PRAKTIKUM 18

MELEMPAR EXCEPTION DAN MEMBUAT EXCEPTION SENDIRI

A. TUJUAN PEMBELAJARAN

1. Mengetahui cara menangani exception dengan cara melempar exception.
2. Mengetahui cara membuat sendiri class exception.

B. DASAR TEORI

Exception adalah suatu kondisi abnormal yang terjadi pada saat menjalankan program. Karena dalam java segala sesuatu merupakan objek, maka exception juga direpresentasikan dalam sebuah objek yang menjelaskan tentang exception tersebut. Contoh exception adalah pembagian bilangan dengan 0, pengisian elemen array diluar ukuran array, kegagalan koneksi database, file yang akan dibuka tidak ada, dan mengakses objek yang belum diinisialisasi.

Terdapat dua penanganan exception yaitu:

- Menangani sendiri exception tersebut.
- Meneruskannya ke luar dengan cara membuat objek tentang exception tersebut dan melemparkannya (throw) keluar agar ditangani oleh kode yang memanggil method(method yang didalamnya terdapat exception) tersebut.

Ada lima keyword yang digunakan oleh Java untuk menangani exception yaitu try, catch, finally, throw dan throws.

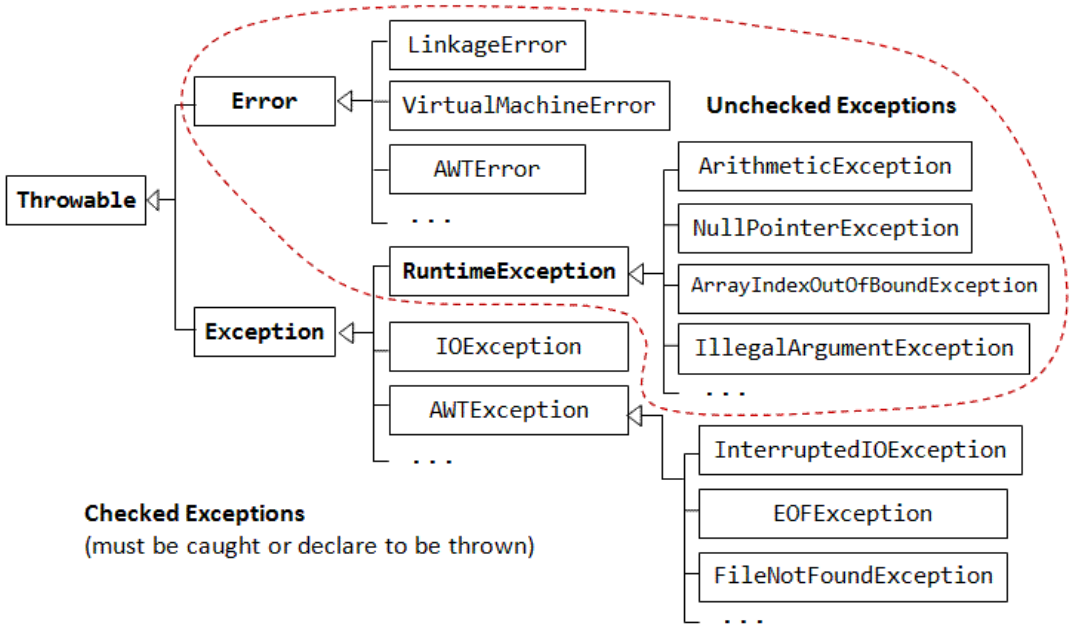
B.1 Tipe-Tipe Exception

Pada exception, superclass tertinggi adalah class Throwable, tetapi kita hampir tidak pernah menggunakan class ini secara langsung. Dibawah class Throwable terdapat dua subclass yaitu class Error dan class Exception. Class Error merupakan tipe exception yang tidak ditangani dengan blok try-catch, karena berhubungan dengan Java run-time system/environment. Untuk exception dengan tipe class Exception, merupakan exception

yang dapat ditangani oleh program. Terdapat subclass dari class Exception diantaranya RunTimeException, IOException, AWTException dan lain-lain.

Semua exception bertipe RunTimeException dan turunannya tidak harus secara eksplisit ditangani dalam program (Unchecked Exception). Contoh subclass dari RunTimeException adalah ArrayIndexOutOfBoundsException, ArithmeticException, NullPointerException dan lain-lain.

Semua tipe exception yang bukan turunan dari class RuntimeException merupakan exception yang harus ditangani atau jika tidak ditangani menyebab error. Dibawah ini adalah hirarki dari exception.



B.2 Penggunaan Blok try-catch

Untuk menangani exception dalam program, dengan meletakkan kode program yang memungkinkan terjadinya exception didalam blok try, diikuti dengan blok catch yang menentukan jenis exception yang ingin ditangani. Contoh :

```
public class Percobaan2 {
    public static void main(String[] args) {
         int a[] = new int[5];
         try{
            a[5] = 100 ;
         }catch(ArrayIndexOutOfBoundsException e){
             System.out.println("Indeks Array melebihi batas");
         }
         System.out.println("Setelah blok try-catch");
    }
```

```
}
```

Output :

```
Terjadi exception karena Indeks Array melebihi batas
Setelah blok try-catch
```

Dapat terjadi kode yang terdapat dalam blok try mengakibatkan lebih dari satu exception. Dalam hal ini, kita dapat menuliskan lebih dari satu blok catch. Contoh :

```
public class Percobaan5 {
    public static void main(String[] args) {
        int bil=10;
        String b[] = {"a","b","c"};
        try{
            System.out.println(bil/0);
            System.out.println(b[3]);
        }catch(ArithmeticException e){
            System.out.println("Error Aritmetik");
        }catch(ArrayIndexOutOfBoundsException e){
            System.out.println("Error Kapasitas Array Melebihi Batas");
        }catch(Exception e){
            System.out.println("Terdapat Error");
        }
    }
}
```

Blok catch yang dijalankan tergantung dengan exception yang terjadi. Java akan menjalankan blok catch yang sesuai dengan tipe exceptionnya saja. Dalam penggunaannya, blok catch dengan tipe subclass harus ditulis lebih dahulu baru diikuti dengan blok catch dengan tipe data superclassnya.

B.3 Menggunakan Keyword ”finally”

Terdapat kode yang harus dijalankan walaupun terjadi atau tidak terjadi exception, misalkan kita membuka file, hal ini memungkinkan terjadinya exception misal file tidak ada, file tidak bisa dibuka, selanjutnya yang harus dilakukan adalah menutup file tersebut.

```
public class Percobaan2 {
    public static void main(String[] args) {
         int a[] = new int[5];
         try{
            a[5] = 100 ;
         }catch(ArrayIndexOutOfBoundsException e){
             System.out.println("Terjadi exception karena Indeks Array
melebihi batas");
         }finally{
```

```
            System.out.println("Selalu Dijalankan");
        }
        System.out.println("Setelah blok try-catch");
    }
}
```

B.4 Menggunakan Keyword ”throw ” dan ”throws”

Secara eksplisit, kita dapat melempar (throw) exception dari program menggunakan keyword throw. Jika exception tersebut adalah checked exception, maka pada method harus ditambahkan throws. Jika exception tersebut adalah unchecked exception, maka pada method tidak perlu ditambahkan throws.

```
public class Percobaan6 {
    public static void method1() throws FileNotFoundException{
        throw new FileNotFoundException("File Tidak Ada");
    }
    public static void main(String[] args) {
        try {
            method1();
        } catch (FileNotFoundException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

B.5 Membuat sendiri Subclass dari Exception

Untuk melakukan ini, kita cukup membuat class baru yang merupakan subclass Exception. Didalam class ini kita cukup mendeklarasikan konstruktor.

```
class Salah extends Exception{
      public Salah(){}
      public Salah(String pesan){
            super(pesan);
      }
}
```

C. TUGAS PENDAHULUAN

Buatlah review 1 halaman mengenai penanganan exception dengan cara melempar exception dan berikan 1 contoh program.

D. PERCOBAAN

Percobaan 1 : Method yang melempar unchecked exception

```
public class Percobaan6 {
    public static void method1(){
        throw new ArrayIndexOutOfBoundsException("Melebihi Kapasitas");
    }
```

```
    public static void main(String[] args) {
        try {
            method1();
        } catch (ArrayIndexOutOfBoundsException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Percobaan 2 : Method yang melempar checked exception

```
public class Percobaan7 {
    public static void method1() throws FileNotFoundException{
        throw new FileNotFoundException("File Tidak Ada");
    }
    public static void main(String[] args) {
        try {
            method1();
        } catch (FileNotFoundException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
```

Percobaan 3 : Memahami mengenai mekanisme exception.

```
public class Exercise1 {
 static void f1() {
   System.out.print("1");
   try {
     System.out.print("2");
     f2();
     System.out.print("3");
   }
   catch (Exception e) { System.out.print("4"); }
   finally {  System.out.print("5"); }
  System.out.println("6");
 }
 static void f2 () throws Exception {
   if (true) throw new Exception();
 }
 public static void main(String s[]) { f1(); }
}
```

Percobaan 4 : Menggunakan konsep Inheritance untuk membuat superclass dan subclass exception. Program menangani exception dengan menangkap subclass exception dengan superclass

```
import javax.swing.*;

class ExceptionA extends Exception {}

class ExceptionB extends ExceptionA {}
```

```
class ExceptionC extends ExceptionB {}

public class Demo {

   public static void main( String args[] )
   {
      try {
         throw new ExceptionC();
      }
      catch( ExceptionA a ) {
         JOptionPane.showMessageDialog(
           null, a.toString(), "Exception",
           JOptionPane.INFORMATION_MESSAGE );
      }

      try {
         throw new ExceptionB();
      }
      catch( ExceptionA b ) {
           JOptionPane.showMessageDialog(
             null, b.toString(), "Exception",
             JOptionPane.INFORMATION_MESSAGE );
      }
      System.exit( 0 );
   }
}
```

Percobaan 5 : Membuat exception sendiri

```
class Salah extends Exception{
      public Salah(){}
      public Salah(String pesan){
            super(pesan);
      }
}
public class TesSalah{
      public static void main(String [] arg) throws Salah{
            Salah s = new Salah("Salah disengaja ha..ha..");
            int i = 0;
            if (i==0)
                 throw s;
      }
}
```

Percobaan 6: Membuat exception sendiri

```
class counterException extends Exception {   // Define
  String complaint;
  public counterException(String c)  {
    this.complaint = c;
  }
  public String toString( ) {
    return "counter Exception " + complaint;
  }
 }
class counter {
```

```
  int n = 0;
  public int zero() { return n=0; }
  public int up() { return ++n; }
  public int down() throws counterException { // Throw
    if (n <= 0)
       throw new counterException
         (n + " count Down failed.");
    return --n;
  }
}
public class Example1 {
 public static void main( String args[] ) {
  counter aCounter = new counter( );
  aCounter.zero( );
  aCounter.up();
  try {  aCounter.down( ); }
  catch (counterException ce) {  // Catch
        System.out.println("" + ce);
  }
  try {  aCounter.down( ); }
  catch (counterException ce) {  // Catch
        System.out.println("" + ce);
  }
  finally  {
       System.out.println(“Finally”);
  }
 }
}
```

```
public class Example2 {
 public static void main( String args[] ) throws Exception {
  counter aCounter = new counter( );
  aCounter.zero( );
  aCounter.up();
  aCounter.down( );
  aCounter.down( );
  System.out.println(“Completed”);
 }
}
```

E. LATIHAN

1. Terdapat dua cara untuk menangani Exception yaitu dengan menangkap Exception dan melempar Exception. Lakukan penanganan exception dengan melempar Exception menggunakan throw. Berilah penjelasan (apakah program termasuk unchecked exceptions atau checked exceptions) !

```
public class ReadFile {
     public static void main(String args[]){
```

```
        File file = new File("Data.txt");
        BufferedReader fileReader ;

        fileReader = new BufferedReader(new FileReader(file));
        while(true){
           String line = fileReader.readLine();
           if (line == null)
              break ;
           System.out.println(line);
        }
     }
}
```

F. TUGAS

1. Buatlah sebuah class Stack, FullStackException dan EmptyStackException. Class Stack ini menggambar Stack yang menerapkan konsep LIFO (Last In First Out). Konsep LIFO ini, data yang terakhir masuk akan keluar pertama kali.

   Class Stack mempunyai atribut:

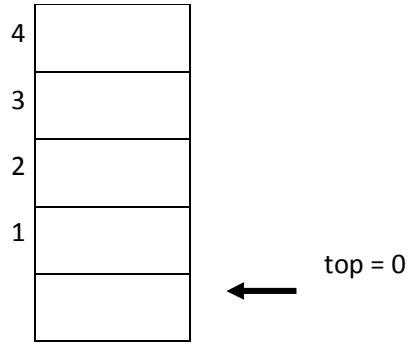
   - size : menentukan besar array untuk menyimpan data. Array berdimensi satu dengan tipe Object.
   - top : merupakan tanda indeks yang paling atas, yang belum terisi. Sehingga data yang akan masuk akan dimasukkan pada indeks tersebut.
   - Object[] elemen : untuk menyimpan data.

   Class Stack mempunyai operasi:

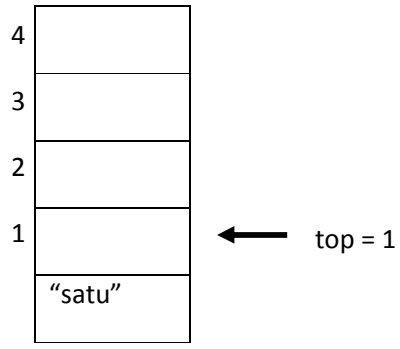
   - public Stack() : jika kita membuat object Stack dengan konstruktor Stack() maka tentukan size = 5.
   - public Stack(int s) : jika kita membuat object Stack dengan konstruktor Stack(int s) maka tentukan size berdasarkan parameter s.
   - public int getSize() : untuk mendapatkan besar array dari Stack.
   - public int getTop() : untuk mendapatkan top dari Stack
   - public void push(Object o) : untuk memasukkan data ke array pada Stack.
   - public Object pop() : untuk mengambil data dari array.

Deskripsi Stack

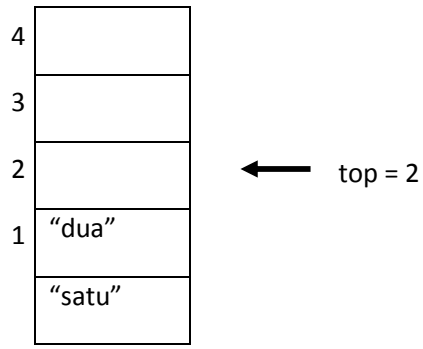
Buat objek Stack dengan nama stack, pertama kali top bernilai 0.



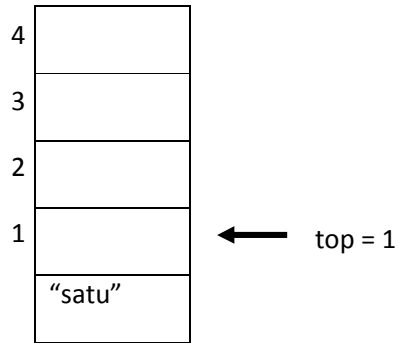
Masukkan sebuah data String “satu”, data “satu” akan dimasukkan pada indeks yang ke-0 (sesuai dengan nilai top) selanjutnya top dinaikkan 1, sehingga top = 1.



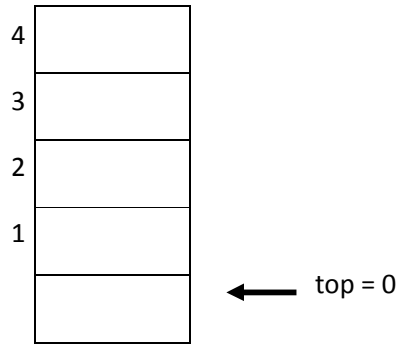
Masukkan sebuah data String “dua”, data “dua” akan dimasukkan pada indeks yang ke-1 (sesuai dengan nilai top) selanjutnya top dinaikkan 1, sehingga top = 2.



Ambil sebuah data dari Stack, data yang diambil adalah data yang dimasukkan terakhir kali (“dua”). Kurangi nilai top dengan 1 sehingga menjadi nilai top = 1. Selanjutnya ambil data pada indeks ke -1.

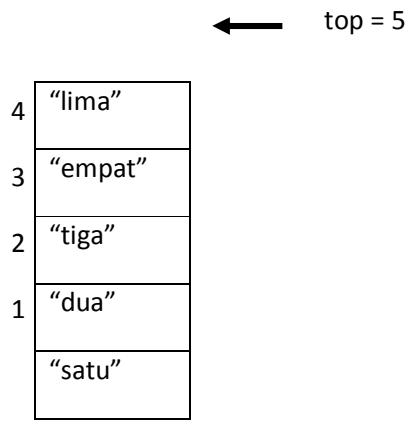


Ambil sebuah data dari Stack, data yang diambil adalah data yang terakhir (“satu”). Kurangi nilai top dengan 1 sehingga menjadi nilai top = 0. Selanjutnya **apabila** ada pengambilan data maka yang diambil adalah data pada indeks ke -0.

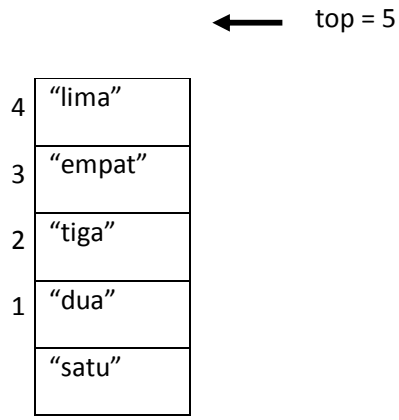


Ambil sebuah data lagi dari Stack, jika kondisi top = 0, maka Stack kosong dan melempar exception EmptyStackException (class ini merupakan subclass dari class RuntimeException), dan muncul peringatan bahwa “Stack kosong”.

Selanjutnya isilah Stack sampai penuh.



Jika kita ingin memasukkan data baru lagi, jika kondisi top = 5, maka akan melempar exception yaitu FullStackException (class ini merupakan subclass dari class RuntimeException).



Masukkan data lagi → Stack Full

A. LAPORAN RESMI

Kerjakan hasil percobaan(D), latihan(E) dan tugas(F) di atas dan tambahkan analisa.