
# Java: Exception

IF2210 – Semester II 2020/2021

by: RSP; rev: SAR

# Exception

- *Exceptional Event*
  - Definisi: sebuah *event* yang terjadi saat eksekusi program, yang mengganggu alur normal dari instruksi program.
- Ketika sebuah *error* terjadi pada sebuah *method*, maka *method* tersebut akan menciptakan sebuah objek dan dilempar ke *runtime system*.
  - Objek yang diciptakan itu disebut *exception object*.

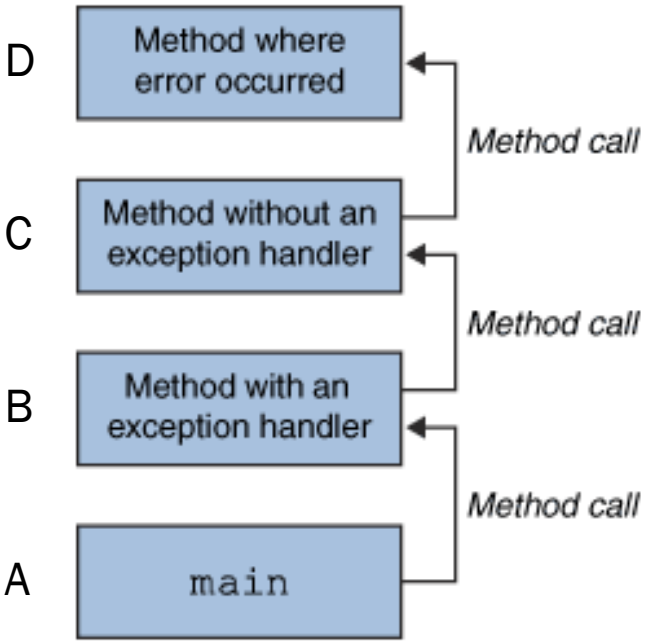
# Exception object

- *Exception object* mengandung informasi tentang *error* tersebut (termasuk tipe dan *state* program ketika *error* terjadi).
- Menciptakan *exception object* dan melempar ke *runtime system* disebut *"throwing an exception"*.

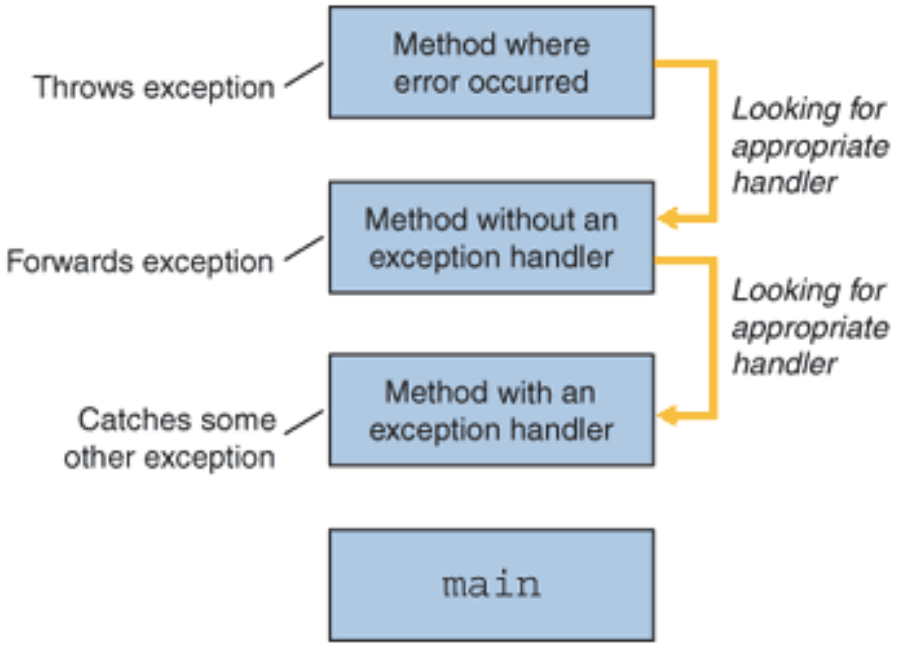
# Exception handler

- Setelah *method* melempar *exception*, *runtime system* akan mencari sesuatu untuk meng-*handle* itu (disebut *exception handler*).
- *Exception handler* akan melakukan *"catching the exception"*.

# Exception handler




The call stack.




Searching the call stack for the exception handler.

# Exception handler

- Komponen: blok-blok *try*, *catch*, dan *finally*.

```
try {
  // do something

} catch (ExceptionType1 exceptionName) {
  // handle a specific exception type

} catch (ExceptionType2 exceptionName) {
  // handle another exception type

} finally {
  // will be executed regardless of exception

}
```

# Try block

- Berisi kode yang mungkin memunculkan *exception*.
- *Try block* bisa dibuat untuk setiap kode yang mungkin menimbulkan *exception*.
- Bisa juga dengan mengumpulkan banyak kode dalam sebuah *try block*.

```
private Vector v;
private static final int SIZE=10;

PrintWriter out = null;

try {

  System.out.println("Entered try block");
  out = new PrintWriter(new FileWriter("out.txt"));
  for (int i=0; i<SIZE; i++) {
    out.println("Value at " + i + ": " + v.elementAt(i));
  }

} catch (/*...*/) {
    // ...

} finally {
    // ...

}
```

# Catch block

- Berisi kode yang merupakan *exception handler*.
- Menangani *exception* dengan tipe yang sesuai dengan tipe yang ditunjukkan pada argumen.
- Satu *try block* dapat diikuti dengan lebih dari satu *catch block*.

```
try {
  // ...

} catch(FileNotFoundException e) {

  System.err.println("FileNotFoundException: " + e.getMessage());
  throw new SampleException(e);

} catch(IOException e) {

  System.err.println("Caught IOException: " + e.getMessage());

} finally {
  // ...

}
```

# Finally block

- Berisi kode yang **pasti** akan dieksekusi setelah *try block*.
- *Finally block* akan dieksekusi baik terjadi *exception* maupun tidak.
- Meskipun di dalam *try block* atau *catch block* ada `return` atau `throw`, *finally block* akan tetap dieksekusi terlebih dahulu sebelum melanjutkan `return` atau `throw`.

```
try {
  // ...

} catch(/*...*/) {
  // ...

} finally {

  if (out != null) {
    System.out.println("Closing PrintWriter");
    out.close();
  } else {
    System.out.println("PrintWriter not open");
  }

}
```

# Kode lengkap

```
private Vector v;
private static final int SIZE=10;

public void writeVector() {
  PrintWriter out = null;
  try {
    System.out.println("Entered try block");
    out = new PrintWriter(new FileWriter("out.txt"));
    for (int i=0; i<SIZE; i++) {
      out.println("Value at " + i + ": " + v.elementAt(i));
    }
  } catch(FileNotFoundException e) {
    System.err.println("FileNotFoundException: " + e.getMessage());
    throw new SampleException(e);
  } catch(IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
  } finally {
    if (out != null) {
      System.out.println("Closing PrintWriter");
      out.close();
    } else {
      System.out.println("PrintWriter not open");
    }
  }
}
```

# Try-with-resources

- Mulai Java 7, ada mekanisme *try-with-resources*.
- Dapat digunakan untuk mendeklarasikan variabel yang akan digunakan di dalam *try block*.
- Variabel tersebut akan otomatis di-`close()` setelah blok selesai, tanpa perlu *finally block*.
- Variabel harus mengimplementasi *interface* `AutoCloseable`.

# Dengan try-with-resources

```
private Vector v;
private static final int SIZE=10;

public void writeVector() {

  try(PrintWriter out = new PrintWriter(new FileWriter("out.txt"))) {

    System.out.println("Entered try block");
    for (int i=0; i<SIZE; i++) {
      out.println("Value at " + i + ": " + v.elementAt(i));
    }

  } catch(FileNotFoundException e) {
    System.err.println("FileNotFoundException: " + e.getMessage());
    throw new SampleException(e);
  } catch(IOException e) {
    System.err.println("Caught IOException: " + e.getMessage());
  }
}
```

# Cara throw exception

- Menggunakan *throw statement*.
- Membutuhkan sebuah argumen berupa objek *throwable*.
- Objek *throwable* adalah instansiasi dari kelas `Throwable` atau turunannya.

```
throw someThrowableObject;
```

- Jika yang di-*throw* adalah *checked exception* (lebih lanjut pada slide-slide setelah ini), deklarasi *method* harus disertai *keyword* `throws`:

```
void doSomethingTo(Object x) throws SomeException { ... }
```

# Contoh

```
public Object pop() {
  int len = size();

  if (len == 0) {
    throw new EmptyStackException();
  }

  Object e = elementAt(len - 1);
  removeElementAt(len - 1);
  return e;
}
```

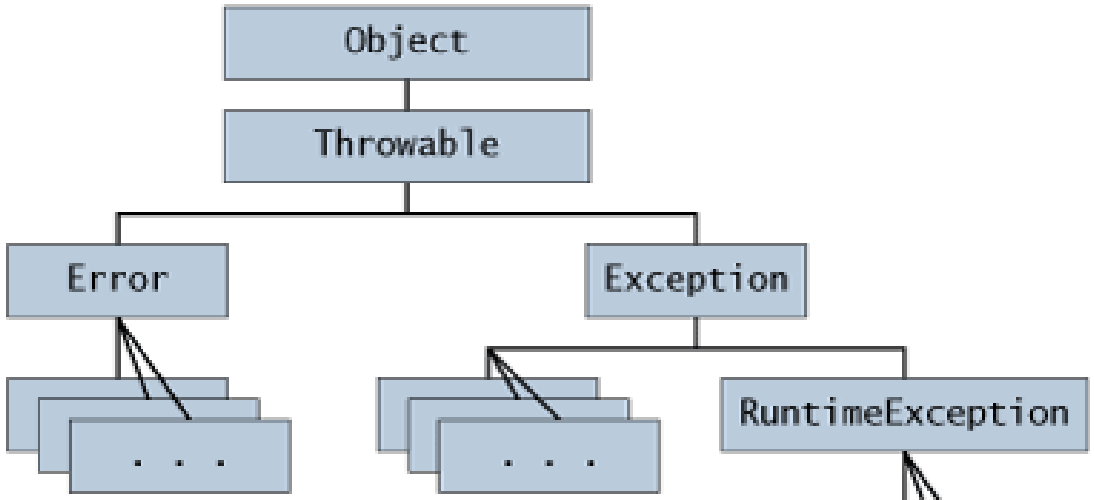
# Chained exception

- Method bisa merespon terjadinya *exception* dengan melempar *exception* lagi
- Digunakan untuk membungkus *exception* dengan informasi tambahan
- Contoh:

```
try {
  // ...

} catch (IOException e) {
  throw new SampleException("Other IOException", e);
}
```

# Kelas Throwable dan turunannya (1)




**should not** be caught
**could** be declared
**irrecoverable** errors

**must** be caught
**must** be declared
**recoverable** errors

**should** be caught
**could** be declared
**recoverable** errors

# Kelas Throwable dan turunannya (2)

- `Error` *indicates serious problems that a reasonable application should not try to catch.*
- `Exception` *indicates conditions that a reasonable application might want to catch.*
- `Error` dan `RuntimeException` bersifat *unchecked*, `Exception` selain itu bersifat *checked*.
- Lebih lanjut, baca: https://stackoverflow.com/questions/5813614/what-is-difference-between-errors-and-exceptions/5813695

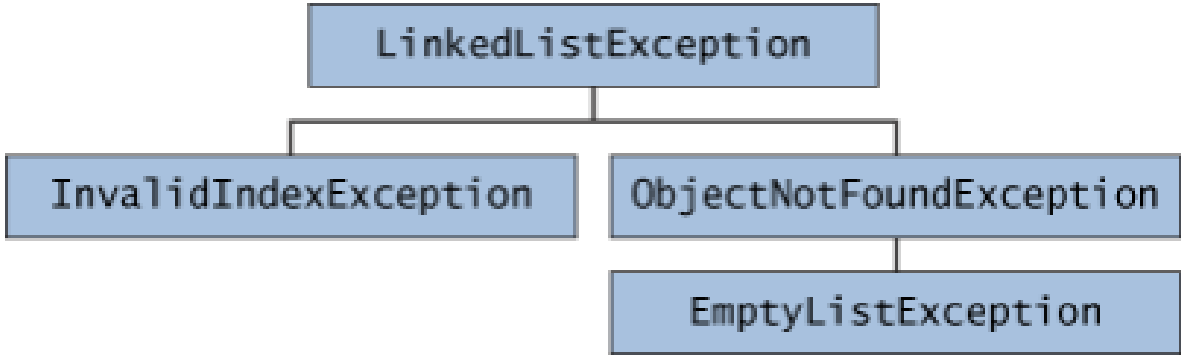
# Membuat kelas exception (1)

- *Exception object* dapat menggunakan kelas yang dibuat orang lain (termasuk yang ada di Java platform) atau membuat sendiri.
- *Exception Subclass* dapat digunakan sebagai *parent*.
- Sebagai konvensi, tambahkan kata *Exception* di nama kelas agar kode lebih mudah dibaca/dipahami.

# Membuat kelas exception (2)

- Alasan membuat *exception class* sendiri:
  - Tidak ada di Java platform.
  - Lebih memudahkan pengguna kelas.
  - Digunakan di banyak tempat.
- Perlu diperhatikan juga independensi package atau *self-contained*.

# Contoh LinkedListException



# Kelebihan exception

- Memisahkan *error handling code* dengan *regular code*.
- Mempropagasikan *error* ke atas (ke *method* pemanggil) di *call stack*.
- Mengelompokkan dan membedakan tipe *error*.