PENANGANAN EKSEPSI

Eksepsi adalah keadaan tidak normal yang muncul pada suatu bagian program pada saat dijalankan. Penanganan eksepsi pada java membawa pengelolaan kesalahan program saat dijalankan kedalam orientasi-objek. Eksepsi java adalah objek yang menjelaskan suatu keadaan eksepsi yang muncul pada suatu bagian program.

Saat suatu keadaan eksepsi muncul, suatu objek exception dibuat dan dimasukkan ke dalam method yang menyebabkan eksepsi. Method tersebut dapat dipilih untuk menangani eksepsi berdasarkan tipe tertentu. Method ini juga menjaga agar tidak keluar terlalu dini melalui suatu eksepsi, dan memiliki suatu blok program yang dijalankan tepat sebelum suatu eksepsi menyebabkan metodenya kembali ke pemanggil.

Eksepsi dapat muncul tidak beraturan dalam suatu method, atau dapat juga dibuat secara manual dan nantinya melaporkan sejumlah keadaan kesalahan ke method yang memanggil.

Dasar-dasar penanganan Eksepsi

Penanganan eksepsi pada java diatur dengan lima kata kunci :

- try,
- catch,
- throw,
- throws dan
- finally.

Pada dasarnya try digunakan untuk mengeksekusi suatu bagian program, dan jika muncul kesalahan, sistem akan melakukan throw suatu eksepsi yang dapat anda catch berdasarkan tipe eksepsinya, atau yang anda berikan finally dengan penangan default.

Berikut ini bentuk dasar bagian penanganan eksepsi :

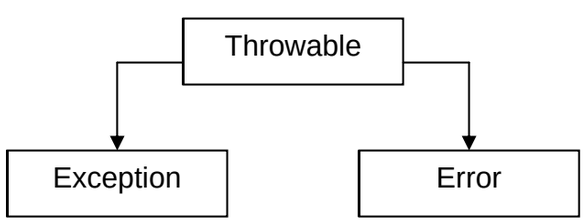
```
try {
// Block of Code
}
catch (ExceptionType1 e) {
// Exception Handler for ExceptionType1
}
catch (ExceptionType2 e) {
// Exception Handler for ExceptionType2
throw (e); // re-throw the Exception…
}
finally { }
```

Tipe Eksepsi

Dipuncak hirarki class eksepsi terdapat satu class yang disebut throwable. Class ini digunakan untuk merepresentasikan semua keadaan eksepsi. Setiap ExceptionType pada bentuk umum diatas adalah subclass dari throwable.

Dua subclass langsung throwable didefinisikan untuk membagi class throwable menjadi dua cabang yang berbeda. Satu, class Exception, digunakan untuk keadaan eksepsi yang harus ditangkap oleh program yang kita buat. Sedangkan yang lain diharapkan dapat menangkap class yang kita subclasskan untuk menghasilkan keadaan eksepsi.

Cabang kedua throwable adalah class error, yang mendefinisikan keadaan yang tidak diharapkan untuk ditangkap dalam lingkungan normal.



Eksepsi Yang Tidak Dapat Ditangkap
Obyek eksepsi secara otomatis dihasilkan oleh runtime java untuk menanggapi suatu keadaan eksepsi. Perhatikan contoh berikut :

```
class Exc0 {
public static void main (String args[]) {
int d = 0;
int a = 42 / d;
}
}
```

Saat runtime java mencoba meng-eksekusi pembagian, akan terlihat bahwa pembaginya adalah nol, dan akan membentuk objek eksepsi baru yang menyebabkan program terhenti dan harus berurusan dengan keadaan kesalahan tersebut. Kita belum mengkodekan suatu penangan eksepsi, sehingga penanganan eksepsi default akan segera dijalankan.

Keluaran dari program diatas :
java.lang.ArithmeticExpression : /by zero
at Exc0.main (Exc0.java:4)

Berikut adalah contoh lainnya dari eksepsi :

```
class Exc1 {
static void subroutine() {
int d = 0;
int a = 42 / d;
}
public static void main (String args[]) {
Exc1.subroutine();
}
}
```

Output-nya :
Exception in thread “ main” java.lang.ArithmeticException : / by zero
at Exc1.subroutine(Exc1.java :4)
at Exc1.main(Exc1.java : 7)

Try dan Catch
Kata kunci try digunakan untuk menentukan suatu blok program yang harus dijaga terhadap semua eksepsi, setelah blok try masukkan bagian catch, yang menentukan tipe eksepsi yang akan ditangkap. Perhatikan contoh berikut:

```
class Exc2 {
public static void main (String args[]) {
try {
int d = 0;
int a = 42 / d;
}
catch (ArithmeticException e) {
System.out.println(“Division By Zero”);
}
}
}
```

Outputnya:
C:\Documents and Settings\noviyanto\My Documents>java Exc2
Division By Zero

Throw
Pernyataan throw digunakan untuk secara eksplisit melemparkan suatu eksepsi. Pertama kita harus mendapatkan penanganan dalam suatu instance throwable, melalui suatu parameter kedalam bagian catch, atau dengan membuatnya menggunakan operator new. Bentuk umum pernyataan throw :
throw ThrowableInstance;

Aliran eksekusi akan segera terhenti setelah pernyataan throw, dan pernyataan selanjutnya tidak akan dicapai. Blok try terdekat akan diperiksa untuk melihat jika telah memiliki bagian catch yang cocok dengan tipe instance Throwable. Jika tidak ditemukan yang cocok, maka pengaturan dipindahkan ke pernyataan tersebut. Jika tidak, maka blok pernyataan try selanjutnya diperiksa, begitu seterusnya sampai penanganan eksepsi terluar menghentikan program dan mencetak penelusuran semua tumpukan sampai pernyataan throw. Contoh :

```
class throwDemo {
static void demoProc() {
try {
throw new NullPointerException(“demo”); }
catch (NullPointerException e) {
System.out.println(“caught inside demoproc”);
throw e; }
}
public static void main (String args[]) {
try { demoProc(); }
catch (NullPointerException e) {
System.out.println(“recaugt : “ + e); }
} }
```

Output :
caught inside demoproc
recaught : java.lang.NullPointerException : demo

Throws
Kata kunci throws digunakan untuk mengenali daftar eksepsi yang mungkin di-throw oleh suatu method. Jika tipe eksepsinya adalah error, atau RuntimeException, atau suatu subclassnya, aturan ini tidak berlaku, karena tidak diharapkan sebagai bagian normal dari kerja program.

Jika suatu method secara eksplisit men-throws suatu intans dari Exception atau subclassnya, diluar RuntimeException, kita harus mendeklarasikan tipenya dengan pernyataan throws. ini mendefinisikan ulang deklarasi method sebelumnya dengan sintaks sbb :
type method-name (arg-list) throws exception-list { }

Contoh :

```
class ThrowsDemo {
static void procedure () throws IllegalAccessException {
System.out.println(“Inside Procedure”);
throw new IllegalAccessException(“demo”);
}
public static void main(String args[]) {
try { procedure(); }
catch (IllegalAccessException e) {
System.out.println(“caught “+ e); }
}
}
```

Output :
Inside procedure

caught java.lang.IllegalAccessException : demo

Finally

Saat suatu eksepsi dilemparkan, alur program dalam suatu method membuat jalur yang cenderung tidak linier melalui method tersebut, melompati baris-baris tertentu, bahkan mungkin akan keluar sebelum waktunya pada kasus dimana tidak ada bagian catch yang cocok. Kadang-kadang perlu dipastikan bahwa bagian program yang diberikan akan berjalan, tidak perduli eksepsi apa yang terjadi dan ditangkap. Kata kunci finally dapat digunakan untuk menentukan bagian program seperti itu.

Setiap try membutuhkan sekurang-kurangnya satu bagian catch atau finally yang cocok. Jika kita tidak mendapatkan bagian catch yang cocok, maka bagian finally akan dieksekusi sebelum akhir program, atau setiap kali suatu method akan kembali ke pemanggilnya, melalui eksepsi yang tidak dapat ditangkap, atau melalui pernyataan return, bagian finally akan dieksekusi sebelum kembali ke method kembali.

Berikut adalah contoh program yang menunjukkan beberapa method yang keluar dengan berbagai cara, tidak satupun tanpa mengeksekusi bagian finally-nyA.

```
class finallyDemo {
static void procA() {
try { System.out.println(“Inside procA..”);
throw new RuntimeException(“Demo”); }
finally { System.out.println(“procA is finally”); }
}
static void procB() {
try { System.out.println(“Inside procB..”);
return; }
finally { System.out.println(“procB is finally”); }
}
public static void main(String args[]) {
try { procA( ); }
catch (Exception e){ };
procB(); }
}
```

Output :

```
Inside procA..
procA is finally
Inside procB..
procB is finally
```