

Exception Handling

Topik

- Kategori Exception
- try, catch, finally
- Method yang melempar exception
- Aturan overriding method dan exception
- Membuat class exception baru

Definisi Exception

- Suatu mekanisme penanganan kesalahan.
- Event yang terjadi ketika program menemui kesalahan saat instruksi program dijalankan.

Exception

- Exception sering digunakan dalam akses sumberdaya non memori.

Catatan:

- Exception = untuk menangani kesalahan ringan (mild error).
- Error = mengindikasikan bahwa error yang terjadi adalah fatal error (severe problem) dimana proses recovery sangat sulit dilakukan bahkan tidak mungkin dilakukan (Contoh : program running out of memory)



Contoh kesalahan yang terjadi:

- Pembagian bilangan dengan 0
- Pengisian elemen array diluar ukuran array
- Kegagalan koneksi database
- File yang akan dibuka tidak exist
- Operand yg akan dimanipulasi out of prescribed range
- Mengakses obyek yang belum diinisialisasi

Common Exception

- ❑ ArithmeticException
 - Hasil dari operasi divide-by-zero pada integer
 - Misal: `int i = 12/0;`
- ❑ NullPointerException
 - Mencoba mengakses atribut atau method suatu object padahal object belum dibuat.
 - Misal: `Date d = null;`
`System.out.println(d.toString());`
- ❑ NegativeArraySizeException
 - Mencoba membuat array dengan ukuran negatif.
- ❑ ArrayIndexOutOfBoundsException
 - Mencoba mengakses elemen array dimana index nya melebihi ukuran array.
- ❑ SecurityException
 - Biasanya dilempar ke browser, class security manager melempar exception untuk applet yang mencoba melakukan:
 - Mengakses lokal file
 - Open socket ke host yg berbeda dgn host yg di open oleh applet

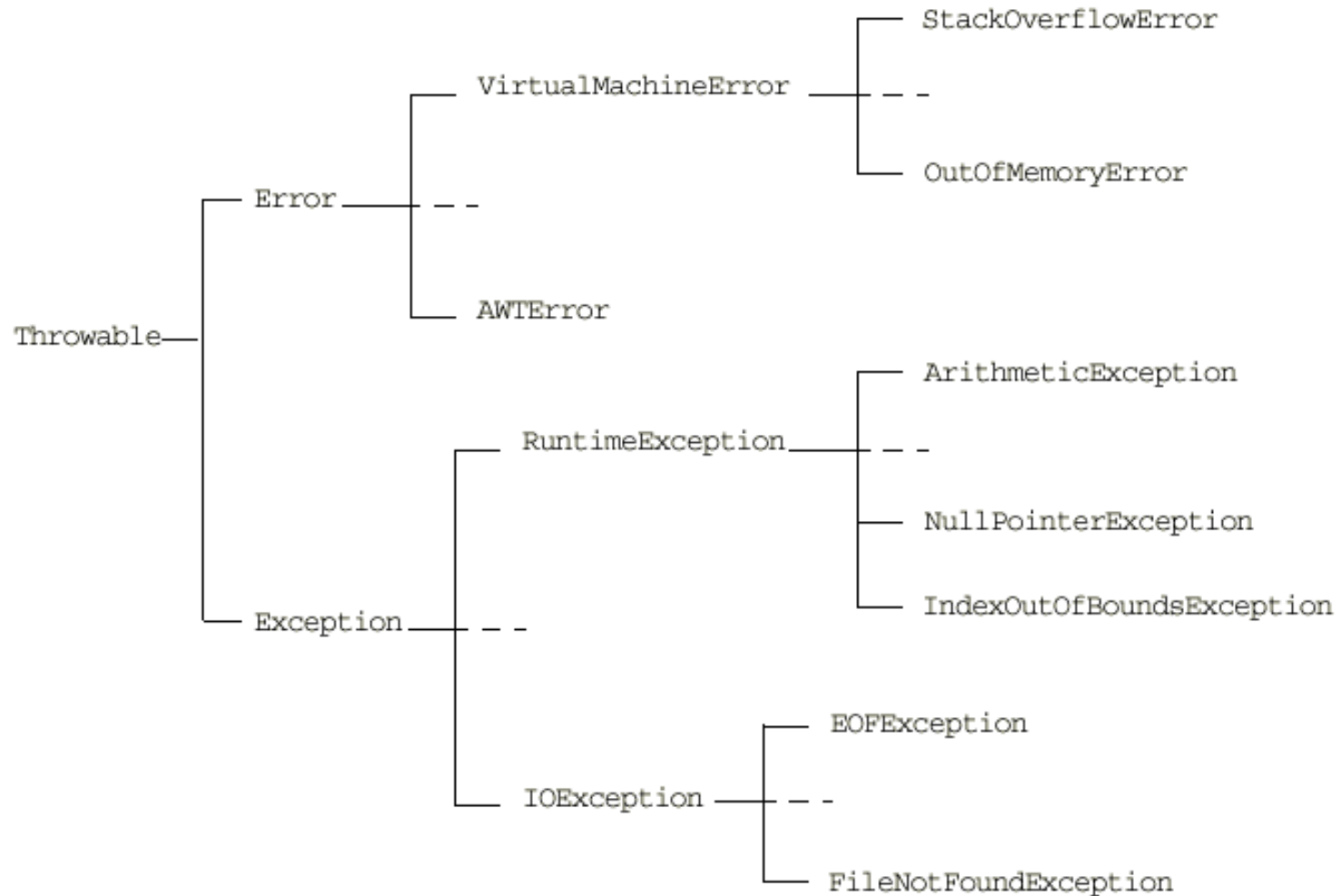
Contoh Exception

```
Class DivByZero {  
    public static void main(String args[]) {  
        System.out.println(3/0);  
        System.out.println("Pls. print me.");  
    }  
}
```

- Menampilkan pesan error

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at DivByZero.main(DivByZero.java:3)
```

Kategori Exception





Apa yang terjadi jika terjadi kesalahan?

- Secara otomatis akan dilempar sebuah object yang disebut dgn **exception**.
- Exception dapat diproses lebih lanjut oleh fungsi-fungsi yang siap menangani kesalahan.
- Proses pelemparan exception disebut dgn **throwing exception**.
- Proses penerimaan exception disebut dengan **catch exception**.

Contoh kejadian error – Cara lama (loading file from the disk)

```
int status = loadTexfile();  
If (status != 1) {  
    // something unusual happened, describe it  
    switch (status) {  
        case 2:  
            // file not found  
            break;  
        case 3:  
            //disk error  
            break;  
        case 4:  
            //file corrupted  
            break;  
        default:  
            // other error  
    }  
} else {  
    // file loaded OK, continue with program  
}
```

Contoh program

Fungsi bacaFile

BukaFile

BacaBarisFileSampaiHabis

TutupFile

Ditambahkan program untuk pengecekan berhasil
tidaknya pembacaan file

Fungsi bacaFile

BukaFile

Jika Gagal Buka File

Lakukan Sesuatu

Jika Berhasil Buka File

BacaBarisFileSampaiHabis

TutupFile

- Bagaimana bila ditambahkan program untuk pengecekan terhadap status pembacaan file?
- Bagaimana bila ditambahkan program untuk pengecekan terhadap status penutupan file?
- Maka program akan menjadi sangat panjang dan banyak terdapat nested if-else.

Solusi?

Gunakan exception

Bentuk:

```
try {  
    .....  
} catch (ExceptionType x) {  
    .....  
}
```

- Blok try : digunakan untuk menempatkan kode-kode program java yang mengandung kode program yang mungkin melemparkan exception.
- Blok catch : digunakan untuk menempatkan kode-kode program java yang digunakan untuk menangani sebuah exception tertentu.

Implementasi 1

```
try {  
    Fungsi bacaFile  
        BukaFile  
        BacaBarisFileSampaiHabis  
        TutupFile  
} catch (KesalahanBukaFile) {  
    // lakukan sesuatu  
}
```


Try dgn banyak catch

- ❑ Dapat digunakan beberapa blok catch untuk satu blok try.
- ❑ Exception dalam satu program bisa mengatasi banyak exception.
- ❑ Contoh implementasi:
- ❑ Misal dalam satu blok try terdapat kemungkinan terjadi:
 - ❑ NullPointerException
 - ❑ IndexOutOfBoundsException
 - ❑ ArithmeticException

```
try {  
    .....  
} catch (ExceptionType1 x1) {  
    .....  
} catch (ExceptionType2 x2) {  
    .....  
}
```

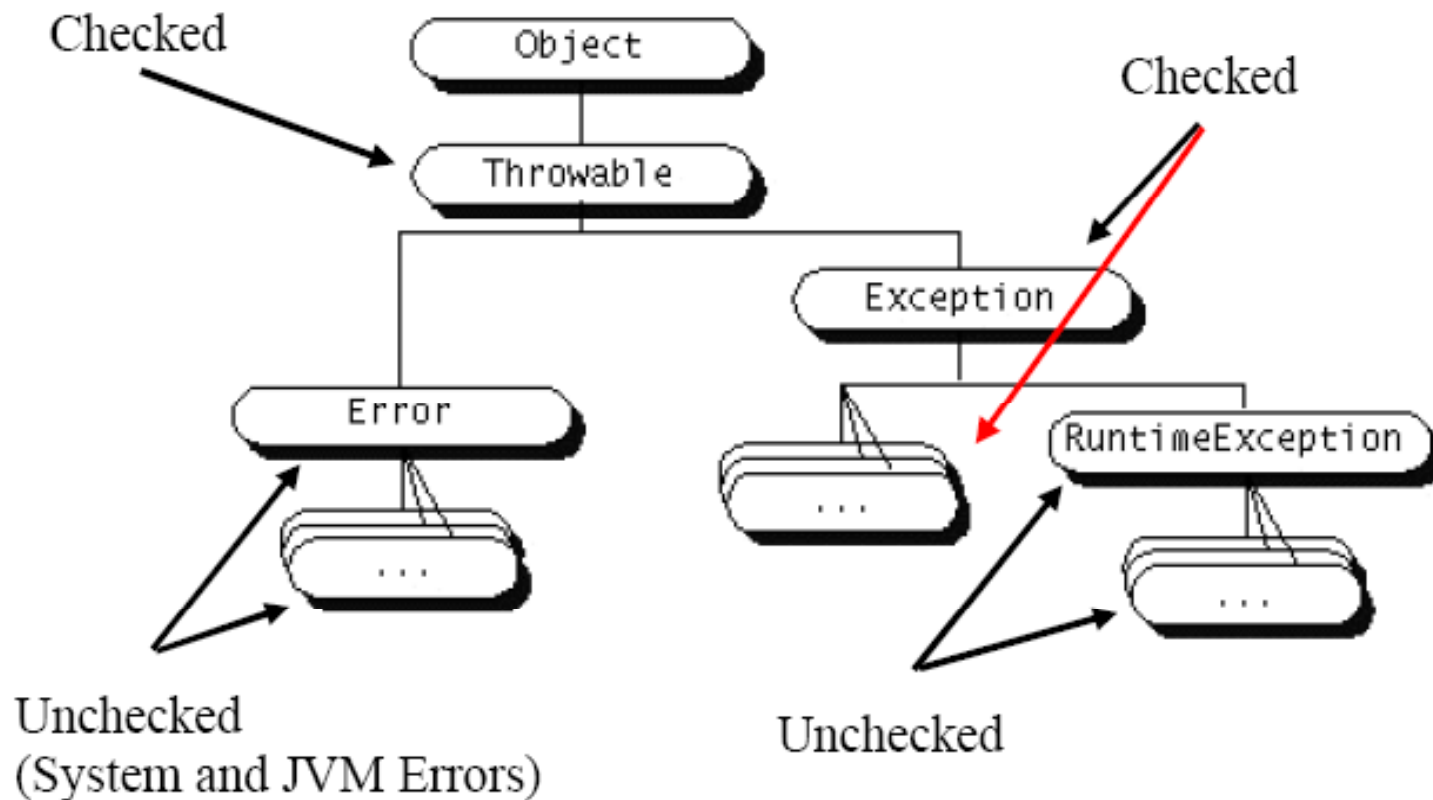
Implementasi 2

```
try {  
    Fungsi bacaFile  
    BukaFile  
    BacaBarisFileSampaiHabis  
    TutupFile  
} catch (KesalahanBukaFile) {  
    // lakukan sesuatu  
} catch (KesalahanAlokasiMemori) {  
    // lakukan sesuatu  
} catch (KesalahanTutupFile) {  
    // lakukan sesuatu  
}
```

Checked/Unchecked Exceptions

- Exception bisa *checked* atau *unchecked*
 - Checked = dicek oleh the compiler
- *Checked exception* hanya dapat ditangani dalam try block atau method yang didesain untuk melempar exception.
 - Compiler akan memberitahu jika checked exception tidak ditangani secara tepat.
 - Contoh : IOException.
- *Unchecked exception* tidak memerlukan penanganan langsung . Pada saat compile **tidak ada pemberitahuan kesalahan**.
 - Contoh : RuntimeException dan turunannya

Checked/Unchecked Exceptions



Melempar Exception - Throw

- Java memperbolehkan untuk melempar exception (generate exception)

throw <exception object>;

- Exception yang dilempar adalah sebuah object exception (Object exception yang sudah disediakan oleh java atau yang di create sendiri)
- Contoh:

throw new ArithmeticException(“testing...”);



Contoh Melempar Exception - Throw

```
1 class ThrowDemo {
2     public static void main(String args[]){
3         String input = "invalid input";
4         try {
5             if (input.equals("invalid input")) {
6                 throw new RuntimeException("throw demo");
7             } else {
8                 System.out.println(input);
9             }
10            System.out.println("After throwing");
11        } catch (RuntimeException e) {
12            System.out.println("Exception caught:" + e);
13        }
14    }
15 }
```

Mendefinisikan method yang menghasilkan exception

- Dilakukan bila method tidak ingin menangani exception sendiri.
- Method tertentu dlm program mungkin akan menghasilkan error yang tidak dikenali secara otomatis oleh Java Virtual Machine.
- Berlaku bagi kategori exception yg bukan subclass dari RuntimeException.
- Contoh: EOFException, MalformedURLException
- Dengan cara membuat method yang dapat melempar exception.
- Sintaks

```
<type> <methodName> (<parameterList>)  
throws <exceptionList> {  
    <methodBody>  
}
```

Contoh method yang mendefinisikan exception

```
Class methodKu{  
    public Image loadImage(String s)  
    throws EOFException, MalformedURLException {  
        If(kondisierroryangterjadi)  
            throw new EOFException()  
    }  
}
```




Contoh method yang mendefinisikan exception

```
1 class ThrowingClass {
2     static void meth() throws ClassNotFoundException {
3         throw new ClassNotFoundException ("demo");
4     }
5 }
6 class ThrowsDemo {
7     public static void main(String args[]) {
8         try {
9             ThrowingClass.meth();
10        } catch (ClassNotFoundException e) {
11            System.out.println(e);
12        }
13    }
14 }
```



Contoh method yang mendefinisikan exception

- Beberapa method java menggunakan statement throw untuk exception

```
public Object pop() throws EmptyStackException {  
    Object obj;  
    if (size == 0)  
        throw new EmptyStackException();  
    obj = objectAt(size - 1);  
    setObjectAt(size - 1, null);  
    size--;  
    return obj;  
}
```

Pada program Stack, jika user mengambil elemen pada stack, jika stack kosong maka akan melempar exception `EmptyStackException`

[source: java.sun.com]

Exception Propagation

- Exception dapat ditangani pada tingkat yang lebih tinggi, jika di tempat terjadinya exception tidak ditangani secara tepat.
- Exception merambat naik(*propagate up*) melalui method calling hierarchy sampai exception ditangkap atau sampai mencapai level method main.

Contoh Exception Propagation

```
void method1 throws IOException {
    throw new IOException("Error in method1");
}

void method2 throws IOException {
    // do stuff, but no catch, just specify
    method1();
}

void method3 throws IOException {
    // do stuff, but no catch, just specify
    method2();
}

public static void main (String args[]){
    // catch if just specify error to console
    try {
        method3();
    } catch (IOException e){
        // handle the exception from method1
    }
}
```

Object Exception

- Object exception yang dihasilkan dapat dimanfaatkan untuk mengetahui lebih lanjut mengenai error atau exception yang terjadi.
- Exception merupakan subclass dari class Throwable.



Method yang diwarisi oleh exception:

- getMessage()
method ini mengembalikan isi pesan untuk menggambarkan exception yang terjadi
- printStackTrace()
method ini menampilkan pesan error dan stack trace ke standard error output stream yang biasanya merupakan konsol window apabila program merupakan program konsol.
- printStackTrace(PrintStream s)
method ini mengembalikan pesan error ke objek PrintStream yang dijadikan parameter. Apabila ingin menampilkan pesan ke konsol, anda dapat menggunakan `system.out` sebagai parameter.

Blok try – catch bertingkat

```
try {  
    try {  
        .....  
    } catch (Exception x) {  
        .....  
    }  
  
    try {  
        .....  
    } catch (Exception x) {  
        .....  
    }  
  
} catch (Exception x) {  
    .....  
}
```

Blok Try – Catch - Finally

- Blok finally : digunakan untuk mendefinisikan kode program yang selalu dieksekusi baik ada exception yang terjadi maupun bila tidak terjadi exception sama sekali.

- Bentuk:

```
try {  
    .....  
} catch (Exception e) {  
    .....  
} finally {  
    .....  
}
```


Contoh finally

```
try {  
    out = new PrintWriter(new FileWriter("out.txt"));  
    // statements that throws exceptions  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.err.println("Caught array error");  
} catch (IOException e) {  
    System.err.println("Caught I/O error");  
} finally {  
    if (out != null) {  
        System.out.println("Closing file);  
        out.close();  
    }  
}
```

Contoh: Tanpa Exception Handling

```
1  public class HelloWorld {
2      public static void main (String[] args) {
3          int i = 0;
4
5          String greetings [] = {
6              "Hello world!",
7              "No, I mean it!",
8              "HELLO WORLD!!"
9          };
10
11         while (i < 4) {
12             System.out.println (greetings[i]);
13             i++;
14         }
15     }
16 }
```



Contoh: Dengan Exception Handling

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         int i = 0;
4
5         String[] greetings = {
6             "Hello world!",
7             "No, I mean it!",
8             "HELLO WORLD!!"
9         };
10
11        while (i < 4) {
12            try {
13                System.out.println(greetings[i]);
14                i++;
15            } catch (ArrayIndexOutOfBoundsException e) {
16                System.out.println("Re-setting Index Value");
17                i = 0;
18            } finally {
19                System.out.println("This is always printed");
20            }
21        }
22    }
23 }
```



Membuat class exception baru

- Sebuah subclass dari exception dapat dibuat sendiri oleh programmer untuk mendefinisikan sendiri secara lebih rinci tentang exception yang dapat terjadi.
- Class exception baru ini harus merupakan subclass dari `java.lang.Exception`.



Overriding Method dan Exception

- Overriding method hanya boleh melempar exception yang merupakan subclass dari exception yang dilempar oleh overridden method atau sama.
- Overriding method boleh mendeklarasikan exception lebih sedikit dari jumlah exception kepunyaan overridden method.

Catatan;

- Overriding method = method yang mengoveride.
- Overridden method = method yang dioveride.

Contoh 1: Method Overriding



```
1 public class TestA {
2     public void methodA() throws RuntimeException {
3         // do some number crunching
4     }
5 }

1 public class TestB1 extends TestA {
2     public void methodA() throws ArithmeticException {
3         // do some number crunching
4     }
5 }

1 public class TestB2 extends TestA {
2     public void methodA() throws Exception {
3         // do some number crunching
4     }
5 }
```

- Class TestB1 → ok karena ArithmeticException merupakan subclass dari RuntimeException.
- Class TestB2 → error karena Exception merupakan superclass dari RuntimeException.

Contoh 2: Method Overriding

```
1 import java.io.*;
2
3 public class TestMultiA {
4     public void methodA()
5         throws IOException, RuntimeException {
6         // do some IO stuff
7     }
8 }

1 import java.io.*;
2
3 public class TestMultiB1 extends TestMultiA {
4     public void methodA()
5         throws FileNotFoundException, UTFDataFormatException,
6         ArithmeticException {
7         // do some IO and number crunching stuff
8     }
9 }
```

- Class TestMultiB1 → ok karena FileNotFoundException dan UTFDataFormatException merupakan subclass dari IOException
- Dan Arithmetic Exception merupakan subclass dari RuntimeException.

Contoh 3: Method Overriding

```
1  import java.io.*;
2
3  public class TestMultiA {
4      public void methodA()
5          throws IOException, RuntimeException {
6          // do some IO stuff
7      }
8  }

1  import java.io.*;
2  import java.sql.*;
3
4  public class TestMultiB2 extends TestMultiA {
5      public void methodA()
6          throws FileNotFoundException, UTFDataFormatException,
7          ArithmeticException, SQLException {
8          // do some IO, number crunching, and SQL stuff
9      }
10 }
```

- Class TestMultiB2 → error karena SQLException atau superclass dari SQLException tidak dideklarasikan pada class TestMultiA.
- TestMultiB2 tidak boleh menambahkan exception baru

Contoh 4: Method Overriding

```
1 import java.io.*;
2
3 public class TestMultiA {
4     public void methodA()
5         throws IOException, RuntimeException {
6         // do some IO stuff
7     }
8 }

```

```
1 public class TestMultiB3 extends TestMultiA {
2     public void methodA() throws java.io.FileNotFoundException {
3         // do some file IO
4     }
5 }
```

- Class TestMultiB3 → ok karena FileNotFoundException adalah subclass dari IOException.
- Contoh diatas menunjukkan bahwa overriding method boleh mendeklarasikan exception yang lebih sedikit dari exception kepunyaan override method.

Membuat Exception

- Tujuan: mendefinisikan class exception yang lebih spesifik untuk keperluan tertentu.
- Untuk membuat class exception baru maka class itu harus merupakan subclass dari class Exception.

Contoh 1:

Membuat class exception baru

```
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;  
    }  
}
```

Contoh 2

Membuat class exception baru

```
1 class HateStringExp extends RuntimeException {  
2     /* some code */  
3 }
```

```
1 class TestHateString {  
2     public static void main(String args[]) {  
3         String input = "invalid input";  
4         try {  
5             if (input.equals("invalid input"))  
6                 throw new HateStringExp();  
7         }  
8         System.out.println("Accept string.");  
9         } catch (HateStringExp e) {  
10            System.out.println("Hate string!");  
11        }  
12    }  
13 }
```