RED HAT
DEVELOPERS

Effective and Clean Java Code? Tips and Tricks from the Real World

Edson Yanaga
Director of Developer Experience
@yanaga

Java Champion

Microsoft MVP

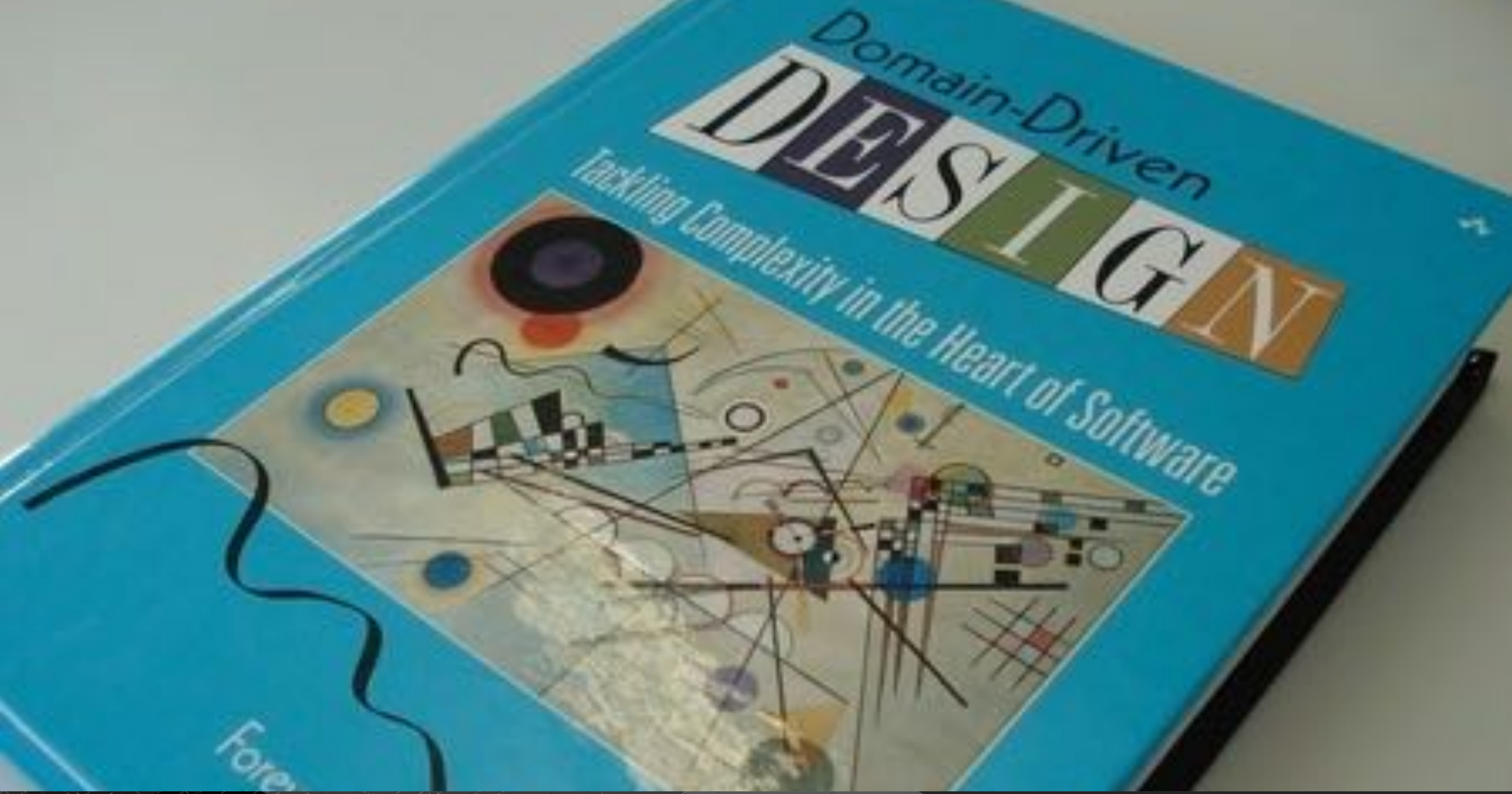
Software is a Craft

“…computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty”

–Donald E. Knuth

*“No matter what other methods you apply to achieve **competence** in a social system, in the end it all depends on whether people actually **care**.”*

–Jurgen Appelo


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software

Code Smells

Primitive Obsession

design

Editor: Martin Fowler ■ ThoughtWorks ■ fowler@acm.org

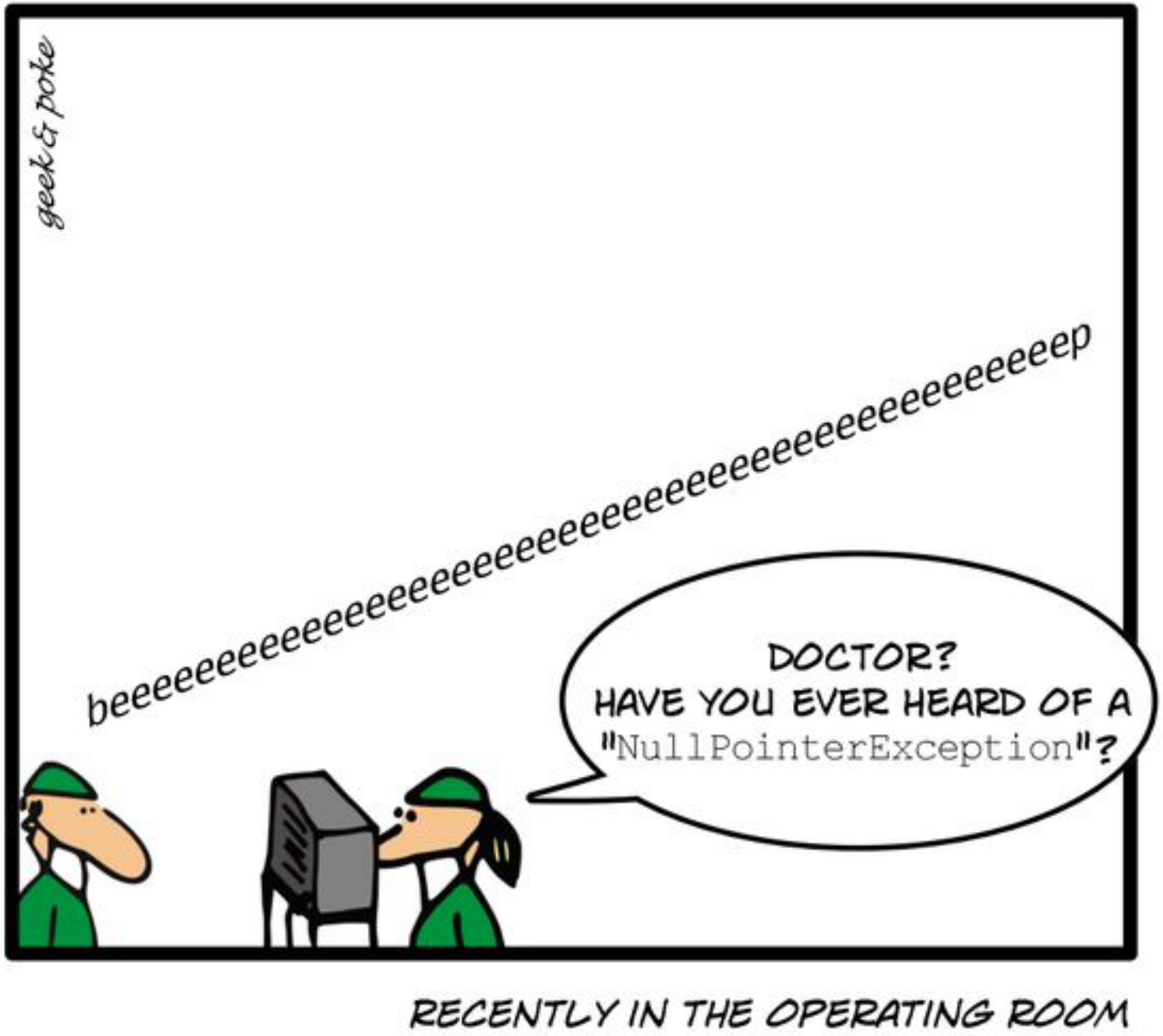
When to Make a Type

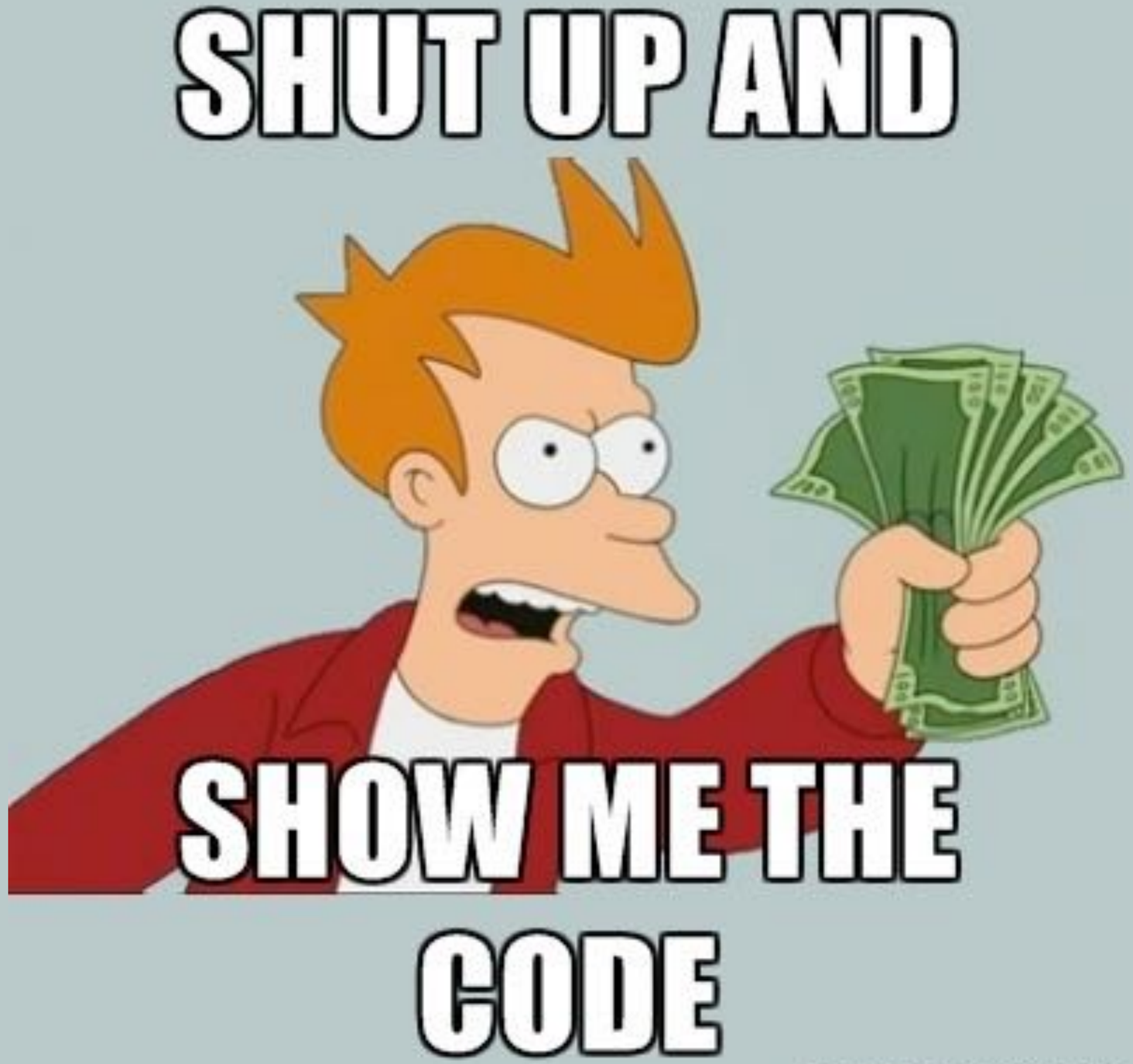
Martin Fowler

When I started programming computers, I began with fairly primitive languages, such as Fortran 4 and various early flavors of Basic. One of the first things you learn using such languages—indeed, even using more up-to-date languages—is which types your language supports. Being oriented toward number crunching, Fortran supported integer and real types, with the interesting rule that any variable whose name started with the letters I through N was an integer, and all other variables were floats. I'm glad that convention hasn't caught on, although Perl is close. Furthermore, using object-oriented languages, you can define your own types and in the best languages, they act just as well as built-in ones.

My favorite example is money. A lot of computer horsepower is dedicated to manipulating money, accounting, billing, trading, and so forth—few things burn more cycles. Despite all this attention, no mainstream language has a built-in type for money. Such a type could reduce errors by being currency aware, helping us, for example, avoid embarrassing moments of adding our dollars to our yen. It can also avoid more insidious rounding errors. It would not only remove the temptation to use floats for money (never, ever do that) but also help us deal with tricky problems such as how to split $10 equally between three people. In addition, it could simplify a lot of printing and parsing code. For more on this (why write the column if I can't plug my books?), see *Patterns of Enterprise Application Architecture* (Addison-Wesley, 2002).

The nice thing about OO programs is that you can easily define a type like this if




SHUT UP AND
SHOW ME THE
CODE
memegenerator.net

Immutables

stars 1,196

Java annotation processors to generate simple, safe and consistent value objects. Do not repeat yourself, try Immutables, the most comprehensive tool in this field!

Get started! | Read guide... | v2.3.2

News and announcements

```java
import org.immutables.value.Value;
// Define abstract value type
@Value.Immutable
public interface ValueObject {
  String name();
  List<Integer> counts();
  Optional<String> description();
}

// Or you can configure different @Value.Style
@Value.Immutable
abstract class AbstractItem {
  abstract String getName();
  abstract Set<String> getTags();
  abstract Optional<String> getDescription();
}
```

```java
// Use generated immutable implementation
ValueObject valueObject =
    ImmutableValueObject.builder()
        .name("My value")
        .addCounts(1)
        .addCounts(2)
        .build();


// Use generated value object
Item namelessItem = Item.builder()
    .setName("Nameless")
    .addTags("important", "relevant")
    .setDescription("Description provided")
    .build();

Item namedValue = namelessItem.withName("Named");
```

Values and Builders

With *Immutables* you can generate state of the art immutable objects and

Easy to use

Just add jar to classpath and use. No required runtime dependencies! Guava

Feature packed 2.2

Lazy, derived and optional attributes. Comprehensive support for collections

Clean code

Immutables has much higher standards for code readability than other

Concept in Code

A new powerful collection library saves us from source.stream().really("?").collect(sink()) pipelines.

```
List<String> names = persons
    .filter(p -> p.age > 12)
    .map(Person::getName);
```

GO FUNCTIONAL


Better Software,
Better World

Join
developers.redhat.com

Feedback welcome!
@yanaga

RED HAT DEVELOPERS

RED HAT
DEVELOPERS

Thank you!


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHatNews


youtube.com/user/RedHatVideos