Edson Yanaga
@yanaga

Applied DDD in a Java EE 7 and Open Source World

#JAVAONE

EDSON YANAGA

- Computer Science Bachelor's Degree
- Electrical Engineer Master's Degree
- Java Developers since 1997
- Unix Systems Administrator since 1999
- Professor of graduate and undergraduate courses since 2000

CERTIFICATIONS

- Oracle Certified Professional, Java Platform, Enterprise Edition 6 Enterprise JavaBeans Developer
- Sun Certified Enterprise Architect for the Java Platform,Enterprise Edition 5 (i)
- Certified ScrumMaster
- Sun Certified Developer for Java Web Services 5
- Sun Certified Specialist for NetBeans IDE
- Sun Certified Web Component Developer for J2EE 1.4
- Sun Certified Programmer for Java 2 Platform 1.4

SOFTWARE IS A CRAFT

Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Manifesto for Software Craftsmanship

Raising the bar.

As aspiring Software Craftsmen we are raising the bar of professional software development by practicing it and helping others learn the craft. Through this work we have come to value:

Not only working software,
but also **well-crafted software**

Not only responding to change,
but also **steadily adding value**

Not only individuals and interactions,
but also **a community of professionals**

Not only customer collaboration,
but also **productive partnerships**

That is, in pursuit of the items on the left we have found the items on the right to be indispensable.

WHAT HAVE YOU LEARNED ABOUT OBJECT ORIENTED ANALYSIS & DESIGN?

Inheritance

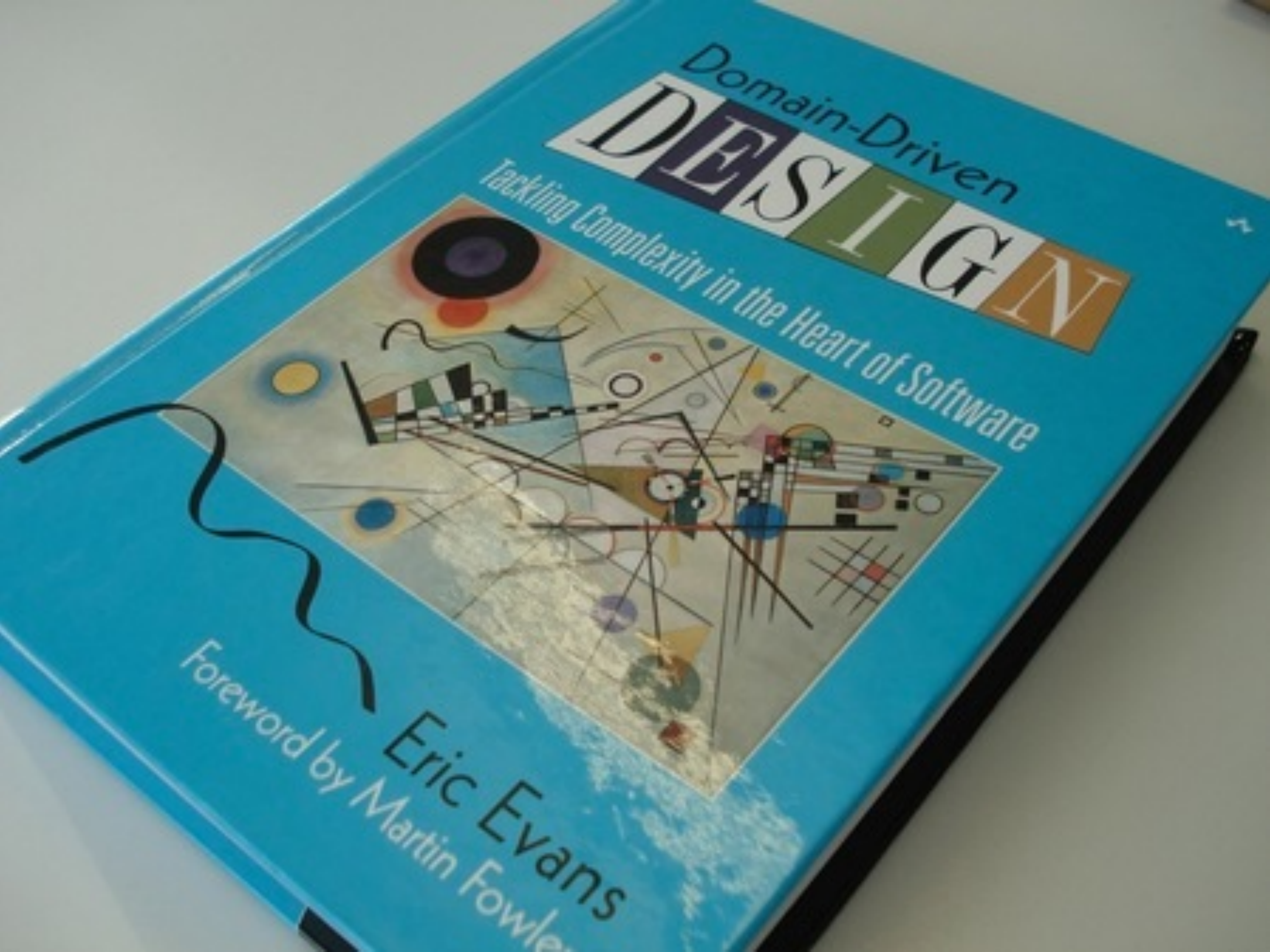
Polymorphism

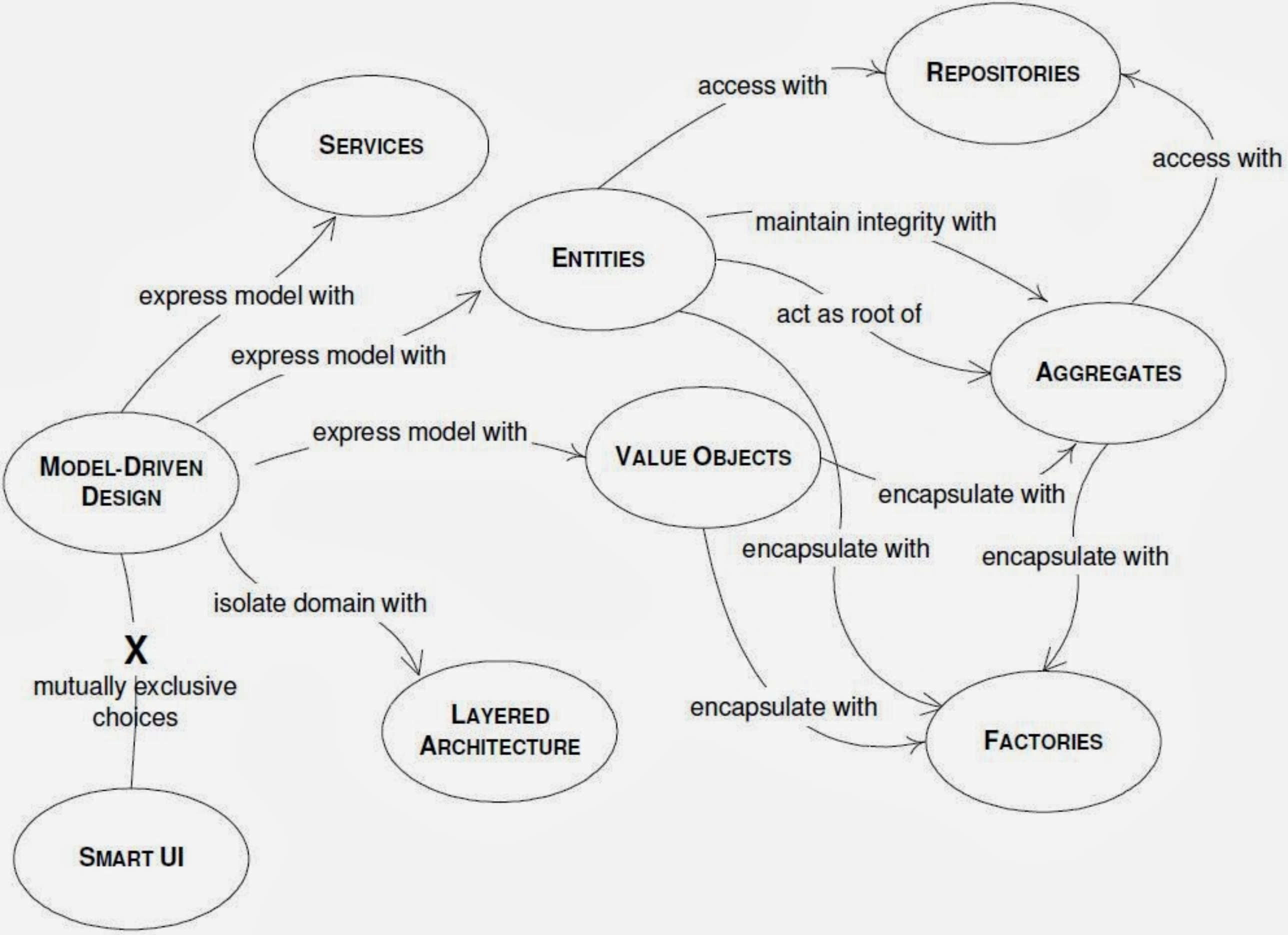
Encapsulation

Encapsulation

Polymorphism

Inheritance


Domain-Driven
DESIGN
Tackling Complexity in the Heart of Software
Eric Evans
Foreword by Martin Fowler


SERVICES
access with
REPOSITORIES
access with
express model with
ENTITIES
maintain integrity with
express model with
act as root of
AGGREGATES
express model with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
encapsulate with
encapsulate with
isolate domain with
X
mutually exclusive choices
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
SMART UI

```
public class Person {

  private String name;

  private String ssn;

  private String telephone;

  private Date birth;

}
```

CODE SMELLS

PRIMITIVE OBSESSION

design

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

When to Make a Type

Martin Fowler

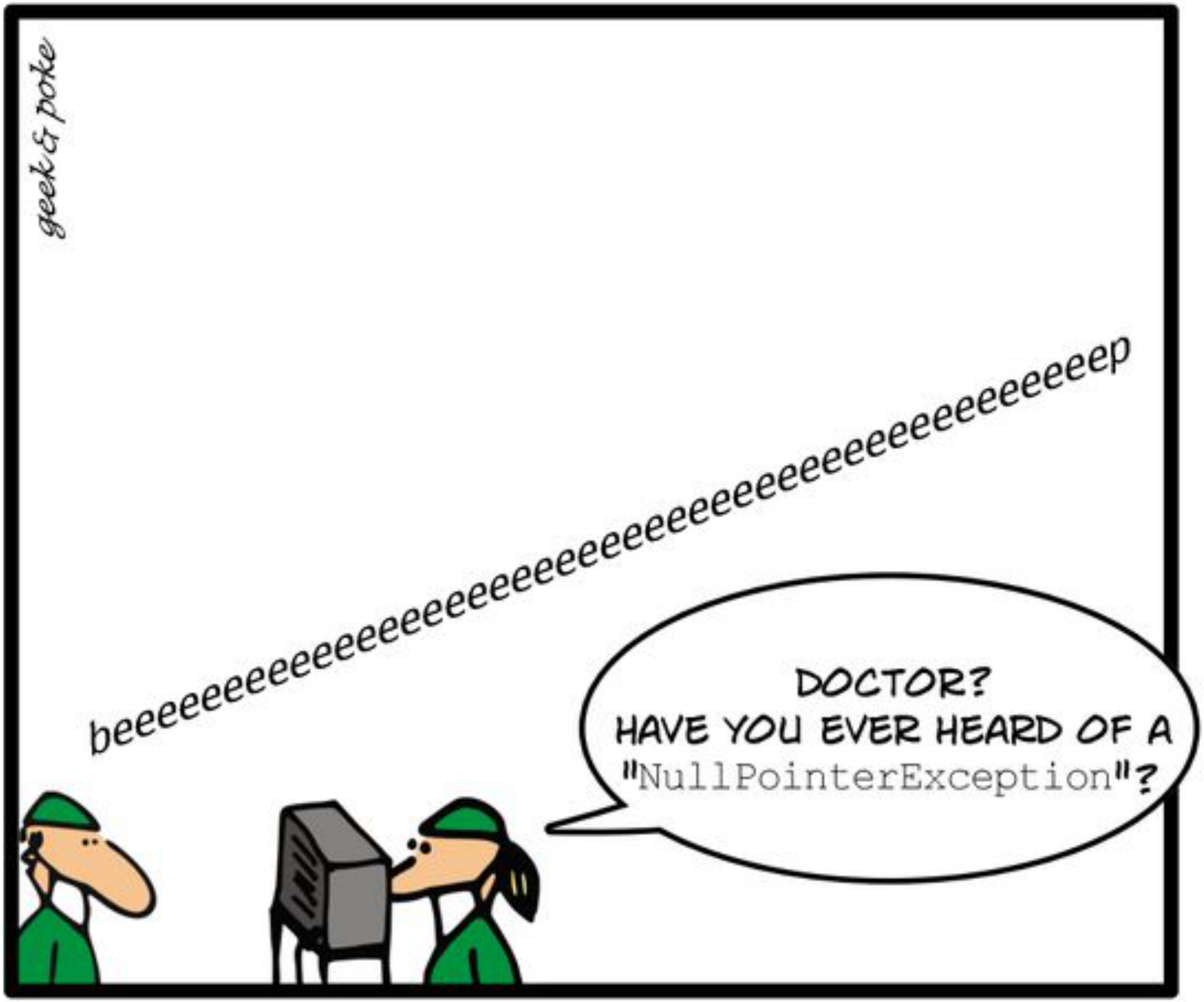
When I started programming computers, I began with fairly primitive languages, such as Fortran 4 and various early flavors of Basic. One of the first things you learn using such languages—indeed, even using more up-to-date languages—is which types your language supports. Being oriented toward number 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 the language and libraries don't include it

```
public class Person {

  private Name name;

  private Ssn ssn;

  private Telephone telephone;

  private Birth birth;

}
```

TELL, DON'T ASK



RECENTLY IN THE OPERATING ROOM

“Don’t talk to strangers”

–DEMETER LAW

Mario Fusco
@mariofusco

Follow

Mutability is the new goto

Reply Retweet Favorite ••• More

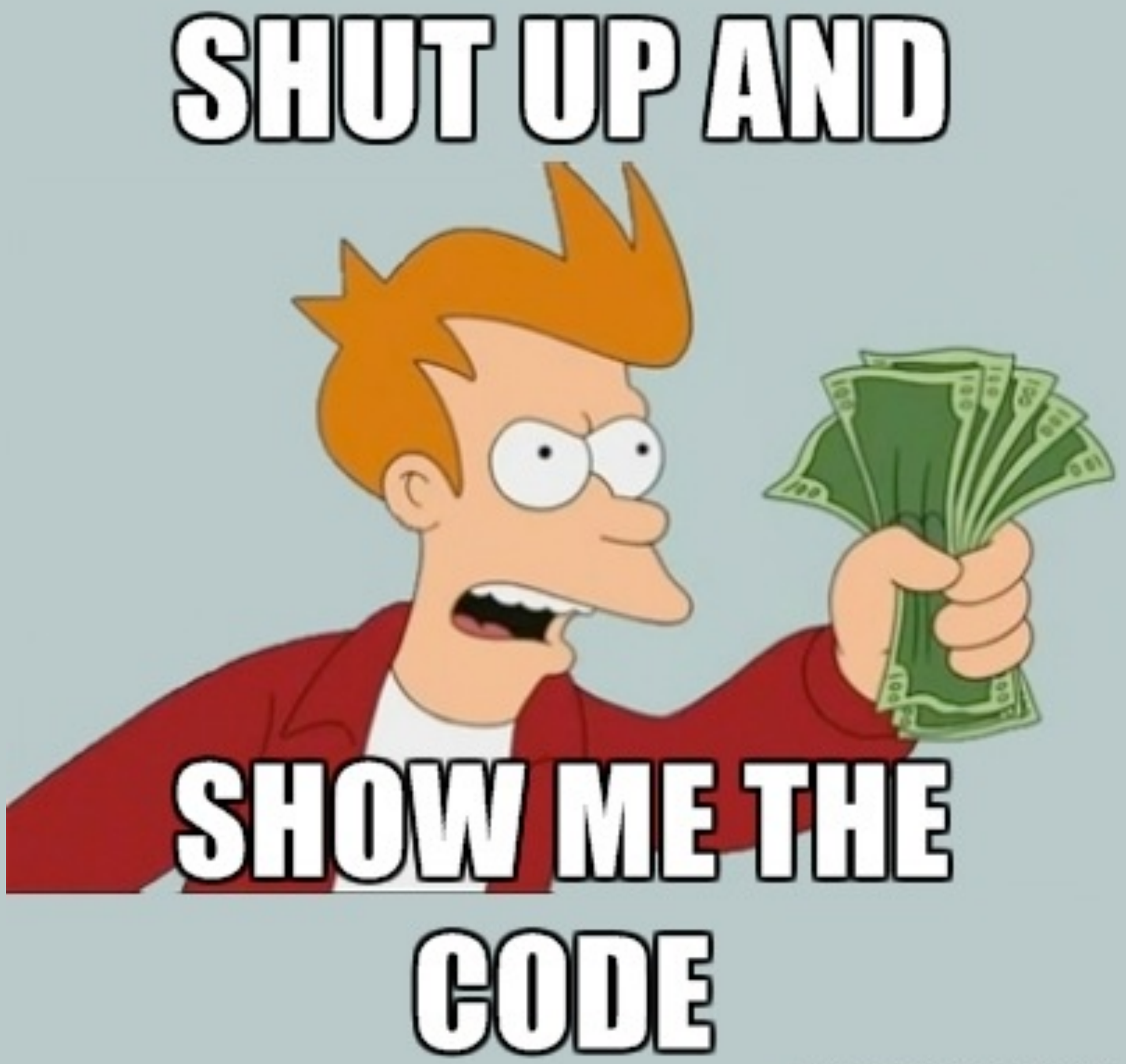
RETWEETS	FAVORITES
221	47

4:35 AM - 28 Oct 2012


QUERY
DSL

```
CriteriaQuery query = builder.createQuery();
Root<Person> men = query.from( Person.class );
Root<Person> women = query.from( Person.class );
Predicate menRestriction = builder.and(
   builder.equal( men.get( Person_.gender ), Gender.MALE ),
   builder.equal( men.get( Person_.relationshipStatus ),
RelationshipStatus.SINGLE ));
Predicate womenRestriction = builder.and(
   builder.equal( women.get( Person_.gender ), Gender.FEMALE ),
   builder.equal( women.get( Person_.relationshipStatus ),
RelationshipStatus.SINGLE ));
query.where( builder.and( menRestriction, womenRestriction ) );
```

```
JPAQuery query = new JPAQuery(em);
QPerson men = new QPerson("men");
QPerson women = new QPerson("women");
query.from(men, women).where(
men.gender.eq(Gender.MALE),
men.relationshipStatus.eq(RelationshipStatus.SINGLE),
women.gender.eq(Gender.FEMALE),
women.relationshipStatus.eq(RelationshipStatus.SINGLE));
```


SHUT UP AND
SHOW ME THE
CODE
memegenerator.net

This is everybody's fault but mine.

HOMER SIMPSON

DON'T CARE?

THEN DO IT FOR YOU!

If you think you can do a thing
or think you can't do a thing,
you're right.

–HENRY FORD


BETTER SOFTWARE,
BETTER WORLD

SOURCE CODE AVAILABLE ON:

https://github.com/yanaga/ddd-javaee7

EDSON YANAGA

edson@yanaga.com.br
@yanaga
www.yanaga.com.br