Dasar-Dasar Pemrograman 2

Lab 08

Exception Handling & Text I/O

Outline:

I. 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*, *hard disk* 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 **try block, catch block,** dan **finally block**:

- **Try block** berisi potongan program yang dapat memunculkan *exception*.
- **Catch block** berisi perintah-perintah yang dilakukan ketika menangani *exception* yang muncul pada *try block*. *Catch* dapat ditulis berkali-kali dengan menangkap berbagai jenis *exception* yang berbeda-beda.
- **Finally block** akan selalu tereksekusi ketika keluar dari *try block*. Hal ini memastikan bahwa perintah-perintah yang berada di dalam *finally block* 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 *finally block* merupakan praktik yang bagus meskipun tidak akan muncul *exception*.

Contoh **try block** dan **catch block**:

```
class Example {
    public static void main(String[] args) {
        int num1, num2;
        // Try block
        try {
            num1 = 0;
            num2 = 96 / num1;
            System.out.println(num2);
            System.out.println("=== END OF TRY BLOCK ===");
        }
        // Catch block
        catch (ArithmeticException e) {
            // Blok ini hanya tereksekusi jika ArithmetixException muncul
            System.out.println("You should not divide a number by zero.")
        }
        catch (Exception e) {
            /**
             * Blok ini menangkap exception umum; dieksekusi ketika ada
             * exception yang belum ter-handle oleh catch sebelum-sebelumnya
             */
            System.out.println("Exception occured.")
        }
```

```
        System.out.println("=== END OF TRY-CATCH BLOCKS ===")
    }
}
```

Contoh **try, catch, finally blocks**:

```
// Try block
try {
    int n = Integer.parseInt(scanner.nextLine());}
// Catch block
catch (NumberFormatException e) {
    System.out.println("Tried to parse non-integer.")
}
// Finally block
finally {
    System.out.println("Yes, finally it's here!")
}
```

Throwing Exceptions

Throw

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 adalah cara menggunakan **throw**:

```
throw <instance>

// Contoh
throw new ArithmeticException("Division by zero.");
```

Pada contoh di atas, blok kode yang mengandung *throw* tersebut akan menghasilkan sebuah *exception* bernama ArithmeticException. *Exception* tersebut kemudian dapat di-*handle* oleh yang menggunakan *method* yang mengandung potongan program tersebut dengan melakukan *try-catch* yang telah dijelaskan pada sub-materi *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 muncul.

Contoh penggunaan **throws**:

```
public int divide(int x, int y) throws ArithmeticException {
    return x / y; // exception jika y bernilai 0
}
```

Pada contoh di atas, *exception* akan dilemparkan dari hasil pembagian ketika peubah y 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 muncul exception seakan-akan dihasilkan oleh method divide
    int x = divide(8, 0);
}
```

Custom Exception

Custom Exception merupakan *exception* yang kita buat sendiri secara *custom*. Misal kita ingin *health point* (hp) dari sebuah monster tidak boleh kurang dari 0 dan kita ingin kode kita mengeluarkan sebuah *exception* yang bernama NegativeHealthPointException. 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 di atas dapat dilihat bahwa *constructor* yang dibuat harus menerima parameter *string* yang dapat digunakan apabila *exception* yang diberikan mengandung parameter *string*. Sama dengan *exception* pada umumnya, kita juga dapat melakukan *throw* pada *custom exception*.

Misal terdapat kasus yang sama dengan kasus di atas, di mana kita akan melempar *exception* apabila *health point* bernilai negatif menggunakan **throw**:

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

II. Text I/O

Text I/O adalah cara untuk membaca Input dari Console atau file lain dan/atau mengeluarkan Output ke sebuah file lain yang pada umumnya bertipe data text (seperti .txt). Selain itu di sini kita akan membahas bagaimana cara membuat suatu file dapat digunakan di dalam suatu program Java. Ada banyak *class* di Java yang menyediakan fungsionalitas untuk *read/write* dari/ke file lain.

Pembaca File

Scanner

Kita telah mengetahui dan familiar dengan **Scanner** untuk membaca sebuah data. Untuk menggunakan Scanner, kita dapat membuat objek Scanner dan menggunakan *method* yang ada pada *class* Scanner untuk mengolah data yang diinput.

Tautan ini dapat memberikan informasi mengenai *method* yang ada pada *class* Scanner.

Contoh penggunaan **Scanner**:

```
import java.util.Scanner;

class ExampleOfScanner {
  public static void main(String[] args) {
    Scanner s = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username: ");

    String userName = s.nextLine();  // Read user input
```

```
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

FileReader

FileReader adalah suatu class di Java yang bisa digunakan untuk membaca karakter dalam file.

Selengkapnya, tautan ini berisi dokumentasi *class* FileReader.

Contoh penggunaan **FileReader**:

```
// Hello.txt
This is an example

// FileRead.java
import java.io.*;

public class FileRead {
    public static void main(String[] args) {
        // Creates FileReader object
        FileReader fr = new FileReader("Hello.txt");
        /**
         * Kita dapat menyimpan seluruh karakter di file ke
         * dalam array dengan method read(char[] c) dengan
         * parameter array char tempat kita ingin
         * menyimpan hasil file yang di-read
         */
        char[] a = new char[50];
        fr.read(a); // reads the content of Hello.txt to array a
        for (char c : a) {
            System.out.print(c);
        } // prints character one by one
        fr.close();
    }
}

// Output
This
```

```
is
an
example
```

BufferedReader

BufferedReader adalah suatu *class* di Java yang berfungsi untuk menyederhanakan pembacaan teks dari suatu *input stream* (misalnya file). Secara singkat, BufferedReader meminimalisir penggunaan I/O *operation* dengan membaca potongan karakter dan menyimpannya di dalam internal *buffer* yang bisa menghasilkan pembacaan yang lebih cepat, karena tidak perlu berinteraksi lagi dengan file yang dibuka. BufferedReader biasanya digunakan dengan membaca suatu objek FileReader dan memasukkannya ke dalam *buffer*.

Selengkapnya, tautan ini berisi dokumentasi *class* BufferedReader.

Contoh penggunaan **BufferedReader**:

```
// input.txt
Hello world!

// ReadFile.java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {
    /**
     * Pembacaan file input.txt dengan BufferedReader bisa saja terdapat
     * error yaitu ketika file yang ingin dibaca tidak ditemukan sehingga
     * digunakan syntax throws IOException seperti di bawah ini
     */
    public static void main(String[] args) throws IOException {
        BufferedReader br = new BufferedReader(new FileReader("input.txt"));
        System.out.println(br.readLine());
        br.close()
    }
}

// Output
```

```
Hello world!
```

Contoh penggunaan BufferedReader di atas juga dapat menggunakan ***try-catch blocks***:

```
// ReadFile.java
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {
    public static void main(String[] args) {
        try {
            BufferedReader br = new BufferedReader(new FileReader("input.txt"));
            System.out.println(br.readLine());
            br.close()
        }
        catch (IOException e) {
            System.out.println("File not found.")
        }
    }
}
```

Kalian juga bisa mencoba bagaimana jika program ReadFile.java tersebut tidak menggunakan 'throws IOException' ataupun 'try-catch' *statement*.

File

File adalah suatu *class* di Java yang merepresentasikan suatu file dalam program Java. Objek dari kelas File dapat menyimpan berbagai atribut dari file dalam komputer itu sendiri, seperti *path* dan sifatnya (apakah hanya bisa dibaca, apakah hanya bisa di-*write*, dan sebagainya). Sekali dibuat, suatu *instance* dari *class* File tidak dapat diubah *path*-nya.

Tautan ini dapat memberikan informasi mengenai *method* yang ada pada *class* File.

Contoh penggunaan **File**:

```
import java.io.File;

class ExampleOfFile {
    public static void main(String[] args) {
        File f = new File("input.txt"); // opens input.txt file
```

```
        System.out.println(f.getName()); // input.txt
        System.out.println(f.getPath()); // input.txt
        System.out.println(f.getAbsolutePath()); // C:\Users\Tasput\input.txt
    }
}
```

Penulis File

PrintWriter

PrintWriter adalah sebuah *class* yang membantu program Java untuk mencetak secara langsung ke dalam *class* File yang telah dibuat. Perbedaan PrintWriter dengan beberapa *class* untuk mencetak lainnya adalah PrintWriter ditujukan untuk mencetak sebuah "teks" ke dalam file, sehingga PrintWriter dilengkapi dengan *method* `format()` dan `printf()`. PrintWriter mengimplementasikan *method* yang ada dalam PrintStream, sehingga PrintWriter harus di-`close()` ketika sudah selesai penggunaannya. Perhatikan bahwa PrintWriter dapat memicu adanya FileNotFoundException pada pembuatan *instance*-nya.

Selengkapnya, tautan ini berisi dokumentasi *class* PrintWriter.

Contoh penggunaan **PrintWriter**:

```
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;

class ExampleOfFile {
    public static void main(String[] args) {
        File f = new File("input.txt"); // opens input.txt file

        System.out.println(f.getName()); // input.txt
        System.out.println(f.getPath()); // input.txt
        System.out.println(f.getAbsolutePath()); // C:\Users\Tasput\input.txt

        PrintWriter pw = new PrintWriter(f);
        pw.println("hello world");
        pw.printf("%d", 99);
        pw.close()
    }
}
```

Soal Lab 08

AVATAR KINGDOM

Sumber: Google Images

WARNING: Sebelum mengerjakan PAHAMI materi Exception Handling dan Text I/O dengan baik, dan juga BACALAH deskripsi soal dengan saksama agar mudah menentukan mana yang try block dan exception block.

Dalam perjalanan pulang setelah menemukan planet baru, Salman tidak sabar untuk mengabarkan kepada penduduk bumi kalau ada planet lain yang bisa dihinggapi. Namun tiba-tiba roket yang Salman kendarai menabrak meteor hingga Salman pun terdampar di tempat yang antah berantah, dalam kondisi tidak sadar.

Tanpa disadari, Salman ditolong oleh mahluk setempat dan dibawa ke sebuah kerajaan yang bernama Avatar Kingdom. Setelah 3 hari 3 malam, akhirnya Salman Terbangun dari tidurnya. Dan seketika Salman pun kaget, dan berkata “DIMANA AKU?? SIAPA KALIAN??”, dengan kebingungan yang Salman miliki, Salman akhirnya ditemui oleh pemimpin kerajaan ini yaitu seorang Avatar yang bernama Aang.

Avatar Aang mengajak Salman untuk berkeliling ke seluruh pelosok dari Avatar Kingdom ini. Setelah kurang lebih 2 jam Salman dan Avatar Aang bercerita dan

melihat-lihat isi dari Avatar Kingdom, mata Salman tertuju ke suatu tempat yaitu tempat dimana para Avatar melakukan berkompetisi dalam sebuah turnamen. Dari rasa penasaran yang dimiliki Salman, Salman pun bertanya kepada Avatar Aang mengenai Tempat Turnamen tersebut, lalu Avatar Aang menceritakan detail dan sistem yang diterapkan dalam turnamen tersebut.

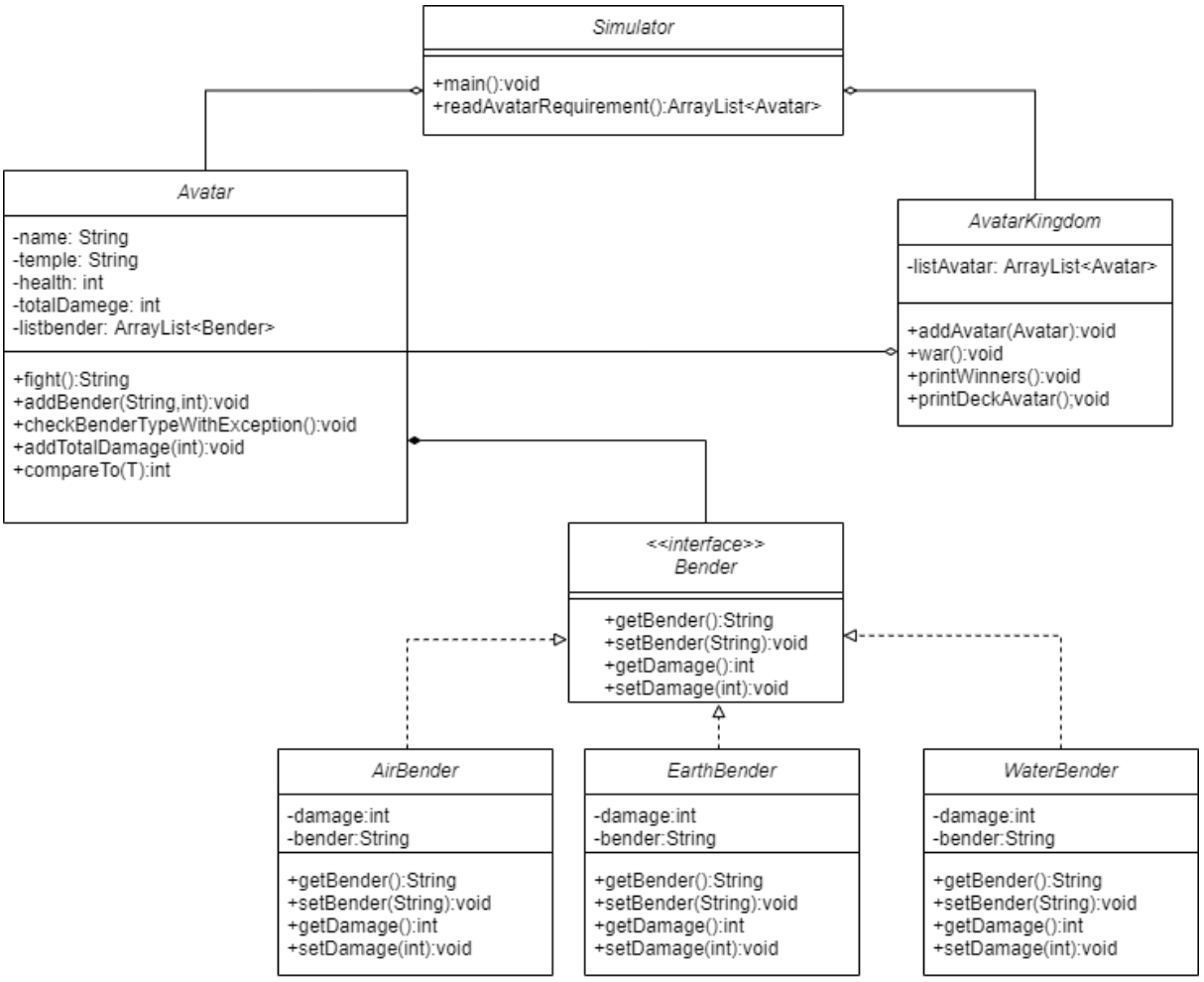
Sejarahnya, turnamen tersebut dibuat ketika negara angin, air, dan bumi sedang mempersiapkan tentara untuk menaklukkan negara api. Nantinya, pemenang dari turnamen tersebut dijadikan pemimpin yang akan memimpin 2020 avatar untuk menyerang negara api. Sedihnya, dalam pertarungan yang sangat dahsyat, negara api runtuh, dan tidak menyisakan orang yang mampu menguasai elemen api, sehingga Fire Bender telah punah di Bumi.

Sistem dari pertarungan turname sangat simpel, yaitu mempertemukan masing-masing avatar untuk saling berperang satu sama lain, nantinya jika avatar yang diserang ataupun menyerang telah mati, maka pertarungan dilanjutkan oleh petarung avatar selanjutnya. Urutan penyerangan dilakukan secara terurut, contoh Avatar 1 menyerang Avatar 2,3,4. Lalu Avatar 2 menyerang Avatar 1,3,4 dan seterusnya. Ketidakberuntungan Avatar 4 yang memiliki kemungkinan sudah mati sebelum dapat menyerang merupakan konsekuensi mengikuti turnamen ini. Urutan 1,2,3 dan 4 diurutkan berdasarkan avatar yang lebih dulu mendaftarkan diri dalam turnamen.

Setelah Salman mendengar sejarah dan sistem yang diterapkan dalam turnamen tersebut, Salman merasa bahwa sistem yang diterapkan sudah kuno, sehingga Salman berpikir dapat membantu Avatar Kingdom untuk memperbaiki sistem turnamennya. Tapi Salman tidak bisa memperbaiki sistemnya sendiri, Salman membutuhkan DekDepe untuk mengimpentasikan program untuk sistem ini.

Spesifikasi Program:

UML Diagram



Classes:

1. Bender (interface)

   Berisi method yang harus diimplementasikan dalam class AirBender, WaterBender, dan EarthBender

2. AirBender, WaterBender, EarthBender
   - Constructor
   - Setter getter
3. Avatar
   - Constructor
   - fight(Avatar other)

        - Melemparkan CanNotAttackException, SuccessAttackException ketika memenuhi beberapa kondisi tertentu, kondisi tersebut sudah tertera di penjelasan Exception Classes
    - addBender(String bender, int damage)
        - Membuat objek Bender sesuai dengan catch exception tertentu
        - Memasukkan objek Bender tersebut kedalam listBender
        - Print pesan exception yang tersedia
    - checkBenderTypeWithException(String bender)
        - Melemparkan AirBenderException, WaterBenderException, dan EarthBenderException ketika memenuhi beberapa kondisi tertentu, kondisi tersebut sudah tertera di penjelasan Exception Classes
    - addTotalDamage(int amountAdded)
        - Menambahkan damage yang dimiliki oleh avatar, total damage didapatkan dari hasil penjumlahan masing-masing bender.damage
    - compareTo(Avatar other)
        - Membandingkan avatar berdasarkan health
4. Avatar Kingdom
    - Constructor
    - addAvatar(Avatar avatar)
        - Menambah avatar ke dalam list
    - war()
        - Mempertemukan masing-masing avatar yang ikut serta dalam sebuah turnamen
        - Mengimplementasikan CanNotAttackException dan SuccessAttackException
        - Jika Avatar A yang hendak menyerang Avatar B, nyawanya habis, maka hanya akan print pesan error sekali saja.
        - Jika SuccessAttackException terpanggil, maka nyawa avatar yang diserang akan dikurangi sejumlah damage yang dimiliki oleh avatar yang menyerang.
    - printWinners()
        - Winner diurutkan dari health yang tertinggi ke health terendah, jika ada health yang sama diantara 2 atau 3 avatar, maka urutan akan dibebaskan.
        - Akan melakukan print sesuai format berikut ini:

          <nomor urut juara>. <nama avatar> dari temple <nama temple>, dengan sisa nyawa <jumlah health>

- printDeckAvatar()
  - Print deck tidak perlu diurutkan berdasarkan nyawa
  - Akan melakukan print sesuai format berikut ini:

    Avatar <nama avatar> dari temple <nama temple>, dengan nyawa <jumlah health>, dan dengan total damage <total damage>, siap bertarung!

5. Exception Classes
   - Membuat Class CanNotAttackException **akan** digunakan dalam method fight (class Avatar) yang **memenuhi** kondisi berikut:
     i. Salah satu dari Avatar nyawanya sudah habis atau menyentuh angka 0
        - Pesan exception diisi dengan format:

          "Nyawa dari Avatar <Nama Avatar> sudah habis, tidak bisa melakukan pertarungan!"

   - Membuat Class SuccessAttackException **akan** digunakan dalam method fight (class Avatar) yang **Tidak Memenuhi** kondisi yang tercantum di CanNotAttackException
     - Pesan exception diisi dengan format:

       "Avatar <nama avatar1>berhasil menyerang <nama avatar2>"

   - AirBenderException, WaterBenderException, dan EarthBenderException terdapat dalam Avatar.java, akan digunakan dalam method checkBenderTypeWithException ketika bender sesuai dengan nama exceptionnya.
     - Pesan exception diisi dengan format:

       "Avatar <nama> menguasai bender <nama bender>"

6. Simulator Class

   Avatar Kingdom dan Avatar akan dibuat didalam simulator, pembuatan Avatar tersebut memiliki requirement khusus yang terdapat dalam avatarReq.java. I/O template sudah dibuat, yang belum di implementasikan yaitu:

   - Membuat objek avatar dan dimasukkan ke dalam array list. Dalam avatarReq, 15 baris pertama memiliki ketentuan
     <nama avatar>;<nama temple>;<nyawa yang dimiliki>

- Menambahkan bender ke dalam avatar. Dalam avatarReq.txt dari baris ke-16 sampai habis memiliki ketentuan
  <avatar ke-n>;<nama bender>;damage

*Asumsi untuk parameter berupa String tidak mungkin ada yang lower case atau upper case untuk semua hurufnya, ex: String type selalu “Water” tidak mungkin “waTer”, “WatEr”, dan sebagainya.

*nyawa tidak boleh negatif

Komponen Penilaian

- 30% Implementasi war pada class AvatarKingdom
- 30%Implementasi class Avatar
- 20% Implementation class Exception
- 15% Implementasi setter, getter, serta constructor
- 5% Dokumentasi dan kerapian kode

Kumpulkan berkas .java yang telah di-zip dengan format penamaan seperti berikut.
`Lab08_[Kelas]_[KodeAsdos]_[NPM]_[NamaLengkap].zip`
Contoh:
`Lab08_A_SMA_1234567890_DekDepe.zip`