CMSC 132:
Object-Oriented Programming II

Effective Java

Department of Computer Science
University of Maryland, College Park

Effective Java

Title
Effective Java Programming Language Guide

Author
Joshua Bloch

Contents
Learn to use Java language and its libraries more effectively

Java Puzzlers (By J. Bloch)

- Java
  - Simple and elegant
  - Need to avoid some sharp corners!
- Puzzlers
  - Java code fragments
  - Expose some tricky aspects of Java
- Effective Java
  - Patterns and idioms to emulate
  - Pitfalls to avoid

What's In A Name?

public class Name {
  private String myName;
  public Name(String n) { myName = n; }
  public boolean equals(Object o) {
    if (!(o instanceof Name)) return false;
    Name n = (Name)o;
    return myName.equals(n.myName);
  }
  public static void main(String[] args) {
    Set s = new HashSet();
    s.add(new Name("Donald"));
    System.out.println(s.contains(new Name("Donald")));
  }
}

Output
1. True
2. False
3. It Varies

Name class violates Java hashCode() contract.
If you override equals(), must also override hashCode()!

You're Such A Character

public class Trivial {
  public static void main(String args[]) {
    System.out.print("H" + "a");
    System.out.print('H' + 'a');
  }
}

Output
1. Ha
2. HaHa
3. Neither

'H' + 'a' evaluated as int, then converted to String!
Use string concatenation (+) with care. At least one operand must be a String

The Confusing Constructor

public class Confusing {
  public Confusing(Object o) {
    System.out.println("Object");
  }
  public Confusing(double[] dArray) {
    System.out.println("double array");
  }
  public static void main(String args[]) {
    new Confusing(null);
  }
}

Output
1. Object
2. double array
3. Neither

When multiple overloading applies, the most specific wins
Avoid overloading. If you overload, avoid ambiguity
Time For A Change

Problem
- If you pay $2.00 for a gasket that costs $1.10, how much change do you get?

```java
public class Change {
    public static void main(String[] args) {
        System.out.println(2.00 - 1.10);
    }
}
```

Output
1. 0.9
2. 0.90
3. Neither

Avoid float or double where exact answers are required. Use BigDecimal, int, or long instead.

Effective Java Topics
1. Creating and Destroying Objects
2. Methods Common to All Objects
3. Classes and Interfaces
4. Substitutes for C Constructs
5. Methods
6. General Programming
7. Exceptions
8. Threads
9. Serialization

Creating and Destroying Objects
- Consider providing static factory methods instead of constructors
- Enforce singleton property with a private constructor
- Enforce noninstantiability with a private constructor
- Avoid creating duplicate objects
- Eliminate obsolete object references
- Avoid finalizers

A Private Matter

class Base {
    public String name = "Base";
}
class Derived extends Base {
    private String name = "Derived";
}
class PrivateMatter {
    public static void main(String[] args) {
        System.out.println(new Derived().name);
    }
}

Output
1. Derived
2. Base
3. Neither

Compiler error in class PrivateMatter: Can’t access name
Private field can hide public. Avoid hiding & public fields

Methods Common to All Objects
- Obey the general contract when overriding equals
- Always override hashCode when you override equals
- Always override toString
- Override clone judiciously
- Consider implementing Comparable

Classes and Interfaces
- Minimize the accessibility of classes and members
- Favor immutability
- Favor composition over inheritance
- Design and document for inheritance or else prohibit it
- Prefer interfaces to abstract classes
- Use interfaces only to define types
- Favor static member classes over nonstatic
Methods
- Check parameters for validity
- Make defensive copies when needed
- Design method signatures carefully
- Use overloading judiciously
- Return zero-length arrays, not nulls
- Write doc comments for all exposed API elements

General Programming
- Minimize the scope of local variables
- Know and use the libraries
- Avoid float and double if exact answers are required
- Avoid strings where other types are more appropriate
- Beware the performance of string concatenation
- Refer to objects by their interfaces
- Prefer interfaces to reflection
- Use native methods judiciously
- Optimize judiciously
- Adhere to generally accepted naming conventions

Exceptions
- Use exceptions only for exceptional conditions
- Use checked exceptions for recoverable conditions and run-time exceptions for programming errors
- Avoid unnecessary use of checked exceptions
- Favor the use of standard exceptions
- Throw exceptions appropriate to the abstraction
- Document all exceptions thrown by each method
- Include failure-capture information in detail messages
- Strive for failure atomicity
- Don't ignore exceptions

Threads
- Synchronize access to shared mutable data
- Avoid excessive synchronization
- Never invoke wait outside a loop
- Don't depend on the thread scheduler
- Document thread safety
- Avoid thread groups