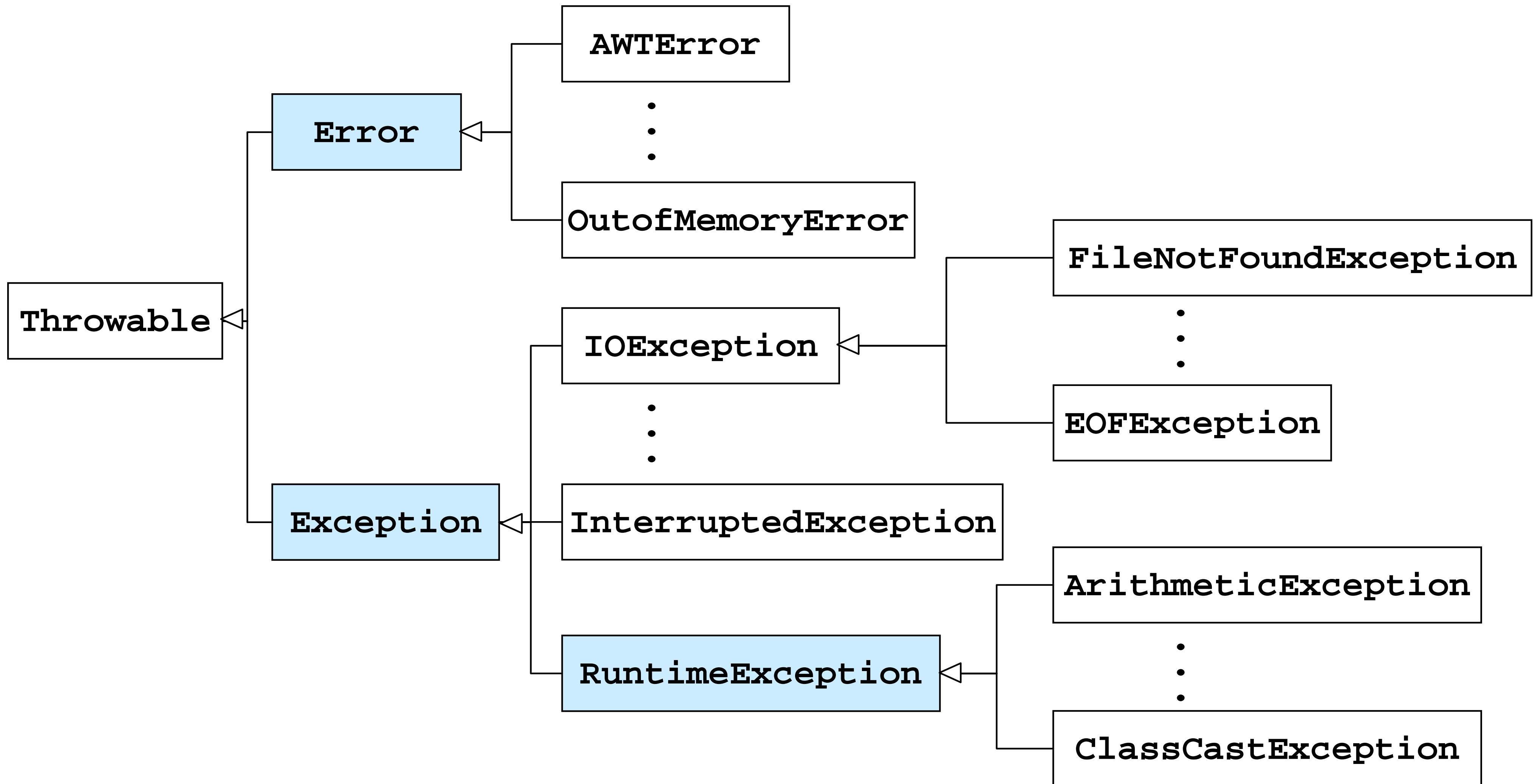
วาจาจาวา – ตอนที่ ๖

Exceptions & Assertions

สมชาย ประสิทธิ์จูตระกูล

Throwables



© S.Prasitjutrakul 2004

Use exceptions only for exceptional conditions

- อย่าใช้ exception แทน control flow แบบปกติ
  - ช้ากว่า มี bug ไม่รู้
- state-dependent class ควรมี
  - state-testing method
  - state-dependent method ที่คืนค่าพิเศษบอกสถานะ

```
try {
  Iterator i = c.iterator();
  while (true) {
    Goo g = (Goo) i.next();
    ...
  }
} catch (NoSuchElementException e) {}
```

```
for (Iterator i = c.iterator();;) {
  Goo g = (Goo) i.next();
  if (g == null) break;
  ...
}
```

```
for (Iterator i = c.iterator(); i.hasNext();) {
  Goo g = (Goo) i.next();
  ...
}
```

RuntimeExceptions และลูกหลาน

- recovery is impossible
- continued execution would do more harm than good
- indicate programming errors, precondition violations

Checked Exceptions

- exceptional conditions
  - can't be prevented by proper use of the API
  - from which the caller can be expected to recover
- checked exceptions force the caller
  - to handle the exception in a catch clause
  - or to propagate the exception outward
- provide helper method to recover

```
public class HttpRetryException extends IOException {
  public HttpRetryException(String detail, int code) {...}
  public HttpRetryException(String detail, int code,
                            String location) {...}
  public int responseCode() {...}
  public String getReason() {...}
  public String getLocation() {...}
  ...
```

Favor the use of standard exceptions

- IllegalArgumentException
- IllegalStateException
- NullPointerException
- IndexOutOfBoundsException
- UnsupportedOperationException
- ConcurrentModificationException

Throw exceptions appropriate to the abstraction

- don't propagate low-level exceptions from a higher-level class (expose implementation)
- use exception translation, exception chaining

```
public Object get(int index) {
  ListIterator e = listIterator(index);
  try {
    return(e.next());
  } catch (NoSuchElementException exc) {
    throw new IndexOutOfBoundsException("Index: " + index);
  }
}
```

```
public Employee getEmployee() throws EmployeeLookupException {
  ...
  try {
    stmt.execute(sql);
  } catch (SQLException ex) {
    throw new EmployeeLookupException(ex);
  }          // exception chaining
  ...
}
```

Exception chaining (Java 1.4)

```java
try {
  lowLevelOp();
} catch (LowLevelException le) {
  // Chaining-aware constructor
  throw new HighLevelException(le);
}
```

```java
try {
  lowLevelOp();
} catch (LowLevelException le) {
  // Legacy constructor
  throw (HighLevelException)
        new HighLevelException().initCause(le);
}
```

Include failure-capture info in detail messages

- contains value of all parameters and fields that contributed to the exception
- unimportant to include a lot of "prose"
- require this information in the constructors

```
public class IndexOutOfBoundsException extends RuntimeException {
  ...
  /**
   * Constructs an IndexOutOfBoundsException.
   * @param lower the lowest legal index value.
   * @param upper the highest legal index value plus one.
   * @param index the actual index value
   */
  public IndexOutOfBoundsException(int lower, int upper, int index) {
    super("lower: " + lower + ", upper: " + upper +
          ", index: " + index));
    )
  }
}
```

Document all exceptions thrown by each method

- declare checked exception individually
  - don't declare some superclass of exceptions e.g., ... throws Exception, ... throws Throwable
- don't declare unchecked exceptions
- document each one using @throws tag

```
/**
 * Skip characters.
 *
 * @param   n   The number of characters to skip
 * @return      The number of characters actually skipped
 * @throws      IllegalArgumentException  If n is negative.
 * @throws      IOException  If an I/O error occurs
 */
public long skip(long n) throws IOException {
    ...
}
```

© S.Prasitjutrakul 2004

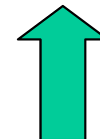
Strive for failure atomicity

- failed method invocation leaves the object
  - in the state it was prior to the invocation
  - in a reasonable state
- to achieve failure atomicity
  - failure atomicity is free for immutable objects
  - validate parameters before performing
  - write recovery code to roll back object's state
  - perform the operation on a temporary copy
- failure atomicity
  - important for checked exceptions
  - is not always achievable
  - is not always desirable (increase cost)

Don't ignore exceptions

- อย่า "กลืน" exception objects
  - empty catch block
- หากมั่นใจสุด ๆ ว่าไม่มีทางเกิด exception ก็
  - comment
  - assert false;
- บ้านติดสัญญาณเตือนไฟไหม้ แต่ปิดระบบ

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



© S.Prasitjutrakul 2004

Be Specific

```
class Account {
  protected long balance;
  public void withdraw(long amount)
              throw IOException {
    if (balance < amount) throw new IOException();
    balance -= amount;
  }
  ...
}
```

ไม่สื่อความหมาย

```
public void withdraw(long amount) throw Exception {
  if (balance < amount) throw new Exception();
  balance -= amount;
}
```

ความหมายกว้างไป

```
public void withdraw(long amount)
            throw InsufficientFundException {
  if (balance < amount)
    throw new InsufficientFundException();
  balance -= amount;
}
```

ชื่อสื่อความหมาย

© S.Prasitjutrakul 2004

Assertion : Syntax

```
assert booleanExpression;
```

```
assert booleanExpression : expression;
```

String ที่แทนข้อความแสดงรายละเอียดของความผิดพลาด

```
private void heatReactor(int temp) {
  assert 200 < temp && temp < 1000 : "too hot " + temp;
  reactor.setHeat(temp);
  ...
}
```

Assertion

- คำสั่งที่ตามด้วย boolean expression ที่โปรแกรมเมอร์มั่นใจว่าต้องเป็นจริง
- ถ้าเกิดเท็จขณะทำงาน จะเกิด AssertionError
- disable การตรวจสอบได้
- โดยทั่วไป enable ตอนพัฒนาและทดสอบ เมื่อมั่นใจว่าปลอดภัย ก็ disable ตอนใช้จริง

```
javac -source 1.4 Test.java

java -ea Test
```

ใช้ได้เฉพาะรุ่น 1.4 เป็นต้นไป

Assertion : มีไปทำไม ?

- เขียนให้คนเข้าใจและให้เครื่องตรวจสอบ ข้อสมมติของโปรแกรมเมอร์
- ช่วยในการหา bug
- เพิ่มความมั่นใจว่าซอฟต์แวร์ที่ใช้งานไม่มี bug
- ดีกว่า exception ตรงที่ disable การตรวจสอบได้ตอน run-time โดยไม่เสียประสิทธิการทำงานเลย

Assertion : จะเขียนไว้ที่ใด

- precondition ของ non-public methods
- postcondition ของ methods ทั่วไป
- แทน comment ในโปรแกรมด้วย assertion
- เพิ่ม assert ใน switch ที่ไม่มี default
- เพิ่ม assert ใน control flow ที่ไม่มีทางไปถึง

Assertion : Preconditions

```
private void heatReactor(int temp) {
  reactor.setHeat(temp);
  ...
}
```

มั่นใจมาก private method เขียนเอง
รับรองว่า temp ที่ส่งมาให้ไม่ร้อนเกินแน่ๆ

```
private void heatReactor(int temp) {
  assert 200 < temp && temp < 1000 : "too hot " + temp;
  reactor.setHeat(temp);
  ...
}
```

กันไว้ดีกว่าแก้ ตรวจสอบให้แน่ใจ ถ้าอยู่นอกช่วงก็ถือว่าเป็น bug

Assertion : Postconditions

```
public void add(Object v) {
  Entry t = root;
  while (t != null) {
    ...
  }
}
```

มั่นใจมาก เราเขียน put เอง ทดสอบหลายครั้งแล้ว ถูกแน่ ๆ

```
public void add(Object v) {
  Entry t = root;
  while (t != null) {
    ...
  }
  assert isBalanced() : "no longer balanced";
}
```

กันไว้ดีกว่าแก้ ตรวจสอบก่อนเสร็จ ว่ายังคงเป็น balanced tree

Assertion : แทน comment

มั่นใจมาก เขียนให้คนอ่านก็พอ

```
void goo() {
  int x, y;
  ...
  // here, x is definitely non-zero
  int m = y / x;
  ...
}
```

```
void goo() {
  int x, y;
  ...
  assert x != 0;
  int m = y / x;
  ...
}
```

มั่นใจมาก เขียนให้คนอ่าน
และให้เครื่องตรวจสอบ

Assertion : ตรงจุดที่มั่นใจว่าไปไม่ถึง

```
Icecream get(int flavor) {
  swith(flavor) {
    case VANILLA :
      ...
    case COCONUT :
      ...
    case LEMON :
      ...
  }
}
```

```
Icecream get(int flavor) {
  swith(flavor) {
    case VANILLA :
      ...
    case COCONUT :
      ...
    case LEMON :
      ...
    default :
      assert false : flavor;
  }
}
```

Assertion : ตรงจุดที่มั่นใจว่าไปไม่ถึง

```
void goo() {
  for (...) {
    ...
    if (...) return;
    ...
  }
  // never reach here
}
```

```
void goo() {
  for (...) {
    ...
    if (...) return;
    ...
  }
  assert false;
}
```

ข้อแนะนำ

- expression ของ assert ต้องไม่มี side effect
- เพิ่ม assert ขณะกำลังเขียน ไม่ใช่เพิ่มหลังเขียนเสร็จ
- โปรย assert ให้ทั่ว ๆ
- ไม่ต้องห่วงเรื่องเวลาการทำงาน เพราะ disable ได้
- Checked exception ใช้กับสิ่งผิดปกติที่คาดว่ามีสิทธิ์เกิด ถ้าเกิดแล้วมีวิธีจัดการ
- Assertion ใช้กับสิ่งที่ต้องไม่เกิด ถ้าเกิดแสดงว่าผิด
- Exception handling เพิ่ม robustness
- Assertion เพิ่ม reliability