**Trapping Error/Error Handling/Exception Handling**

**try – catch**

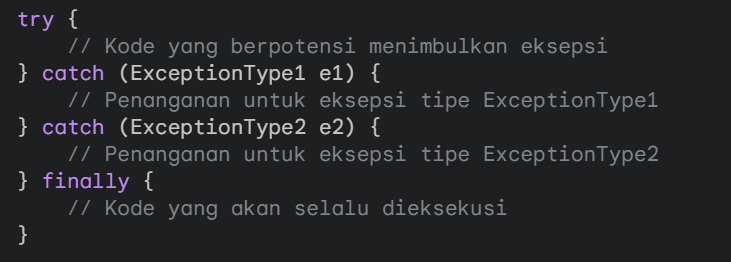
**Pendahuluan**

Dalam pemrograman, kesalahan atau kondisi tak terduga dapat terjadi saat program berjalan. Kondisi ini disebut *eksepsi* atau *exception*. Jika tidak ditangani dengan baik, eksepsi dapat menyebabkan program berhenti secara tiba-tiba atau menghasilkan output yang tidak diinginkan. Untuk mengatasi hal ini, Java menyediakan mekanisme *penanganan eksepsi* atau *exception handling* menggunakan blok try-catch.

**Konsep Dasar**

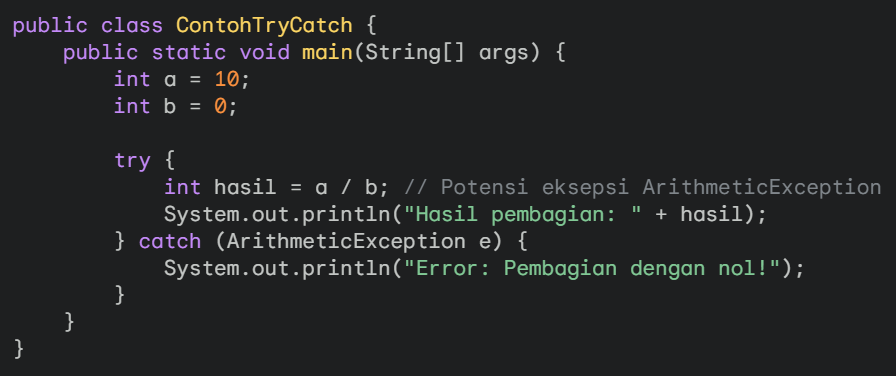
* **Eksepsi (Exception):** Sebuah objek yang mewakili kesalahan atau kondisi tak terduga yang dapat terjadi saat program berjalan.
* **Blok try:** Blok kode yang berisi kode yang dapat berpotensi menimbulkan eksepsi (kesalahan).
* **Blok catch:** Blok kode yang menangkap dan menangani eksepsi yang terjadi di dalam blok try.
* **Blok finally (opsional):** Blok kode yang akan selalu dieksekusi, baik terjadi eksepsi atau tidak. Biasanya digunakan untuk membersihkan sumber daya (misalnya, menutup file atau koneksi database).

**Sintaks Dasar**



Contoh Program

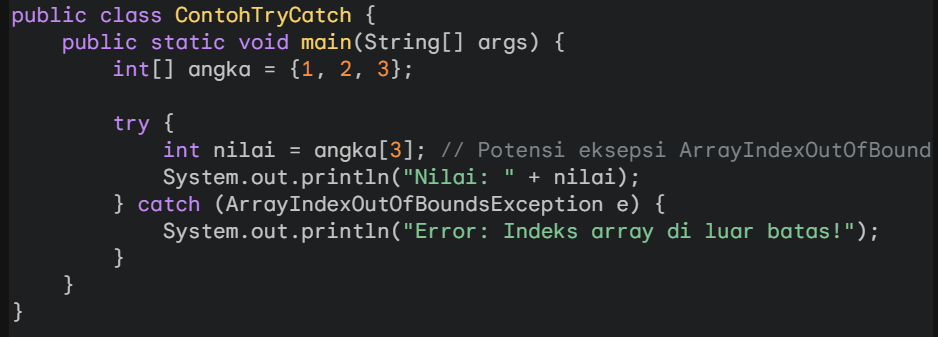
1. Pembagian dengan Nol



**Output:**

Error: Pembagian dengan nol!

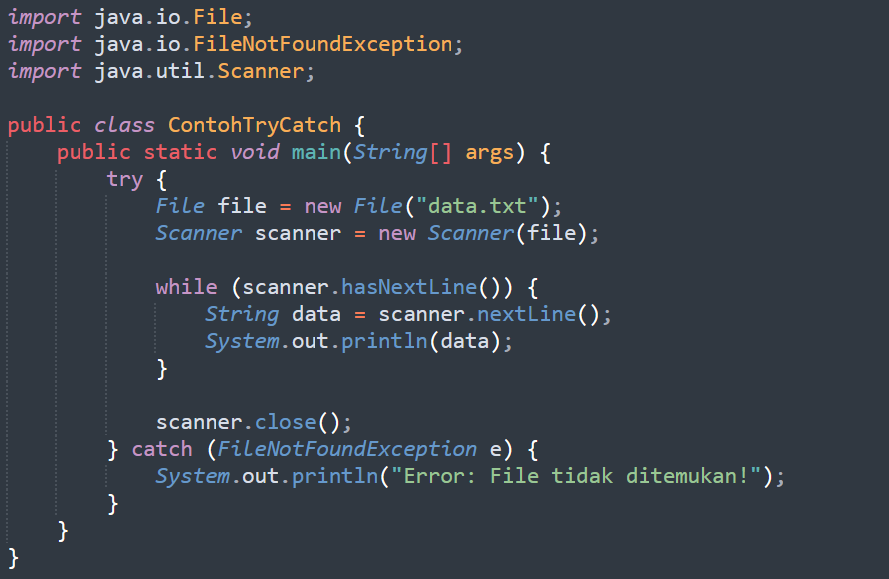
1. ArrayIndexOutOfBoundsException



Output:

Error: Indeks array di luar batas!

1. FileNotFoundException



**Output (jika file** data.txt **tidak ada):**

Error: File tidak ditemukan!

1. Penggunaan finally



**Output (jika file data.txt tidak ada):**

Error: File tidak ditemukan!

Scanner ditutup.

**Jenis-jenis Eksepsi**

* **Checked exceptions:** Eksepsi yang harus ditangani secara eksplisit menggunakan try-catch atau dideklarasikan menggunakan throws pada method signature. Contoh: IOException, SQLException.
* **Unchecked exceptions:** Eksepsi yang tidak perlu ditangani secara eksplisit. Biasanya disebabkan oleh kesalahan pemrograman. Contoh: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException.

**Best Practices**

* **Tangani eksepsi sespesifik mungkin:** Gunakan blok catch untuk menangkap tipe eksepsi yang spesifik, bukan hanya Exception secara umum.
* **Berikan pesan error yang informatif:** Cetak pesan error yang jelas dan membantu pengguna memahami masalahnya.
* **Gunakan finally untuk membersihkan sumber daya:** Pastikan sumber daya seperti file atau koneksi database selalu ditutup, baik terjadi eksepsi atau tidak.
* **Jangan sembunyikan eksepsi:** Hindari menggunakan blok catch kosong atau hanya mencetak pesan error tanpa melakukan tindakan perbaikan.
* **Pertimbangkan untuk membuat custom exception:** Untuk kasus-kasus khusus dalam aplikasi Anda, buatlah kelas eksepsi sendiri yang mewarisi dari kelas Exception atau subclass-nya.

**Kesimpulan**

Penanganan eksepsi dengan try-catch adalah mekanisme penting dalam Java untuk membuat program yang lebih robust dan tahan terhadap kesalahan. Dengan memahami konsep dasar, jenis-jenis eksepsi, dan best practices, Anda dapat menulis kode yang lebih baik dan menangani kondisi tak terduga dengan lebih efektif.

**Latihan:**

* Modifikasi contoh program di atas untuk menangani berbagai jenis eksepsi.
* Buatlah program yang membaca input dari pengguna dan tangani potensi eksepsi seperti NumberFormatException jika input tidak valid.
* Coba buat custom exception sendiri dan gunakan dalam program Anda.



**Penjelasan Kode:**

1. **Import java.util.Scanner:** Mengimpor kelas Scanner untuk membaca input dari pengguna.
2. **Membuat objek Scanner:** Membuat objek scanner untuk membaca input dari System.in (standar input, biasanya keyboard).
3. **while (true):** Loop tak terbatas yang akan terus meminta input dari pengguna sampai input valid diberikan.
4. **try block:**
   * Meminta pengguna untuk memasukkan angka bulat.
   * Membaca input sebagai string menggunakan scanner.nextLine().
   * Mengonversi string input menjadi integer menggunakan Integer.parseInt(). Jika input bukan angka bulat yang valid, akan terjadi eksepsi NumberFormatException.
   * Jika konversi berhasil, mencetak angka yang dimasukkan dan keluar dari loop menggunakan break.
5. **catch block:**
   * Menangkap eksepsi NumberFormatException jika terjadi.
   * Mencetak pesan error yang informatif kepada pengguna.
6. **finally block (opsional):**
   * Tidak digunakan dalam contoh ini, tetapi bisa ditambahkan jika ada sumber daya yang perlu dibersihkan (misalnya, menutup file atau koneksi database).
7. **scanner.close():** Menutup objek scanner setelah selesai digunakan (baik practice).

**Cara Kerja Program:**

1. Program meminta pengguna untuk memasukkan angka bulat.
2. Program mencoba mengonversi input menjadi integer.
3. Jika konversi berhasil, program mencetak angka tersebut dan keluar dari loop.
4. Jika konversi gagal (input tidak valid), program menangkap eksepsi NumberFormatException, mencetak pesan error, dan meminta pengguna untuk memasukkan ulang angka.
5. Proses ini berulang sampai pengguna memasukkan angka bulat yang valid.

**Tambahan:**

* Anda dapat mengganti Integer.parseInt() dengan Double.parseDouble() jika ingin membaca input berupa angka desimal (floating-point).
* Anda dapat menambahkan lebih banyak blok catch untuk menangani jenis eksepsi lain yang mungkin terjadi, misalnya IOException jika terjadi kesalahan saat membaca input.
* Anda dapat menggunakan blok finally untuk memastikan bahwa sumber daya (seperti scanner) selalu ditutup, bahkan jika terjadi eksepsi.