PERCOBAAN 6
EXCEPTION

Pokok Bahasan
- Penanganan Eksepsi
- Menangkap Eksepsi
- Catch Secara bertingkat
- Melontarkan Eksepsi
- Melontarkan kembali Eksepsi
- Klausa Throws

Tujuan Belajar
Dengan praktikum ini mahasiswa diharapkan dapat:
- Dapat memahami cara penanganan Eksepsi
- Memahamkan cara Menangkap Eksepsi
- Dapat menjelaskan konsep catch secara bertingkat
- Memahamkan cara melontarkan eksepsi
- Memahamkan cara melontarkan kembali eksepsi
- Dapat menjelaskan tentang klausa throws

Dasar Teori
A. Pemahaman Dasar
Eksepsi (exeption) adalah suatu mekanisme yang digunakan oleh beberapa bahasa pemrograman untuk mendeskripsikan apa yang harus dilakukan jika ada suatu kondisi yang tidak diinginkan terjadi. Eksepsi dapat dijumpai saat:
- Mengakses method dengan argumen yang tidak sesuai
- Membuka file yang tidak ada
- Koneksi jaringan yang terganggu
- Manipulasi operan yang nilainya keluar dari batasan yang didefinisikan
- Pemanggilan class yang tidak ada

Java menyediakan dua kategori besar untuk eksepsi yang disebut sebagai ***checked exception*** dan ***unchecked exception.***
- *Checked Exception* adalah eksepsi yang diantisipasi oleh programmer untuk dihandle dalam program dan terjadi dikarenakan oleh kondisi luar yang siap muncul saat program berjalan. Misalnya membuka file yang tidak ada atau gangguan jaringan.
- *Unchecked Exception* bisa muncul dari kondisi yang merepresentasikan adanya *bug* atau situasi yang secara umum dianggap terlalu sulit bagi program untuk menghandlenya. Disebut sebagai *unchecked* karena kita tidak perlu mengeceknya atau melakukan sesuatu jika kondisi ini terjadi. Eksepsi yang muncul dari kategori situasi yang merepresentasikan bug ini disebut sebagai *runtime exception*. Misalnya mengakses array melebihi size yang dimilikinya.
- Sedangkan eksepsi yang muncul sebagai akibat dari isu environment software –yang ini jarang sekali atau sulit sekali untuk dihandle- disebut sebagai ***error,*** misalnya *running out memory*.

Jadi, class *Exception* mendefinisikan kondisi error yang ringan yang dijumpai oleh program.

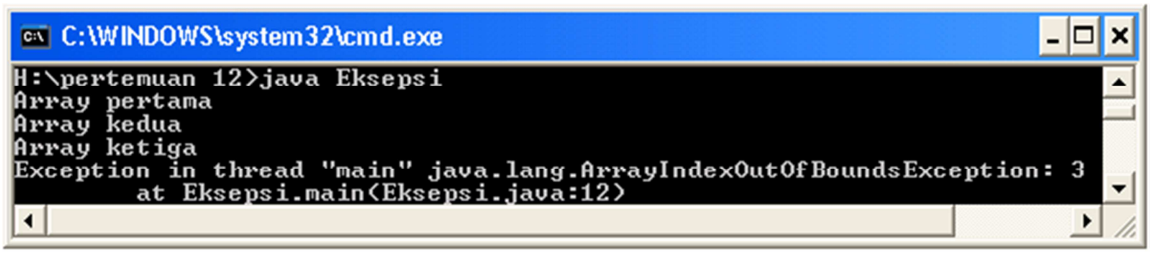
Sedangkan untuk kondisi error yang berat didefinisikan dengan *Error*.

- *Class Exception* adalah sebuah class dasar yang merepresentasikan checked exception. Dalam hal ini, bukannya membiarkan terjadinya penghentian program, sebaliknya Anda harus menuliskan beberapa kode untuk menghandle eksepsi dan berikutnya melanjutkan program.
- *Class Error* adalah class dasar yang digunakan untuk kondisi error serius yang tidak terdeteksi. Dalam banyak kasus, Anda harus membiarkan program diterminasi.
- *Class RuntimeException* adalah class dasar yang digunakan untuk unchecked exception yang bisa muncul sebagai akibat dari bug program. Pada banyak kasus, Anda harus membiarkan program dihentikan.

Ketika dalam program terjadi eksepsi, method yang menemukan error tersebut bisa menghandle sendiri atau melemparkannya (*"throw"*) kembali kepada pemanggilnya untuk memberi sinyal bahwa telah terjadi suatu masalah. Sebagai contoh, Coba jalankan program ini. Apa hasilnya?

```
public class Eksepsi {
   public static void main (String[] args) {
     int i = 0;
     String myarray[] = {"Array pertama", "Array kedua", "Array
ketiga" };

     while (i < 4) {
        System.out.println (myarray[i]);
        i++;
      }
    }
 }
```

Program akan diterminasi dengan sebuah pesan kesalahan seperti contoh hasil runtime program di atas.



Upaya menghandle eksepsi memungkinkan program untuk menangkap eksepsi, menghandlenya kemudian melanjutkan eksekusi program. Dalam hal ini digunakan statemen *try* dan *catch*. Dengan perincian sbb :

- Blok *try* → adalah kode yang memungkinkan terjadinya pelemparan (throw) eksepsi tertentu
- Blok *catch* → adalah kode yang akan dieksekusi jika sebuah tipe eksepsi dilemparkan

```
try {
      //code that might throw a particular exception
} catch(MyExceptionType myExcept) {
      //code to execute if MyExceptionType exception is thrown
} catch(Exception otherExcept) {
      //code to execute if a general Exception exception is thrown
}
```

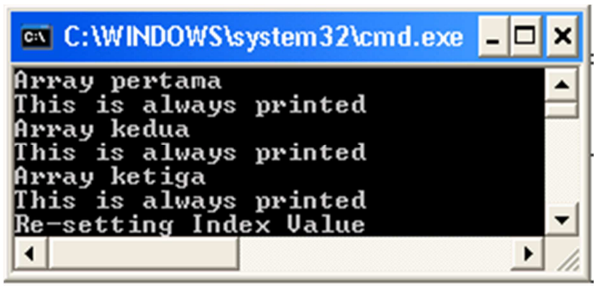
Dimungkinkan ada beberapa blok catch setelah sebuah blok try, yang masing-masing akan menghandle tipe eksepsi yang berbeda

Statemen *finally*

Statemen finally digunakan untuk mendefinisikan sebuah blok kode yang akan senantiasa dieksekusi jika eksepsi berhasil ditangkap dalam blok catch. Perhatikan contoh di bawah ini

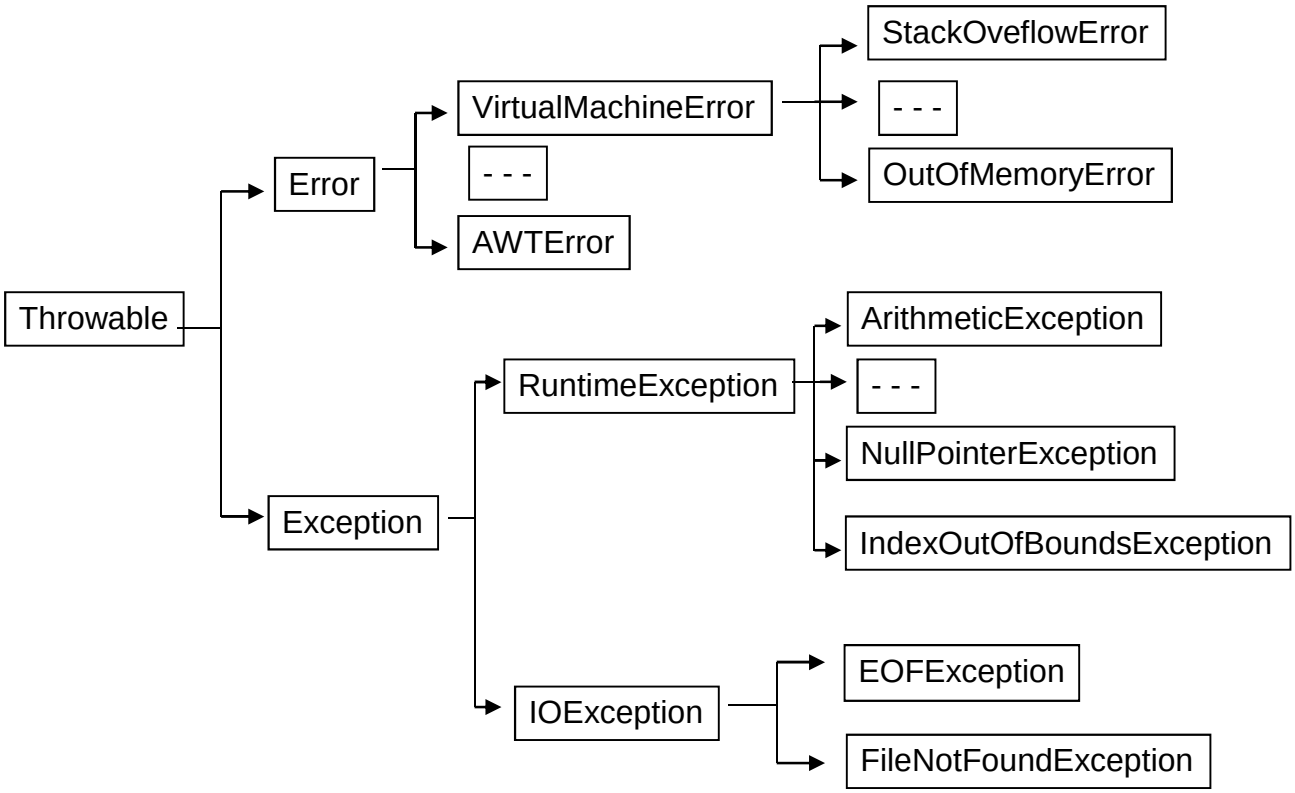
```
public class Eksepsi2 {
   public static void main(String[] args) {
     int i = 0;
     String myarray[] = {"Array pertama","Array kedua","Array ketiga"};

     while (i < 4) {
        try {
              System.out.println(myarray[i]);
              i++;
        }  catch (ArrayIndexOutOfBoundsException e){
              System.out.println("Re-setting Index Value");
              i = 0;
        } finally {
              System.out.println("This is always printed");
         }
    }
  }
}
```

Pengesetan indeks kembali menjadi 0 mengakibatkan program dieksekusi dalam infinite loop.



Kategori eksepsi

Di bawah ini adalah hirarki class yang merepresentasikan pengkategorian eksepsi. Dalam hal ini class java.lang.Throwable merupakan class parent dari seluruh objek yang bisa dilempar dan ditangkap menggunakan mekanisme exception handling.



Sehingga dalam contoh di atas terdapat statemen :

```
catch (ArrayIndexOutOfBoundsException e)
```

merupakan jenis RuntimeException. Jika Anda tidak hafal jenis eksepsinya secara pasti, bisa mengambil parent classnya.

Eksepsi yang umum

Java menyediakan beberapa eksepsi yang telah didefinisikan. Beberapa eksepsi yang umum, di antaranya adalah :

- **ArithmeticException** → hasil dari operasi *divide-by-zero* terhadap integer
  Contoh : `int i = 12 / 0;`
- **NullPointerException** → mengakses membernya (atribut atau method) ketika reference-nya masih menunjuk ke null, misalnya ketika belum dicreate instance objectnya
  Contoh :
  ```
  Date d = null;    //tanpa membuat instance object
  System.out.println(d.toString());
  ```
- **NegativeArraySizeException** → membuat array dengan size yang diset negatif
- **ArrayIndexOutOfBoundsException** → mengakses array melebihi indeks terbesarnya
- **SecurityException** → biasanya terjadi pada sebuah browser, ketika class SecurityManager melempar eksepsi kepada applet yang melakukan operasi yang membahayakan host atau file-filenya (tidak berhak mengaksesnya), misalnya mengakses file system lokal, membuka soket ke sebuah host yang berbeda dengan host yang melayani applet, dll

Overriding method & eksepsi

Ketika meng-override method yang melempar eksepsi, maka method yang meng-override hanya bisa mendeklarasikan eksepsi yang sama classnya atau subclass dari eksepsi yang dilemparkan oleh metode yang di-override. Perhatikan contoh berikut ini :

```
public class TestA {
     public void methodA() throws RuntimeException {
          …
     }
}
```

```
public class TestB1 extends TestA {
     public void methodA() throws ArithmeticException {
          …
     }
}
```

Bisa karena *ArithmeticException* adalah subclass dari *RuntimeException*

```
public class TestB2 extends TestA {
     public void methodA() throws Exception {
          …
     }
}
```

Gagal karena *Exception* adalah class induk dari *RuntimeException*

Tugas Pendahuluan

1. Apa perbedaan antara error dan eksepsi!
2. Apakah try dapat mengandung except dan finally sekaligus? Beri contoh programnya!