Effective Java 物件篇

Writing programs that are clear, correct, usable, robust, flexible, and maintainable

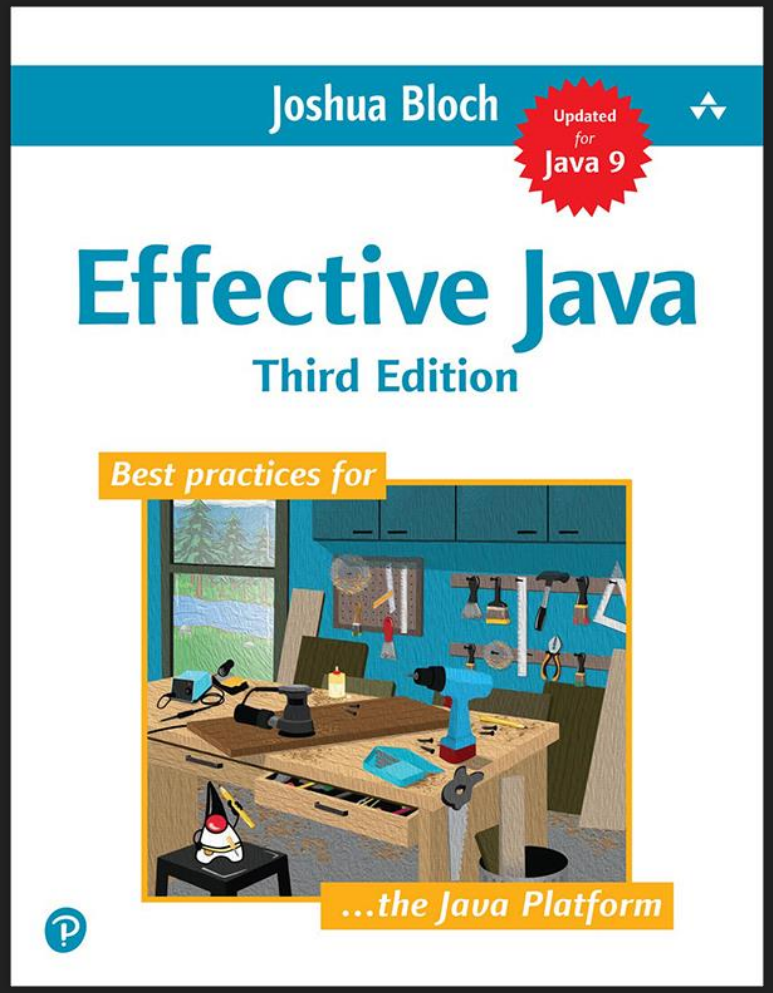
Chou, Peiyuan

周培源

Outline

- 前言
- 建立與摧毀物件(9條)
- Objects通用方法(5條)

前言


Joshua Bloch
Updated for Java 9
Effective Java
Third Edition
Best practices for
...the Java Platform

建立與摧毀物件(9條)

- 考慮使用static factory method而非只用Constructor
- Constructor參數過多時，考慮使用builder
- Singleton可以使用private constructor或Enum的方式
- 不希望讓人起始物件可以使用private constructor
- Dependency Injection優於Hardwiring Resource
- 避免建立不需要的物件
- 刪除過時的物件參照
- 避免使用finalizers與cleaners(Java 9)
- 使用try with resources優於try…finally

考慮使用static factory method而非只用Constructor

- 優點：
  - 可以有自己的方法名
  - 不需要每次呼叫都建立一個新的instance
  - 可以回傳子類別，不一定都要回傳本來的類別
  - 根據不同參數回傳不同的子類別
  - 回傳的物件不一定要在你寫方法的當下就存在
- 缺點：
  - 只有static factories方法而無public或protected constructor的類別無法被繼承
  - 不好找到
    - 命名建議：from, of, valueOf, getInstance, create, newInstance, getXXX, newXXX, {type}

Constructor參數過多時，考慮使用builder

- 列舉法
- JavaBeans模式
- Builder模式

Singleton可以使用private constructor或Enum的方式

- public final field method
- static factory method
- single-element enum method

不希望讓人起始物件可以使用private constructor

- private constructor

Dependency Injection優於Hardwiring Resource

- Spell checker example
- Pass the resource into the constructor when creating a new instance

避免建立不需要的物件

- Prefer primitives to boxed primitives
- keySet() in Map
- Don't create a new object when you should reuse an existing one
- String.matches

刪除過時的物件參照

- Memory Leak?
- Whenever a class manages its own memory, the programmer should be alert for memory leaks.

避免使用finalizers與cleaners(Java 9)

- finalize is not final block!
- Never do anything time-critical in a finalizer or cleaner
- finalizer attacks
- Implements AutoCloseable
- legitimate use:
  - Safety net
  - Native object

使用try with resources優於try…finally

- Multi-resources
- Exception tracking

Objects通用方法(5條)

- Overriding equals注意事項
- Override hashCode注意事項
- Override toString
- Override Clone ?
- 使用Comparable

Overriding equals注意事項

- equivalence relation
  - Reflexive: x=x
  - Symmetric: x=y則y=x
  - Transitive: x=y, y=z則x=z
  - Consistent: x=y 則只要x,y沒變動，x=y
- Non-nullity
- High quality equals():
  - == operator確認是否是自己
  - instanceof確認是否是正確type
  - Cast為正確型別
  - 重要的欄位要確認是否相同
- Always override hashCode when you override equals.

Override hashCode注意事項

- hashCode與equals的規則
- 建議寫法(result*質數 + hashCode => loop)
  - boolean => f ? 0 : 1
  - byte, char, short, int => (int)f
  - long => (int)(f ^ (f >>> 32))
  - float => Float.floatToIntBits(f)
  - double => Double.doubleToLongBits(f)
  - Object => hashCode()
  - array => recursive calculate each element
- Or just use java.util.Objects.hash(...) if not performance critical

Override toString

- 文件要寫清楚是否有特定規格
- 不Override的話，預設就是className@HashCode

Override Clone ?

- 建議使用Copy constructor或Copy factory

```
// Copy constructor
public Yum(Yum yum) { ... };

// Copy factory
public static Yum newInstance(Yum yum) { ... };
```

使用Comparable

- Implements Comparable表示支援natural ordering
- compareTo()規則
  - x.compareTo(y) == -y.compareTo(x)
  - x.compareTo(y) > 0 && y.compareTo(z) > 0) => x.compareTo(z) > 0
  - x.compareTo(y) == 0 => x.compareTo(z) == y.compareTo(z)
  - (x.compareTo(y) == 0) == (x.equals(y)) (非強制)
- 可使用Comparator來撰寫Comparable
- 不建議在compareTo()內使用>或<