temukan. Dan penulis kode serta reviewer terlibat dalam diskusi tentang perubahan kode.
- f. **Revisions:** Penulis kode membuat kode berdasarkan umpan balik yang diterima
- g. **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.
- h. **Follow-Up:** Setelah diintegrasikan, kode pada lingkungan produksi dipantau untuk memastikan bahwa perubahan berfungsi sesuai harapan dan tidak menimbulkan masalah baru
- i. **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.



```
Code Reviews

package SimpleCalculator;

public class SimpleCalculator {
    public int multiplication(int a, int b) {
        return a * b;
    }

    public int distribution(int a, int b) {
        return a/b;
    }

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

        //Testing the multiplication() method
        int multi1 = calculator.multiplication(5, 6);
        System.out.println("5 x 6 = " +multi1); //Expected output: 30

        int multi2 = calculator.multiplication(10, 10);
        System.out.println("10 x 10 = " +multi2); //Expected output: 100

        //Testing the distribution() method
        int dis1 = calculator.distribution(40, 5);
        System.out.println("40 : 5 = " +dis1); //Expected output: 8

        int dis2 = calculator.distribution(36, 4);
        System.out.println("36 : 4 = " +dis2); //Expected output: 9
    }
}
```



## 5. Testing Fundamentals and Test-Case Generation

Testing adalah proses penting dalam siklus software development. Testing melibatkan verifikasi (mengacu pada serangkaian tugas yang memastikan bahwa software mengimplementasikan fungsi tertentu dengan benar) dan validasi (mengacu pada serangkaian tugas berbeda yang memastikan bahwa software yang telah dibangun dapat ditelusuri ke kebutuhan pengguna) bahwa aplikasi bebas dari bug, memenuhi persyaratan teknis yang ditetapkan oleh pengembangannya, dan memenuhi persyaratan pengguna secara efisien dan efektif. Dengan mengidentifikasi dan memperbaiki masalah secara sistematis, testing membantu menghasilkan software yang berkualitas tinggi dan berfungsi sesuai harapan pada berbagai skenario.

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:

- a. 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.
- b. Random approach - Random approach menghasilkan kasus uji berdasarkan asumsi error dan kesalahan sistem.
- c. Specification-based technique - Model ini menghasilkan kasus uji berdasarkan spesifikasi kebutuhan formal.
- d. 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.
- e. Sketch-diagram-based approach - Jenis pendekatan pembuatan kasus ini mengikuti diagram Unified Modeling Language (UML) untuk merumuskan test case.

## 6. Unit Testing

Unit Testing adalah teknik pengujian perangkat lunak dimana masing-masing komponen atau unit aplikasi perangkat lunak diuji secara terpisah untuk memastikan kinerjanya sesuai dengan yang diharapkan. Tujuannya adalah untuk mengidentifikasi dan memperbaiki bug di awal proses

pengembangan dan memastikan setiap unit bekerja dengan benar sebelum diintegrasikan ke sistem yang lebih besar. Keuntungannya dapat mempercepat pengembangan, mempermudah pemeliharaan kode, dan meningkatkan kepercayaan dalam perangkat lunak. Tools yang digunakan untuk melakukan unit testing diantaranya Junit (Java), NUnit (.NET), EMMA (Java), Unit PHP (PHP), PyTest (Python). Unit Testing biasanya terdiri dari empat fase:

1. **Merencanakan dan menyiapkan lingkungan.** Pengembang mempertimbangkan unit mana dalam kode yang perlu mereka uji, dan bagaimana menjalankan semua fungsi yang relevan dari setiap unit untuk mengujinya secara efektif
2. **Menulis case pengujian dan skrip.** Pengembang menulis kode pengujian unit dan menyiapkan skrip untuk mengeksekusi kode tersebut.
3. **Mengeksekusi unit testing.** Pengujian unit berjalan dan memperlihatkan bagaimana kode berperilaku untuk setiap kasus pengujian.
4. **Analisa Hasil.** Pengembang dapat mengidentifikasi kesalahan atau masalah dalam kode dan memperbaikinya

Berikut ini best practice yang dapat digunakan untuk membuat pengujian unit lebih efektif:

- **Write Readable Tests**

Pengujian yang mudah dibaca membantu pengembang lain memahami cara kerja kode, tujuannya, dan apa yang salah jika pengujian gagal. Dan membuat lebih mudah untuk memperbarui pengujian ketika kode yang mendasari berubah

- **Write Deterministic Tests**

Pengujian deterministik selalu lolos (jika tidak ada masalah) atau selalu gagal (bila ada masalah) pada bagian kode yang sama. Hasil tes tidak akan berubah selama kode juga tidak diubah. Sedangkan **Non-Deterministic** juga dikenal sebagai pengujian tidak stabil, sehingga pengujian mungkin lulus atau gagal karena berbagai kondisi meskipun kode yang di test sama atau belum diubah.

- **Don't Use Multiple Asserts in a Single Unit Test**

Agar Unit Testing lebih efektif dan mudah dikelola, setiap pengujian sebaliknya hanya memiliki satu kasus pengujian. Yang artinya, tes tersebut seharusnya hanya mempunyai satu pernyataan. Hal ini, karena ketika pengujian menggunakan banyak kasus dan gagal, hal ini membuat tidak jelas apa penyebab utama bug tersebut.

Contoh Unit Testing pada Java:

```

Unit Testing

import org.junit.Test;
import static org.junit.Assert.*;

public class StringUtilsTest {
    @Test
    public void testConcatenate() {
        String result = StringUtils.concatenate("Hello", "World");
        assertEquals("HelloWorld", result); // ✓ Hasil yang diharapkan: "HelloWorld"
    }

    @Test
    public void testConcatenateFailure() {
        String result = StringUtils.concatenate("Hello", "World");
        assertEquals("Hello World", result); // ✗ Hasil yang diharapkan: "Hello World", namun
        // hasil sebenarnya: "HelloWorld"
    }

    @Test
    public void testToUpperCase() {
        String result = StringUtils.toUpperCase("hello");
        assertEquals("HELLO", result); // ✓ Hasil yang diharapkan: "HELLO"
    }

    @Test
    public void testToLowerCase() {
        String result = StringUtils.toLowerCase("WORLD");
        assertEquals("world", result); // ✓ Hasil yang diharapkan: "world"
    }
}

```

Output:

```

Run: StringUtilsTest
Tests failed: 1, passed: 3 of 4 tests - 19 ms

StringUtilsTest (ModuleUnitTest) 19 ms
  ✓ testToLowerCase 7 ms
  ✓ testToUpperCase 1 ms
  ✗ testConcatenateFailure 11 ms
  ✓ testConcatenate 0 ms

org.junit.ComparisonFailure:
Expected :Hello World
Actual   :HelloWorld
<Click to see difference>

<2 internal lines>
at Modul1UnitTesting.StringUtilsTest.testConcatenateFailure(StringUtilsTest.java:16) <25 internal lines>

```

Untuk 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. Hal ini terjadi karena:

```

Unit Testing

@Test
public void testConcatenateFailure() {
    String result = StringUtils.concatenate("Hello", "World");
    assertEquals("Hello World", result); // X Hasil yang diharapkan: "Hello World", namun
    hasil sebenarnya: "HelloWorld"
}

```

fungsi **testConcatenateFailure()** mempunyai tugas untuk menghubungkan String a dan String b. Oleh karena itu output yang dihasilkan dari kode diatas seharusnya adalah “HelloWorld”, dan ketika dilakukan percobaan untuk “Hello World” hasil testingnya failed.

---

## CODELAB

### CODELAB 1: LOGIC ERROR

Berikut merupakan sebuah program untuk menghitung total pada array angka, dimana program tersebut memiliki logic error. Identifikasi kesalahan dan perbaiki programnya agar sesuai dengan yang diharapkan.

```

Codelab 1

public class Codelab1 {
    public static void main(String[] args) {
        int[] angka = {22, 67, 31, 11, 5};
        int hasil = hitungTotal(angka);
        System.out.println("Hasilnya adalah: " + hasil);
    }

    public static int hitungTotal(int[] array) {
        int hasil = 0;
        for (int i = 1; i < array.length - 1; i++) {
            hasil += array[i];
        }
        return hasil;
    }
}

```

### CODELAB 2: EXCEPTION HANDLING

Buatlah sebuah program dalam bahasa Java yang melakukan validasi alamat email. Program ini harus meminta pengguna untuk memasukkan alamat email dan kemudian memeriksa apakah alamat email tersebut valid. Alamat email dianggap valid jika mengandung karakter '@webmail.umm.ac.id'. Jika alamat email tidak valid, lemparkan custom exception 'InvalidEmailException' dengan pesan kesalahan yang sesuai.

```

Exception Handling

package Latihan2Modul1;

import java.util.Scanner;

// Custom exception untuk email tidak valid
class InvalidEmailException extends Exception {
    public InvalidEmailException(String message) {
        super(message);
    }
}

public class EmailValidation {
    // Method untuk validasi email
    public static void validateEmail(String email) throws InvalidEmailException {
        if (!email.endsWith("@webmail.umm.ac.id")) {
            throw new InvalidEmailException("Alamat email harus diakhiri dengan '@webmail.umm.ac.id'");
        }
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        boolean isValidEmail = false;

        while (!isValidEmail) {
            System.out.print("Masukkan alamat email Anda (harus diakhiri dengan '@webmail.umm.ac.id'): ");
            String email = scanner.nextLine().trim();

            try {
                validateEmail(email);
                isValidEmail = true;
                System.out.println("Alamat email Anda adalah: " + email);
            } catch (InvalidEmailException e) {
                System.out.println("Kesalahan validasi email: " + e.getMessage());
                System.out.println("Silakan coba lagi.");
            }
        }

        scanner.close();
    }
}

```

Pada kode di atas, program menggunakan exception handling untuk menangani kesalahan saat pengguna memasukkan input yang tidak diakhiri '@webmail.umm.ac.id'. Jika pengguna memasukkan input selain yang diakhiri dengan '@webmail.umm.ac.id', blok catch akan dieksekusi dan pesan "Alamat email harus diakhiri dengan '@webmail.umm.ac.id'. Silakan coba lagi." akan ditampilkan.

#### Output:

```

Masukkan alamat email Anda (harus diakhiri dengan '@webmail.umm.ac.id'): fia@gmail.com
Kesalahan validasi email: Alamat email harus diakhiri dengan '@webmail.umm.ac.id'.
Silakan coba lagi.
Masukkan alamat email Anda (harus diakhiri dengan '@webmail.umm.ac.id'): fia@webmail.umm.ac.id
Alamat email Anda adalah: fia@webmail.umm.ac.id

```

### CODELAB 3: CODE REVIEW

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

```
CodeLab 3

public class HitungGaji {

    public int perhitunganGaji(int jamKerja, int gajiPerJam) {
        int gajiTotal = 0;
        for (int i = 0; i < jamKerja; i++) {
            gajiTotal += gajiPerJam;
        }
        return gajiTotal;
    }

    public static void main(String[] args) {
        HitungGaji hitung = new HitungGaji();

        int gajiTotal = hitung.perhitunganGaji(40, 250000);
        System.out.println("Gaji karyawan dengan 40 jam kerja = " + gajiTotal);
    }
}
```

Output:

```
Output

Gaji karyawan dengan 40 jam kerja = 10000000
```

---

## TUGAS

### TUGAS 1: ANALISIS KESALAHAN

Program berikut seharusnya dapat menghitung, mengubah, dan menemukan data pada array buah . Namun, terdapat beberapa kesalahan dalam kode tersebut. Temukan dan perbaiki kesalahan-kesalahan tersebut. Setelah itu, tulislah analisis tentang kesalahan yang ditemukan.

```

Tugas 1

import java.util.*;

public class NinjaFruit {

    public static void main(String[] args) {
        List<String> buah = Arrays.asList("apel", "pisang", "kiwi", "anggur", "semangka");

        System.out.println("Total huruf: " + calculateTotalLength(buah));
        System.out.println("Kata terpanjang: " + findLongestWord(buah));
        System.out.println("Daftar kata dalam huruf kapital: " + capitalizeWords(buah));
        System.out.println("Panjang masing-masing kata: " + wordLengths(buah));
    }

    public static int calculateTotalLength(List<String> words) { //menghitung jumlah huruf
        int totalLength = 0;
        for (int i = 0; i <= words.size(); i++) {
            totalLength += words.get(i).length();
        }
        return totalLength;
    }

    public static String findLongestWord(List<String> words) { //menemukan kata terpanjang
        if (words.isEmpty()) {
            throw new IllegalArgumentException("List kosong");
        }
        String longestWord = "";
        for (String word : words) {
            if (word != null && word.length() > longestWord.length()) {
                longestWord = word;
            } else if (word != null && word.length() <= longestWord.length()) {
                longestWord = word;
            }
        }
        return longestWord;
    }

    public static List<String> capitalizeWords(List<String> words) { //mengubah setiap huruf
        menjadi kapital
        List<String> capitalized = new ArrayList<>();
        for (String word : words) {
            capitalized.add(word.substring(1).toUpperCase());
        }
        return capitalized;
    }

    public static List<Integer> wordLengths(List<String> words) { //mengitung panjang kata
        List<Integer> lengths = new ArrayList<>();
        for (String word : words) {
            if (word == null) {
                lengths.add(word.length());
            }
        }
        return lengths;
    }
}

```

## TUGAS 2: COMPLEX EXCEPTION HANDLING

Buatlah sebuah program untuk melakukan validasi password. Program ini harus meminta pengguna untuk memasukkan password dan kemudian memeriksa apakah password tersebut memenuhi syarat-syarat yang ditentukan. Password dianggap valid jika memenuhi semua kriteria berikut:

1. Memiliki panjang minimal 8 karakter.
2. Mengandung setidaknya satu karakter huruf besar (A-Z).
3. Mengandung setidaknya satu karakter huruf kecil (a-z).
4. Mengandung setidaknya satu karakter angka (0-9).
5. Mengandung setidaknya satu karakter khusus ('!', '@', '#', '\$', '%', '^', '&', '\*').

Jika password tidak memenuhi salah satu atau lebih dari kriteria di atas, lemparkan custom exception **InvalidPasswordException** dengan pesan kesalahan yang sesuai.

## TUGAS 3: PROGRAM SPECIFICATION

Buatlah sebuah program dan spesifikasi deskripsinya baik secara formal dan informal dari penjelasan berikut:

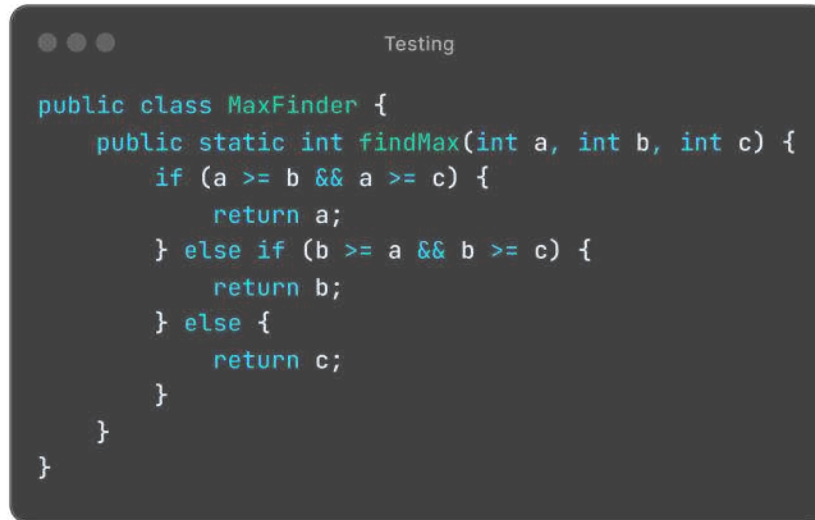
Seorang pemilik wisata ingin membuat sebuah pembelian tiket secara online dimana pembelian tiket wisata tersebut memiliki 2 jenis tiket, yaitu tiket reguler dengan harga 75 ribu untuk dewasa dan 60 untuk anak-anak dan tiket terusan dengan harga 100 ribu untuk dewasa dan 85 ribu untuk anak-anak, jika memasukkan hari sabtu dan minggu maka harga tiket akan naik sebanyak 20% dari harga normal. Pengguna diharuskan untuk memasukkan nama, hari, dan tanggal untuk pergi ke wisata tersebut, jika salah satu kosong akan menimbulkan catatan berisi 'data harus diisi'.

Catatan: Pembuatan deskripsi bisa ditulis menggunakan Google Doc, Word, dll.

## TUGAS 4: TESTING

Buat sebuah fungsi findMax yang menerima tiga bilangan bulat sebagai parameter dan mengembalikan nilai maksimum dari ketiganya.





```

public class MaxFinder {
    public static int findMax(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 findMax dengan mempertimbangkan beberapa skenario, termasuk:

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

---

## REFERENSI

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

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

[Defensive Programming Techniques Explained with Examples | GoLinuxCloud](#)

[Defensive programming in R | geeksforgeeks](#)

[Program specification - Stanford Reference](#)

[Program specification - Oxford Reference](#)

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

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

[What is Test Case Generation? - Definition from Techopedia](#)

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

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

[How Unit Tests Work? | Bright](#)

[What is Unit Testing? Definition from WhatIs.com \(techtarget.com\)](#)

---

**KRITERIA & DETAIL PENILAIAN**

KRITERIA PENILAIAN	POIN
CODELAB 1	5
CODELAB 2	5
CODELAB 3	5
TUGAS 1	10
TUGAS 2	15
TUGAS 3	15
TUGAS 4	15
PEMAHAMAN	30
<b>TOTAL</b>	<b>100%</b>