Exception

Exception adalah *event* (kejadian) yang mengacaukan jalannya suatu program. *Worst case scenario* ketika suatu program mengalami exception adalah *termination*. *Termination* (penutupan) program adalah hal yang harus dihindari. Untuk itu kita harus menangani exception yang terjadi di program, atau yang biasa disebut sebagai *handle exception*.

Kode yang baik adalah yang terhindar dari segala bentuk kejadian yang menyebabkan efek buruk kepada program. Oleh karena itu mari kita kenali dulu berbagai macam exception yang ada di Java.

Pada dasarnya ada 3 jenis exception berdasarkan kategorinya.

1. Checked Exception, adalah exception yang terjadi saat compile time. Di sini programmer harus menambahkan kode untuk meng-handle exception kategori ini. Jika tidak di-handle maka kode yang dituliskan akan error dan tidak akan bisa dikompilasi. Contohnya adalah exception `java.io.FileNotFoundException`.
2. Unchecked Exception, adalah exception yang terjadi saat execution time. Eror ini terjadi dalam lingkup internal dari aplikasi kita, biasanya terjadi karena salah penulisan kode atau penggunaan salah terhadap satu API. Contohnya adalah NullPointerException.
3. Error, adalah exception yang diluar kendali user atau programmer. Eror ini terjadi di lingkup eksternal dari aplikasi kita. Ketika exception ini terjadi maka tidak ada yang bisa kita lakukan untuk mengatasinya, contohnya ketika perangkat kerasnya rusak saat kita ingin membaca data.

Dari definisi di atas maka error exception dan unchecked exception termasuk dari exception yang berada pada execution time. Sebabnya, keduanya hanya dialami ketika program sudah berjalan. Perbedaanya adalah unchecked berada di dalam internal program kita, sedangkan error exception berada di eksternal program kita.

Kemudian kode apa yang harus kita tambahkan untuk mengatasi berbagai macam exception tersebut? Kita harus kenal dengan 3 block kode yaitu try, catch, dan finally.

Try-Catch

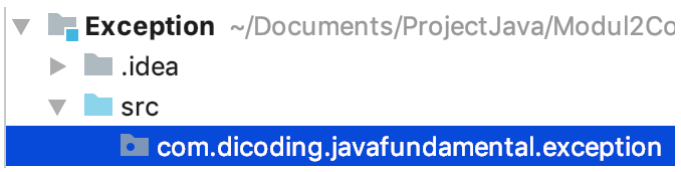
Kode yang rawan dengan exception kita masukkan ke dalam block try-catch. Kode yang kita masukkan ke dalam block try-catch biasa disebut sebagai protected code. Kodenya sepert ini.

```
1. try{
2.    // Protected code
3. } catch (Exception e){
4.    // Catch block
5. }
```

Codelab Try-Catch

Misalnya, program kita mencoba akses ke filereader. `FileReader` merupakan komponen yang dapat membaca suatu berkas.

1. Buatlah proyek baru dengan nama Exception dengan nama package `com.dicoding.javafundamental.exception` di dalamnya:



2. Buatlah sebuah kelas baru di dalamnya dengan nama `Main`, kemudian tambahkan kode berikut:

```
1.  package com.dicoding.javafundamental.exception;
2.
3.  import java.io.File;
4.  import java.io.FileReader;
5.
6.  public class Main {
7.      public static void main(String[] args) {
8.          String location = "D://namafile.txt";
9.          File file = new File(location);
10.         FileReader fr = new FileReader(file);
11.     }
12. }
```

Jika diperhatikan, kode di atas akan mengalami eror "Unhandled exception: java.io.FileNotFoundException".Ini dikarenakan exception `java.io.FileNotFoundException` merupakan checked exception. Maka kita perlu memasukkannya ke dalam try-catch agar bisa di -*compile*.

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

        String location = "D://namafile.txt";
        File file = new File(location);
        FileReader fr = new FileReader(file);
    }
}
```

Unhandled exception: java.io.FileNotFoundException

Lalu, bagaimana jalanya program ketika suatu kode kita masukkan ke dalam try-catch? Normalnya, program akan menjalankan kode yang ada di dalam block try. Jika tidak ada kesalahan yang terjadi, maka program akan berjalan normal sampai baris kode di dalam try selesai. Dan jika ketika terjadi eror di dalam try, maka program akan keluar dari block try dan menjalankan block catch.

3. Kita gunakan contoh kode yang sebelumnya yaitu membaca berkas dengan menggunakan komponen filereader. Berkas yang akan kita coba baca mempunyai nama latihan.txt. Ubahlah kelas `Main` menjadi seperti ini:

```java
package com.dicoding.javafundamental.exception;

import java.io.File;
import java.io.FileReader;

public class Main {
    public static void main(String[] args) {
        try {
            // Mencoba membaca berkas latihan.txt
            File file = new File("D://latihan.txt");
            FileReader fr = new FileReader(file);
            // Jika berhasil maka tampilkan pesan
            System.out.println("Read file berhasil");
        } catch (Exception e) {
            // Jika terjadi kesalahan maka tampilkan pesan
```

4. Kita asumsikan ada 2 skenario yang bisa terjadi ketika kode di atas dijalankan. Pertama, ada berkas latihan.txt di drive d: dan program berhasil membaca berkasnya. Kedua, berkas latihan.txt tidak ada di drive d:. Jika dijalankan dengan kedua skenario maka output-nya adalah :

   Output skenario berkas ditemukan:

   ```
   Read file berhasil
   ```

   Output skenario berkas tidak ditemukan:

   ```
   D:\latihan.txt (The system cannot find the file specified)
   ```

   Dari output di atas maka dapat disimpulkan bahwa kode untuk menampilkan pesan “Read file berhasil” tidak dijalankan karena terjadi exception ketika mencoba membaca berkas `latihan.txt`. Program akan langsung menuju ke block catch dan menampilkan pesan penyebab dari erornya.

Multiple catch

Dari kode sebelumnya kita menggunakan Exception untuk menangani semua exception yang terjadi.
Sebenarnya, kita bisa memilih tipe exception apa saja yang hanya ingin kita handle. Ini sangat berguna ketika kita ingin meng-handle tiap exception dengan perlakuan yang berbeda.

Lebih lanjut, tidak hanya 1 exception saja yang bisa kita tangani. Kita bisa menambahkan lebih dari 1 block catch.
Misalnya seperti ini.

```
try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}
```

Dan setelah Java 7 kita bisa menggunakan 1 baris kode untuk meng-*handle* multi catch. Seperti ini contohnya.

```
catch (IOException|SQLException ex) {
    logger.log(ex);
    throw ex;
}
```

Finally

Block finally adalah block yang di tambahkan di akhir block try-catch. Finally akan selalu dijalankan setelah try-catch baik terjadi exception atau tidak. Finally bermanfaat ketika kita ingin melakukan cleanup code. Cleanup code di sini maksudnya adalah de-alokasi sumber daya, Artinya semua sumber daya yang dibuka atau digunakan pada blok try seperti koneksi jaringan, database, berkas, stream, dll akan ditutup dengan menjalankan instruksi yang ditulis pada blok finally.

Contohnya adalah seperti ini.

```
package com.dicoding.javafundamental.exception;

public class Main {
    public static void main(String[] args) {
        int[] a = new int[5];
        try {
            System.out.println("Akses elemen ke 5 :" + a[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Exception thrown :" + e);
        } finally {
            a[4] = 10;
            System.out.println("Nilai elemen terakhir: " + a[4]);
        }
    }
}
```

Output

```
Exception thrown :java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 5
Nilai elemen terakhir: 10
```

Contoh program di atas adalah contoh akses elemen array yang melebihi dari jumlah indeks array-nya. Program akan masuk ke block catch, kemudian akan menjalankan kode yang ada di block finally, yaitu mengisi nilai indeks ke-4 array dengan 10.

‹ Sebelumnya

Selanjutnya ›