

Dasar-Dasar Pemrograman 2

Tutorial 09 Kelas B, D & E

Exception Handling

Kamis, 25 April 2019 - 16:00 WIB



FAKULTAS
ILMU
KOMPUTER

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.

```

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* terlewat karena *return*, *continue*, atau *break*. Meletakkan kode *cleanup* di dalam blok *finally* merupakan praktik yang bagus meskipun tidak akan muncul *exception*.

```

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**:

```
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 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.

```
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 *exception*nya 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).

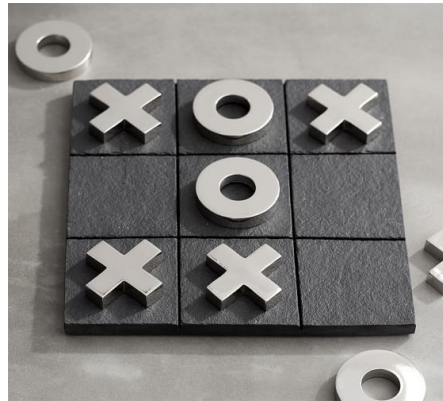
```
class NegativeHealthPointException extends Exception{
    public NegativeHealthPointException(String message) {
        super(message);
    }
}
```

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");
}
```

Tic Tac Toe



Dek Depe dan Dek Depy menemukan papan tic tac toe di sekre. Mereka berniat memainkan [tic tac toe](#) tersebut. Namun Dek Depe menyadari papan tic tac toe tersebut habis dimainkan Kak Pewe. Dek Depe bertanya apakah mungkin papan tersebut masih bisa dimainkan? Dek Depe meminta bantuanmu, mahasiswa Fakultas Ilmu Komputer yang jago programming, untuk membuat program memeriksa apakah papan tic tac toe tersebut masih bisa dimainkan atau tidak dengan **Exception Handling**. Pada tutorial ini hanya ada 4 kasus yang harus dihandle, yaitu:

Exception	Penjelasan	Contoh Board
BoardFormatException	Board terdiri dari karakter 'X', 'O', dan '.' (Terdapat karakter null dan 'A')	<pre> A X X . . O . . </pre>
BoardStateException	Tic tac toe dimainkan secara bergantian, maka jumlah X = 3 dan O = 1 menyalahi aturan.	<pre> X O X . . X . . . </pre>
WinnerFoundException	Telah ditemukan pemenangnya.	<pre> X X X O O </pre>
BoardFullException	Board sudah terisi penuh.	<pre> X O X O X O O X O </pre>

Anda **wajib** membuat *class Exception* yang sesuai dan ubahlah *code* Simulator.java serta Board.java yang memanfaatkan *class Exception* yang sudah kalian buat.

Prioritas exception jika terjadi kondisi yang melanggar lebih dari satu exception :

1. Exception bawaan java
2. BoardFormatException
3. BoardStateException
4. WinnerFoundException
5. BoardFullException

Contoh

Input	Output
.	Board dapat dimainkan.
XO. OXX OXO	Board dapat dimainkan.
X. . .X. .O.	Board dapat dimainkan.
XXX OOO XXX	BoardStateException: Menyalahi aturan tic tac toe.
XXX OOO XOX	WinnerFoundException: Pemenang sudah ditemukan.
XXO OOX XXO	BoardFullException: Papan tic tac toe sudah penuh.
xOX . .X . .X	BoardFormatException: Terdapat karakter selain 'O', 'X', dan '.'
XXX OOO ...	WinnerFoundException: Pemenang sudah ditemukan.
XOXO OXOX XOXO	java.lang.ArrayIndexOutOfBoundsException: 3

XX OX OO	BoardFormatException: Terdapat karakter selain 'O' , 'X', dan '.'
----------------	---

*) pesan exception dibebaskan (tidak harus seperti contoh)

Komponen Penilaian :

Komponen	Penjelasan	Bobot
Custom Exception class untuk masing - masing kasus exception	Membuat exception class yang akan di-throw sesuai kasus yang terjadi diatas. Custom Exception yang dibuat harus menggunakan nama class yang deskriptif.	45%
Throw Exception pada Board.java	Melemparkan exception yang sesuai kasus diatas 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 Board 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 :

Kamis, 25 April 2019

Pukul **17:40 WIB**

Format Pengumpulan :

Satukan semua class-class (.java) 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 :

RDO_DepeDepi_E_1801234567_Lab9.zip