DEPARTMENT OF
COMPUTER SCIENCE

CMSC 132

Object-Oriented Programming II

Spring 2025

Week 14 Lecture 2 **Effective Java and Review**

UNIVERSITY OF
MARYLAND

FEARLESSLY
FORWARD

Announcements

Project 9 Graphs final submission is Sunday May 11th

Quiz 6 Make-up is *Monday May 12* in IRB 1108

Final Exam Review Guide

TAs will be holding a final exam review session next Wednesday (May 14th) from 4:00-7:00 PM at IRB 0324. We will go over all major topics covered in this course. If you plan on coming, please fill out this form so we know which topics to focus on.

CMSC132 Course Evaluation

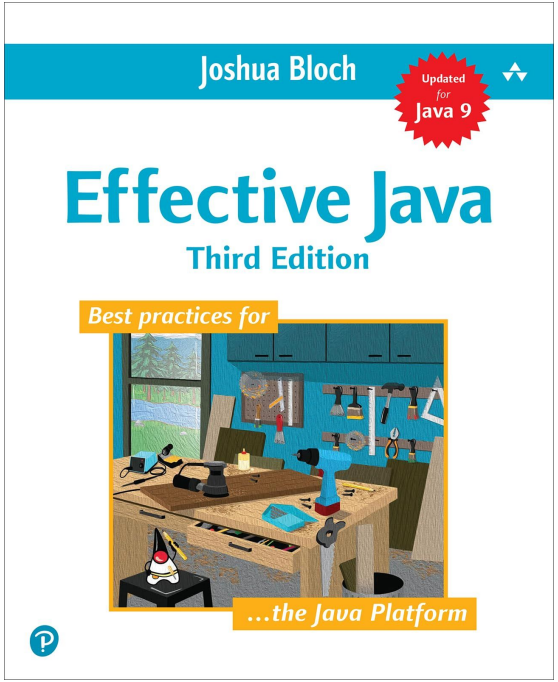
- Please take a few minutes to complete the course evaluation.
- All feedback is read and appreciated

Log in to Student Feedback on Course Experiences: https://CourseExp.umd.edu

Effective Java

TextBook: Effective Java

Effective Java Third Edition
Author: Joshua Bloch



Contents include:

- Learn to use the Java language and its libraries more effectively
- 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")));
} }
```

What is the output?

a. True
b. False
c. It varies

You're Such a Character

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

What is the output?

a. Ha
b. HaHa
c. Neither

Time for a Change

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

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

What is the output?

a. 0.9
b. 0.90
c. Neither

High Level Class Design

Classes and Interfaces

- Minimize the accessibility of classes and members
- Favor immutability
- Favor composition over inheritance
- Prefer interfaces to abstract classes
- Always override toString
  - Make your class more pleasant to use and makes systems using the class easier to debug

Classes and Interfaces

- Consider implementing Comparable for a class
  - You class will interoperate with all of the many generic algorithms and collection implementations available
- A file should store a single top-level class
  - You can have multiple top level class if only one (or none) are public
- Prefer lambdas to anonymous classes
  - Omit the types of lambda parameters unless their presence improves program's clarity
- Use a standard functional interfaces when possible (instead of a purpose-built one)

Methods

- Check parameters for validity
- Make defensive copies when needed
- Use overloading judiciously (use good judgement)
- Return zero-length arrays, not nulls
- Write doc comments for all exposed API elements
- Prefer alternatives to Java Serialization
    - Other mechanisms exist that avoid the dangers associated with Java serialization

General Programming

- Minimize the scope of local variables
  - Declare them close to where they are used
- Prefer for-each loops to traditional for loops
- For loops over while loops if the iteration variable will not be used after the loop is over
- Know and use the libraries
  - Every programmer should be familiar with java.lang, java.util, java.io

General Programming

- Prefer primitive types to boxed primitives
- Avoid float and double if exact answers are required
- Beware the performance of string concatenation
- Adhere to generally accepted naming conventions
- Refer to objects by their interfaces

Exceptions

- Use exceptions only for exceptional conditions
- Use checked exceptions for recoverable conditions and run-time exceptions for programming errors
- Favor the use of standard exceptions
- Throw exceptions appropriate to the abstraction
- Document all exceptions thrown by each method
- Don't ignore exceptions (e.g., empty catch clauses)

Generics

- Don’t use raw types
  - E.g., raw type for List<E> is List
- Prefer lists to arrays
- Favor generic types and methods
  - Define classes and methods using generics when possible
- Use bounded wildcards to increase API flexibility

Avoid Duplicate Object Creation

- Reuse existing object instead
  - Reuse improves clarity and performance
- Simplest example
  - String s = new String("DON’T DO THIS!");
  - String s = "Do this instead";
  - Why? Since Strings constants are reused
- In loops, savings can be substantial
- But don't be afraid to create objects
  - Object creation is cheap on modern JVMs

Object Duplication Example

```
public class Person {
   private final Date birthDate;
   public Person(Date birthDate){
      this.birthDate = birthDate;
   }
   // UNNECESSARY OBJECT CREATION
  public boolean bornBefore2000(){
  Calendar gmtCal = Calendar.getInstance(
                    TimeZone.getTimeZone("GMT"));
      gmtCal.set(2000,Calendar.JANUARY,1,0,0,0);
      Date MILLENIUM = gmtCal.getTime();
      return birthDate.before(MILLENIUM);
   }
```


Object Duplication Example

```
public class Person {
    …
    // STATIC INITIALIZATION CREATES OBJECT ONCE
    private static final Date MILLENIUM;
  static {
  Calendar gmtCal = Calendar.getInstance(
                    TimeZone.getTimeZone("GMT"));
  gmtCal.set(2000,Calendar.JANUARY,1,0,0,0);
  Date MILLENIUM = gmtCal.getTime();
  }
  public boolean bornBefore2000(){ // FASTER!
     return birthDate.before(MILLENIUM);
  }
}
```


Immutable Classes

Immutable Classes

- Classes whose instance cannot be modified
- Examples:
  - String
  - Integer
  - BigInteger

How to Write an Immutable Class

- Do not provide any mutator methods (no set methods)
- Ensure that no methods may be overwritten (define the call as final)
- Make all fields final
- Make all fields private
- Ensure exclusive access to any mutable components

Immutable Class Example

```
public final class Fval {
   private final float f;
   public Fval(float f) {
       this.f = f;
   }
   // ACCESSORS WITHOUT CORRESPONDING MUTATORS
   public float value( ) { return f; }

   // ALL OPERATIONS RETURN NEW Fval
   public Fval add(Fval x) {
       return new Fval(f + x.f);
   }
   // SUBTRACT, MULTIPLY, ETC. SIMILAR TO ADD
```

Immutable Class Example

```
public boolean equals(Object o) {
     if (o == this) return true;
     if (!(o instanceof Fval))
        return false;
     Fval c = (Fval) o;
     return (Float.floatToIntBits(f) ==
                        Float.floatToIntBits(c.f));
  }
```

Immutable Class Advantages

Simplicity

- Instances have exactly one state
- Constructors establish invariants
- Invariants can never be corrupted

Immutable Class Advantages

Inherently Thread Safe

- No need for synchronization
  - Internal or external
  - Since no writes to shared data
- Cannot be corrupted by concurrent access
- By far the easiest approach to thread safety

Immutable Class Advantages

Can be shared freely

```
// EXPORTED CONSTANTS
public static final Fval ZERO = new Fval(0);
public static final Fval ONE  = new Fval(1);
// STATIC FACTORY CAN CACHE COMMON VALUES
public static Fval valueOf(float f) { ...}
// PRIVATE CONSTRUCTOR MAKES FACTORY MANDATORY
private Fval (float f) {
   this.f = f;
}
```

Immutable Class Advantages

No Copies

- No need for defensive copies
- No need for any copies at all
- No need for clone or copy constructor
- Not well understood in the early days
  - public String(String s); // Should not exist

Immutable Class Advantages

Composability

- Excellent building blocks
- Easier to maintain invariants
- If component objects won't change

Immutable Class Disadvantages

- Separate instance for each distinct value
- Creating these instances can be costly
  - BigInteger moby = ...; // A million bits
  - moby = moby.flipBit(0); // Ouch!
- Problem magnified for multistep operations
  - Provide common multistep operations as primitives
  - Alternatively, provide mutable companion class

Immutable Class

When to make a class Immutable?

- Always, unless there's a good reason not to
- Always make small “value classes” immutable
  - Examples
    - Color
    - PhoneNumber
    - Price
- Date and Point (both mutable) were mistakes!

Immutable Class

When to make a class Mutable?

- Class represents entity whose state changes
  - Real-world - BankAccount, TrafficLight
  - Abstract - Iterator, Matcher, Collection
  - Process classes - Thread, Timer
- If class must be mutable, minimize mutability
  - Constructors should fully initialize instance
  - Avoid reinitialize methods