VERSION 2.1

FEBRUARI, 2023

PEMROGRAMAN BERORIENTASI OBJEK

MODUL 6 - JAVA EXCEPTIONS

DISUSUN OLEH:

Muhammad Nizar Zulmi Rohmatulloh

Jody Yuantoro

DIAUDIT OLEH:

Aminudin, S.Kom., M.Cs.

PRESENTED BY: TIM LAB. IT

UNIVERSITAS MUHAMMADIYAH MALANG

PEMROGRAMAN BERORIENTASI OBJEK

TUJUAN

1. Mahasiswa dapat memahami *exception* dan *error* pada program.
2. Mahasiswa dapat menerapkan *exception handling* dengan Java.
3. Mahasiswa dapat menerapkan pelemparan *exception* (*throw statement*).
4. Mahasiswa dapat menerapkan pembuatan kelas *custom exception*.

TARGET MODUL

1. Mahasiswa dapat menerapkan *error/exception handling*.

PERSIAPAN

1. Java Development Kit.
2. Text Editor / IDE (Visual Studio Code, Netbeans, Intellij IDEA, atau yang lainnya).

KEYWORDS

- Error
- Exception Handling
- Checked Exception
- Runtime Exception

TEORI

- **Java Exceptions**

  Exception (atau *exceptional event*) adalah suatu masalah yang muncul saat eksekusi suatu program. Saat *exception* terjadi, program yang berjalan secara normal terganggu dan berhenti secara abnormal (biasa disebut *error*), maka *exceptions* ini perlu ditangani (*exception handling*). *Exception* dapat terjadi dengan berbagai alasan, misalnya skenario berikut: *user* memasukkan data yang *invalid* (format tidak benar); suatu *file* yang akan dibuka ternyata tidak ada; ketika akan terhubung ke jaringan tetapi ada kesalahan baik *software* maupun *hardware*, dan lain sebagainya.

  Exception dapat dikategorikan menjadi 3 jenis:

  1) **Checked Exception**

     Merupakan *exception* yang dicek saat kode di-*compile* sehingga *exception* ini harus ditangani jika tidak program tidak akan berhasil di-*compile*. Misalnya, jika ingin membuka file tapi file tidak ditemukan, maka *exception* akan muncul.

  2) **Unchecked Exception (Runtime Exception)**

     Kebalikan dari *checked exception*, jenis ini akan berhasil di-*compile* tetapi akan muncul saat program berjalan. Misalnya perhitungan dibagi dengan 0; suatu variabel bernilai null; mengakses indeks suatu array melebihi indeks terbesar.

  3) **Error**

     Error sebenarnya berbeda dengan *exception*, mudahnya, *error* adalah selain kedua hal di atas, yaitu di luar kendali *user* dan *programmer*, seperti kehabisan memori (OutOfMemoryError). Error tidak dapat di-*resolve*.

- **Try … Catch … Finally**

  Penerapan *exceptions handling* adalah menggunakan scope try-catch dan finally (opsional), kode yang ada pada blok try adalah kode yang kemungkinan akan menimbulkan *exception*, sedangkan kode pada blok catch adalah bagaimana kita akan menangani exception tersebut, yang terakhir blok finally adalah blok kode yang akan

selalu dieksekusi baik terjadi exception maupun tidak. Penulisannya adalah sebagai berikut.

```
try {
    /*
     * Kode yang kemungkinan akan
     * menimbulkan exception
     */
} catch ([TipeException] [variabelException]) {
    /*
     * Kode penanganan exception yang muncul.
     */
} catch (ArithmeticException aException) { // ← contoh tipe
    /*
     * Blok catch dapat lebih dari 1 (min. 1).
     */
} finally {
    /*
     * Kode yang akan selalu dieksekusi baik
     * muncul exception ataupun tidak.
     * Blok finally bersifat opsional.
     */
}
```

Contoh:

1. **Mengakses variabel / objek yang bernilai null.**

```
package edu.praktikum.pbo.modul6;

public class Main {
    public static void main(String[] args) {
        String text = null;
        try {
            System.out.println(text.length());
        } catch (NullPointerException nPE) {
            System.err.println(nPE.getMessage());
            System.out.println("Pada blok finally akan diisi
string kosong");
        } finally {
            text = ""; // string tidak lagi null.
        }
        System.out.println(text); // tereksekusi
    }
}

Output baris pertama adalah pesan dari nPE.getMessage():

Cannot invoke "String.length()" because "text" is null
Pada blok finally akan diisi string kosong
// baris ini akan kosong hasil dari println(text) text kosong.
```

2. **Mengakses array lebih dari panjangnya.**

```
package edu.praktikum.pbo.modul6;

public class Main {
    public static void main(String[] args) {
        try {
            int a[] = new int[5];
            a[10] = 50;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(e.getMessage());
        }
    }
}

Output:

Index 10 out of bounds for length 5
```

3. **Bilangan dibagi oleh bilangan nol.**

```
package edu.praktikum.pbo.modul6;

public class Main {
    public static void main(String[] args) {
        try {
            int a = 50 / 0; // ArithmeticException
        } catch (Exception e) {
            System.out.println(e.getClass());
            System.out.println(e.getMessage());
        }
    }
}

Output:
class java.lang.ArithmeticException
/ by zero
```

Perhatikan pada contoh nomor 3, meskipun yang kita tangkap adalah *class* Exception, Java akan tahu dari *class* yang mana *exception* tersebut dilempar (throwed), Anda bisa mengeceknya dengan *method* `getClass()` seperti pada contoh.

- **Custom Exception**

  Anda dapat membuat *Custom Exception* (kelas turunan **`Exception`** selain *built-in* di JDK) yang bertujuan agar kode Anda lebih mudah dibaca, selain itu membuat *Custom Exception* bisa digunakan untuk mengkategorikan atau memfilter saat nanti *exception* tersebut di-*catch.* Berikut contohnya:

```
package edu.praktikum.pbo.modul6;
```

```
public class InvalidRangeException extends Exception {

    InvalidRangeException(String message) {
        super(message);
    }

    InvalidRangeException(String message, Throwable cause) {
        super(message, cause);
    }

}
```

Contoh di atas kita membuat *exception* baru yang bernama **`InvalidRangeException`**, pembuatan *exception* harus mewarisi kelas **`Exception`**, dengan konstruktor berparameter **`message`** untuk pesan apa yang akan kita sampaikan saat *exception* tersebut di-*catch,* sedangkan untuk parameter **`cause`** untuk penggunaan yang lebih lanjut, jika Anda tertarik silakan mempelajari ***Throwable*** dan ***Exception Chaining*** karena topik tersebut tidak di-*cover* di sini.

- **Throw dan Throws**

  Setelah membuat *custom exception* di atas, kita bisa menggunakannya dengan *keyword* ***throw*** dan menandai suatu method yang melempar exception dengan *keyword* ***throws.*** Berikut contohnya:

```java
package edu.praktikum.pbo.modul6;

public class Main {

    public static boolean validateAge(int age) throws InvalidRangeException {
        if (age < 0 || age > 150) {
            throw new InvalidRangeException(message:"Umur harus dalam rentang 0 - 150");
        }
        return true;
    }

    Run | Debug
    public static void main(String args[]) {
        int age = -1;

        try {
            validateAge(age);
        } catch (InvalidRangeException iRangeException) {
            System.out.println(iRangeException.getMessage());
            // Output baris di atas: Umur harus dalam rentang 0 - 150
        }
    }

}
```

Perhatikan kode di atas, digunakan ***`throw new`*** melempar *exception* sedangkan *keyword* ***`throws`*** (dengan **`s`**) sebelum kurung kurawal pembuka (**`{`**) digunakan untuk menandai bahwa method tersebut melempar exception, pada bagian ini kita bisa menandainya dengan multiple exception sesuai dengan apa exception apa saja yang dilempar.

Contoh:

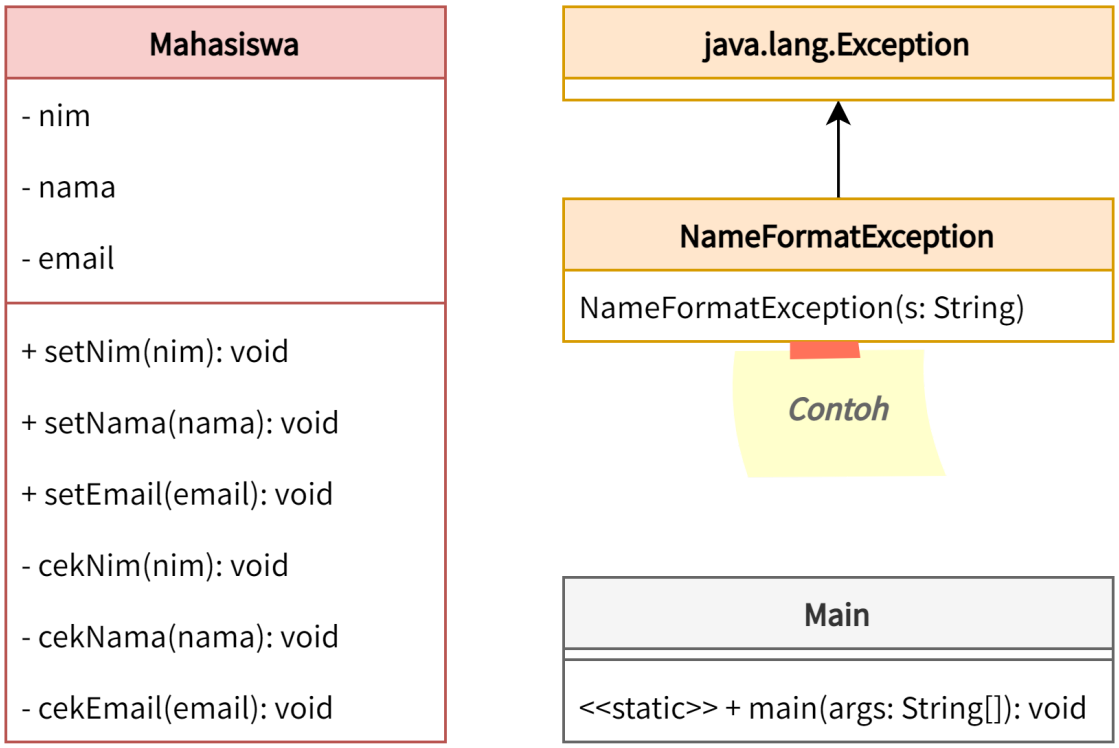
```java
public void validateName(String name) throws NameFormatException,
NameTooShortException {

        // Implementasi kode lain
        throw new NameFormatException("Nama mengandung angka atau simbol");
        // Implementasi kode lain
        throw new NameTooShortException("Nama minimal 8 huruf");
}
```

CODELAB

Buatlah program inputan data diri mahasiswa informatika dengan spesifikasi sebagai berikut:

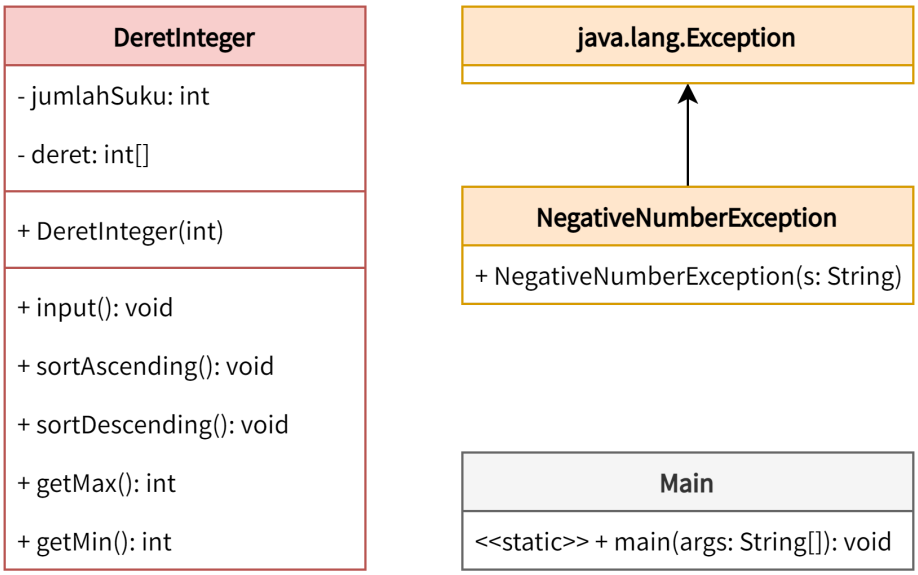
- Inputan yang ada minimal terdiri dari nama, nim, dan email.
- Hanya NIM mahasiswa informatika dengan kode "10370311".
- NIM tidak boleh kurang atau lebih dari 15 digit dan harus berupa angka, tidak diperbolehkan mengandung simbol dan alfabet.
- *Catch* semua *exception* yang mungkin terjadi.
- Usahakan juga menggunakan keyword `finally`. Jika tidak berikan alasannya.
- Buatlah *custom exceptions* untuk setiap *exception* yang ada (minimal 1). Misalnya: NIM kurang atau lebih dari 15 digit; nama mengandung selain alfabet; email tidak benar; dlsb.
- Method-method pengecekan bersifat *private*. Inputan menggunakan method *setters*.
- Jika pengecekan salah tampilkan pesan error, kemudian meminta inputan kembali.
- ***Hint***: Untuk melakukan pengecekan string Anda dapat mencari referensi **Pattern Matching (Regular Expression)**.
- Kira-kira seperti di bawah ini diagramnya:



TUGAS

1. Anda diminta untuk membuat *class* dengan nama **DeretInteger** yang melakukan penyimpanan dan manipulasi suatu deret angka dengan tipe data integer.
   a. Class tersebut mempunyai konstruktor yang akan diigunakan untuk menginisialisasi jumlah suku yang ada di dalam deret integer tersebut.
   b. Mempunyai *method* **`input()`** yang digunakan untuk memasukan inputan elemen-elemen deret integer (inputan dimasukan satu persatu ke dalam variabel), ke dalam *class*.
   c. Di dalam *method* **`input()`** tersebut terdapat *exception handling* yang mampu mengecek apakah inputan yang dimasukan tersebut bilangan integer atau bukan. Jika yang dimasukan bukan nilai integer maka proses inputan tersebut akan diulang pada indeks elemen yang sama.
   d. Di dalam *method* **`input()`** tersebut juga terdapat *exception handling* yang akan melempar *exception* ke *class* **NegativeNumberException** apabila inputan merupakan bilangan negatif.
   e. Di dalam *method* **`input()`** tersebut juga terdapat *exception handling* yang melakukan pengecekan untuk inputan bilangan 0. Apabila yang dimasukan tersebut adalah bilangan 0, maka proses inputan akan diulang pada indeks yang sama.
   f. Tambahkan *method* yang digunakan untuk melakukan pengurutan deret secara *ascending*, *descending*, dan *method* untuk mencari nilai maksimal dan minimal yang ada di dalam deret. Masing-masing *method* tersebut memiliki *exception handling* yang digunakan untuk menangani apabila proses yang terjadi di dalam *method* tersebut mengakibatkan terjadinya *exception* (contoh: ketika mengakses index deret yang tidak mempunyai elemen).
2. Pastikan juga membuat *custom exception* berupa *class* **NegativeNumberException** yang merupakan turunan dari *class* **Exception**. *Class* ini akan meng-*handle exception* yang terjadi pada waktu method **`input()`** pada *class* di atas dijalankan.

3. Seperti biasa, **Main** *class* harus ada untuk menjalankan objek dari *class* **DeretInteger** di atas.
4. Kira-kira beginilah diagram dari deskripsi tugas di atas.



RUBRIK PENILAIAN

ASPEK PENILAIAN	POIN
Codelab	20
Tugas	35
Pemahaman	45
TOTAL	**100**

Selamat Mengerjakan
Tetap Semangat