วาจาจาวา – ตอนที่ ๔

เมท็อด

สมชาย ประสิทธิ์จูตระกูล

พารามิเตอร์ของ public method

- ตรวจก่อนใช้งาน : ผิด throw exception ทันที
- รวม constructor ด้วย
- กรณีรับพารามิเตอร์มาเก็บไว้ใช้ภายหลังก็ต้องตรวจสอบเลย

```
public Object get(int index) {
  if (index < 0 || index >= size())
    throw new IndexOutOfBoundsException("index = " + index);
  ...
}
```

```
public class Period {
  private final Date start, end;
  public Period(Date start, Date end) {
    if (start == null || end == null)
      throw new NullPointerException();
    if (start.compareTo(end) > 0)
      throw new IllegalArgumentException();
    this.start = date; this.end = end;
  }
  ...
```

ข้อกำหนดของพารามิเตอร์

- เขียนรายละเอียดใน javadoc comment เสมือนเป็น "สัญญา" ให้ผู้ใช้ปฏิบัติตาม
- ใช้ @throw ระบุ exception ที่จะโยนถ้าผิดสัญญา

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param  index index of element to return
 * @return the element at the specified position in this list
 * @throws IndexOutOfBoundsException if index is out of range
 *         (index < 0 || index >= size()).
 */
public Object get(int index) {
  if (index < 0 || index >= size())
    throw new IndexOutOfBoundsException("index = " + index);
  ...
}
...
```

พารามิเตอร์ของ private method

- ใช้ assertion (ผิดคือ bug ของเราเอง)
- รวมเมท็อดแบบ package-private ด้วย

```
private void setTemperature(int temp) {
  assert 200 < temp && temp < 1000 : temp;
  ...
}
```

Defensive Programming

- ป้องกันไม่ให้มีการเปลี่ยน state ของออปเจกต์โดยที่เราไม่รับรู้
  - พารามิเตอร์ที่เรารับไปเก็บไว้อาจเปลี่ยนแปลงได้
  - field ที่คืนให้ผู้อื่น เขาอาจนำไปเปลี่ยนแปลงได้

```
public class Period {
  private final Date start, end;
  public Period(Date start, Date end) {
    this.start = date; this.end = end;
  }
  public Date start() {
    return start;
  }
  public Date end() {
    return end;
  }
  ...
```

```
Date e = new Date(99, 12, 30);
Period p = new Period(e, e);
e.setYear(80);
...
Date s = p.start();
s.setYear(70);
```

ควรทำสำเนา mutable parameters

- constructor ต้องทำสำเนา mutable parameters
  - ทำสำเนาก่อนตรวจสอบความถูกต้อง
  - อย่าใช้ clone กับพารามิเตอร์ของคลาสที่ subclass ได้

```
public class Period {
  private final Date start, end;
  public Period(Date start, Date end) {
    this.start = new Date(start.getTime());
    this.end = new Date(end.getTime());
    if (this.start.compareTo(this.end) > 0)
      throw new IllegalArgumentException();
  }
  ...
```

ควรคืนสำเนาของ mutable fields

- อย่าคืน reference ของ internal fields
- ทำสำเนา แล้วคืน reference ของสำเนาแทน
- ใช้ clone ได้
- อย่าลืม : nonzero-length array ก็ mutable

```
public class Period {
  private final Date start, end;
  ...
  public Date start() {
    return (Date) start.clone();
  }
  public Date end() {
    return new Date(end.getTime());
  }
  ...
```

Immutable fields ปลอดภัย

- ไม่ต้อง clone immutable objects
- ไม่ต้องห่วง primitive-type fields
- รับ mutable มาแปลงเป็น immutable ก่อนเก็บ

```
public class Period {
  private final long start, end;
  public Period(Date start, Date end) {
    this.start = start.getTime();
    this.end = end.getTime();
    if (this.start > this.end)
      throw new IllegalArgumentException();
  }
  public Date start() { return new Date(start); }
  public Date end() { return new Date(end); }
  ...
}
```

Method Signatures

- ตั้งชื่อให้สื่อความหมาย คาดเดาได้ สอดคล้องกันตลอด package เลียนแบบชื่ออื่น ๆ ในวงการ
- มีเมท็อดให้บริการแต่พอควร
- พารามิเตอร์ต้องมีไม่มาก ($\leq$ 3 ตัว)
- ถ้าพารามิเตอร์มีประเภทเดียวกัน จำยาก
- ใช้ interface กำหนดประเภทของพารามิเตอร์

Overloading และ Overriding

- compiler เลือก overloaded method
  - จากประเภทของ object reference
  - เลือกตัวที่ "most specific"
- jvm เลือก overridden method
  - ที่มี signature ตามที่ compiler ได้เลือกไว้ จากคลาสของ object

```
class A {
  void f(int x, double y) {}
  void f(int x, long y) {}
  void f(int x, String s) {}
}

class B extends A {
  void f(int x, String s) {}
  void f(int x, int y) {}
  void f(long x, int y) {}
}
```

```
A a = new B();
B b = (B) a;
a.f(2, 3);
b.f(2, 3);
a.f(2, "3");
a.f(2L, 3); // error
```

การลดความสับสนของ Overloading

- ไม่ควรมีจำนวนพารามิเตอร์เท่ากัน
- ถ้าเท่า ก็ให้พารามิเตอร์ต่างกันจนเห็นได้ชัด

```
public class Container extends Component {
  Component add(Component comp) {...}
  Component add(Component comp, int index) {...}
  Component add(String name, Component comp) {...}
  void add(Component comp, Object cs) {...}
  void add(Component comp, Object cs, int index) {...}
  ...
}
```

```
public class ObjectOutputStream extends OutputStream {
  ...
  void write(byte[] buf) {...}
  void write(byte[] buf, int off, int len) {...}
  void write(int val) {...}
  void writeBoolean(boolean val) {...}
  void writeByte(int val) {...}
  void writeDouble(double val) {...}
  void writeFloat(float val) {...}
  void writeInt(int val) {...}
  ...
```

คืนอาเรย์ศูนย์ช่องดีกว่าคืน null

```
class CheeseShop {
  private List cheesesInStock = new ArrayList();
  public Cheese[] getCheeses() {
    if (cheesesInStock.size() == 0)  return null;
    return (Cheese[]) cheesesInStock.toArray();
  }
  ...
```

```
public Cheese[] getCheeses() {
  if (cheesesInStock.size() == 0)  return new Cheese[0];
  return (Cheese[]) cheesesInStock.toArray();
}
```

```
class CheeseShop {
  private List cheesesInStock = new ArrayList();
  private final static Cheese[] NO_CHEESE = new Cheese[0];
  public Cheese[] getCheeses() {
    if (cheesesInStock.size() == 0)  return NO_CHEESE;
    return (Cheese[]) cheesesInStock.toArray();
  }
  ...
```

```
public Cheese[] getCheeses() {
  return (Cheese[]) cheesesInStock.toArray(NO_CHEESE);
}
```

API ต้องมี javadoc comment

- เมท็อด
  - ทำอะไร (ถ้าเป็น overridable อาจต้องบอกว่าทำอย่างไร)
  - preconditions
    - ระบุเงื่อนไขของพารามิเตอร์ด้วย @param
    - ระบุอ้อม ๆ กรณีที่จะเปิด unchecked exception ผ่าน @throws
  - side effects
  - ประเด็นเรื่อง thread-safe
- ข้อความจนถึงเครื่องหมายจุดตัวแรกคือสรุป
  - เขียนสรุป method และ constructor เป็น verb phrase
  - เขียนสรุป class, interface และ field เป็น noun phrase

Javadoc comment

```
/**
 * Linked list implementation of the <tt>List</tt> interface.
 * Implements all optional list operations, and permits all elements
 * (including <tt>null</tt>). * ...
 */
public class LinkedList extends AbstractSequentialList {

 /**
  * Appends all of the elements in the specified collection to the
  * end of this list, in the order that they are returned by the
  * specified collection's iterator. The behavior of this operation
  * is undefined if the specified collection is modified while the
  * operation is in progress.  ...
  *
  * @param c the elements to be inserted into this list.
  * @return <tt>true</tt> if this list changed as a result of the call
  * @throws NullPointerException if the specified collection is null.
  */
 public boolean addAll(Collection c) {
   return addAll(size, c);
 }
 ...
```