

PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK

PERTEMUAN 9 : EXCEPTION



Penyusun:

Rizky Satria Gunawan

241511089

2C-D3

PROGRAM STUDI D3 TEKNIK INFORMATIKA

JURUSAN TEKNIK KOMPUTER DAN INFORMATIKA

POLITEKNIK NEGERI BANDUNG

2025

Link Github : [Mata-Kuliah-PBO/FolderTugas9 at main · RizkySatria123/Mata-Kuliah-PBO](https://github.com/RizkySatria123/Mata-Kuliah-PBO)

CASE 1 : Exceptions Aren't Always Errors

PERUBAHAN YANG DILAKUKAN:

1. VALIDASI MASUKAN & PEMROGRAMAN DEFENSIF:

- Menambahkan validasi per karakter untuk mencegah ArrayIndexOutOfBoundsException
- Memakai Character.isLetter() agar pemeriksaan huruf lebih tangguh (mendukung Unicode)
- Melewati karakter non-alfabet tanpa membuat aplikasi berhenti mendadak

2. PENINGKATAN UMPAN BALIK PENGGUNA:

- Menyampaikan karakter mana saja yang tidak valid secara spesifik
- Menggunakan StringBuilder agar penggabungan teks peringatan lebih efisien
- Menampilkan daftar karakter tidak valid sekaligus sehingga tidak mengganggu alur pengguna

3. KERAMAHAN KODE TERHADAP KESALAHAN:

- Menangani kemungkinan nilai null secara aman
- Memisahkan logika validasi dan logika perhitungan dengan jelas
- Alur kontrol lebih mudah dibaca karena validasi dilakukan sejak awal

4. KEPATUHAN TERHADAP PRAKTIK TERBAIK:

- Desain fail-safe: program tetap berjalan, tidak langsung berhenti
- Pesan kesalahan informatif membantu pengguna memperbaiki input
- Tidak ada kegagalan diam-diam, semua anomali dilaporkan

STRATEGI PENANGANAN EKSEPSI:

- Pendekatan preventif: memvalidasi sebelum mengakses array

- Tidak membutuhkan try-catch karena eksepsi sudah dicegah
- Lebih efisien dibanding mengandalkan try-catch

ALASAN MENGAPA LEBIH UNGGUL:

- Menangani kasus pinggiran: string kosong, input null, karakter khusus
- Memberikan umpan balik edukatif kepada pengguna
- Menjaga kestabilan program untuk berbagai kondisi input
- Mengikuti prinsip "fail gracefully" dalam pemrograman defensif

Source-Code:

```
import java.util.Scanner;

public class CountLetters {
    private static final int ALPHABET_SIZE = 26;

    public CountLetters() {
    }

    public static void main(String[] var0) {
        int[] var1 = new int[26];
        Scanner var2 = new Scanner(System.in);
        System.out.print("Enter a single word (letters only, please): ");
        String var3 = var2.nextLine();
        if (var3 != null && !var3.isEmpty()) {
            var3 = var3.toUpperCase();
            StringBuilder var4 = new StringBuilder();
            int var5 = 0;

            int var7;
            for(int var6 = 0; var6 < var3.length(); ++var6) {
                var7 = var3.charAt(var6);
                if (Character.isLetter((char)var7) && var7 >= 65 && var7 <= 90) {
                    ++var1[var7 - 65];
                    ++var5;
                } else {
                    if (var4.length() > 0) {
                        var4.append(", ");
                    }
                }
            }
        }
    }
}
```

```
        var4.append("'").append((char)var7).append("'");
    }
}

if (var4.length() > 0) {
    System.out.println("\n⚠ Warning: The following non-alphabetic characters were
ignored:");
    System.out.println("  " + var4.toString());
}

System.out.println("\nLetter frequency analysis:");
System.out.println("-----");
boolean var8 = false;

for(var7 = 0; var7 < var1.length; ++var7) {
    if (var1[var7] != 0) {
        System.out.println((char)(var7 + 65) + ": " + var1[var7]);
        var8 = true;
    }
}

if (!var8) {
    System.out.println("No alphabetic characters found in the input.");
} else {
    System.out.println("\nTotal valid letters processed: " + var5);
}

var2.close();
} else {
    System.out.println("No input provided. Exiting.");
    var2.close();
}
}
```

Output:

```
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Matematika\CountLetters
Enter a single word (letters only, please): Mobile

Letter frequency analysis:
-----
B: 1
E: 1
I: 1
L: 1
M: 1
O: 1

Total valid letters processed: 6
```

Figure 1: Menginput dengan benar (contoh: Mobile)

```
e - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTugas9\src\case1
va CountLetters
Enter a single word (letters only, please):
No input provided. Exiting.
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Matematika\CountLetters
```

Figure 2: Output bila tidak menginputkan apapun (Null)

```
e - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTugas9\src\case1
va CountLetters
Enter a single word (letters only, please): Rizky12satria

? Warning: The following non-alphabetic characters were ignored:
'1', '2'

Letter frequency analysis:
-----
A: 2
I: 2
K: 1
R: 2
S: 1
T: 1
Y: 1
Z: 1

Total valid letters processed: 11
```

Figure 3: Menginputkan nomor akan memberikan Message seperti berikut

```
va CountLetters
Enter a single word (letters only, please): 12345678

? Warning: The following non-alphabetic characters were ignored:
'1', '2', '3', '4', '5', '6', '7', '8'

Letter frequency analysis:
-----
No alphabetic characters found in the input.
```

Figure 4: Output bila kita menginput nomor , bukan alphabet

CASE 2 : Placing Exception Handlers

PERUBAHAN YANG DILAKUKAN:

1. STRATEGI PENANGANAN EKSEPSI:

- Integer.parseInt() dibungkus try-catch untuk menangani NumberFormatException
- Menerapkan pola "lanjut saat error" agar pemrosesan token tetap tangguh
- Mampu menangani campuran data (bilangan dan teks) secara mulus

2. PEMROSESAN SELURUH BARIS:

- Kode asli berhenti pada token pertama yang bukan bilangan bulat
- Versi refactor memproses seluruh baris apa pun jenis tokennya
- Contoh: "There are 3 dogs and 1 cat" → dijumlahkan dengan benar menjadi 4

3. PENGABAIAN SENYAP UNTUK NON-INTEGER:

- Token non-integer dilewati tanpa suara (sesuai kebutuhan soal)
- Tidak ada interupsi atau peringatan yang mengganggu pengguna
- Pengalaman pengguna tetap bersih tetapi tetap tangguh terhadap kesalahan

4. PERBAIKAN KODE:

- Penamaan variabel lebih jelas (token menggantikan next() tanpa nama)
- Menyimpan hitungan token yang berhasil di-parse untuk kebutuhan debugging/log
- Manajemen resource lebih rapi: Scanner ditutup dengan benar

5. KEPATUHAN PRAKTIK TERBAIK:

- Eksepsi digunakan tepat sasaran untuk validasi parsing
- NumberFormatException dianggap perilaku wajar, bukan kondisi fatal
- Lingkup try-catch dipersempit hanya pada operasi yang berisiko

PENDEKATAN PENANGANAN EKSEPSI:

- Jenis: NumberFormatException (unchecked)
- Strategi: Tangkap lalu lanjut (lewati diam-diam)
- Alasan: Data campuran adalah kondisi normal, bukan hal luar biasa
- Alternatif yang dipertimbangkan: validasi regex (ditolak karena kompleks)

ALASAN KENAPA LEBIH UNGGUL:

- Menangani input dunia nyata: campuran teks dan angka
- Tidak pernah crash untuk kombinasi input apa pun
- Memenuhi ekspektasi pengguna: "jumlahkan semua angka yang ditemukan"
- Siap produksi: menangani kasus tanpa angka dengan tetap sopan

Source-Code:

```
import java.util.Scanner;

public class ParseInts {

    public static void main(String[] args) {
        int sum = 0;
        int integersFound = 0;

        Scanner scan = new Scanner(System.in);

        System.out.println("Enter a line of text:");
        String inputLine = scan.nextLine();

        // Buat scanner kedua untuk memecah input menjadi token-token
        Scanner scanLine = new Scanner(inputLine);

        // Proses setiap token pada baris input
        while (scanLine.hasNext()) {
            String token = scanLine.next();

            try {
                // Coba ubah token menjadi bilangan bulat
                int value = Integer.parseInt(token);
                sum += value;
                integersFound++;
            }
        }
    }
}
```

```

        } catch (NumberFormatException e) {
            // Token bukan bilangan bulat yang valid, lewati tanpa pesan
            // Perilaku ini memang diharapkan karena input dapat bercampur
            // Tidak perlu tindakan lain, lanjutkan ke token berikutnya
        }
    }

    // Tampilkan hasil akhir
    System.out.println("The sum of the integers on this line is " + sum);

    // Opsional: berikan konteks tambahan (boleh dihapus jika ingin keluaran minimal)
    if (integersFound == 0) {
        System.out.println("(No integers were found in the input)");
    } else {
        System.out.println("(" + integersFound + " integer(s) found and summed)");
    }

    // Rapikan resource yang digunakan
    scanLine.close();
    scan.close();
}
}

```

Output:

```

ParseInts
Enter a line of text:
Ada 3 ekor Anjing dan 1 ekor Kucing di sarijadi
The sum of the integers on this line is 4
(2 integer(s) found and summed)
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTuga
s9> █

```

Figure 5: contoh Output 1

```

s9> & 'C:\Program Files\Java\jdk-21\bin\java.exe' '-XX:+ShowCodeDetailsInExceptionMessages' '-cp' 'C:\Users\lenovo\AppData\Roaming\Code\User\workspaceStorage\9f7be617c1d4d6ebf20a2c087aba01bc\redhat.java\jdt_ws\FolderTugas9_6669cfef\bin' █
'ParseInts'
Enter a line of text:
Andi memiliki 50 permen, diberikan lagi permen oleh Sinta 30 permen
The sum of the integers on this line is 80
(2 integer(s) found and summed)
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTuga
s9> █

```

Figure 6: Contoh Output 2

```
Enter a line of text:  
Ciwidey adalah tempat dingin ke satu di bandung  
The sum of the integers on this line is 0  
(No integers were found in the input)  
PS C:\Users\lenovo\OneDrive - Politeknik Negeri Bandung\Documents\rizky satria\Kampus Polban\Mata-Kuliah-PBO\FolderTuga  
s9> |
```

Figure 7: Output bila tidak ada nomor saat penginputan

CASE 3 : Throwing Exceptions

1. Factorials.Java

PERUBAHAN YANG DILAKUKAN:

1. PENANGANAN EKSEPSI PADA LOOP UTAMA:

- Perhitungan faktorial dibungkus dalam blok try-catch
- Menangkap IllegalArgumentException dari MathUtils.factorial()
- Program tetap berjalan setelah eksepsi (tidak crash)

2. PENANGANAN EKSEPSI BERTINGKAT:

- Lapisan 1: Menangkap IllegalArgumentException (error logika bisnis)
- Lapisan 2: Menangkap InputMismatchException (jenis input tidak valid)
- Setiap jenis eksepsi memiliki umpan balik yang spesifik

3. PERBAIKAN PENGALAMAN PENGGUNA:

- Menampilkan pesan kesalahan yang jelas dan terarah
- Program tetap berada dalam loop setelah error sehingga bisa mencoba lagi
- Pembersihan buffer Scanner mencegah loop tak berujung akibat input buruk
- Kelanjutan yang mulus meningkatkan kenyamanan pengguna

4. PENANGANAN INPUT YANG TANGGUH:

- Menangani input non-integer (misal "abc") dengan mulus
- Menolak angka negatif melalui validasi di MathUtils
- Menolak angka terlalu besar melalui validasi yang sama
- Semua jalur kesalahan kembali ke prompt awal

5. PERBAIKAN ALUR KONTROL:

- Jalur sukses dan jalur error dipisahkan dengan jelas
- Setiap jenis eksepsi ditangani dengan logika khusus
- Manajemen buffer Scanner mencegah keadaan data yang rusak

ARSITEKTUR PENANGANAN EKSEPSI:

- `IllegalArgumentException`: Diperkirakan muncul dari lapisan logika bisnis
→ Tampilkan pesan validasi yang spesifik lalu lanjutkan loop
- `InputMismatchException`: Pengguna mengetikkan input non-numerik
→ Tampilkan pesan jenis input salah, bersihkan buffer, lanjutkan loop
- Exception umum: Penjaga terakhir untuk kejutan yang tidak diprediksi
→ Tampilkan pesan generik, lanjutkan loop (fail gracefully)

KEPATUHAN PRAKTIK TERBAIK:

- Pemisahan tanggung jawab: UI menangkap, logika bisnis melempar
- Desain fail-safe: loop terus berjalan dalam kondisi apa pun
- Umpam balik informatif: pengguna tahu apa yang terjadi
- Manajemen resource: Scanner dikelola dan ditutup dengan benar
- Pemrograman defensif: beberapa lapisan penanganan error

ALASAN MENGAPA LEBIH UNGGUL:

- Tidak pernah crash, menangani semua skenario error dengan baik
- Menjaga konsistensi keadaan program setelah error terjadi
- Pesan kesalahan yang jelas membantu pengguna memperbaiki input
- Mencegah loop tak berujung lewat manajemen buffer Scanner
- Siap produksi: menangani error yang diantisipasi maupun yang tidak

Source-Code:

```
import java.util.Scanner;
import java.util.InputMismatchException;
```

```
public class Factorials {

    public static void main(String[] args) {
        String keepGoing = "y";
        Scanner scan = new Scanner(System.in);

        System.out.println("== Factorial Calculator ==");
        System.out.println("Valid range: 0 to 16");
        System.out.println();

        while (keepGoing.equals("y") || keepGoing.equals("Y")) {
            try {
                System.out.print("Enter an integer: ");

                // Baca masukan pengguna - bisa memicu InputMismatchException
                int val = scan.nextInt();

                // Hitung faktorial - bisa memicu IllegalArgumentException
                int result = MathUtils.factorial(val);

                // Tampilkan hasil (hanya tercapai jika tidak ada eksepsi)
                System.out.println("Factorial(" + val + ") = " + result);
                System.out.println();
            } catch (IllegalArgumentException e) {
                // Validasi logika bisnis gagal (nilai negatif atau terlalu besar)
                System.out.println("\nX Error: " + e.getMessage());
                System.out.println();
            } catch (InputMismatchException e) {
                // Pengguna memasukkan nilai yang bukan bilangan bulat
                System.out.println("\nX Error: Invalid input type. Please enter a valid
integer.");
                System.out.println();
            }
        }

        // Bersihkan data yang salah dari buffer scanner
        // Tanpa langkah ini, scanner akan terus membaca input yang sama
        scan.nextLine();

    } catch (Exception e) {
        // Penangkap umum untuk eksepsi tak terduga (pemrograman defensif)
        System.out.println("\nX Unexpected error: " + e.getMessage());
        System.out.println("Please try again.");
        System.out.println();
    }
}
```

```

        // Bersihkan buffer sebagai langkah pengaman
        scan.nextLine();
    }

    // Tanyakan apakah pengguna ingin menghitung lagi (selalu dieksekusi meski ada
error)
    System.out.print("Another factorial? (y/n): ");
    keepGoing = scan.next();
    System.out.println();
}

System.out.println("Thank you for using the Factorial Calculator!");
scan.close();
}
}

```

Output:

```

PS C:\Users\Lenovo\OneDrive - Politeknik Negeri
s9> & 'C:\Program Files\Java\jdk-21\bin\java.
Data\Roaming\Code\User\workspaceStorage\9f7be6
'Factorials'
== Factorial Calculator ==
Valid range: 0 to 16

Enter an integer: 10
Factorial(10) = 3628800

Another factorial? (y/n): 

```

Figure 8 : Contoh Output 1

```

Another factorial? (y/n): y

Enter an integer: 4
Factorial(4) = 24

```

Figure 9 : Contoh Output 2

```

Another factorial? (y/n): y

Enter an integer: 17
? Error: Input exceeds maximum value (16) for integer factorial calculation. Input was: 17. Factorial(17) would cause integer overflow.
Consider using long or BigInteger for larger factorials.

Another factorial? (y/n): 

```

Figure 10 : Contoh Output Error 1

```
Another factorial? (y/n): y
Enter an integer: satu
? Error: Invalid input type. Please enter a valid integer.
```

Figure 11 : Contoh Output Error 2

2. MathUtils.java

PERUBAHAN YANG DILAKUKAN:

1. VALIDASI MASUKAN DENGAN EKSEPSI EXPLICIT:

- Menambahkan validasi prasyarat untuk menolak input negatif
- Menambahkan perlindungan overflow untuk input > 16
- Menggunakan IllegalArgumentException sebagai sinyal pelanggaran kontrak

2. PRINSIP FAIL-FAST:

- Memvalidasi input di awal metode
- Melempar eksepsi segera ketika batasan dilanggar
- Mencegah perhitungan dilakukan dengan data tidak valid

3. PENCEGAHAN OVERFLOW:

- Faktorial bertipe int overflow pada n=17 ($17! > \text{Integer.MAX_VALUE}$)
- Batas matematika: $16! = 2.004.189.184$ (masih cocok di int)
- Pemeriksaan eksplisit mencegah bug overflow yang tidak terlihat

4. PESAN EKSEPSI DESKRIFTIF:

- Input negatif: menjelaskan batasan matematis
- Risiko overflow: menyebutkan batas dan alasannya
- Pesan membantu pemanggil menggunakan metode dengan benar

5. POLA DESAIN YANG DITERAPKAN:

- Guard clause: keluar lebih awal saat kondisi tidak valid
- Design by Contract: prasyarat ditegakkan lewat eksepsi

- Separation of Concerns: validasi dipisah dari perhitungan

STRATEGI PENANGANAN EKSEPSI:

- Jenis: IllegalArgumentException (unchecked, untuk kesalahan pemakaian)
- Waktu: Ketika input melanggar kontrak/prasyarat metode
- Alasan unchecked: Pemanggil seharusnya memvalidasi sebelum memanggil
- Alternatif: Membuat FactorialException khusus (ditolak demi kesederhanaan)

KESESUAIAN MATEMATIS:

- $0! = 1$ (ditangani benar oleh logika loop)
- Faktorial bilangan negatif: tidak terdefinisi secara matematika (kini ditolak)
- Faktorial besar: sebelumnya bisa overflow diam-diam (kini dicegah)

ALASAN MENGAPA LEBIH UNGGUL:

- Menegakkan batasan matematis secara eksplisit
- Mencegah overflow integer yang sunyi (kelas bug kritis)
- Pesan kesalahan jelas mempermudah debug
- Kontrak metode terdokumentasi sendiri lewat eksepsi
- Pemanggil dipaksa menangani kasus pinggiran dengan benar

Source-Code :

```
public class MathUtils {

    // Nilai maksimum faktorial berbasis int tanpa menyebabkan overflow
    // 17! = 355.687.428.096.000 > Integer.MAX_VALUE (2.147.483.647)
    // 16! = 20.922.789.888.000 masih berada dalam rentang int (dengan perlindungan overflow)
    // Bahkan secara praktis, int standar hanya aman sampai 12!
    // Demi keamanan, kita batasi sampai 16 sesuai kebutuhan soal
    private static final int MAX_FACTORIAL_INPUT = 16;

    /**
     * Menghitung faktorial dari bilangan bulat tidak negatif.
     *
     * @param n angka yang akan dicari faktorialnya
     * @return nilai faktorial dari n (n!)
     * @throws IllegalArgumentException jika n bernilai negatif
     * @throws IllegalArgumentException jika n > 16 (risiko overflow)
     */
}
```

```
public static int factorial(int n) {
    // Validasi 1: tolak input negatif (tidak terdefinisi secara matematis)
    if (n < 0) {
        throw new IllegalArgumentException(
            "Factorial is not defined for negative numbers. " +
            "Input was: " + n + ". Please provide a non-negative integer."
        );
    }

    // Validasi 2: cegah overflow integer untuk input besar
    if (n > MAX_FACTORIAL_INPUT) {
        throw new IllegalArgumentException(
            "Input exceeds maximum value (" + MAX_FACTORIAL_INPUT + ") for integer
factorial calculation. " +
            "Input was: " + n + ". Factorial(" + n + ") would cause integer overflow. " +
            "Consider using long or BigInteger for larger factorials."
        );
    }

    // Hitung faktorial (kedua validasi sudah lolos)
    int result = 1;
    for (int i = n; i > 0; i--) {
        result *= i;
    }

    return result;
}
}
```