NETWORK LABORATORY
Electrical Engineering Department, 2nd floor
Universitas Indonesia
Depok. 16424

PRAKTIKUM PEMROGRAMAN BERORIENTASI OBJEK 2017/2018

CASE STUDY

MODUL 7: Exception and Threading

Pada modul ini praktikan diharapkan dapat mengimplementasi dan menggunakan **Try-Catch, Exception, User-Defined Exception, dan MultiThreading** pada Java. Pada modul ini, Anda diminta untuk membuat beberapa kelas baru untuk fungsi **User-Defined Exception** dan melakukan implementasinya pada fungsi main() di JHotel**. Sedangkan untuk threading akan membuat program yang terpisah dari JHotel.**

Pada modul ini data yang dikumpulkan adalah:

- **2 commit** pada Github dengan masing-masing batas waktu adalah **19 April 2018 18:00 (Tugas 1 sampai 11)** dan **21 April 2018 23:55 (Tugas 12)**
- **1 dokumentasi** untuk hasil ***running program*** **pada Tugas 11** sama dengan workspace yang Anda commit ke github batas waktu adalah **19 April 2018 18:00.**

Tugas 1: Penyesuaian UML dan Syarat Method

1. **UML Modul 7 yang baru hanya terdapat pada kelas baru dan Customer saja.**
   Note: Pada modul 6 sebelumnya, constructor pada **Room** seharusnya berupa seperti ini:

```
public Room(Hotel hotel, String nomor_kamar)
{
    this.hotel = hotel;
    this.nomor_kamar = nomor_kamar;
    statusKamar = StatusKamar.VACANT;
}
```

   SingleRoom, DoubleRoom, dan PremiumRoom juga hanya memiliki *constructor* **hotel** dan **nomor_kamar** saja.
2. Tambahkan **email** pada seluruh *constructor* **Customer**.
3. Pada *method* **"addCustomer",** tambahkan syarat apabila ditemukan ID atau email yang sama maka penambahan customer gagal dilakukan **(*return false*).**
4. Pada *method* **"addHotel",** tambahkan syarat apabila ditemukan ID atau nama hotel dengan lokasi yang sama maka penambahan hotel gagal dilakukan **(*return false*).**

Tugas 2: Membuat Kelas "PelangganSudahAdaException"

1. Buatlah sebuah kelas baru bernama "**PelangganSudahAdaException**"
2. Extend kelas "**Exception**" pada kelas **"PelangganSudahAdaException**"
3. Implementasi *field* dan *method* pada kelas "**PelangganSudahAdaException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**PelangganSudahAdaException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Pelanggan dengan data : "**
6. Pada konstruktor "**PelangganTidakDitemukanException**", simpan *variable lokal* "pelanggan_input" ke *variable instance* "pelanggan_error"
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override super.getMessage seperti di bawah ini:

```
super.getMessage() + pelanggan_error + " sudah terdaftar.";
```

Tugas 3: Membuat Kelas "PelangganTidakDitemukanException"

1. Buatlah sebuah kelas baru bernama "**PelangganTidakDitemukanException**"
2. Extend kelas "**Exception**" pada kelas **"PelangganTidakDitemukanException**"
3. Implementasi *field* dan *method* pada kelas "**PelangganTidakDitemukanException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**PelangganTidakDitemukanException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Data Customer dengan ID : "**
6. Pada konstruktor "**PelangganTidakDitemukanException**", simpan *variable lokal* "pelanggan_input" ke *variable instance* "pelanggan_error"

7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + pelanggan_error + " tidak ditemukan.";
```

Tugas 4: Membuat Kelas "PesananTidakDitemukanException"

1. Buatlah sebuah kelas baru bernama "**PesananTidakDitemukanException**"
2. Extend kelas "**Exception**" pada kelas "**PesananTidakDitemukanException**"
3. Implementasi *field* dan *method* pada kelas "**PesananTidakDitemukanException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**PesananTidakDitemukanException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Pesanan yang dipesan oleh :"**
6. Pada konstruktor "**PesananTidakDitemukanException**", simpan *variable lokal* "pelanggan_input" ke *variable instance* "pelanggan_error".
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + pesanan_error.getPelanggan().getNama() + " tidak ditemukan.";
```

Tugas 5: Membuat Kelas "PesananSudahAdaException"

1. Buatlah sebuah kelas baru bernama "**PesananSudahAdaException**"
2. Extend kelas "**Exception**" pada kelas "**PesananSudahAdaException**"
3. Implementasi *field* dan *method* pada kelas "**PesananSudahAdaException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**PesananSudahAdaException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah

pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**

5. Pesan pada kelas induk diinginkan adalah **"Pesanan yang dipesan oleh : "**
6. Pada konstruktor "**PesananSudahAdaException**", simpan *variable lokal* "pesanan_input" ke *variable instance* "pesanan_error"
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + pesanan_error.getPelanggan().getNama() + " sudah melakukan
pemesanan.”
```

Tugas 6: Membuat Kelas "HotelSudahAdaException"

1. Buatlah sebuah kelas baru bernama "**HotelSudahAdaException**"
2. Extend kelas "**Exception**" pada kelas "**HotelSudahAdaException**"
3. Implementasi *field* dan *method* pada kelas "**HotelSudahAdaException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**HotelSudahAdaException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Hotel dengan nama : "**
6. Pada konstruktor "**HotelSudahAdaException**", simpan *variable lokal* "hotel_input" ke *variable instance* "hotel_error"
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + hotel_error.getNama() + “ sudah terdaftar.”;
```

Tugas 7: Membuat Kelas "HotelTidakDitemukanException"

1. Buatlah sebuah kelas baru bernama "**HotelTidakDitemukanException**"
2. Extend kelas "**Exception**" pada kelas "**HotelTidakDitemukanException**"

3. Implementasi *field* dan *method* pada kelas "**HotelTidakDitemukanException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**HotelTidakDitemukanException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Hotel dengan ID : "**
6. Pada konstruktor "**HotelTidakDitemukanException**", simpan *variable lokal* "hotel_input" ke *variable instance* "hotel_error"
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini::

```
super.getMessage() + hotel_error + " tidak ditemukan.";
```

Tugas 8: Membuat Kelas "RoomSudahAdaException"

1. Buatlah sebuah kelas baru bernama "**RoomSudahAdaException**"
2. Extend kelas "**Exception**" pada kelas "**RoomSudahAdaException**"
3. Implementasi *field* dan *method* pada kelas "**RoomSudahAdaException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**RoomSudahAdaException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Kamar dengan nomor ruang "**
6. Pada konstruktor "**RoomSudahAdaException**", simpan *variable lokal* "room_input" ke *variable instance* "room_error"
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + room_error.getRoomNumber() + " pada " + room_error.getHotel() +
"sudah terdaftar.";
```

Tugas 9: Membuat Kelas "RoomTidakDitemukanException"

1. Buatlah sebuah kelas baru bernama "**RoomTidakDitemukanException**"
2. Extend kelas "**Exception**" pada kelas "**RoomTidakDitemukanException**"
3. Implementasi *field* dan *method* pada kelas "**RoomTidakDitemukanException**" dengan mengikuti UML terbaru.
4. Pada konstruktor "**RoomTidakDitemukanException**", inisialisasi kelas yang diextend dengan menggunakan *method* **super().** Gunakan *method* **super()** ini untuk mengubah pesan (*default*) yang akan dikembalikan oleh **super.getMessage() super.getMessage()** pada method **getPesan().**
5. Pesan pada kelas induk diinginkan adalah **"Kamar yang terletak di : "**
6. Pada konstruktor "**RoomTidakDitemukanException**", simpan *variable lokal* "hotel_error" ke *variable instance* "hotel_input" dan “room_error” ke “room_input”.
7. Pada method **getPesan(),** buatlah pengembalian nilai yang akan meng-override **super.getMessage()** seperti di bawah ini:

```
super.getMessage() + hotel_error + “ dan dengan nomor kamar “ + room_error + “ tidak ditemukan.”;
```

Tugas 10: Implementasi Exception

1. Pada kelas “**DatabasePesanan**”, buatlah metode **“addPesanan”** untuk melemparkan *exception* (throws) "**PesananSudahAdaException**". Method ini baru melemparkan exception jika *return false;*
2. Pada kelas **“DatabasePesanan”**, buatlah metode “**removePesanan”** melemparkan *exception* **"PesananTidakDitemukanException".** Method ini baru melemparkan exception jika *return false;*
3. Pada kelas “**DatabaseCustomer**”, buatlah metode **“addCustomer”** untuk melemparkan *exception* (throws) "**PelangganSudahAdaException**". Method ini baru melemparkan exception jika *return false;*
4. Pada kelas **“DatabaseCustomer”**, buatlah metode **“removeCustomer”** melemparkan *exception* **"PelangganTidakDitemukanException".** Method ini baru melemparkan

exception jika *return false.* Karena metode ini juga akan menyesuaikan dengan "**removePesanan"** pada "**DatabasePesanan"**, maka lakukan **try-catch**.

5. Pada kelas "**DatabaseRoom**", buatlah metode **"addRoom"** untuk melemparkan *exception* (throws) "**RoomSudahAdaException**". Method ini baru melemparkan exception jika *return false;*
6. Pada kelas **"DatabaseRoom"**, buatlah metode "**removeRoom"** melemparkan *exception* **"RoomTidakDitemukanException".** Method ini baru melemparkan exception jika *return false;*
7. Pada kelas "**DatabaseHotel**", buatlah metode **"addHotel"** untuk melemparkan *exception* (throws) "**HotelSudahAdaException**". Method ini baru melemparkan exception jika *return false;*
8. Pada kelas **"DatabaseHotel"**, buatlah metode **"removeHotel"** melemparkan *exception* **"HotelTidakDitemukanException".** Method ini baru melemparkan *exception* jika *return false.* Karena metode ini juga akan menyesuaikan dengan "**removeRoom"** pada "**DatabaseRoom"**, maka lakukan **try-catch**.

Tugas 11: Implementasi Exception pada JHotel

1. Karena ada beberapa metode yang sudah melemparkan **exception**, maka Anda harus menyesuaikan **try-catch** pada beberapa fungsi yang ada di dalam program. Untuk itu, Anda harus mengetes keberhasilan **exception** pada kelas JHotel.

Petunjuk:

- Buatlah 3 objek Customer, 4 objek Hotel, 4 objek Room, 3 objek Pesanan dengan data bebas **(apabila ingin menggunakan hasil dari demo Modul 6 dipersilahkan).** Sesuaikan **try-catch** pada seluruh *method* tersebut.
- Buatlah skenario program sedemikian rupa yang dapat mengetes **seluruh keberhasilan kelas exception.** Contohnya, apabila ingin mengetes **"PesananSudahAdaException",** maka lakukan **"addPesanan"** sekali lagi dengan data yang sama persis seperti Anda buat sebelumnya. **Petunjuk: Try-catch diimplementasikan pada seluruh metode yang melemparkan exception seperti pada tugas 9.**

Contoh hasil yang diharapkan:

```
---------TES EXCEPTION-------------------------------------------------------------------------------------------------

-Exception Pesanan Sudah Ada-
Pesanan yang dipesan oleh : Abi sudah melakukan pemesanan.

-Exception Pelanggan Sudah Ada-
Pelanggan dengan data : Customer{id=4, nama='Abi', email='abi@gmail.com', dob=01 Feb 1990} sudah terdaftar.

-Exception Hotel Sudah Ada-
Hotel dengan nama : Margo Hotel sudah terdaftar.

-Exception Room Sudah Ada-
Kamar dengan nomor ruang : A101 pada Hotel{nama='Ibis', lokasi='Margonda', bintang=5} sudah terdaftar.

-Exception Pesanan Tidak Ditemukan-
Pesanan yang dipesan oleh : Abi tidak ditemukan

-Exception Pelanggan Tidak Ditemukan-
Data customer dengan ID : 3 tidak ditemukan.

-Exception Room Tidak Ditemukan-
Kamar yang terletak di : Hotel{nama='Ibis', lokasi='Margonda', bintang=5} dan dengan nomor kamar A101 tidak ditemukan.

-Exception Hotel Tidak Ditemukan-
Hotel dengan ID : 2 tidak ditemukan.
```

- Print hasil data terakhir yang Anda peroleh.

  Contoh hasil yang diharapkan:

```
---------------------------------HASIL-------------------------------------
Customer{id=1, nama='Abi', email='abi@gmail.com', dob=01 Feb 1990}
Customer{id=2, nama='Budi', email='budi@gmail.com', dob=02 Mar 1990, booking order is in progress}

Hotel{nama='Ibis', lokasi='Margonda', bintang=5}
Hotel{nama='Aston', lokasi='Kutek', bintang=4}
Hotel{nama='Margo Hotel', lokasi='Margonda', bintang=5}

Room{hotel=Aston, roomNumber='C303', tipeKamar='Premium', dailyTariff=0.0, statusKamar=Vacant}
Room{hotel=Ibis, roomNumber='D404', tipeKamar='Premium', dailyTariff=0.0, statusKamar=Booked, pelanggan='Budi'}

Pesanan{pelanggan=Budi, hotel=Ibis, kamar=D404, tipeKamar=Premium, status='DIPROSES'}
```

Tugas 12: Implementasi Threading (Tidak Berada dalam Program JHotel)

Buatlah program yang dapat melakukan perhitungan secara *parallel,* sehingga dapat mengetahui siapa *thread* yang lebih cepat sampai garis *finish*.

Berikut ini merupakan spesifikasi dari program yang harus dibuat:

Program ini merupakan program dengan ***threading***, menjalankan proses yang sama dalam setiap *thread*-nya. Setiap *thread* merepresentasikan seorang kontestan lomba berlari, ketiga kontestan (ibarat untuk *thread*) adalah **Lionel**, **Andres**, dan **Messi**. Agar tidak ada kecurangan, implementasikan **interface runnable** dalam membuat *thread* lombanya.

1. Berikut adalah tahapan-tahapan dalam membuat program:

1. Buat project baru
   (berbeda dengan JHotel).
2. Buat **kelas RunForYourLife** dan implementasikan **interface runnable**.
3. Tentukan *instance variable* yang dibutuhkan beserta *constructor*, *setter*, dan *getter* kelas **RunForYourLife.**
   (Petunjuk: Anda harus menentukan sendiri dengan berdasarkan pada ketentuannya yang ada di nomor 2, begitu juga dengan isi *method* lainnya).
4. Buatlah *method* **run()** dan isi *method* ini dengan proses untuk melakukan lomba lari seperti yang telah ditentukan. Gunakan method **Thread.sleep()** untuk menentukan waktu jeda dan gunakan mekanisme **Try-Catch** untuk *interrupt exception handling*.
5. Buatlah *method* **start()** dan implementasikan *threading*.
6. Buatlah **kelas LetsGo** dan tentukan *instance variable* yang dibutuhkan.
7. Buatlah *method* **random** untuk menentukan bilangan random.
8. Buat *method* **main**, lalu buatlah algoritma untuk menentukan waktu jeda setiap kontestan. Setelah itu panggil objek dari class **RunForYourLife** lalu jalankan *thread* menggunakan *method* **start().**

2. Berikut adalah ketentuan pembuatan program:

1. Setiap kontestan akan berlari dari **checkpoint 1 hingga ke 20.**
2. Waktu jeda berlari setiap kontestan berbeda-beda dan ditentukan secara **random (100-500 ms).**
3. Karena Lionel baru saja cedera, maka dirinya memiliki waktu berlari **lebih lama** dari Andres dan Messi. Sedangkan Andres yang baru saja sembuh juga memiliki waktu berlari **lebih lama** dibandingkan Messi.

4. Saat **thread dibuat**, maka program akan mencetak:

```
<Nama>, ready.
```

5. Saat **thread dimulai** maka program akan mencetak:

```
<Nama>, set …
```

6. Saat **thread telah dijalankan** saat pertama kali berlari, program akan mencetak:

```
<Nama>, go!
```

7. Selama **thread berlangsung** maka dia akan mencetak:

```
<Nama> has passed checkpoint (<Angka>)
```

8. Jika **terjadi interupsi** pada saat thread berlangsung maka akan mencetak:

```
<Nama> was interrupted.
```

9. Setelah **sampai ke garis akhir** maka program akan mencetak:

```
<Nama> has finished!
```

Contoh *output* yang diharapkan dari program, sebagai berikut **(checkpoint harus berjumlah 20)**: