Dasar-Dasar Pemrograman 2

Tutorial 09 Kelas C & F

Exception Handling

Selasa, 23 April 2019 - 16:00 WIB

Pada tutorial/lab sebelumnya, kalian telah mempelajari bagaimana cara memodelkan masalah *object-oriented* dengan *abstract class* dan *interface*. Pada lab kali ini, kalian akan belajar bagaimana cara mengatasi munculnya suatu error dengan *Exception Handling*.

Exception Handling adalah event yang terjadi ketika program menemui kesalahan pada saat instruksi program dijalankan. Banyak hal yang dapat menimbulkan event ini, misalnya crash, harddisk rusak dengan tiba-tiba, sehingga program-program tidak bisa mengakses file-file tertentu. Programmer pun dapat menimbulkan event ini, misalnya dengan melakukan pembagian dengan bilangan nol, atau pengisian elemen array melebihi jumlah elemen array yang dialokasikan dan sebagainya.

Try, Catch, Finally

Catching merupakan salah satu cara yang dapat dilakukan untuk menangani *exception* yang muncul. Untuk melakukan *catching exceptions*, kita akan mengenal dengan yang namanya blok *Try* dan blok *Catch*. Pada blok *try* berisi potongan program yang dapat memunculkan *exception*. Sedangkan pada blok *catch*, berisi perintah-perintah yang dilakukan ketika menangani *exception* yang muncul pada blok *try*. *Catch* dapat ditulis berkali-kali dengan menangkap berbagai jenis *exception* yang berbeda-beda.

```java
class Example1 {
Run | Debug
public static void main(String args[]) {
        int num1, num2;
        try {
            // Kita memprediksi akan terjadi exception pada blok ini
            // dengan begitu kita memasukkannya ke dalam blok try.
            num1 = 0;
            num2 = 62 / num1;
            System.out.println(num2);
            System.out.println("Hey I'm at the end of try block");
        }
        catch (ArithmeticException e) {
            // Blok ini hanya akan terjadi ketika Arithmetic Exception muncul
            System.out.println("You should not divide a number by zero");
        }
        catch (Exception e) {
            // Blok ini menangkap Exception umum, akan dieksekusi ketika
            // ada exception yang belum ter-handle oleh catch sebelum-sebelumnya
            System.out.println("Exception occurred");
        }
        System.out.println("I'm out of try-catch block in Java.");
    }
 }
```

Selain blok *try* dan *catch*, terdapat satu blok lagi yang disebut dengan ***finally***. Blok *finally* akan selalu tereksekusi ketika keluar dari blok *try*. Hal ini memastikan bahwa perintah-perintah yang berada di dalam blok *finally* akan selalu tereksekusi meskipun nanti ternyata muncul *exception* yang tidak terduga. Kegunaan *finally* lebih dari hanya *handling exception*. *Finally* bisa memungkinkan *programmer* menghindari kode *cleanup* terlewati karena *return*, *continue*, atau *break*. Meletakkan kode *cleanup* di dalam blok *finally* merupakan praktik yang bagus meskipun tidak akan muncul *exception*.

```java
try {
    int n = Integer.parseInt(scanner.nextLine());
}
catch (NumberFormatException e) {
    System.out.println("Tried to parse non integer.");
}
finally {
    // Kode di dalam sini akan selalu tereksekusi
    System.out.println("Finally, it's here!");
}
```

Throwing Exceptions

Throw

Keyword ***throw*** dalam Java digunakan untuk melemparkan *exception* secara eksplisit pada *method* atau suatu blok kode. *Keyword* ***throw*** biasanya digunakan untuk melemparkan *exception* buatan sendiri (*custom exception*). Berikut ini adalah cara menggunakan *throw*:

```
throw <instance>

// Contoh:
throw new ArithmeticException("Pembagian dengan nol");
```

Pada contoh di atas, blok kode yang mengandung ***throw*** tersebut akan menghasilkan sebuah *exception* bernama ArithmeticException. *Exception* tersebut kemudian bisa di-*handle* oleh yang menggunakan *method* yang mengandung potongan program tersebut dengan melakukan *try-catch* yang akan dijelaskan pada submateri Try, Catch, Finally.

Instance class yang bisa dilemparkan oleh ***throw*** haruslah merupakan *subclass* dari *class* Throwable. Biasanya, *class exception* buatan sendiri dibuat dengan meng-*extend class* Exception atau RuntimeException karena kedua *class* tersebut sudah merupakan *subclass* dari Throwable.

Throws

Throws merupakan salah satu cara untuk meng-*handle* kemunculan *exception* selain menggunakan *try-catch*. Dengan menggunakan ***throws***, *exception* yang muncul pada suatu blok kode dalam suatu *method* akan dilemparkan lagi menuju kode yang memanggil *method* tersebut. Dengan demikian, biarkan yang menggunakan *method* tersebut yang melakukan *handle* lebih lanjut dari *exception* yang telah muncul.

Berikut contoh cara penggunaan ***throws***:

```java
public int divide(int a, int b) throws ArithmeticException {
    int result = a / b; // Akan terjadi exception ketika b bernilai 0.
    return result;
}
```

Pada contoh di atas, *exception* akan dilemparkan dari hasil pembagian ketika peubah b bernilai 0. Kemudian kita melakukan *handle* dengan cara melemparkan lagi *exception* tersebut keluar *method* kepada kode yang melakukan pemanggilan *method* divide (saat ini kita tidak perlu memikirkan dulu bagaimana nanti *exception* tersebut di-*handle* oleh kode yang memanggil *method* divide tersebut). Dengan demikian, seakan-akan *method* divide lah yang menghasilkan *exception* tersebut.

```java
public static void main(String[] args) {
    // Akan terjadi exception seakan-akan dihasilkan oleh method divide
    int x = divide(5, 0);
}
```

Custom Exception

Misalkan kita ingin membuat sebuah exception sendiri. Misal kita ingin health point (hp) dari sebuah monster tidak boleh kurang dari 0 dan kita ingin kode kita ingin mengeluarkan sebuah exception sebut saja exceptionnya adalah NegativeHealthPointException maka dengan prinsip inheritance kita dapat membuat sebuah class yang meng-*extends* class Exception (class Exception secara default sudah ada pada java tanpa harus dibuat lagi).

```java
class NegativeHealthPointException {
    public NegativeHealthPointException(String message) {
        super(message);
    }
}
```

Pada kode diatas dapat dilihat juga bahwa *constructor* yang dibuat harus menerima parameter string yang dapat digunakan apabila exception yang diberikan mengandung parameter string.

Sama dengan exception pada biasanya kita juga dapat melakukan throw pada custom exception. Misalkan sama dengan kasus diatas dimana kita akan melempar exception apabila health point bernilai negatif.

```
if (healthPoint < 0) {
    throw new NegativeHealthPointException("HP negatif");
}
```

Pesan Rahasia

Dek Depe sedang senang bertukar pesan dengan temannya, Dek Depy. Namun, Kak Pewe yang menyebalkan selalu *kepo* sehingga ia selalu membaca pesan yang mereka tulis. Merasa dimata-matai, Dek Depe yang sudah mengambil matkul Kriptografi berencana membuat enkripsi pada pesannya agar Kak Pewe tidak bisa membaca pesan mereka.

Enkripsi yang dilakukan terhadap pesan adalah sebagai berikut :

1. Pesan hanya terdiri dari **huruf kapital dan spasi**.
2. Pesan dienkripsi dengan suatu key yang berupa angka dalam bentuk String.
3. Pesan akan dienkripsi per-karakter dengan cara menambahkan nilai ordinal dari karakter tersebut dan nilai dari satu karakter key, hasil penjumlahan ini akan menggantikan karakter yang dienkripsi. Contoh :
    - Pesan = HAI
    - Key = 323
    - Proses enkripsi : H = ord(H) + 3, A = ord(A) + 2, I = ord(I) + 3. Sehingga hasil enkripsinya adalah 756776.
4. Jika panjang key kurang dari panjang pesan, key akan digunakan secara *cyclic.* Contoh:
    - Pesan = SALAM
    - Key = 123
    - Proses enkripsi : S = ord(S) + 1, A = ord(A) + 2, L = ord(L) + 3, A = ord(A) + 1, M = ord(M) + 2. Sehingga hasil enkripsinya adalah 8467796679.

Program untuk mendekripsi pesan telah dibuat oleh Dek Depe dan diberikan kepada Dek Depy untuk mendekripsi (membaca) pesan terenkripsi yang dikirimkan oleh Dek Depe. Namun, karena Dek Depy agak gaptek dan ceroboh, dia kerap kali salah memasukkan pesan dan key nya sehingga program yang dibuat Dek Depe sering error dan membuat Dek Depy bingung. Karena Dek Depe sudah kelelahan membuat program dekripsi, ia meminta bantuanmu, mahasiswa Fakultas Ilmu Komputer yang

jago programming, untuk membuat program dekripsi buatan Dek Depe tidak mudah error dengan **Exception Handling**.

Sudah disediakan template untuk *code* dekripsi. Tugas kalian adalah mengimplementasi exception handling untuk memeriksa apakah pesan yang akan dienkripsi sudah memenuhi syarat-syarat pesan enkripsi atau belum. Exception yang harus kalian handle adalah:

- Key yang diberikan harus berupa String of Numbers.
- Pesan yang dihasilkan hanya berupa **huruf kapital dan spasi** saja.
- Panjang String yang akan didekripsi harus genap.

Buatlah *class Exception* yang sesuai dan ubahlah *code* Simulator.java agar memanfaatkan *class Exception* yang sudah kalian buat.

Contoh input dan output:

```
========================================
Masukan key: 12345
73677983848081
HALOOOO
73677983843377688190 86
HALOO KAMUU
749168697033776881908687888990
IYAAA KAMUUUUUU
847568847033786875 78
SIAPA LAGI
85718769853385728170797368883790 67
TETAP SEMANGAT YA
stop
========================================
```

Komponen Penilaian :

Komponen	Penjelasan	Bobot
Custom Exception class untuk isi pesan yang tidak valid	Membuat exception class yang akan di-*throw* jika isi pesan tidak sesuai dengan spesifikasi. Custom Exception yang dibuat harus menggunakan nama class yang deskriptif.	15%
Custom Exception class untuk	Membuat exception class yang akan di-*throw* jika panjang pesan tidak sesuai dengan spesifikasi. Custom Exception	15%

panjang pesan yang tidak valid	yang dibuat harus menggunakan nama class yang deskriptif.	
Custom Exception class untuk key yang tidak valid	Membuat exception class yang akan di-*throw* jika input key tidak valid. Custom Exception yang dibuat harus menggunakan nama class yang deskriptif.	15%
Throw Exception pada Decryption.java	Melemparkan exception yang sesuai jika pesan, key, atau hal lainnya tidak sesuai dengan spesifikasi. Exception yang di-*throw* harus berisi pesan error yang jelas.	25%
Exception Handling pada Simulator.java	Meng-handle exception yang di-*throw* oleh class Decryption maupun exception yang mungkin terjadi pada Simulator.	20%
Kerapian	Penulisan program mengikuti kaidah dan konvensi yang telah diajarkan. Program ditulis dengan rapi, terstruktur, dan disertakan oleh dokumentasi secukupnya.	10%

Deadline :

Selasa, 23 April 2019
Pukul **17:40 WIB**

Format Pengumpulan :

Satukan semua class-class yang dibutuhkan pada zip archive dan kumpulkan di slot pengumpulan yang telah disediakan di SCeLE dengan format :

[Kode Asdos]_[Nama]_[Kelas]_[NPM]_Lab[X].zip

Contoh :

JO_JonathanChristopherJakub_C_1706040151_Lab0.zip