

MODUL 11

ERROR HANDLING

Error handling (penanganan kesalahan) dalam Java adalah mekanisme untuk menangani situasi tak terduga (error) yang terjadi saat program dijalankan (run-time). Hal ini penting untuk mencegah program berhenti secara tiba-tiba, memberikan informasi kepada pengguna, dan memungkinkan program tetap berjalan atau keluar dengan cara yang terkontrol.

Secara umum, error dalam Java diklasifikasikan ke dalam tiga kategori:

- **Compile-Time Error:** Terjadi saat proses *kompilasi* (sebelum program dijalankan). Contoh: kesalahan sintaks, salah ketik nama variabel/metode, dll. Error ini tidak ditangani oleh mekanisme error handling seperti try-catch.
- **Run-Time Error (Exception):** Terjadi saat program sedang berjalan. Contoh: pembagian dengan nol, akses array di luar indeks, NullPointerException, dsb. Error ini dapat ditangani menggunakan blok try-catch.
- **Logical Error:** Program berjalan tanpa crash, tapi hasilnya salah. Tidak menyebabkan exception; harus ditemukan melalui pengujian/debugging.

11.1 Exception

Exception atau pengecualian adalah masalah atau gangguan yang terjadi ketika program sedang dijalankan. Saat terjadi exception, alur normal program terganggu, sehingga program bisa berhenti secara tidak wajar (crash). Hal ini tidak baik, karena bisa menyebabkan pengalaman pengguna yang buruk atau kehilangan data. Oleh karena itu, Java menyediakan cara untuk menangani exception, agar program tetap bisa berjalan dengan baik.

Exception bisa muncul karena berbagai alasan, contohnya:

- Pengguna memasukkan data yang tidak valid (misalnya input angka, tapi diketik huruf).
- Program ingin membuka file, tapi file tersebut tidak ditemukan.
- Jaringan terputus di tengah proses komunikasi.
- Java kehabisan memori saat program dijalankan.

Agar kita tahu cara menanganinya, Java membagi exception menjadi tiga jenis utama:

1. Checked Exception

Checked Exception adalah jenis exception yang harus ditangani saat compile time. Jika tidak ditangani, program tidak akan berhasil dikompilasi.

Contoh Skenario:

Membaca file yang tidak ada → FileNotFoundException.

```
import java.io.*;

public class CheckedExceptionExample {
    public static void main(String[] args) {
        try {
            FileReader file = new FileReader("data.txt"); // File tidak ada
            BufferedReader reader = new BufferedReader(file);
            System.out.println(reader.readLine());
            reader.close();
        } catch (IOException e) {
            System.out.println("Terjadi kesalahan saat membaca file: " + e.getMessage());
        }
    }
}
```

Karena file "data.txt" tidak ditemukan, Java mewajibkan kita menangani exception ini dengan try-catch atau throws. Jika tidak ditangani, akan muncul error saat compile.

2. Unchecked Exception

Unchecked Exception terjadi saat program dijalankan, bukan saat dikompilasi. Penanganannya tidak wajib, tapi sebaiknya tetap dilakukan.

Contoh Skenario:

- Membagi angka dengan nol → `ArithmeticException`
- Mengakses elemen array di luar batas → `ArrayIndexOutOfBoundsException`

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

        try {
            System.out.println(angka[5]); // indeks di luar batas array
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Terjadi kesalahan: " + e.getMessage());
        }

        try {
            int hasil = 10 / 0; // pembagian dengan nol
        } catch (ArithmeticException e) {
            System.out.println("Tidak bisa membagi dengan nol: " + e.getMessage());
        }
    }
}
```

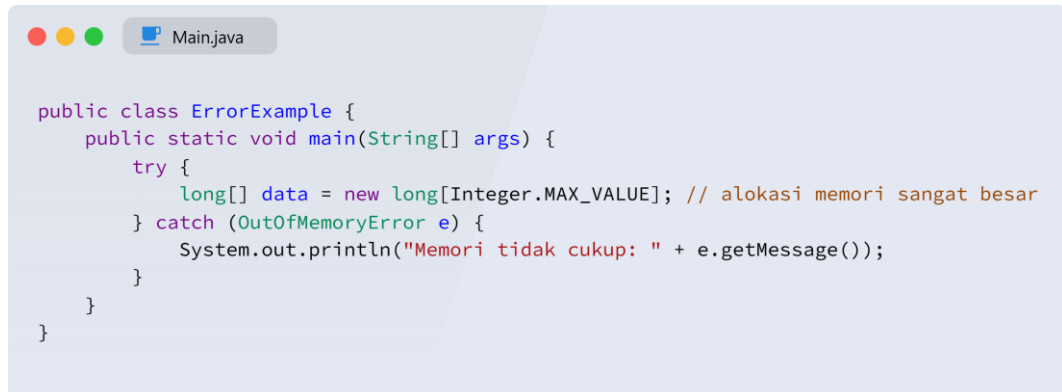
Dari contoh skenario tersebut walaupun tanpa try-catch pun program akan tetap bisa dikompilasi. Tapi jika terjadi error saat dijalankan, program bisa **berhenti tiba-tiba** jika tidak ditangani.

3. Error

Error adalah masalah serius yang tidak seharusnya ditangani dalam kode. Biasanya disebabkan oleh masalah sistem atau JVM, seperti kehabisan memori.

Contoh Skenario:

- Rekursi tak terbatas → `StackOverflowError`
- Mengalokasikan memori terlalu besar → `OutOfMemoryError`



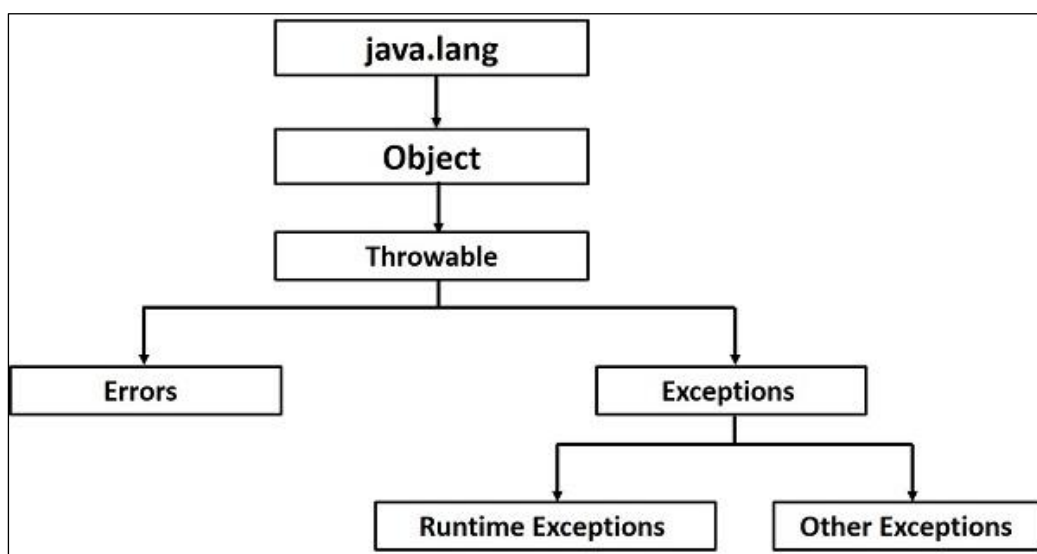
```
public class ErrorExample {
    public static void main(String[] args) {
        try {
            long[] data = new long[Integer.MAX_VALUE]; // alokasi memori sangat besar
        } catch (OutOfMemoryError e) {
            System.out.println("Memori tidak cukup: " + e.getMessage());
        }
    }
}
```

Error seperti `OutOfMemoryError` **tidak bisa diantisipasi sepenuhnya** dan sebaiknya dihindari dengan menulis kode yang efisien. Penanganannya tidak umum seperti exception biasa.

11.2 Exception Hierarchy

Java memiliki struktur turunan (hierarki) untuk semua jenis exception dan error. Semua exception di Java berasal dari kelas `Throwable`, yang merupakan induk dari dua cabang utama yakni:

- **Exception:** Digunakan untuk masalah yang dapat ditangani dalam program.
- **Error:** Digunakan untuk masalah serius yang terjadi di lingkungan runtime (JVM), seperti kehabisan memori, dan tidak seharusnya ditangani oleh program.



Sumber: [Java Exceptions](#)

Contoh Exception Built-in di Java:

Kategori	Contoh Exception	Keterangan
Checked Exception	IOException, FileNotFoundException	Harus ditangani dengan try-catch
Unchecked Exception	ArithmeticException, NullPointerException	Runtime, opsional ditangani
Error	OutOfMemoryError, StackOverflowError	Tidak ditangani program

Metode Penting dari Kelas Throwable:

No.	Metode	Keterangan
1	getMessage()	Mengembalikan pesan detail dari exception
2	getCause()	Menjelaskan penyebab exception (jika ada)
3	toString()	Mengembalikan nama class + pesan error
4	printStackTrace()	Menampilkan jejak error ke konsol (biasanya digunakan untuk debugging)
5	getStackTrace()	Mengembalikan array stack trace
6	fillInStackTrace()	Mengisi ulang stack trace saat ini

11.3 Penanganan Exception

1. Try-Catch Block

try-catch digunakan untuk menangani error (exception) agar program tidak langsung berhenti ketika terjadi kesalahan.

Struktur umum:

```
try {  
    // kode yang mungkin menimbulkan error  
} catch (JenisException e) {  
    // blok ini dijalankan jika terjadi error  
}
```

Contoh:

```
try {  
    int data = 10 / 0; // error: pembagian dengan nol  
} catch (ArithmeticException e) {  
    System.out.println("Terjadi kesalahan: " + e.getMessage());  
}
```

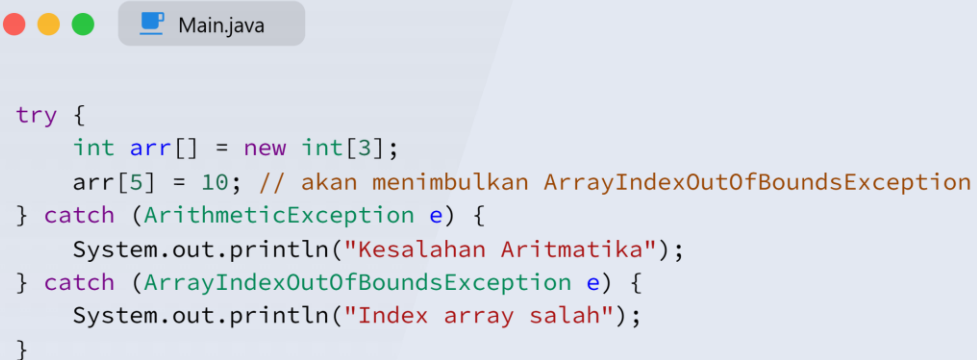
2. Multiple Catch Blocks

Kita bisa menggunakan lebih dari satu catch untuk menangani jenis error yang berbeda.

Struktur:

```
try {
    // kode yang bisa menimbulkan berbagai error
} catch (IOException e) {
    // menangani error IO
} catch (ArithmeticException e) {
    // menangani pembagian nol
} catch (Exception e) {
    // menangani error umum
}
```

Contoh:



```
try {
    int arr[] = new int[3];
    arr[5] = 10; // akan menimbulkan ArrayIndexOutOfBoundsException
} catch (ArithmeticException e) {
    System.out.println("Kesalahan Aritmatika");
} catch (ArrayIndexOutOfBoundsException e) {
    System.out.println("Index array salah");
}
```

3. throw dan throws

A. throw

Digunakan untuk melempar (menghasilkan) exception secara manual.

Struktur:

```
throw new ExceptionType("pesan");
```

Contoh:



```
throw new ArithmeticException("Terjadi pembagian dengan nol");
```

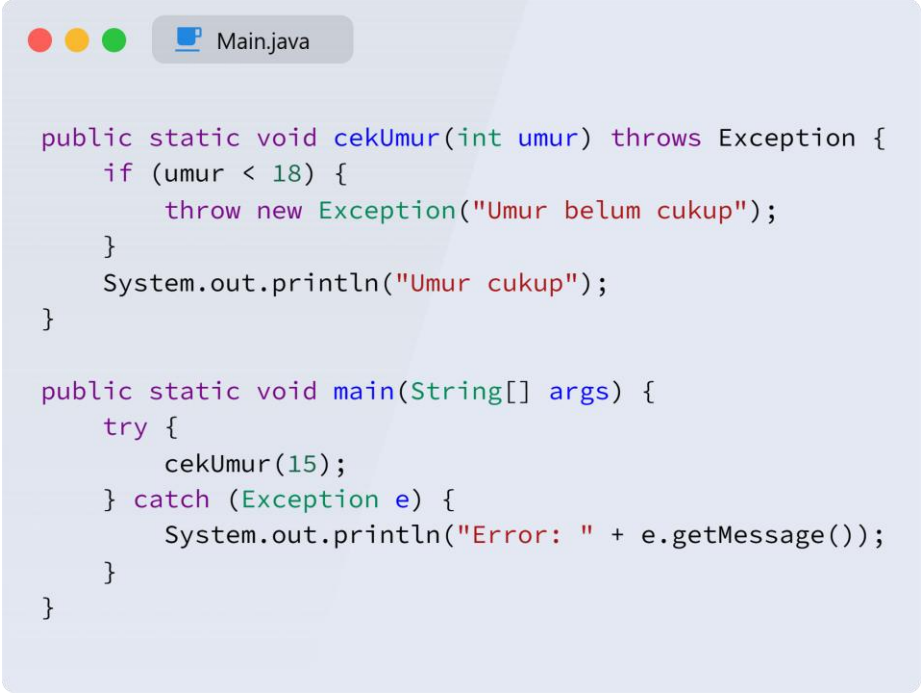
B. throws

Digunakan pada deklarasi method untuk menunjukkan bahwa method tersebut dapat menimbulkan exception.

Struktur:

```
void method() throws IOException {  
    // kode  
}
```

Contoh:



```
public static void cekUmur(int umur) throws Exception {  
    if (umur < 18) {  
        throw new Exception("Umur belum cukup");  
    }  
    System.out.println("Umur cukup");  
}  
  
public static void main(String[] args) {  
    try {  
        cekUmur(15);  
    } catch (Exception e) {  
        System.out.println("Error: " + e.getMessage());  
    }  
}
```

4. finally Block

finally adalah blok kode yang selalu dijalankan, baik terjadi exception atau tidak.

Struktur:

```
try {  
    // kode  
} catch (Exception e) {  
    // penanganan  
} finally {  
    // selalu dijalankan  
}
```

Contoh:

```
try {
    int a = 5 / 0;
} catch (ArithmeticException e) {
    System.out.println("Exception: " + e.getMessage());
} finally {
    System.out.println("Blok finally dijalankan");
}
```

5. User-Defined Exceptions (Exception buatan sendiri)

Kita bisa membuat exception sendiri dengan menurunkan class dari Exception.

Langkah-langkah:

- 1). Buat class baru yang extends Exception.
- 2). Tambahkan constructor.
- 3). Gunakan throw untuk melempar exception tersebut.

Contoh:

```
// 1. Definisi exception buatan
class UmurTidakValidException extends Exception {
    UmurTidakValidException(String pesan) {
        super(pesan);
    }
}

// 2. Penggunaan dalam program
public class CekUmur {
    static void validasiUmur(int umur) throws UmurTidakValidException {
        if (umur < 18) {
            throw new UmurTidakValidException("Umur minimal 18 tahun");
        }
    }

    public static void main(String[] args) {
        try {
            validasiUmur(16);
        } catch (UmurTidakValidException e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

Kesimpulan

Konsep	Fungsi
try-catch	Menangani exception agar program tetap berjalan
Multiple catch	Menangani berbagai jenis exception
throw	Melempar exception secara manual
throws	Menyatakan bahwa method dapat melempar exception
finally	Kode yang selalu dijalankan, cocok untuk menutup file atau koneksi
User-defined Exception	Membuat exception sendiri sesuai kebutuhan

Latihan

Kerjakan soal dibawah ini!

Latihan	
Durasi Pengerjaan	
Start	
End	
Soal	<p>Studi Kasus: Sistem Peminjaman Buku Perpustakaan</p> <p>Deskripsi:</p> <p>Anda diminta untuk membangun aplikasi konsol Java yang digunakan untuk peminjaman buku di perpustakaan kampus. Program akan menerima input dari pengguna seperti:</p> <ul style="list-style-type: none">• Nama pengguna• ID buku• Lama peminjaman (dalam hari) <p>Sistem harus memverifikasi data yang diinput, memastikan ID buku valid (terdaftar dalam sistem), serta tidak membiarkan peminjaman dilakukan dengan input yang salah atau tidak masuk akal.</p> <p>Tujuan:</p> <ul style="list-style-type: none">• Mengimplementasikan exception handling (try-catch, throw, dan throws)• Membuat exception kustom (custom exception)• Menggunakan finally untuk menutup resource• Menangani berbagai jenis exception (input, data tidak ditemukan, logika bisnis) <p>Ketentuan Program:</p> <ol style="list-style-type: none">1. Terdapat daftar ID buku yang tersedia: ["B001", "B002", "B003"]2. Pengguna wajib memasukkan ID buku yang valid. Jika tidak, program harus melempar exception.

	<ol style="list-style-type: none"> 3. Lama peminjaman maksimum adalah 14 hari. Jika melebihi, munculkan exception. 4. Semua input dilakukan melalui Scanner 5. Bila terjadi exception, tampilkan pesan error yang jelas. 6. Gunakan blok finally untuk menutup Scanner. <p>Contoh Alur Input:</p> <pre> hp@BENONY-GABRIEL MINGW64 ~/OneDrive/Documents/Kuliah CS/La \$ cd c:\\Users\\hp\\OneDrive\\Documents\\Kuliah\\ CS\\Lainn e\\ Adoptium\\jdk-17.0.12.7-hotspot\\bin\\java.exe -XX:+Show \\Lainnya\\Asprak\\ PBO\\Public\\Praktikum\\ PBO\\bin week12. Masukkan nama Anda: Benony Gabriel Masukkan ID buku: B001 Masukkan lama peminjaman (hari): 30 Exception: Lama peminjaman harus antara 1 - 14 hari. Program selesai. </pre> <p>Hasil yang Diharapkan:</p> <ul style="list-style-type: none"> • Mahasiswa memahami bagaimana error bisa terjadi saat input atau logika program tidak sesuai. • Mampu menggunakan exception handling untuk mencegah program crash. • Terlatih menulis clean code dengan validasi dan struktur terorganisir.
Link Pengumpulan	