Penanganan Eksepsi

Definisi Eksepsi (exception)

- Exception adalah kondisi yang menyebabkan program menjadi hang atau quit dari alur normal yang telah ditentukan pada saat program dijalankan. Exception ini dipicu oleh runtime eror yakni eror yang terjadi pada saat program dieksekusi oleh interpreter. Pada umumnya eror dalam pemrograman dapat dibagi menjadi 3 kategori antara lain :
  - Eror Kompilasi

    Eror ini terjadi pada saat program sedang dicompile yakni bila java compiler menemukan syntax atau aturan-aturan lainnya pada program.

    Biasanya eror ini relatif lebih mudah untuk ditelusuri (trace) karena compiler akan memberitahu pada baris program yang mana terdapat eror.

    EX: lupa membubuhkan notasi titik koma (;) pada akhir statement.
  - Eror Runtime

    Kesalahan yang disebabkan oleh faktor eksternal lain seperti program gagal menemukan dan membuka file, kesalahan operasi (pembagian bilangan dengan nol), data yang di input tidak sesuai dengan tipe data yang diminta, dll

Kategori Error dalam Pemrograman

1. **Eror Kompilasi**

   Eror ini terjadi pada saat program sedang dicompile yakni bila java compiler menemukan syntax atau aturan-aturan lainnya pada program.

   - Error ini relatif lebih mudah untuk ditelusuri (trace) karena compiler akan memberitahu pada baris program yang mana terdapat eror.
   - lupa membubuhkan notasi titik koma (;) pada akhir statement.

Kategori Error dalam Pemrograman

2. **Eror Runtime**

   Kesalahan yang disebabkan oleh faktor eksternal lain seperti program gagal menemukan dan membuka file, kesalahan operasi (pembagian bilangan dengan nol), data yang di input tidak sesuai dengan tipe data yang diminta, dll

3. **Error Logic**

   Kesalahan dalam logika program.

Eksepsi yang tidak dapat ditangkap

- Eksepsi yang tidak dapat ditangkap

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

Keluaran dari program diatas :

Exception in thread “main” java.lang.ArithmeticException:/by zero at main.main(main.java:4)

java result : 1

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.

- Penanganan eksepsi pada java diatur dengan empat kata kunci, yaitu :
  - Try
  - Catch
  - Throw
  - Finally
- Eksepsi yang tidak dapat ditangkap

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

Keluaran dari program diatas :

Exception in thread “main” java.lang.ArithmeticException:/by zero at main.main(main.java:4)

java result : 1

Penanganan eksepsi pada Java

- Try
- Catch
- Throw
- Finally

Try dan Catch

- Kata kunci **Try** digunakan untuk menentukan suatu blok program yang harus dijaga terhadap semua eksepsi.
- Kata kunci **Catch** digunakan untuk menentukkan tipe eksepsi yang akan ditangkap.
- **Contoh :**

```
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:

Division By Zero

Finally

- Analogikan proses exception Try-Catch dengan mekanisme penyeleksian kondisi switch-case. Switch-case akan memeriksa nilai suatu variable dan bila ditemukkan suatu kondisi yang telah ditetapkan maka variable itu akan diproses.
- Dalam switch-case terdapat keyword default yang digunakan sebagai blok default bila tidak ditemukkan kondisi yang telah ditetapkan. Try-catch memiliki keyword Finally yang hampir sama fungsinya dengan default pada switch-case.
- Ada 3 skenario pemrosesan dengan keyword finally yaitu :
  - Bila tidak terjadi exception, blok finally akan dieksekusi. Setelah selesai, interpreter akan mengeksekusi staement selanjutnya.
  - Bila terjadi exception, interpreter akan berhenti mengeksekusi statement dalam blok try berikutnya, kemudian interpreter akan mencari catch yang bersesuaian, bila ditemukan interpreter akan mengeksekusi catch & finally.
  - Bila exception terjadi namun tidak ada catch yang bersesuaian maka statement-statement try berikutnya yang masih tersisa tidak akan dieksekusi, selanjutnya interpreter akan mengeksekusi blok finally.

- Contoh :

```
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
```

Throw

- Digunakan untuk melemparkan suatu eksepsi.

  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
```

TERIMA KASIH