"Always code as if the person who ends up maintaining your code is a violent psychopath who knows where you live."

Amir Raminfar
Learn About Best Practices

• Cover some principles of programming
• Common pitfalls (Related to Java)
Optimize judiciously

We follow two rules in the matter of optimization:

• Rule 1. Don’t do it.

• Rule 2 (for experts only). Don’t do it yet—that is, not until you have a perfectly clear and unoptimized solution.

—M. A. Jackson [Jackson75]
Optimize judiciously

• Strive to write good programs rather than fast ones

• Strive to avoid design decisions that limit performance.
KISS

• Keep it simple, Stupid!
• Keep it short and simple
• Keep it simple AND stupid
• Don’t over complicate the solution
• Writing less code is better.
DRY

• Don’t Repeat Yourself
Write Code for the Maintainer

• Almost any code that is worth writing is worth maintaining in the future, either by you or by someone else.

• The future you who has to maintain code often remembers as much of the code, as a complete stranger, so you might as well always write for someone else.
Single Responsibility Principle

• Personal favorite

• A component of code (e.g. class or function) should perform a single well defined task
Common Programming Anti-patterns
While vs For loops

• Which is better?
Favor For-each loop always

- Performance
- Less bugs
- Variable scopes
- Less code
StringBuilder vs String

```java
String s = "";
for (int i = 0; i < total; i++) {
    s += String.valueOf(i);
}

StringBuilder sb = new StringBuilder();
for (int i = 0; i < total; i++) {
    sb.append(String.valueOf(i));
}
```

//
StringBuilder vs String

String s = "";
for (int i = 0; i < total; i++) {
    s += String.valueOf(i);
}
// 4828ms
StringBuilder sb = new StringBuilder();
for (int i = 0; i < total; i++) {
    sb.append(String.valueOf(i));
}
// 4ms
Integer vs Int

```java
Integer sum = 0;
for (int i = 0; i < total; i++) {
    sum += 1;
}
// 8ms
```

```java
int sum = 0;
for (int i = 0; i < total; i++) {
    sum += 1;
}
// 0ms
```
Singleton

• What is a singleton?
• How do you do it in Java?
Best Way to Implement Singleton

```java
public enum Elvis {
    INSTANCE;
    public void leaveTheBuilding() { ... }
}
```
Avoid creating unnecessary objects

What’s wrong with this?
String s = new String("stringette");
//Don’t do this!
Avoid creating unnecessary objects

String s = "stringette";
Don’t return null for Collections

```java
public List<?> getList(){
    // do something
    return null;
}
```

• Don’t do this! Explain why?
public List<?> getList(){
    // do something
    return Collections.emptyList();
}
Memory Leak

// Can you spot the "memory leak"?
public class Stack {
    private Object[] elements;
    private int size = 0;

    public Object pop() {
        if (size == 0)
            throw new EmptyStackException();
        return elements[--size];
    }
}
Memory Leak

// Can you spot the "memory leak"?
public class Stack {
    private Object[] elements;
    private int size = 0;

    public Object pop() {
        if (size == 0) {
            if (size == 0)
                throw new EmptyStackException();
        }
        return elements[--size];
    }
}
Hashcode and Equals

- Always overwrite hashcode() and equals() functions together!
- Explain why?
Refer to objects by their interfaces

// Don’t do this!

class

public ArrayList<String> getString()
{
    //...

    return null;
}


Refer to objects by their interfaces

// Do this!

public List<String> getString(){
    //...
}

Building your project

- Make it easy to build your project.
- Use frameworks like -
  - Ant
  - Maven
  - Grape
Static Code Analyzers

- IntelliJ has a code analyzer
- FindBugs
Some good reads

• Some good reads: Effective Java, 2nd Ed [Bloch 2008]
• Software Paradigms [Kaisler 2005]
• Introduction To Algorithms, 3rd ed [Cormen et al. 2009]