

IF2032 – Pemrograman Java: Exception

Achmad Imam Kistijantoro
sumber: Slide Hananto W.
Semester II 2008/2009

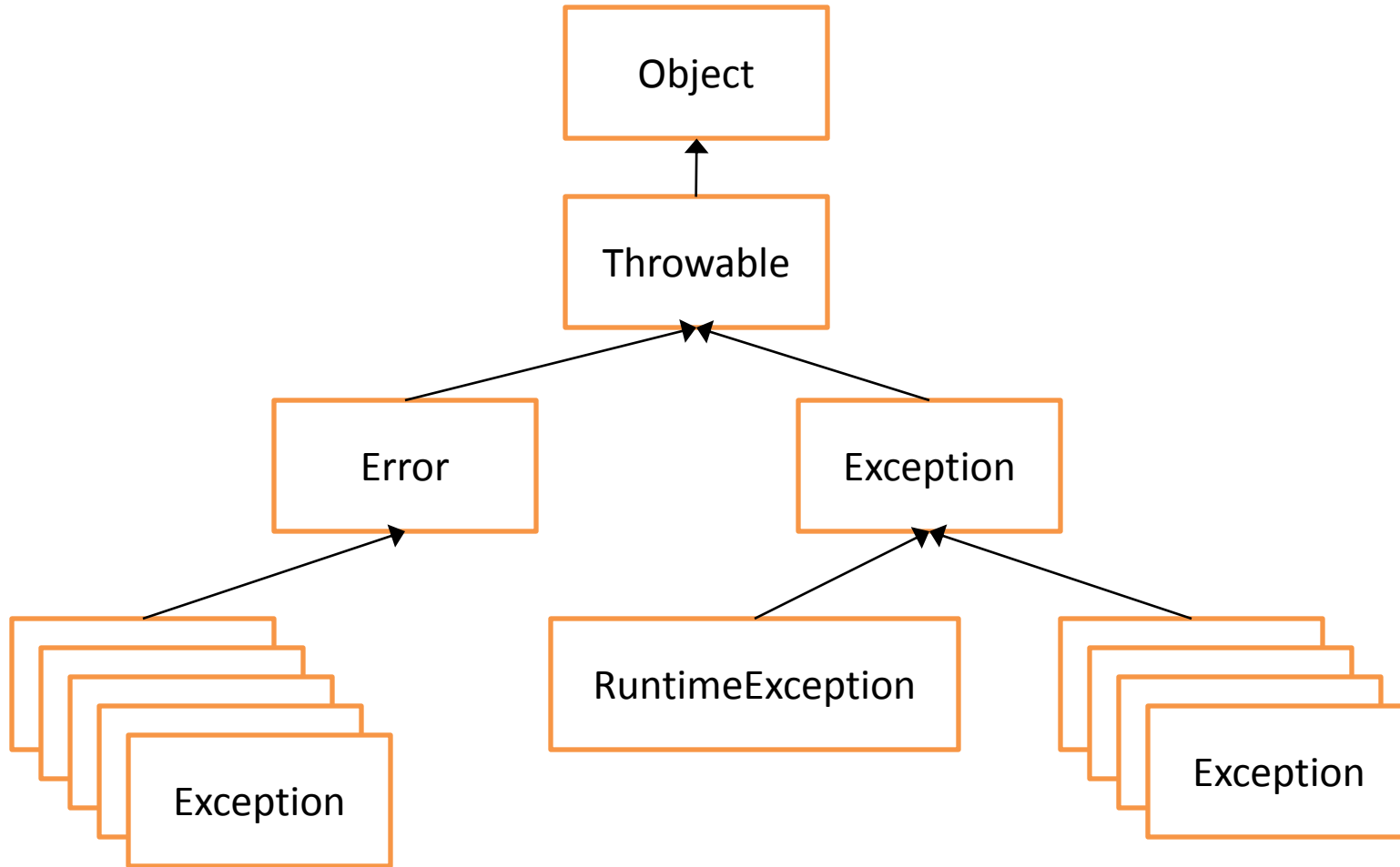
Exceptions

- Sebuah event yang akan menginterupsi alur proses program normal dari sebuah program
- Akan mengakibatkan program terminate abnormally

Exceptions

- Error tidak harus selalu ditangani dengan exception handling (namun exception mempermudah penanganan error)
- Exception handling di Java bekerja dengan cara mengubah alur eksekusi program, sambil melempar suatu objek tertentu sebagai informasi untuk alur yang baru

Hierarki Exception



Runtime Exception

- Exception yang diturunkan dari RuntimeException tidak akan diperiksa pada saat kompilasi. Exception jenis ini baru dapat terdeteksi pada saat eksekusi.
- Contoh: NullPointerException, IndexOutOfBoundsException, dsb.

checked vs unchecked exception

- checked exception: kompilator memeriksa apakah method/blok yang dipanggil dapat menghasilkan exception sesuai dengan yang dideklarasikan
- unchecked exception: kompilator tidak memeriksa apakah exception yang dihasilkan sebuah method/blok sesuai dengan yang dideklarasikan
 - Error
 - RuntimeException

penggunaan

- checked exception: jenis kesalahan yang dapat diperbaiki, programmer diharapkan menyediakan kode penanganan kesalahan yang memadai
 - contoh: FileNotFoundException
- runtime exception: umumnya jenis kesalahan yang diakibatkan programmer, karena tidak mengikuti kontrak yang telah disepakati
 - contoh: ArrayIndexOutOfBoundsException
- Error: kondisi abnormal yang memang sebaiknya tidak ditangani oleh aplikasi
 - LinkageError, OutOfMemoryError

Contoh Exception

- Beberapa contoh exceptions
 - `ArrayIndexOutOfBoundsException` exception, akan terjadi jika kita mengakses suatu array di index yang tidak valid
 - `NumberFormatException`, akan terjadi jika kita melakukan passing parameter non-number ke method `Integer.parseInt()`

Penanganan Exception

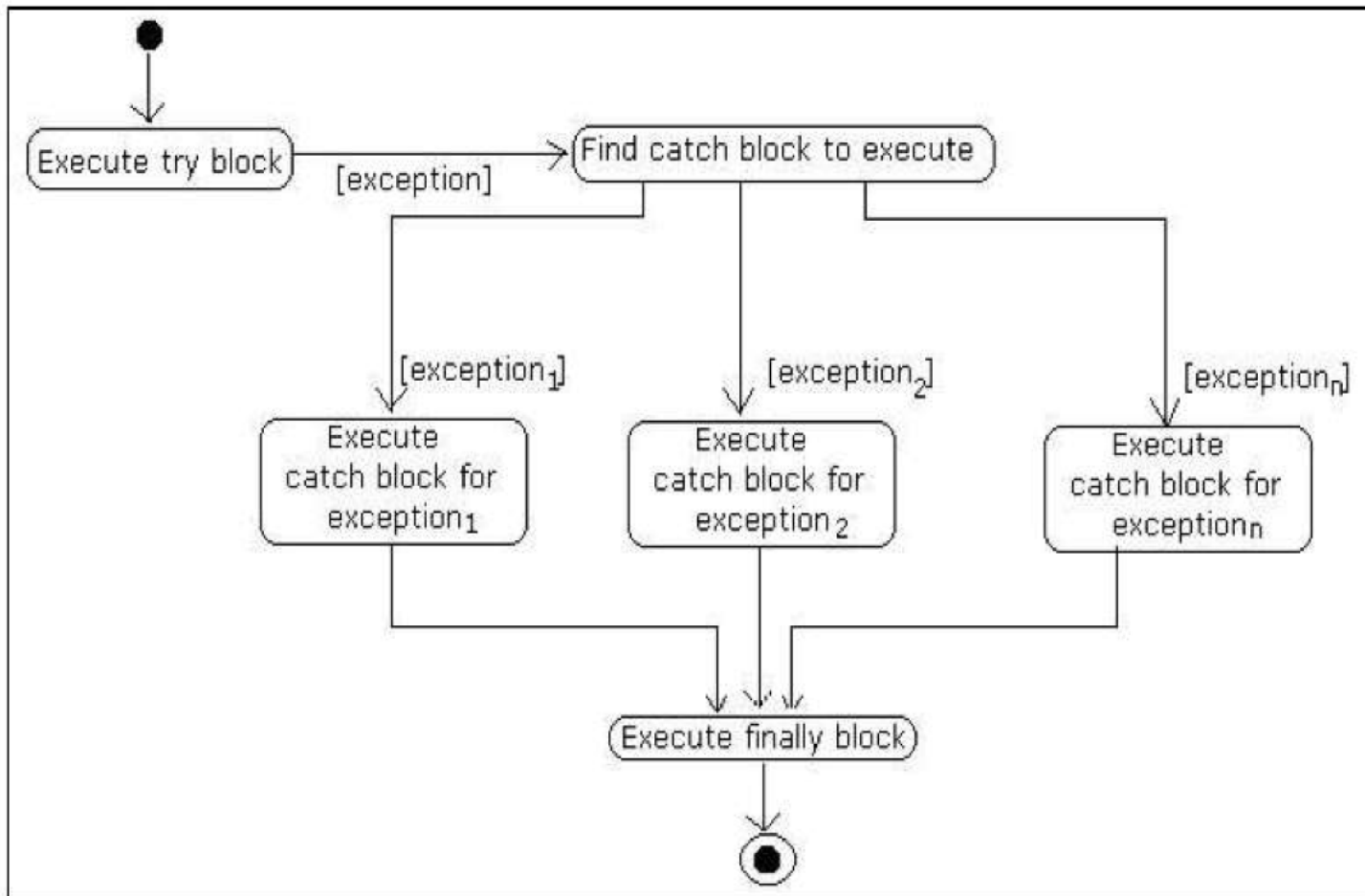
- Untuk menangani exceptions di Java digunakan blok `try-catch-finally`
- Statement program yang memungkinkan terjadi exceptions harus diletakkan dalam blok `try-catch-finally`

blok try-catch-finally

- Bentuk umum dari blok try-catch-finally


```
try{
    //write the statement that can generate an exceptions
    //in this block
}
catch(<exceptionType1> <varName1>){
    //write the action your program will do if an exception of certain
    //type occurred
}
...
}
catch(<exceptionTypen> <varNamen>){
    //write the action your program will do if an exception of certain
    //type occurred
}
finally{
    //add more cleanup code here
}
```

Alur program



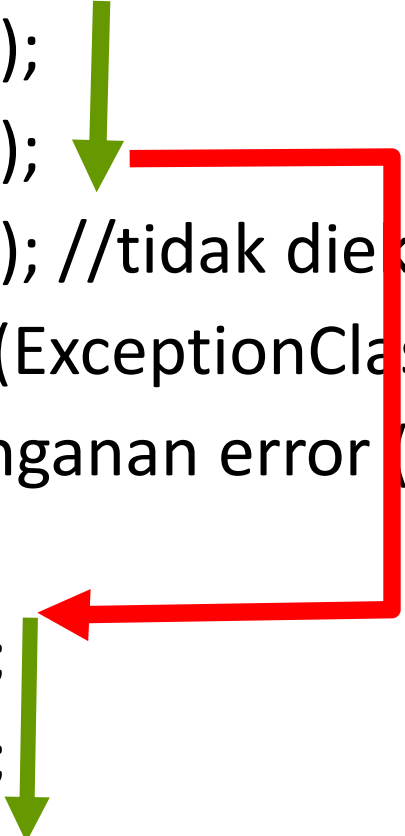
Alur normal

```
try {  
    aksi_1();  
    aksi_2();  
    aksi_3();  
} catch (ExceptionClassName name){  
    //penanganan error (tidak dieksekusi)  
}  
aksi_4();  
aksi_5();
```



Exception dilempar aksi_2

```
try {  
    aksi_1();  
    aksi_2();  
    aksi_3(); //tidak dieksekusi  
} catch (ExceptionClassName name){  
    //penanganan error (dieksekusi)  
}  
aksi_4();  
aksi_5();
```



The diagram illustrates the execution flow of the provided Java code. A green arrow points down from the line `aksi_2();` to the start of the `catch` block, indicating that an exception was thrown at this point. A red arrow then points from the `catch` block down to the line `aksi_4();`, showing that the code following the `try` block is not executed when an exception occurs. Another green arrow points down from `aksi_4();` to `aksi_5();`, indicating the normal flow of execution after the `try` block.

catatan

- Dalam statement
 - `throw new Exception()`, sebuah instans dari kelas `Exception()` diciptakan (konstruktor defaultnya dipanggil)
- Jika kelas memiliki konstruktor yang lain, konstruktor itu dapat dipanggil:
 - `throw new Exception("lebih dari 12");`
- Dalam catch, method kelas exception dapat dipanggil:
 - `System.out.println(e.getMessage());`

nested exception handling

- Boleh ada blok try catch dalam try catch, penangkap adalah blok terdekat

```
try {  
    fungsi_a();  
    try {  
        fungsi(b);  
    } catch (NumberFormatException n) {  
        //throw new FatalException("error baru");  
        throw n;  
    }  
} catch (FatalException fatal) { }
```

```
}
```

Melempar kembali exception

- Dalam catch, exception bisa dithrow:

```
catch (Exception e) {throw e;}
```

- Exception baru bisa dibuat:

```
catch (FileNotFoundException e) {  
    throw FatalException();  
}
```

- Gunanya melempar kembali agar exception ditangkap oleh klausa exception terluar, atau method yang memanggil method ini.

exception pada method

```
void connect_internet() throws Exception {  
    //coba melakukan koneksi  
    if (error) throw new connect_exception("error");  
    //....  
}  
  
//...  
//bagian main  
try {  
    connect_internet();  
} catch (connect_exception e) {  
    /*do something*/  
} catch (Exception ex) {}
```

catatan

- Exception adalah *nama kelas* exception standar milik Java
- Programmer bisa membuat sendiri kelas Exception dengan mengimplementasikan Throwable, atau (yang lebih disarankan) menurunkan dari `java.lang.Exception`

Membuat kelas exception

- Buat class exception yang diturunkan dari class exception lain yang lebih umum
 - Misal class `OutOfDiskSpaceException` bisa diturunkan dari `IOException`
- Sebaiknya turunkan dari class `Exception` karena sudah memiliki method untuk mencatat pesan exception

Contoh: SmallInt exception

```
class SmallIntExcept extends Exception
{
    private static int num_except;
    SmallIntExcept(String msg) {
        super(msg);
        num_except++;
    }
    static int numException () {
        return num_except;
    }
    void response () {
        System.out.println(getMessage());
    }
};
```

kelas SmallInt

```
class SmallInt{
    int value;
    SmallInt(int val){
        value = val;
    }
    void plus(SmallInt X) throws SmallIntExcept{
        value = value + X.value;
        if (value > 10) throw (SmallIntExcept ("TOO BIG"));
        if (value < 0)   throw (SmallIntExcept ("TOO SMALL"));
    }
    public String toString() {
        return Integer.toString(value);
    }
    void ReadVal () {
        Scanner s = new Scanner(System.in);
        value = s.nextInt();
    }
}
```

Main program

```
class SmallIntExample {
    public static void main (String args[]) {
        System.out.println("start of smallint ...");
        SmallInt S1= new SmallInt(1);
        SmallInt S = new SmallInt();
        S.ReadVal ();
        try {
            S1.plus (S);
            System.out.println("hasil S1= S1+S =" +S1);
        }catch (SmallIntExcept e) {
            e.response ();
        }
    }
}
```

exception trace

- kelas Throwable menyediakan informasi eksekusi kode yang mengakibatkan kesalahan, yaitu execution stack trace.
 - `e.printStackTrace()`
 - `StackTraceElement[] e.getStackTrace()`
- kelas Throwable juga menyediakan mekanisme chained exception: sebuah exception yang terjadi akibat exception lainnya

```
try {  
    lowLevelOp();  
} catch (LowLevelException le) {  
    throw new HighLevelException(le);  
}
```

Exception abuse

```
try {  
    int i = 0;  
    while(true)  
        a[i++].f();  
} catch (ArrayIndexOutOfBoundsException  
    e) {  
  
}
```

Contoh kasus: penanganan end of file: EOFException ?

contoh

```
public Object pop() throws EmptyStackException {
    try {
        //      if( size == 0) throw new
        //EmptyStackException( e );

        Object result = elements[--size];
        elements[size] = null;
        return result;
    } catch (ArrayIndexOutOfBoundsException e) {
        throw new EmptyStackException( e );
    }
}
```