

Bahasa Pemrograman 2

Exception Handling

anton@ukdw.ac.id

Exception

- Exception adalah sebuah indikasi **masalah** yang muncul *saat program dijalankan*
- Exception adalah kondisi **abnormal** yang terjadi *saat program dijalankan*
- Saat program dijalankan → run-time
- Exception untuk run-time error (run-time error management)

Exception

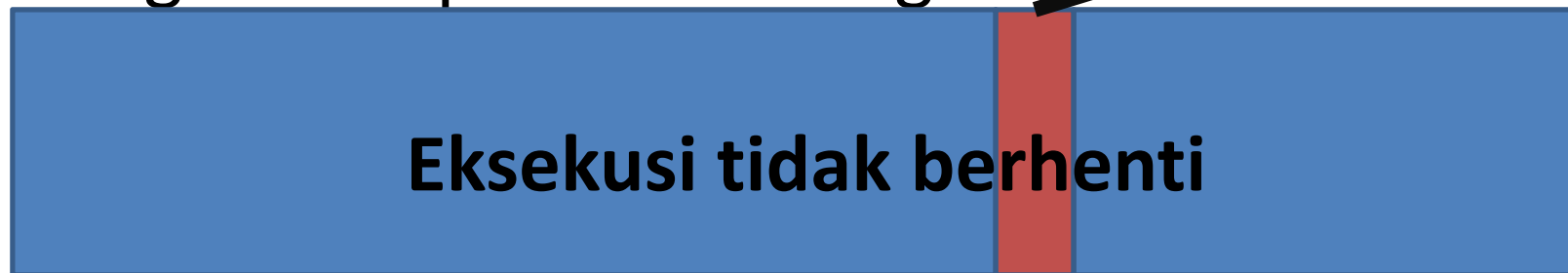
- **Exception** : eksepsi, problem yang muncul jarang terjadi (infrequently)
- Exception perlu ditangani (exception handling)
- Exception handling: memungkinkan program tetap berjalan seakan-akan tidak muncul masalah
- **Fault tolerant** : jika terjadi masalah program tidak berhenti begitu saja

Exception

- Tanpa Exception Handling



- Dengan Exception Handling



Execption pseudocode

- Pseudocode

...

Lakukan proses input

*Jika input tidak valid lakukan
error-processing*

Lakukan proses penghitungan

*Jika proses penghitungan gagal
lakukan error-processing*

Lakukan proses penampilan hasil

...

Overview

- Pada bahasa pemrograman **procedural**, error handling ditangani bersamaan dengan proses-proses dalam program yang dibuat (*inline error handling*)
- Inline error handling membuat program sulit untuk dibaca, dimodifikasi, debugging dan maintenance
- Pada Java, error handling dapat dilakukan bersamaan atau **terpisah** dari kode program utama

Overview

- Java Exception: **object** yang menggambarkan sebuah kondisi eksepsional (exception) pada suatu bagian kode
- Saat terjadi exception, sebuah object yang mewakili exception tersebut **dibuat** dan **dilemparkan (thrown)** dari method dimana exception tersebut terjadi
- Method tersebut dapat memilih untuk menghandle sendiri exception tersebut atau melemparkannya pada yang lain

Overview

- Exception dapat dihasilkan dari **java run-time system** maupun dihasilkan secara **manual** dari kode program

Overview

- Exception yang dihasilkan oleh Java biasanya terjadi karena **kesalahan dasar** seperti melanggar ketentuan-ketentuan dari bahasa pemrograman Java, pelanggaran pada batasan-batasan yang telah didefinisikan dalam **Java run-time**
- Exception yang dihasilkan secara manual (generated by code) digunakan untuk **melaporkan atau menangkap exception** yang terjadi pada suatu **method**

DivisionByZero Error

```
public class Example01 {  
  
    public static void main(String[] args) {  
        int pbg = Integer.parseInt(System.console().readLine("Masukkan bilangan pembilang: "));  
        int pyt = Integer.parseInt(System.console().readLine("Masukkan bilangan penyebut: "));  
        int hasil = pembagian(pbg, pyt);  
        System.out.println("Hasil " + pbg + "/" + pyt + " = " + hasil);  
        System.out.println("Ini kalo error gak muncul");  
    }  
  
    public static int pembagian(int pembilang, int penyebut) {  
        return pembilang/penyebut;  
    }  
}
```

Overview

Muncul Error:

Stack trace:

Exception in thread "main"

java.lang.ArithmeticException: / by zero

at example01.Main.pembagian(Main.java:18)

at example01.Main.main(Main.java:13)

Java Result: 1

DivByZero with Exception

```
public class Example02 {  
  
    public static void main(String[] args) {  
        try {  
            int pbg = Integer.parseInt(System.console().readLine("Masukkan bilangan pembilang: "));  
            int pyt = Integer.parseInt(System.console().readLine("Masukkan bilangan penyebut: "));  
            int hasil = pembagian(pbg, pyt);  
            System.out.println("Hasil " + pbg + "/" + pyt + " = " + hasil);  
        } catch (ArithmeticException ae) {  
            System.out.println("Terjadi exception ArithmeticException");  
        }  
        System.out.println("Ini akan ditampilkan");  
    }  
  
    public static int pembagian(int pembilang, int penyebut)  
        throws ArithmeticException {  
        return pembilang/penyebut;  
    }  
}
```

Demo Exception Flow

- Example03

Kapan exception dapat digunakan?

- Exception handling untuk **synchronous error**
 - Error yang terjadi saat sebuah perintah dijalankan (run)
- Tidak dapat digunakan untuk **asynchronous error**

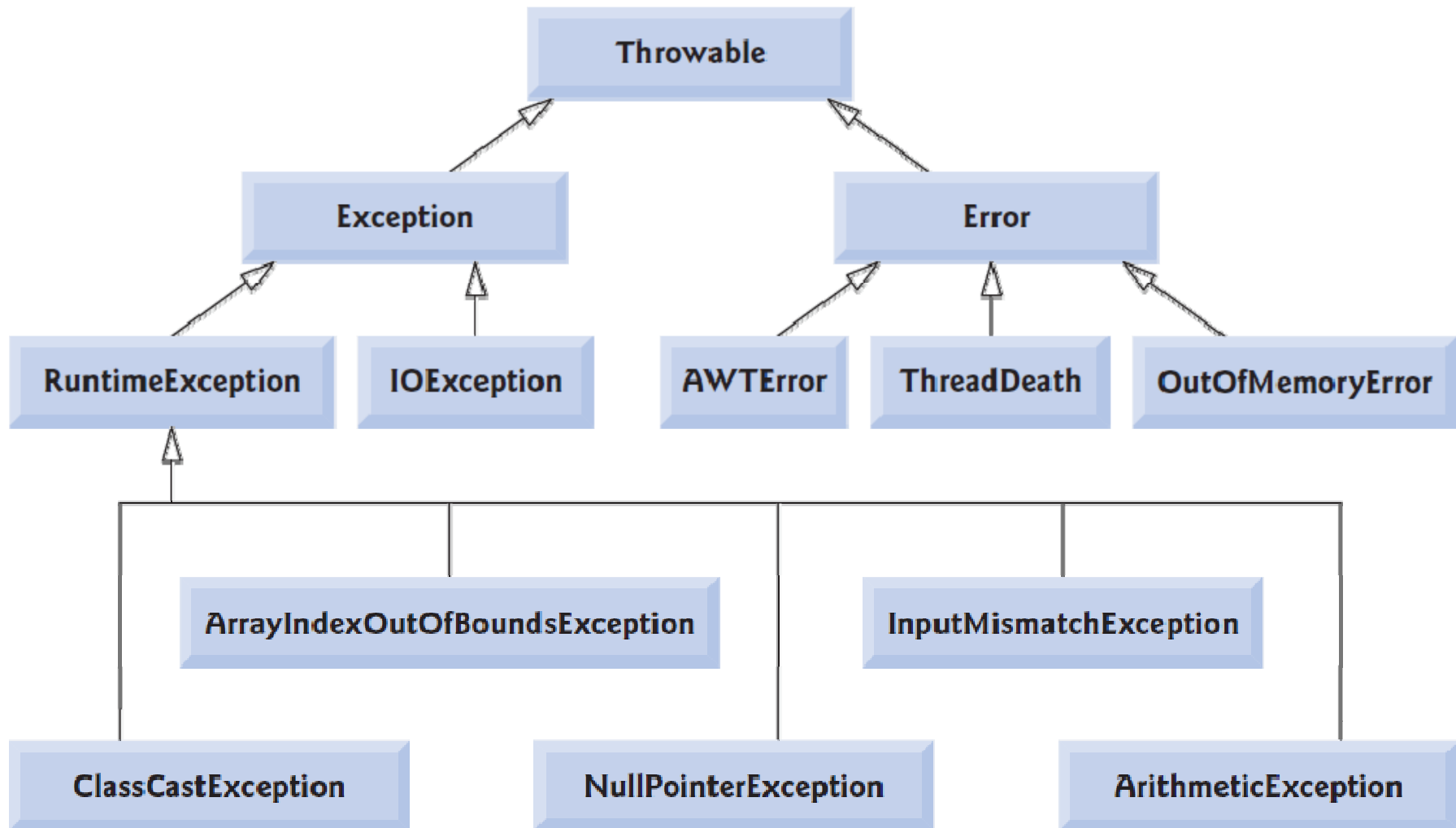
Overview

- **Synchronous Error** : division by zero, out of bound array, overflow, invalid method parameter, etc
- **Asynchronous Error** : Network transfer, mouse clicks, keystrokes, etc → yang terjadi secara paralel dan independen terhadap aliran kontrol program (program flow control)

Hierarki Java Exception

- Semua Java Exception merupakan keturunan (inherits) dari class **Exception**, baik secara langsung maupun tidak langsung

Hierarki Java Exception



Hierarki Exception

- Seluruh class yang merupakan turunan dari class **Exception** *tapi bukan turunan dari `RuntimeException`* merupakan **checked Exception**
- Seluruh class yang merupakan turunan dari class **Error** merupakan **unchecked Exception**

Hierarki Exception

- Turunan dari **Throwable** ada 2, yaitu **Exception** dan **Error**
- **Exception**: dapat ditangani oleh kode program dan eksekusi program dapat dilanjutkan
- **Error**: terjadi kesalahan pada Java Virtual Machine, kesalahan yang ditimbulkan oleh environment/system, tidak dapat dihandle oleh program

Hierarki Exception

- **Checked Exception** : Java Compiler mengecek kode program, apakah suatu method mungkin menghasilkan exception, apakah exception tersebut sudah ditangkap (catch) atau dilempar (throws)
 - invalid user input, database problems, network outages, absent files
- **Unchecked Exception** : Java Compiler tidak mengecek apakah suatu kode program menghasilkan exception atau tidak
 - Berupa logic error
 - IllegalArgumentException, NullPointerException, atau IllegalStateException

Unchecked Exception

```
public static void checkForPositive(int aNumber) {  
    if (aNumber < 1) {  
        throw new IllegalArgumentException(aNumber + " is less than 1");  
    }  
}
```

```
public static void checkForNull(Object aObject) {  
    if ( aObject == null ) {  
        throw new NullPointerException();  
    }  
}
```

Penggunaan Java Exception

- Terdapat 5 keywords:
try, catch, finally, throw, throws

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

Penggunaan Java Exception

- Dengan **try-catch**

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

Penggunaan Java Exception

- Dengan **try-catch-finally**

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

Penggunaan Java Exception

- Block **finally** : pasti dijalankan dalam semua kondisi (terjadi exception atau tidak)
- Digunakan untuk menanggulangi **resource leak** (misal: kehabisan memory, deadlock/starvation)
- Contoh: network error, file handle error, etc

Penggunaan Java Exception

```
public class Example04 {  
    public static void main(String[] args) {  
        try {  
            int pembilang = Integer.parseInt(System.console().readLine("Masukkan bilangan pembilang: "));  
            int penyebut = Integer.parseInt(System.console().readLine("Masukkan bilangan penyebut: "));  
            int hasil = pembilang/penyebut;  
            System.out.println("Hasil " + pembilang + "/" + penyebut + " = " + hasil);  
        } catch (Exception e){  
            System.out.println("Terjadi exception");  
            e.printStackTrace();  
        } finally {  
            System.out.println("Bagian ini akan tetap muncul :)");  
        }  
    }  
}
```

Penggunaan Java Exception

- Selain try-catch-finally, terdapat 2 keyword lagi:
- **throw**
digunakan untuk melempar exception
- **throws**
digunakan untuk mendeklarasikan exception apa saja yang bisa dilempar

Penggunaan Java Exception

- **throw**
melempar exception
- Bentuknya:
throw ThrowableInstance;

ThrowableInstance: merupakan object dari class Throwable atau turunannya

Penggunaan Java Exception

```
public class Example05 {  
  
    public static void main(String[] args) {  
        try {  
            demoproc();  
        } catch (NullPointerException e) {  
            System.out.println("Exception ditangkap di method main: " + e);  
        }  
    }  
  
    static void demoproc() {  
        try {  
            throw new NullPointerException("demo");  
        } catch (NullPointerException e) {  
            System.out.println("Exception ditangkap di method demoproc");  
            throw e; // rethrow the exception  
        }  
    }  
}
```

Penggunaan Java Exception

- **throws** : exception apa saja yang bisa dilempar oleh suatu method
- Bentuk umumnya :

*type method-name(parameter-list) **throws**
exception-list*

{

 // body of method

}

Penggunaan Java Exception

```
public class Example06 {  
  
    public static void throwOne() throws IllegalAccessException {  
        System.out.println("Inside throwOne.");  
        throw new IllegalAccessException("demo");  
    }  
  
    public static void main(String[] args) {  
        try {  
            throwOne();  
        } catch (IllegalAccessException ex) {  
            System.out.println("Illegal Access");  
        }  
    }  
}
```

Penggunaan Java Exception

- Java menyediakan fasilitas stack-trace untuk menampilkan urutan dari terjadinya exception
- Exception bisa terjadi secara berantai (**Chained Exception**)
- Chained Exception : urutan exception dari method ke method

Penggunaan Java Exception

- Demo

→ Chained Exception (Example07)

Pembuatan Exception Baru

- Selain menggunakan Exception yang sudah didefinisikan oleh library Java (bawaan Java), anda juga dapat membuat **exception sendiri** untuk keperluan tertentu
- Harus merupakan **turunan** dari class **Exception**

Pembuatan Exception Baru

- Misal anda ingin meminta input jargon (“client”)
- Definisikan sebuah exception baru (misal namanya: InputJargonException)
- Exception terjadi apabila inputnya bukan “client”

Pembuatan Exception Baru

```
class InputJargonException extends Exception {  
    private String nama;  
    InputJargonException(String n) {  
        nama = n;  
    }  
  
    public String toString() {  
        return "Ini adalah Exception InputJargonException  
            : " + nama;  
    }  
}
```

Pembuatan Exception Baru

```
public static String inputHuruf() throws
    InputJargonException {
    Scanner input = new Scanner(System.in);
    System.out.print("Masukkan jargon : ");
    String hasil = input.next();
    System.out.println("Anda memasukkan " + hasil);
    if(hasil.equalsIgnoreCase("client") == false)
        throw new InputJargonException("Jargon Error");
    return hasil;
}
```

NEXT

- Class Diagram