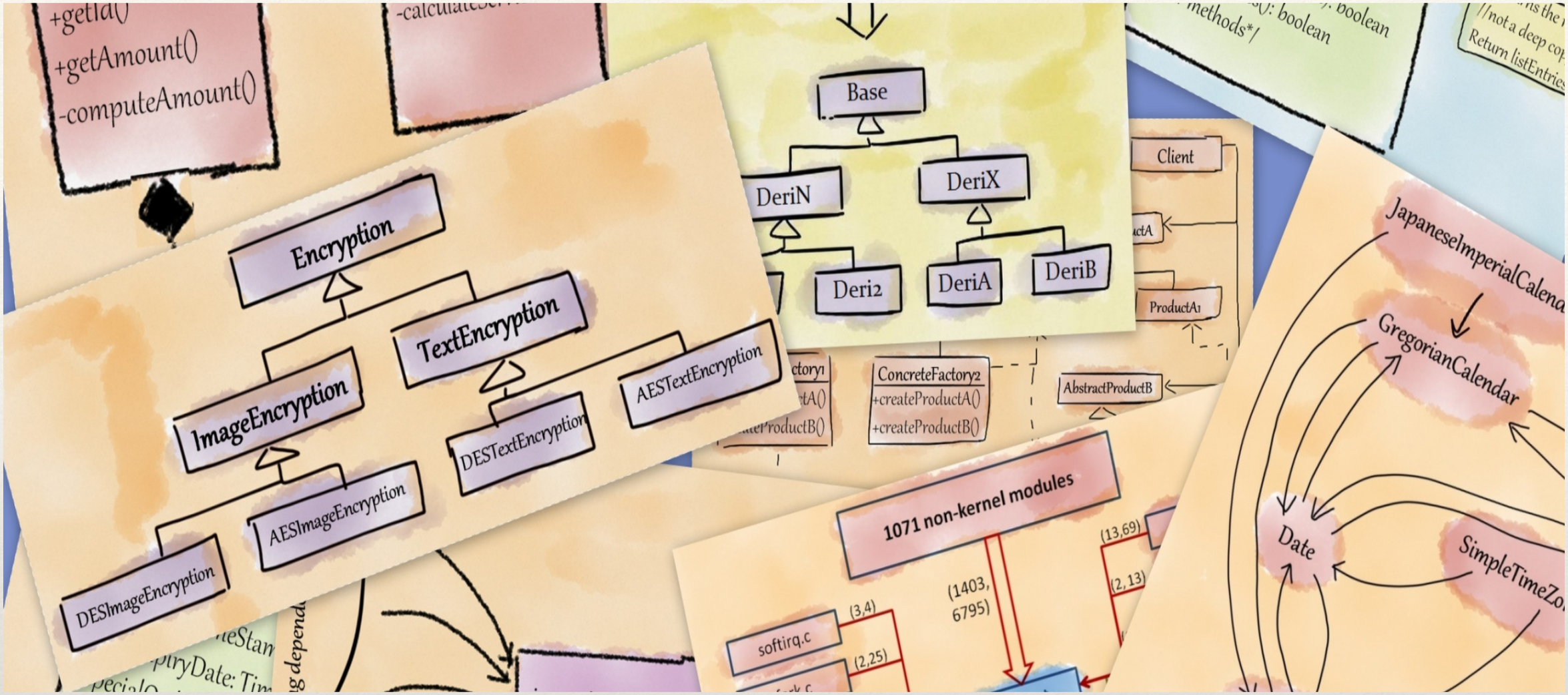


Introduction to Refactoring

Ganesh Samarthyam
ganesh.samarthyam@gmail.com

"Applying design principles is the key to creating high-quality software!"

Architectural principles:
Axis, symmetry, rhythm, datum, hierarchy, transformation

For architects: design is the key!

Agenda

- **Refactoring Foundations**
- Refactoring - Principles
- Refactoring Bad Smells
- Refactoring Tools

Why care about refactoring?

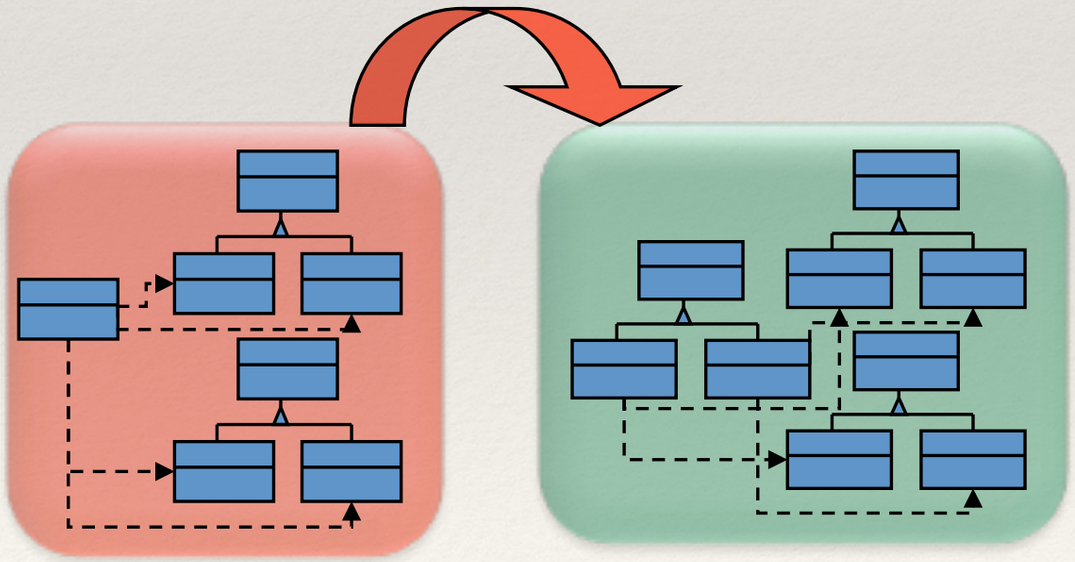
> As an evolving program is continually changed, its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it

- Lehman's law of Increasing Complexity

What is refactoring?

Refactoring (noun): a *change* made to the *internal structure* of software to make it *easier to understand and cheaper to modify* without *changing its observable behavior*

Refactor (verb): to restructure software by applying a series of refactorings without changing its observable behavior



Should this be “refactored”?

```
class Base {
      public Base() {
        foo();
      }
      public void foo() {
        System.out.println("In Base's foo ");
      }
}
class Derived extends Base {
      public Derived() {
        i = new Integer(10);
      }
      public void foo() {
        System.out.println("In Derived's foo " + i.toString() );
      }
      private Integer i;
}
class Test {
      public static void main(String [] s) {
        new Derived();
      }
}
```

Crashes by throwing a NullPointerException

It’s a bug! Fix it - its not called refactoring

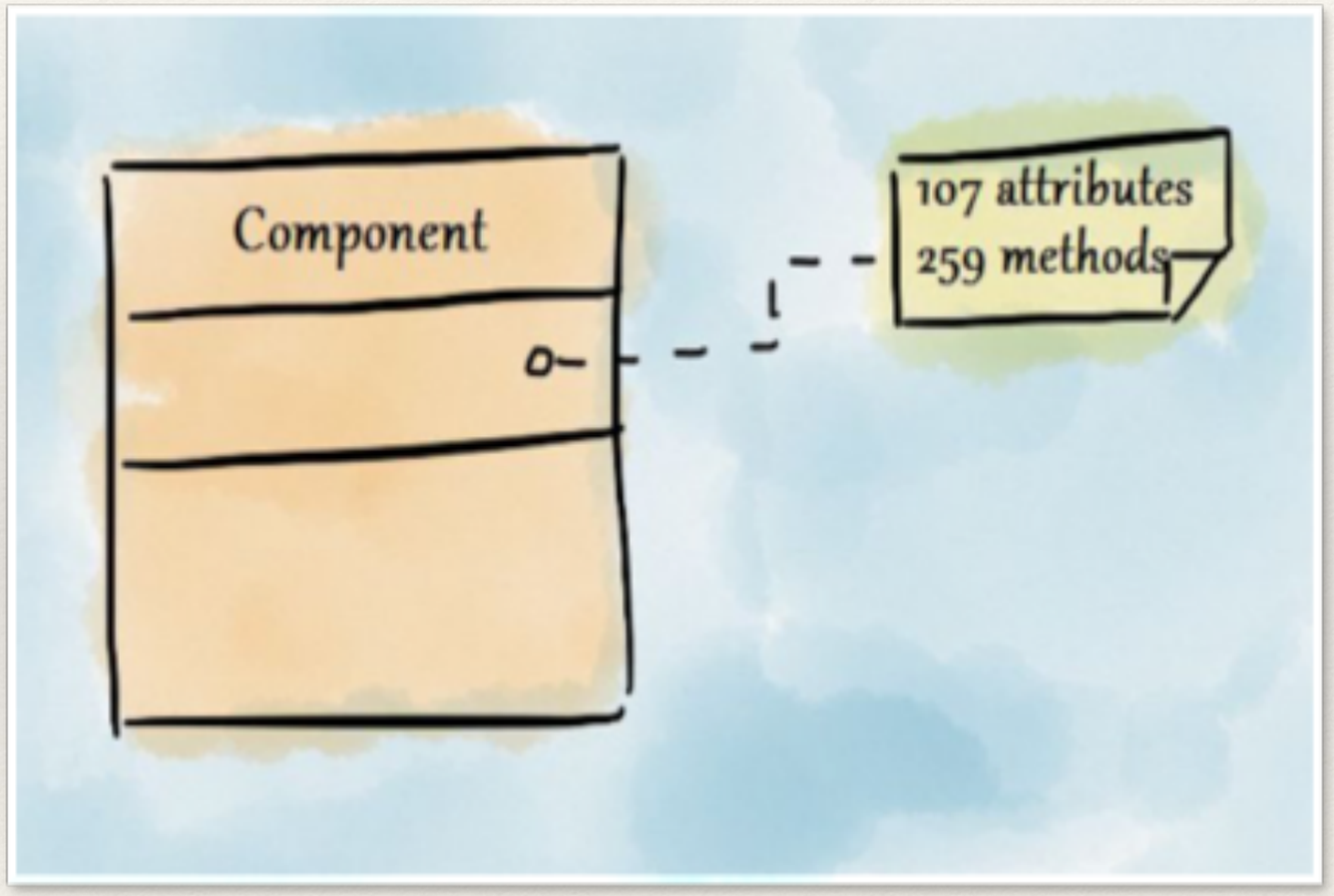
Should this be “refactored”?

Does this have “long message chains” smell?

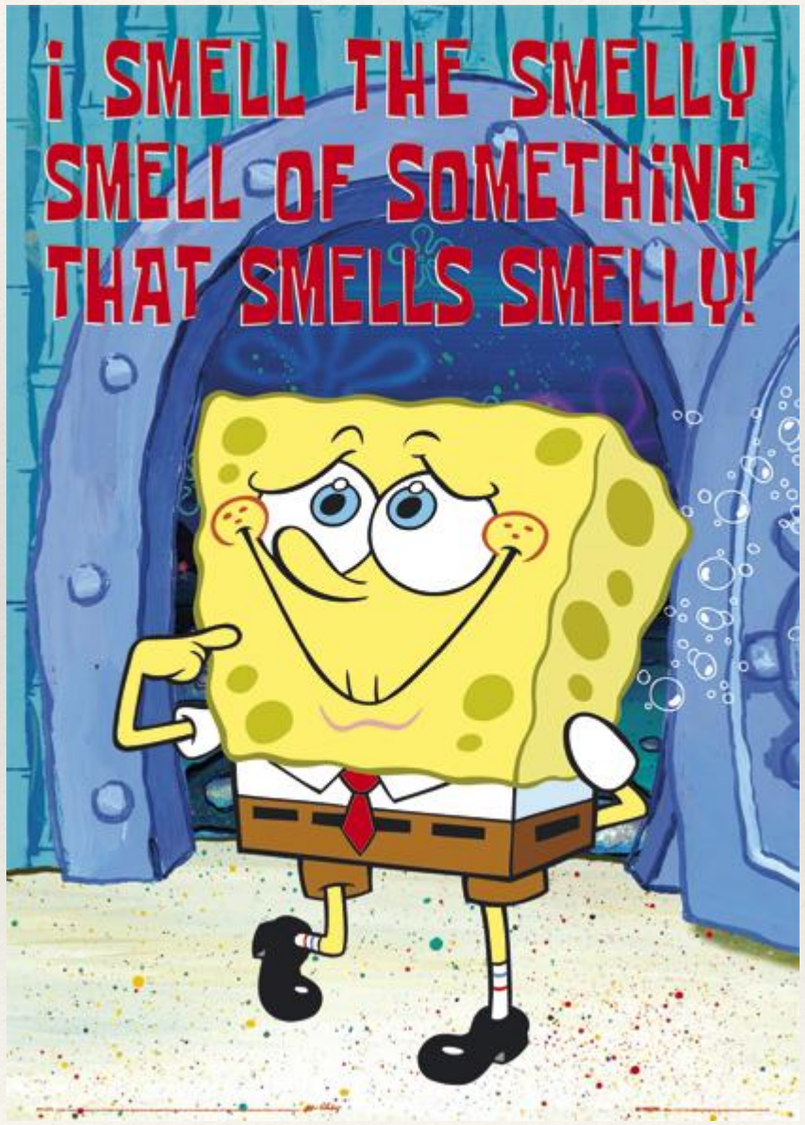
```
Calendar calendar = new Calendar.Builder()
.set(YEAR, 2003)
.set(MONTH, APRIL)
.set(DATE, 6)
.set(HOUR, 15)
.set(MINUTE, 45)
.set(SECOND, 22)
.setTimeZone(TimeZone.getDefault())
.build();
System.out.println(calendar);
```

No - it’s a feature
(use of builder pattern)!

Should this be “refactored”?



Smells approach to refactoring



What are smells?

"Smells are certain structures
in the code that suggest
(sometimes they scream for)
the possibility of refactoring."

Granularity of smells

Architectural
Cyclic dependencies between modules Monolithic modules Layering violations (back layer call, skip layer call, vertical layering, etc)
Design
God class Refused bequest Cyclic dependencies between classes
Code (implementation)
Internal duplication (clones within a class) Large method Temporary field

Agenda

- Refactoring Foundations
- **Refactoring - Principles**
- Refactoring Bad Smells
- Refactoring Tools

3 principles behind patterns

Program to an interface, not to an implementation

Favor object composition over inheritance

Encapsulate what varies

SOLID principles

Single Responsibility Principle (SRP)	• There should never be more than one reason for a class to change
Open Closed Principle (OCP)	• Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification
Liskov's Substitution Principle (LSP)	• Pointers or references to base classes must be able to use objects of derived classes without knowing it
Dependency Inversion Principle (DIP)	• Depend on abstractions, not on concretions
Interface Segregation Principle (ISP)	• Many client-specific interfaces are better than one general-purpose interface

“When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature.”

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

“Refactoring changes the programs in small steps.
If you make a mistake, it is easy to find the bug.”

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

“Don't publish interfaces prematurely. Modify your code ownership policies to smooth refactoring.”

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

"When you get a bug report, start by writing a unit test that exposes the bug."

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

“Before you start refactoring, check that you have a solid suite of tests.”

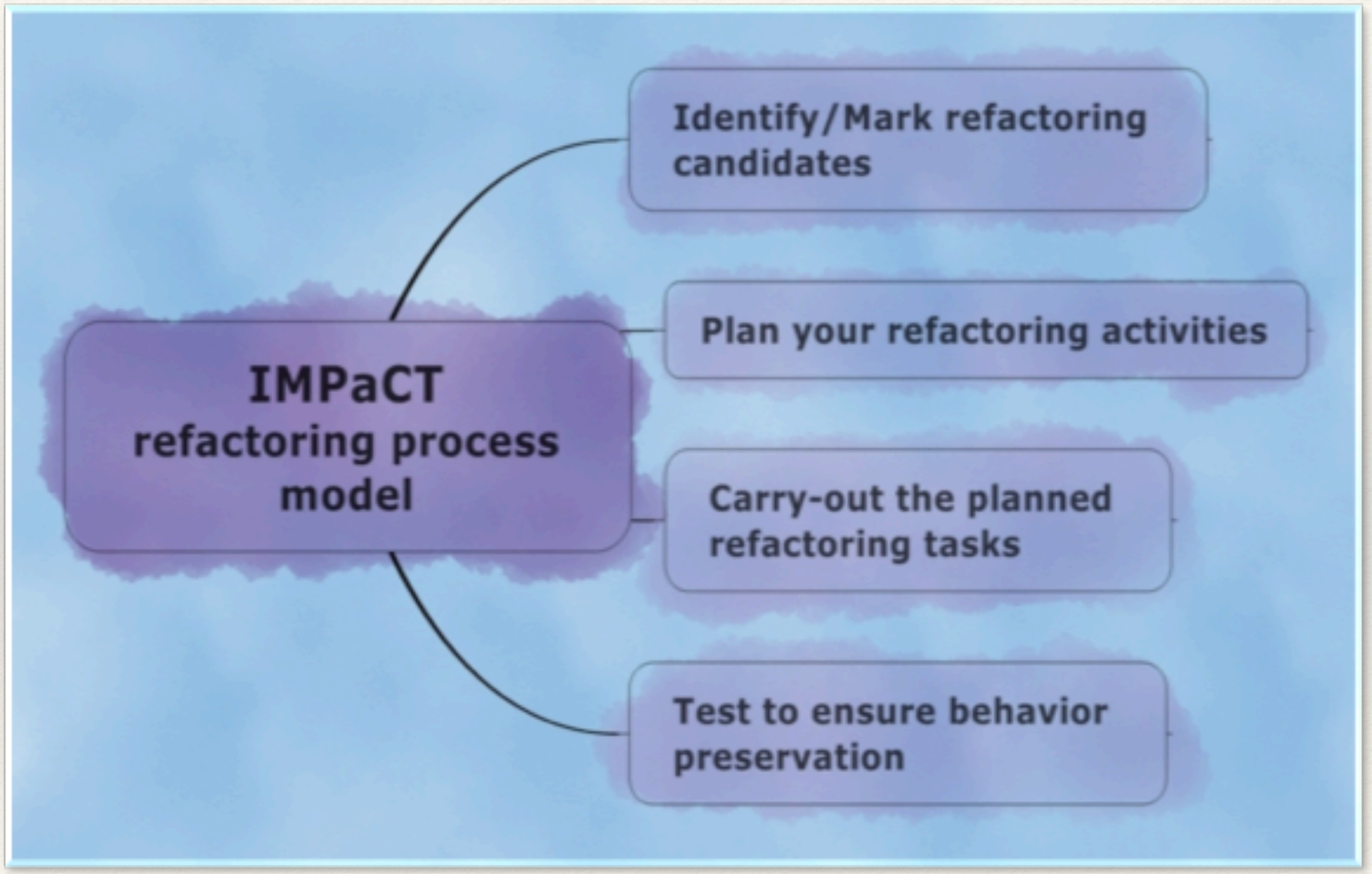
“Make sure all tests are fully automatic and that they check their own results.”

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

“It is better to write and run incomplete tests than not to run complete tests.”

Source: Refactoring, Martin Fowler, Addison Wesley, 2003

IMPaCT Process Model



Refactoring: Practical concerns

"Fear of breaking working code"

Where do I have time for refactoring?

How to refactor code in legacy projects (no automated tests, difficulty in understanding, lack of motivation, …)?

Is management buy-in necessary for refactoring?

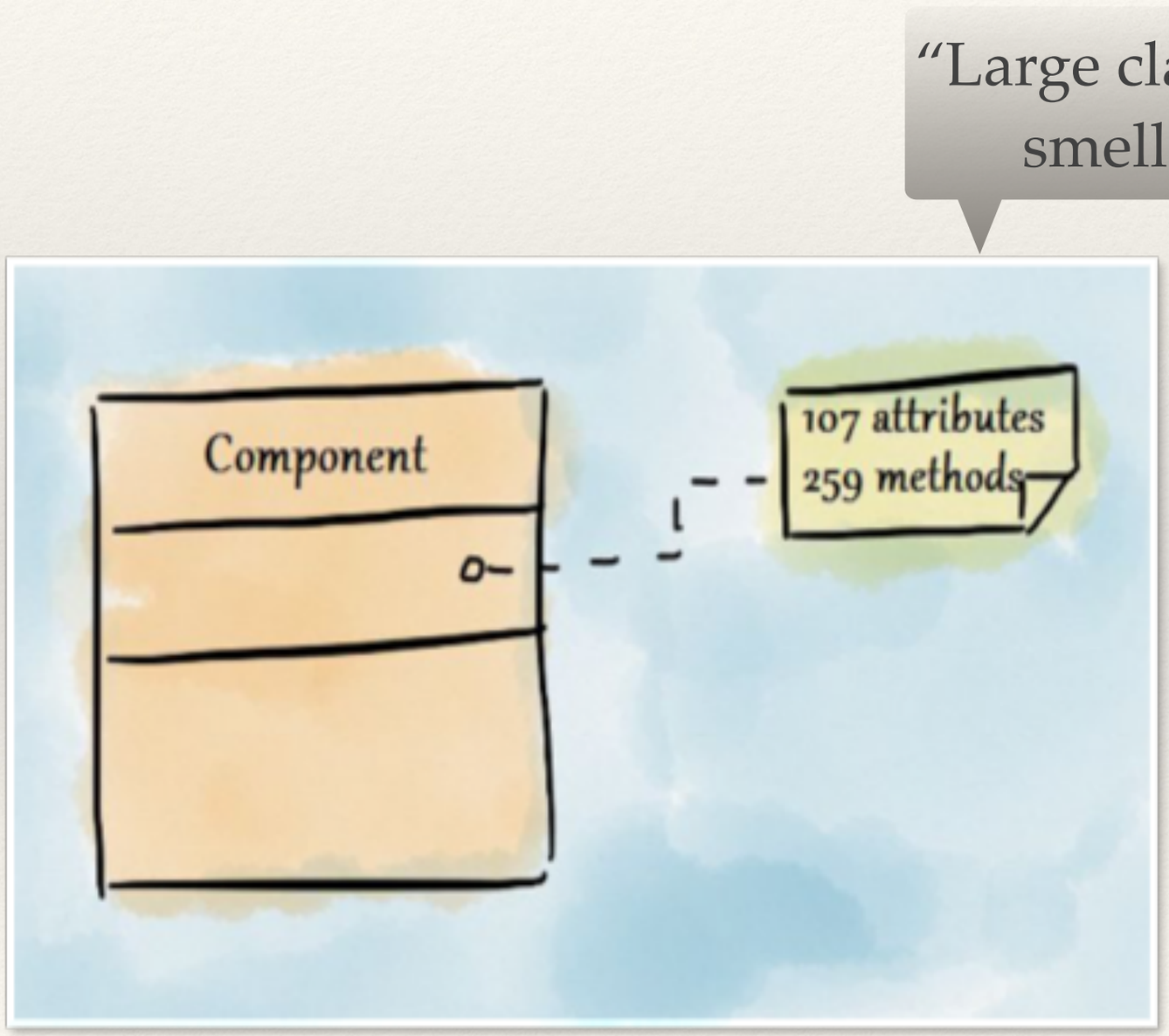
Agenda

- Refactoring Foundations
- Refactoring - Principles
- **Refactoring Bad Smells**
- Refactoring Tools

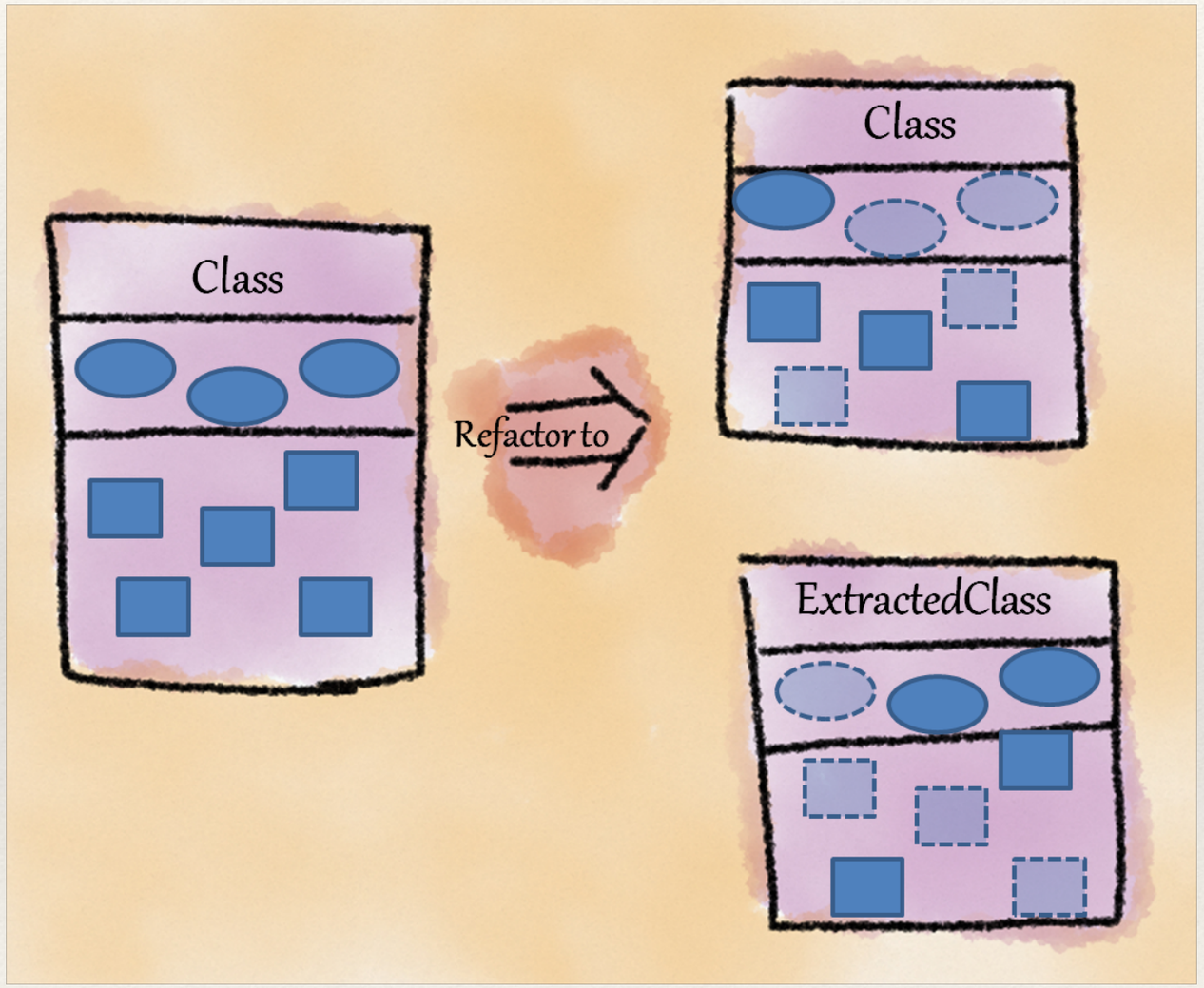
Fowler's bad smells

Alternative Classes
with Different Interfaces
Comments
Data Class
Data Clumps
Divergent Change
Duplicated Code
Feature Envy
Inappropriate Intimacy
Incomplete Library Class
Large Class
Lazy Class
Long Method
Long Parameter List
Message Chains
Middle Man
Parallel Inheritance Hierarchies
Primitive Obsession
Refused Bequest
Shotgun Surgery
Speculative Generality
Switch Statements
Temporary Field

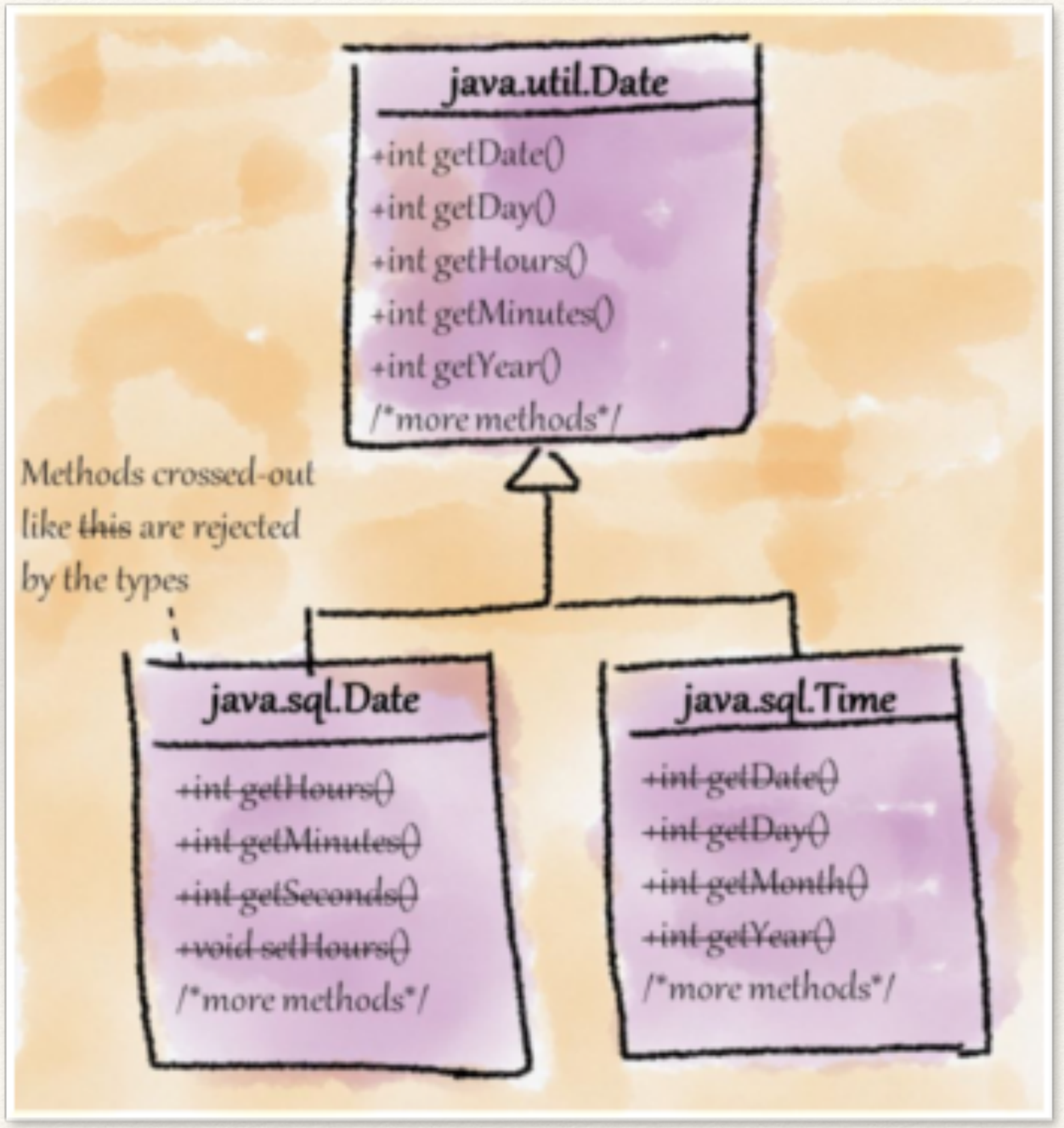
What's that smell?



Refactoring with "extract class"



What's that smell?



"Refused bequest" smell

Refused bequest smell

A class that overrides a method of a base class in such a way that the contract of the base class is not honored by the derived class

What's that smell?

class GraphicsDevice

```
public void setFullScreenWindow(Window w) {
    if (w != null) {
        if (w.getShape() != null) { w.setShape(null); }
        if (w.getOpacity() < 1.0f) { w.setOpacity(1.0f); }
        if (!w.isOpaque()) {
            Color bgColor = w.getBackground();
            bgColor = new Color(bgColor.getRed(),
                                bgColor.getGreen(),
                                bgColor.getBlue(), 255);
            w.setBackground(bgColor);
        }
    }
...
}
```

This code in GraphicsDevice uses more methods of Window than calling its own methods!

"Feature envy" smell

Feature envy smell

Methods that are more interested in the data of other classes than that of their own class

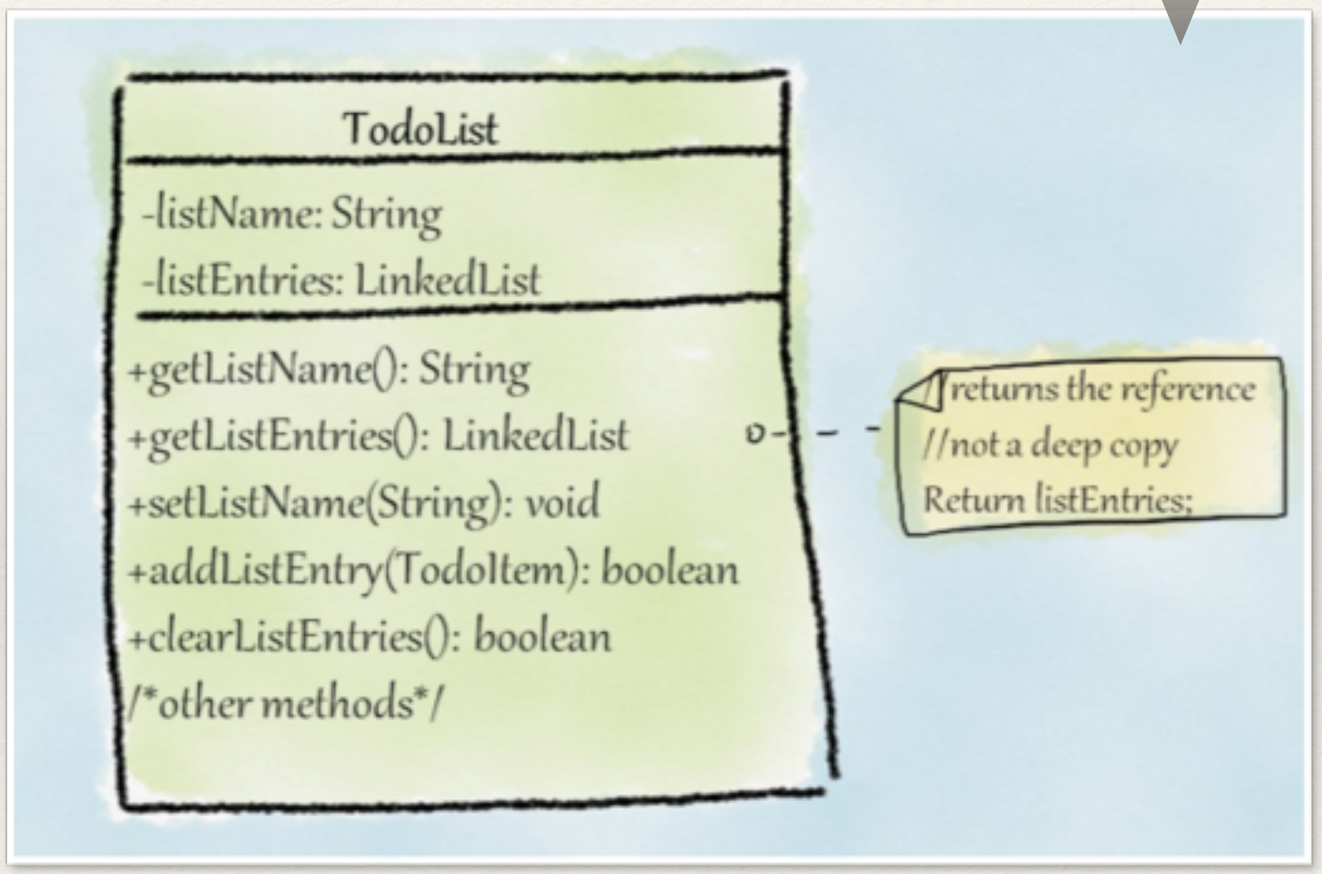
What's that smell?

"Lazy class" smell

```
public class FormattableFlags {
        // Explicit instantiation of this class is prohibited.
        private FormattableFlags() {}
        /** Left-justifies the output.  */
        public static final int LEFT_JUSTIFY = 1<<0; // '-'
        /** Converts the output to upper case */
        public static final int UPPERCASE = 1<<1;    // 'S'
        /**Requires the output to use an alternate form. */
        public static final int ALTERNATE = 1<<2;    // '#'
}
```

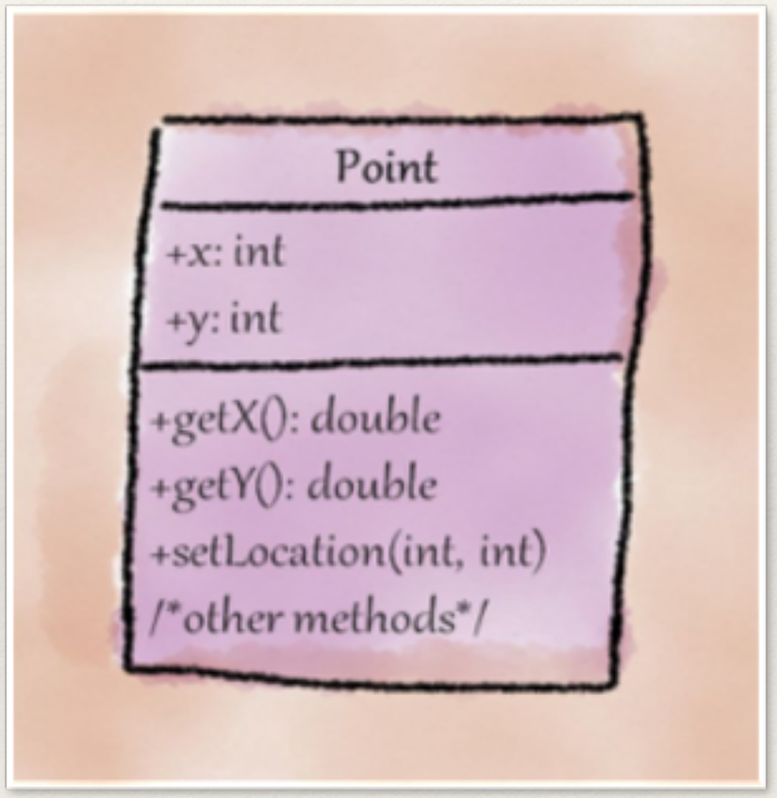
What's that smell?

Results in "Inappropriate intimacy" smell

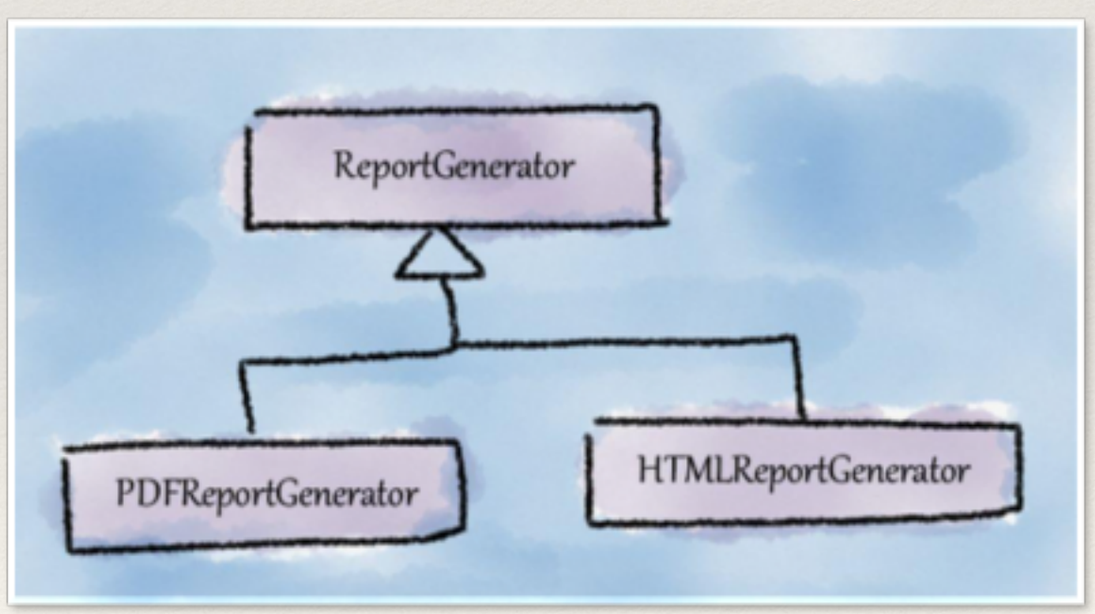


What's that smell?

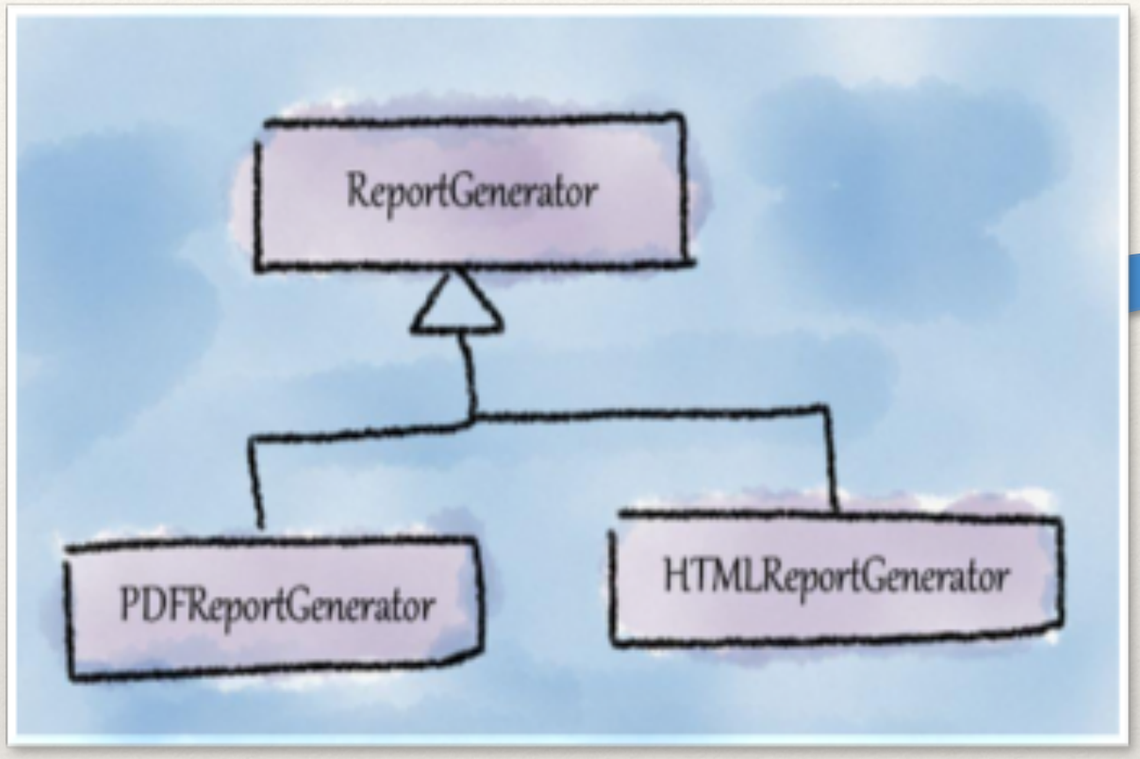
Results in "Inappropriate intimacy" smell



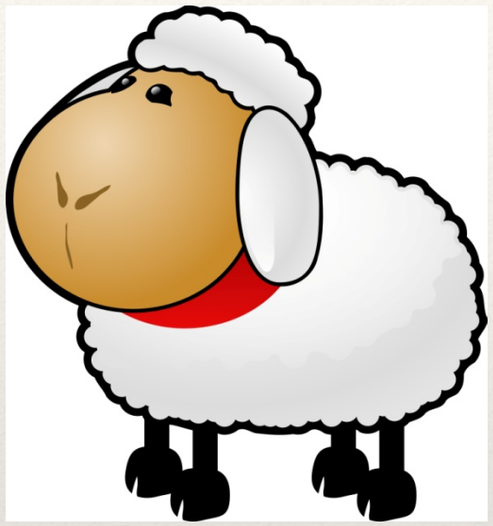
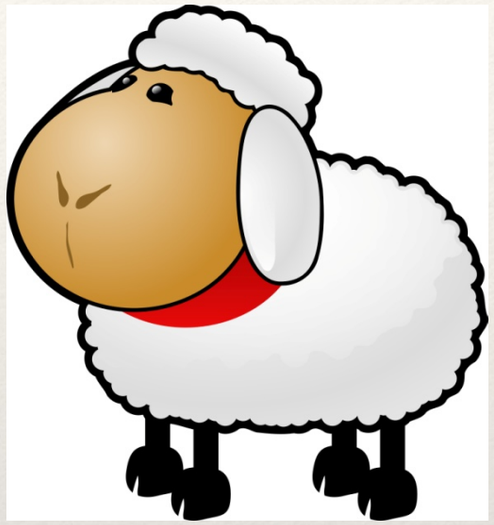
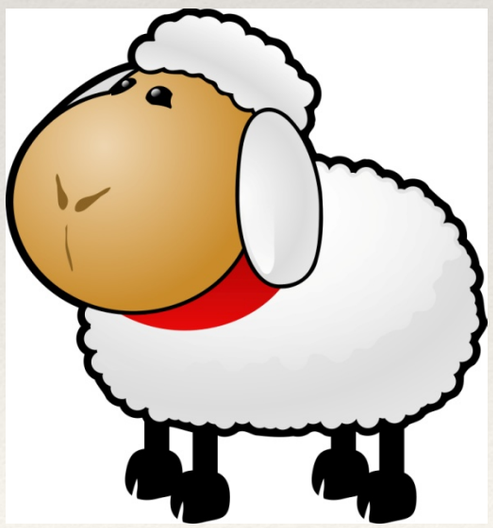
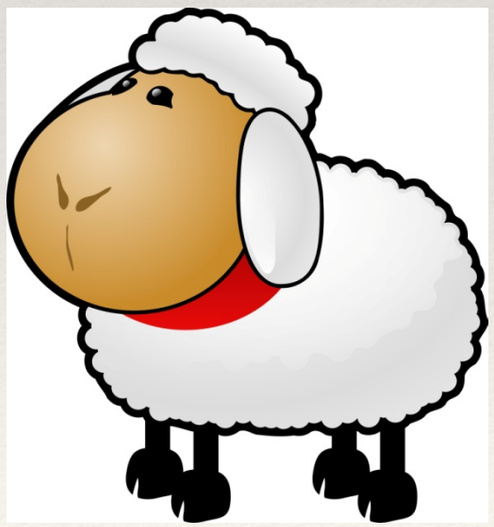
What's that smell?



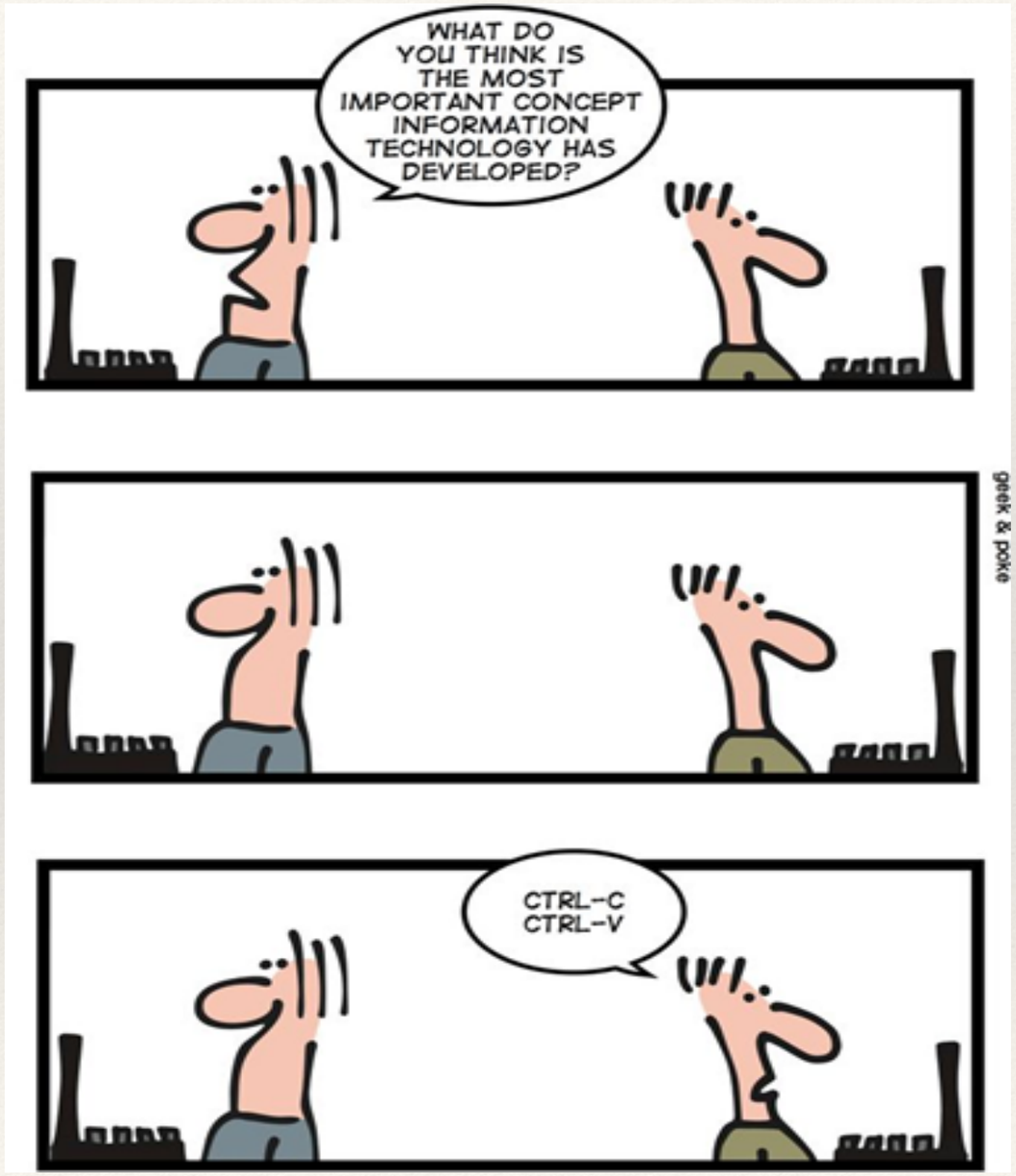
Refactoring “speculative generalization”



Duplicated Code

CTRL-C and CTRL-V



Types of Clones

Type 1

- **exactly identical** except for variations in whitespace, layout, and comments

Type 2

- **syntactically identical** except for variation in symbol names, whitespace, layout, and comments

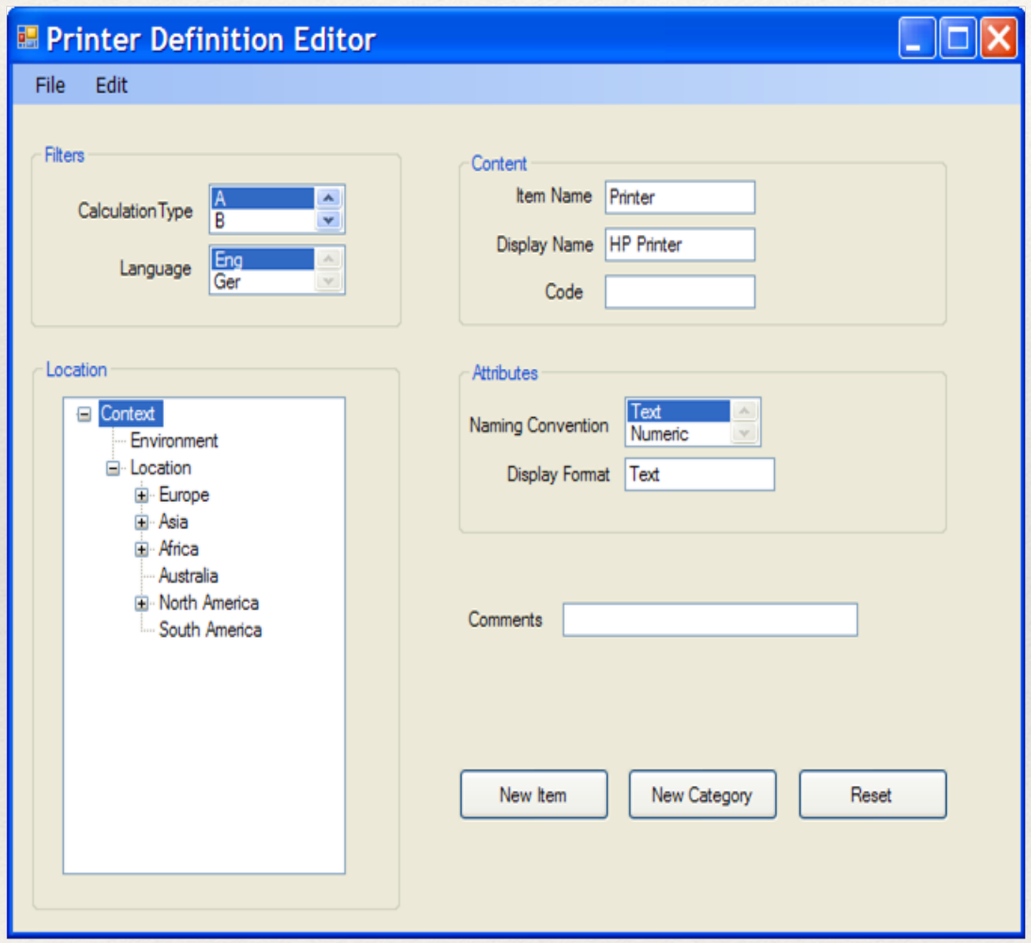
Type 3

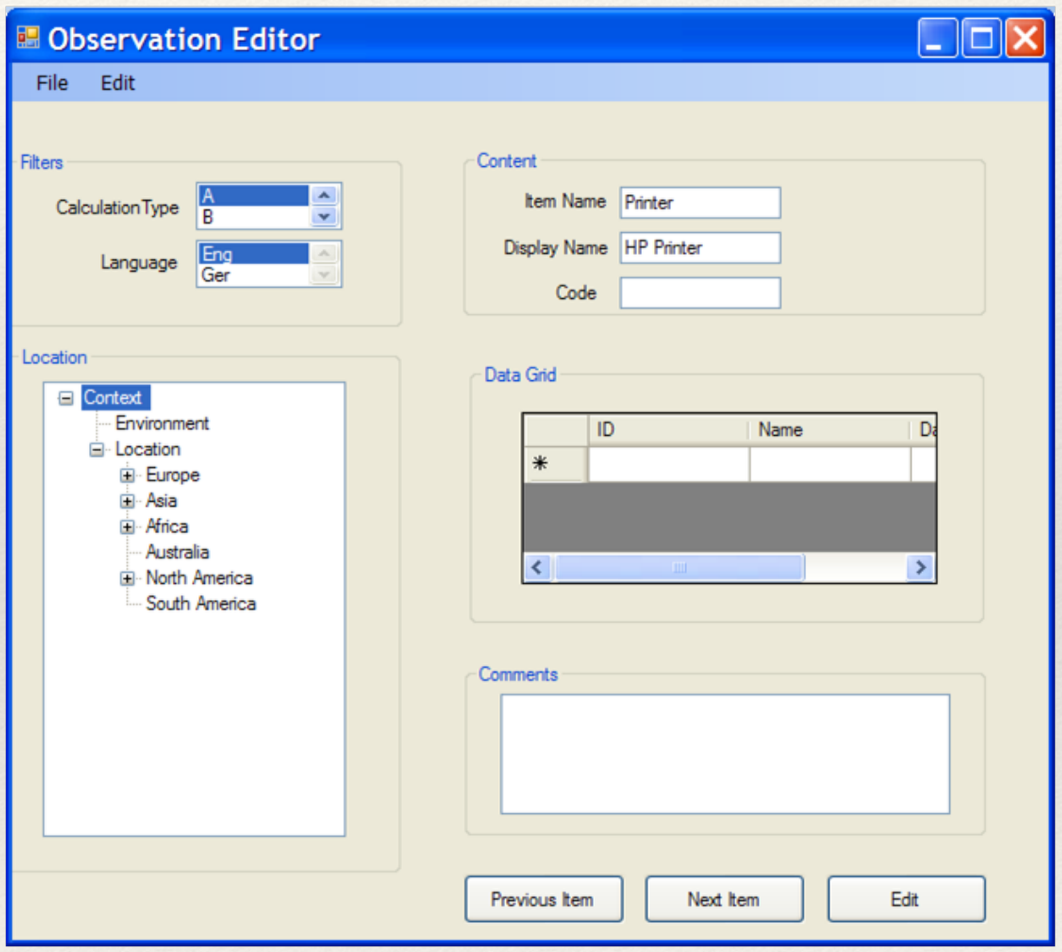
- identical except some **statements changed, added, or removed**

Type 4

- when the fragments are **semantically identical** but implemented by syntactic variants

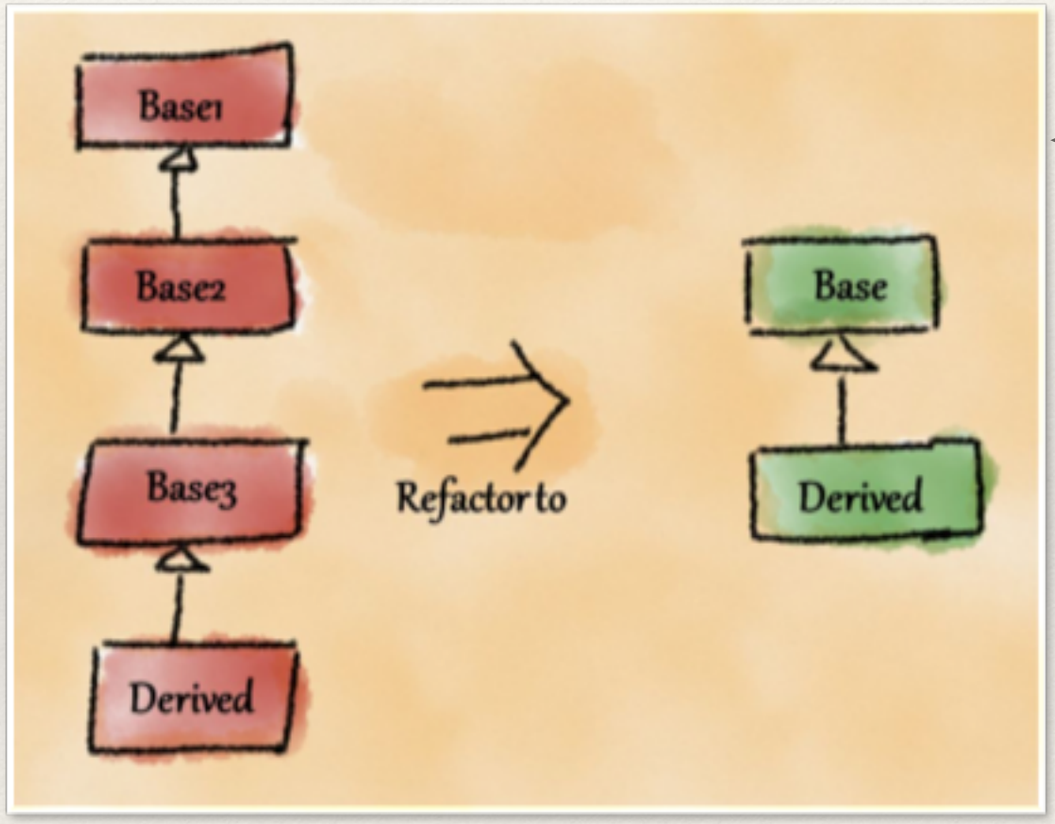
How to deal with duplication?





What's that smell?

Refactoring

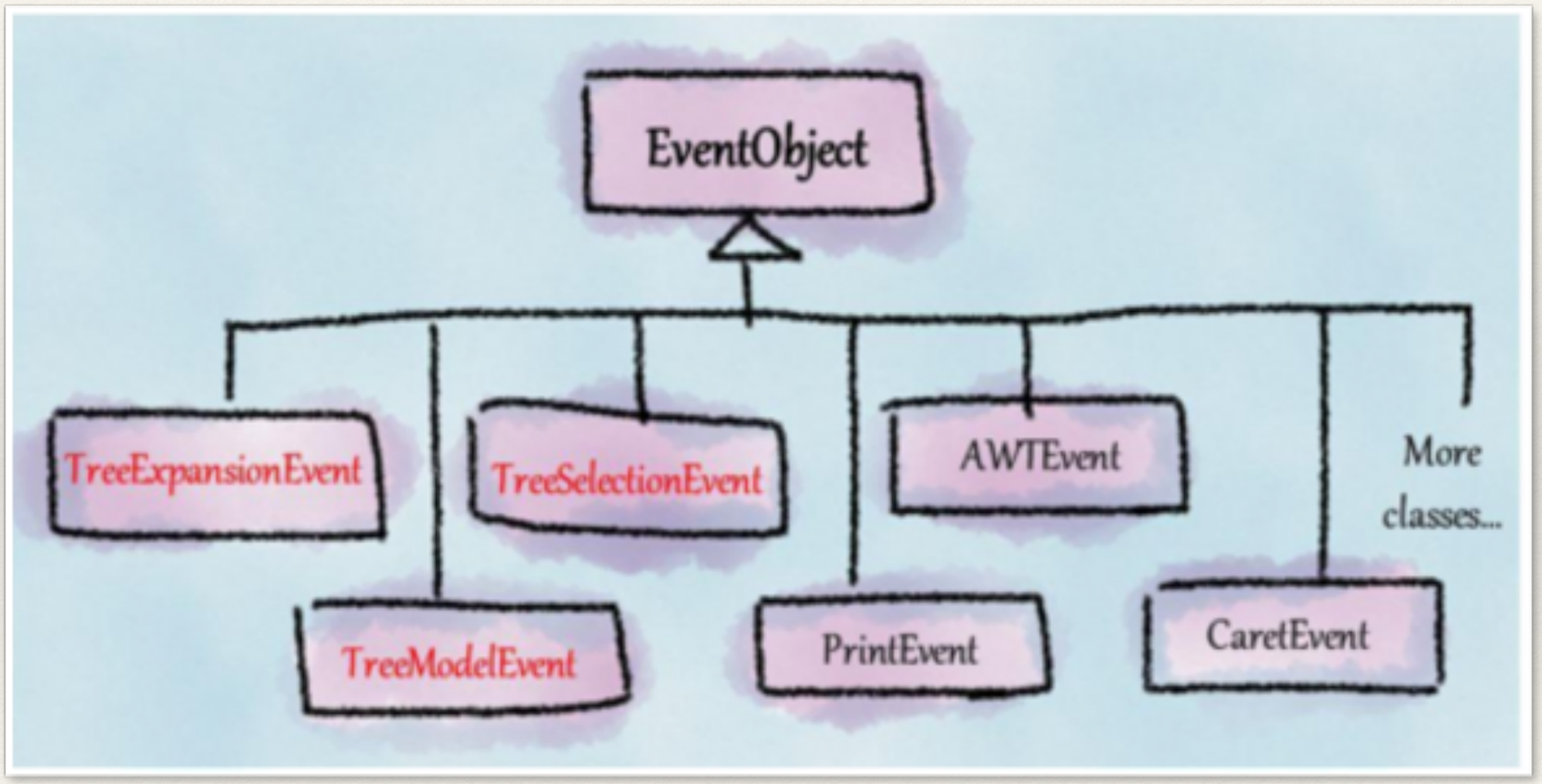


Collapse Hierarchy

Refactoring

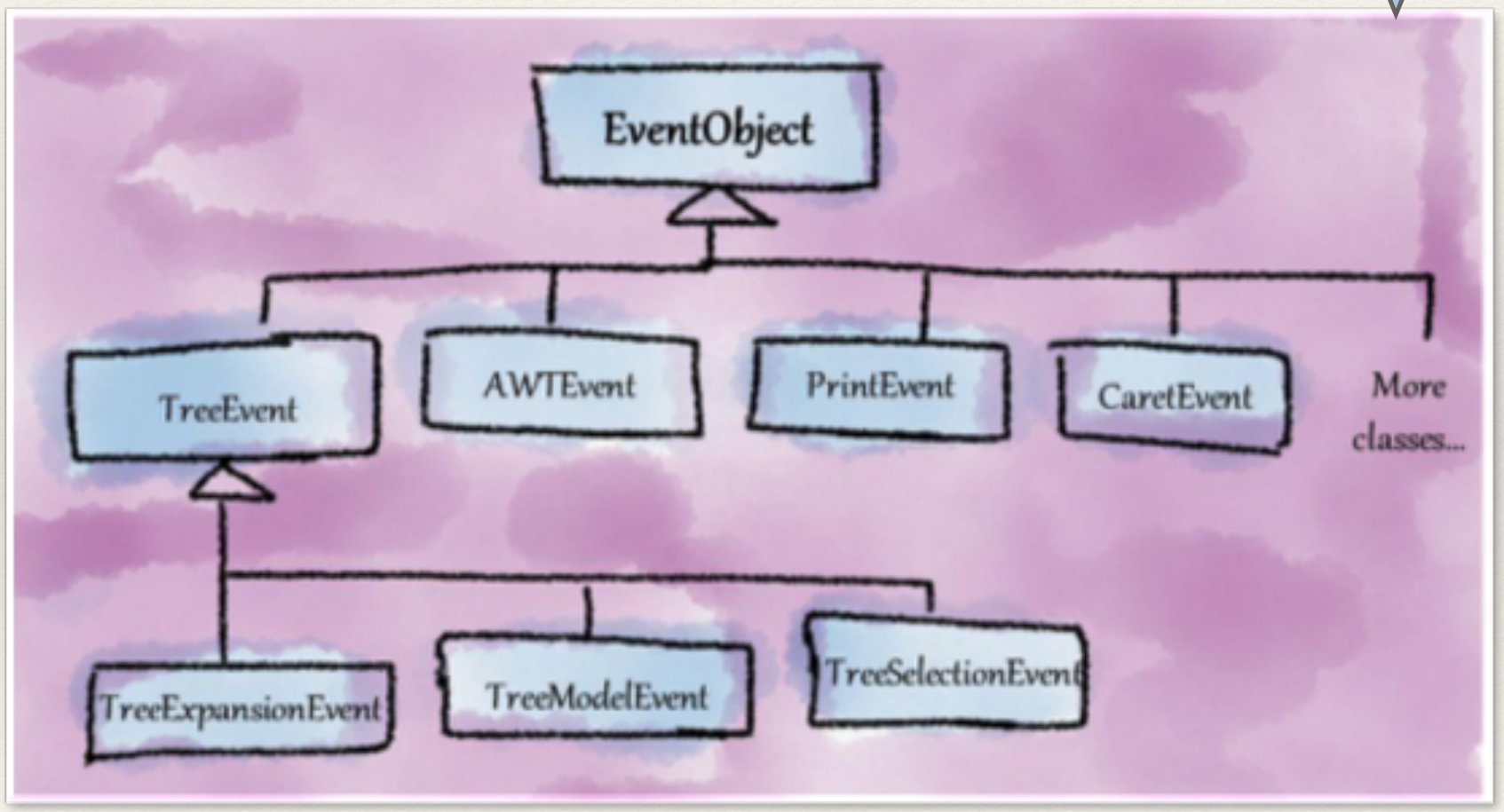
```
/**
 * Pluggable look and feel interface for JButton.
*/
public abstract class ButtonUI extends ComponentUI {
}
```

What's that smell?

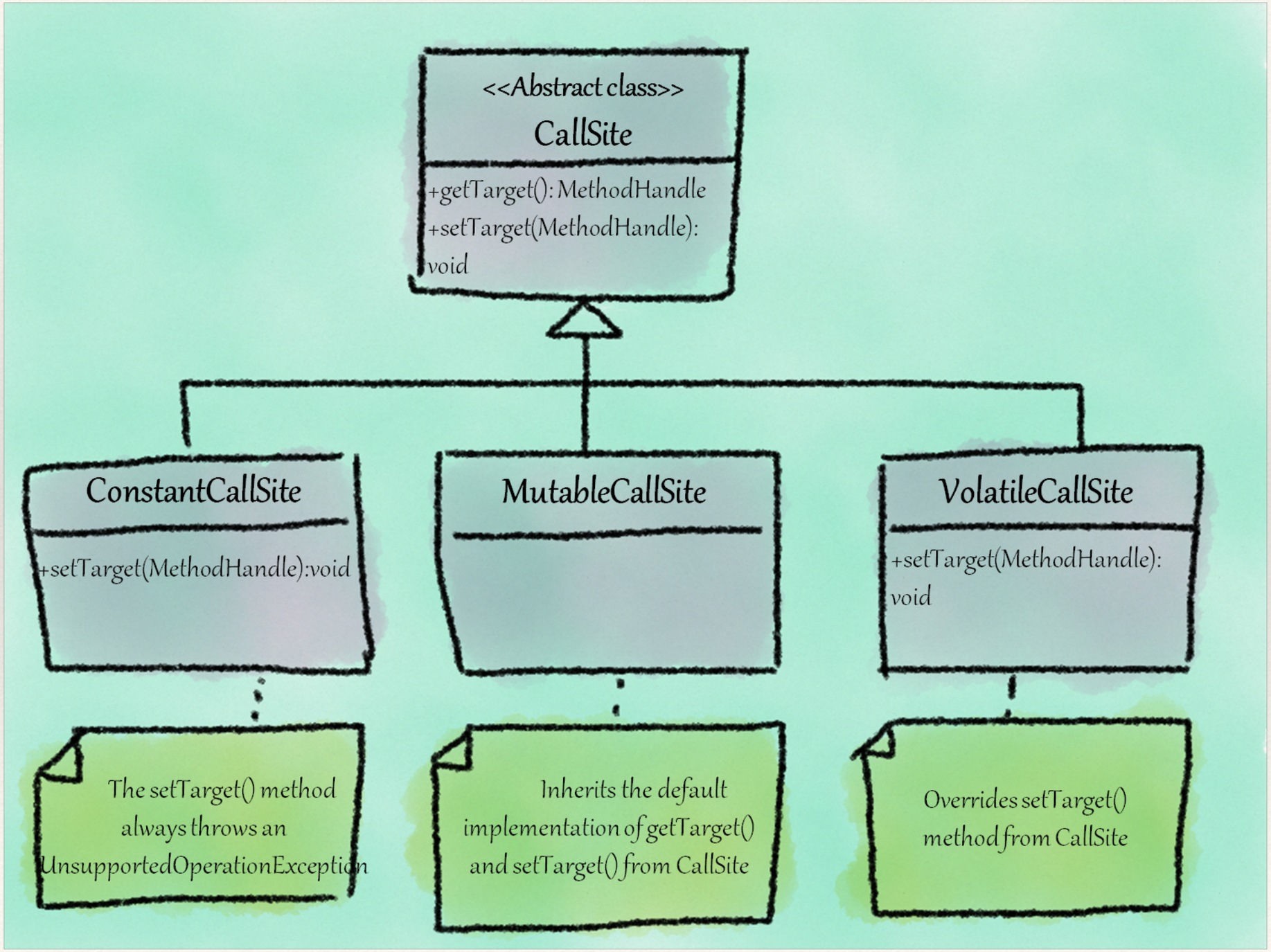


Refactoring

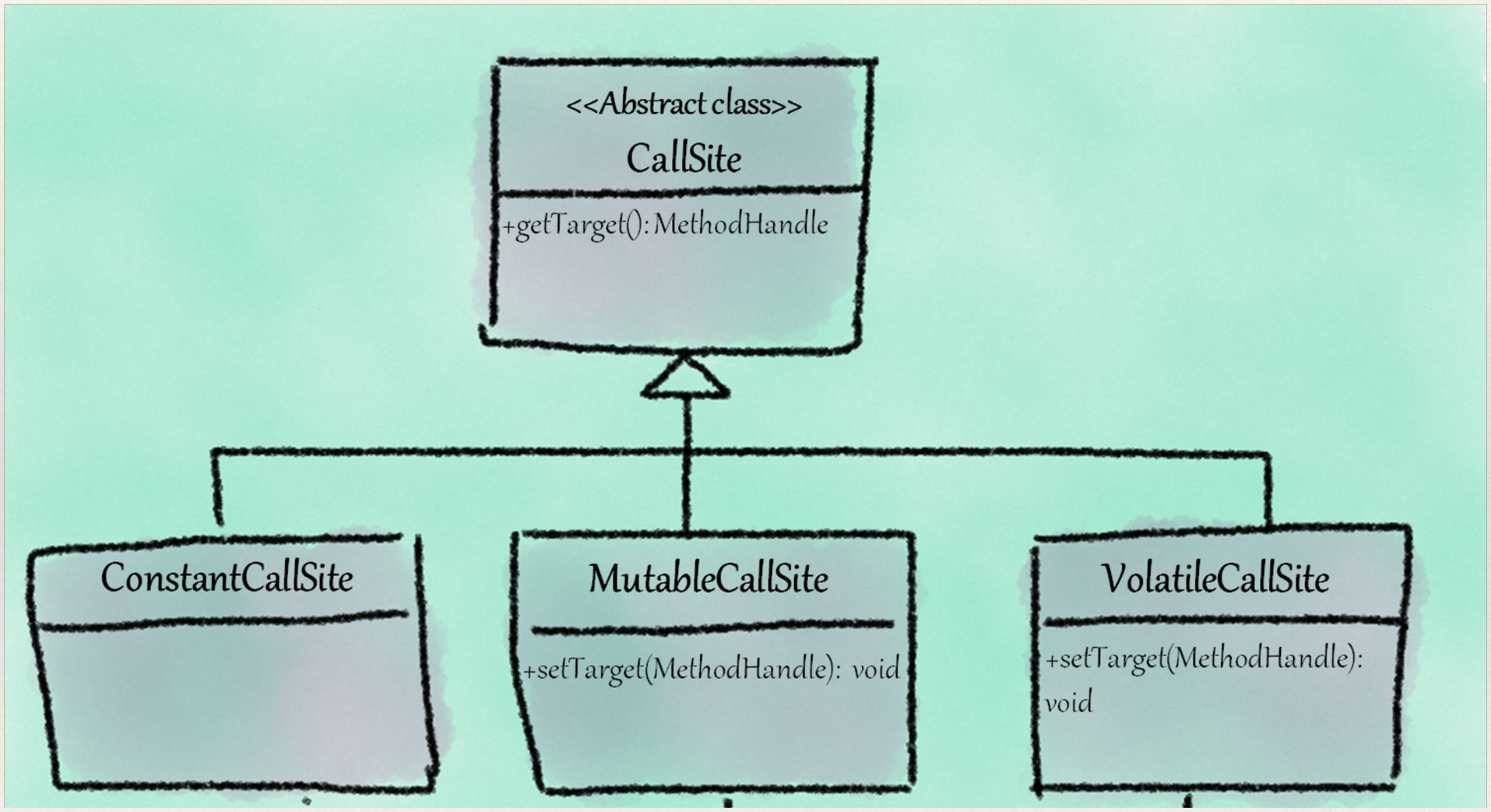
Extract Superclass



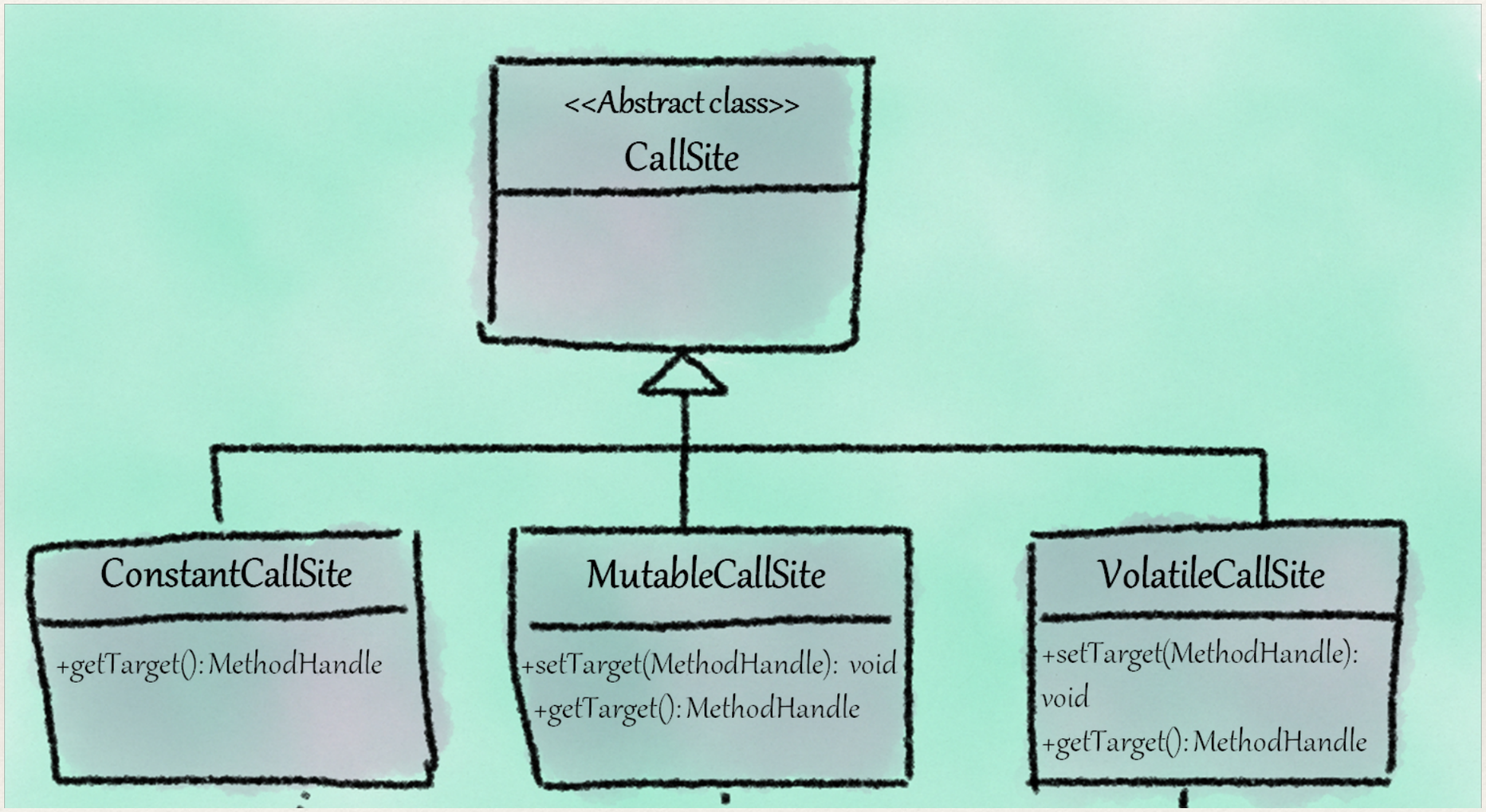
What's that smell?



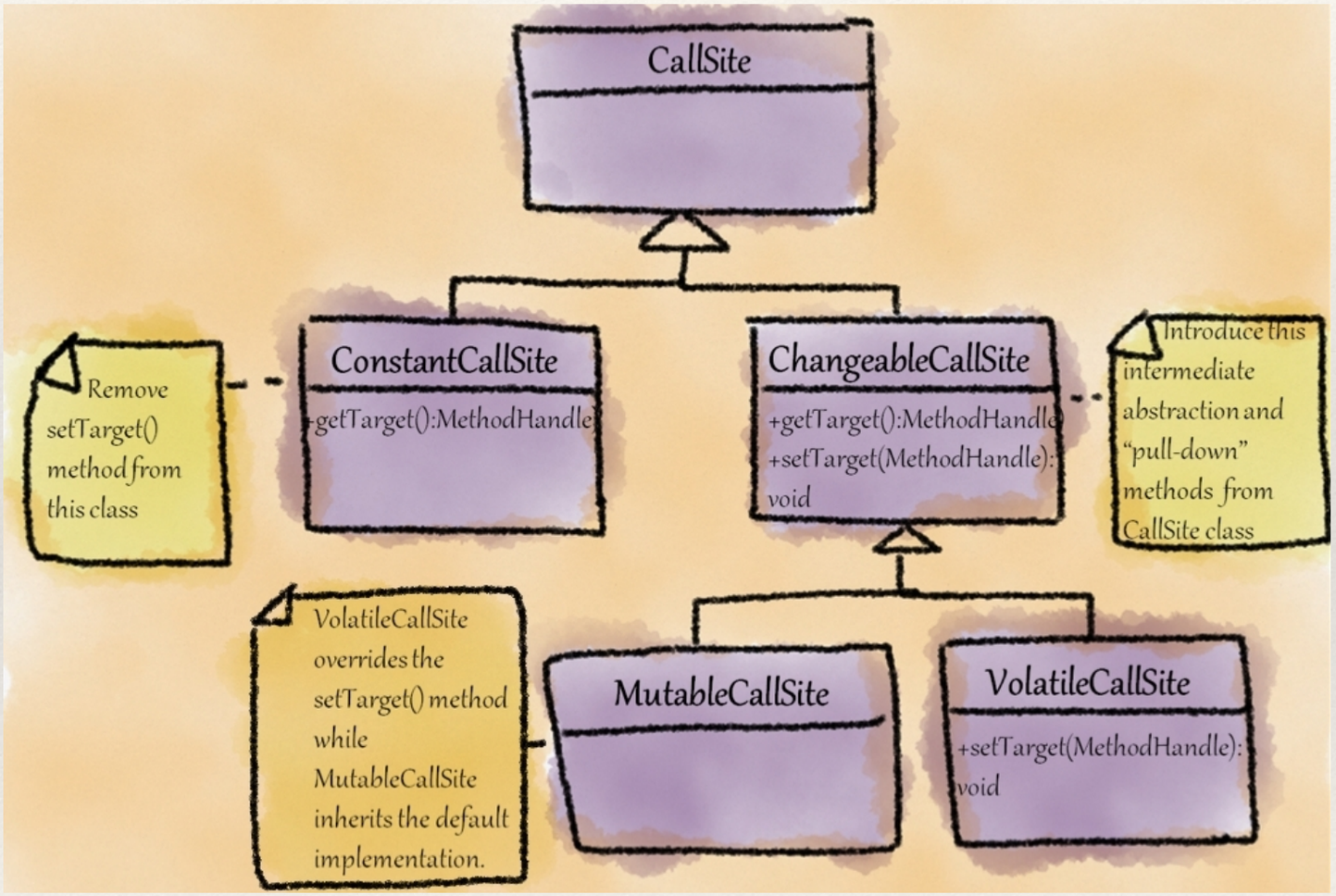
How about this refactoring?



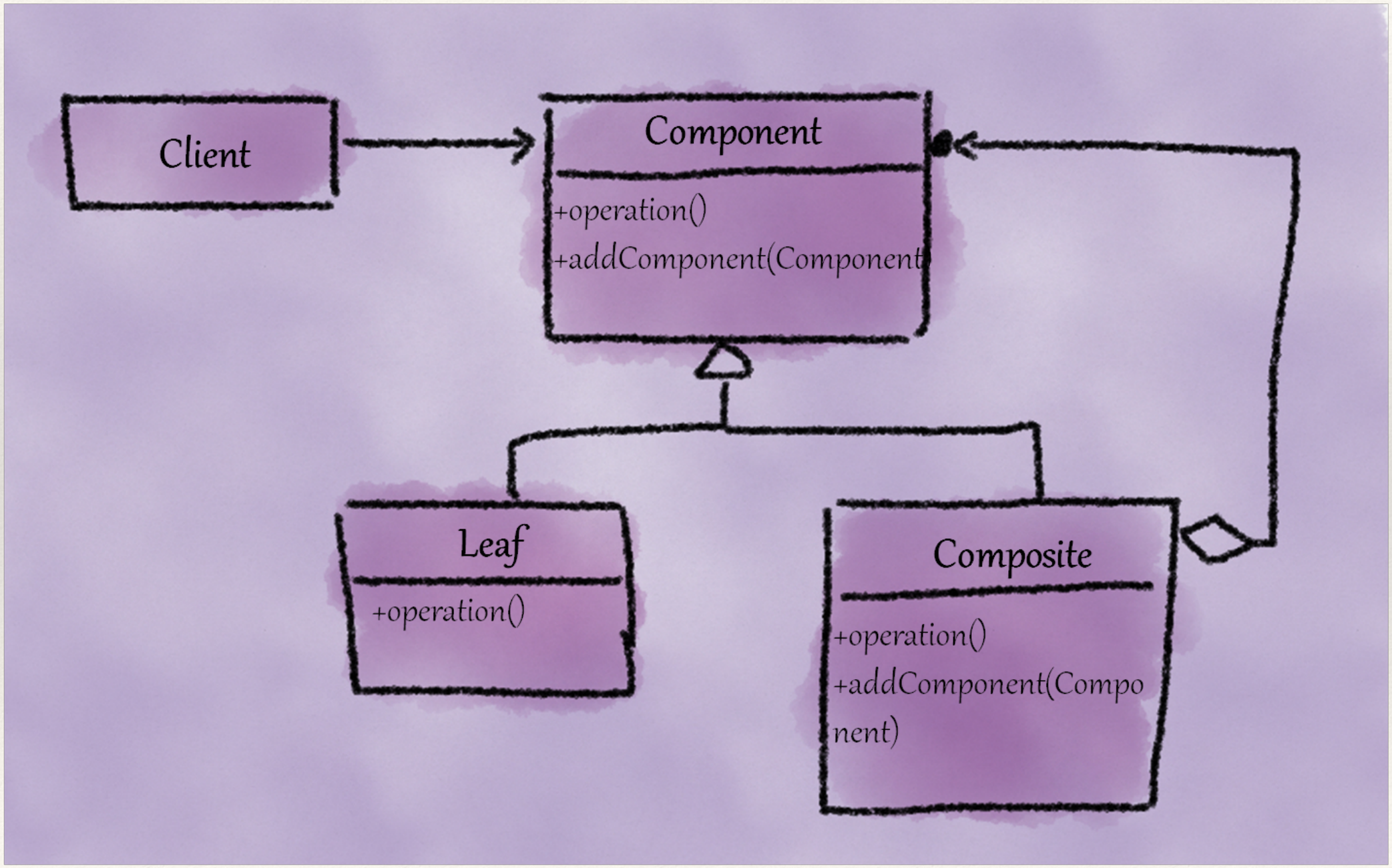
How about this refactoring?



Suggested refactoring

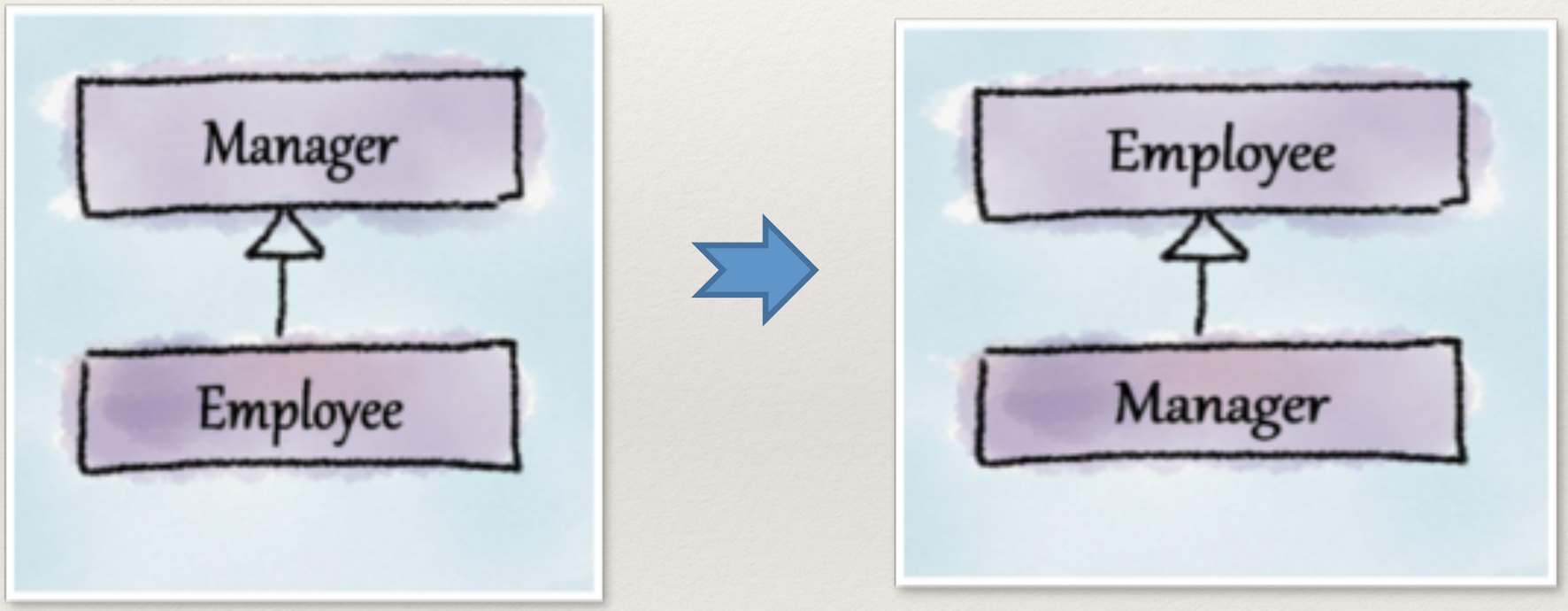


Practical considerations

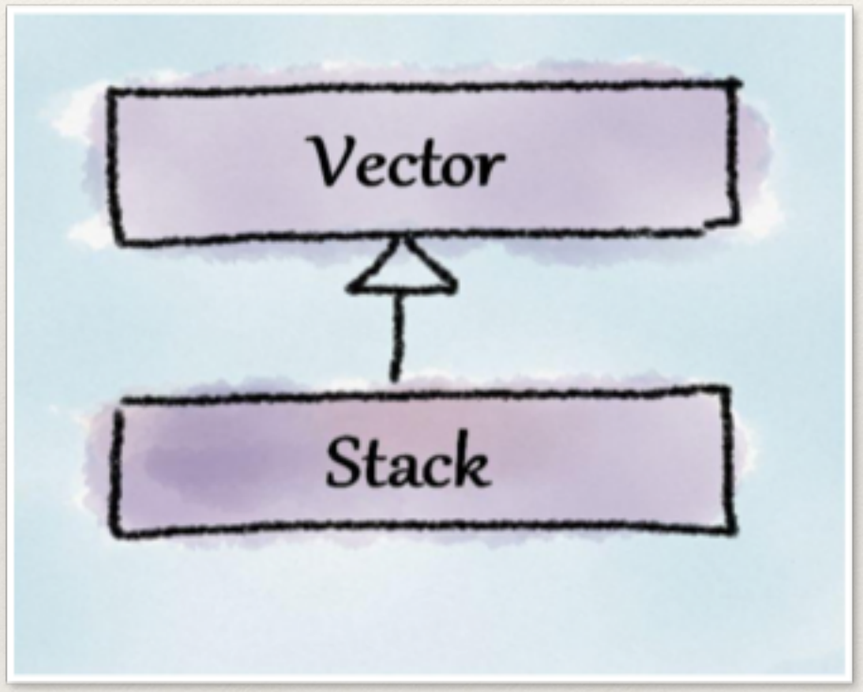


What's that smell?

Refactoring



What's that smell?



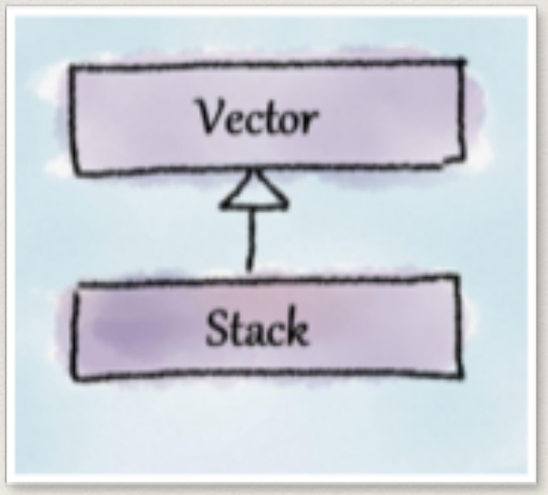
Liskov's Substitution Principle (LSP)

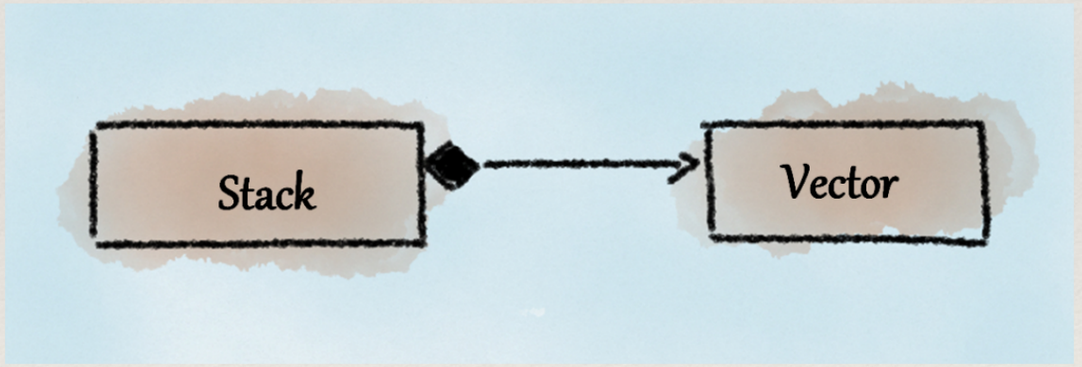
It should be possible to replace objects of supertype with objects of subtypes without altering the desired behavior of the program

Barbara Liskov

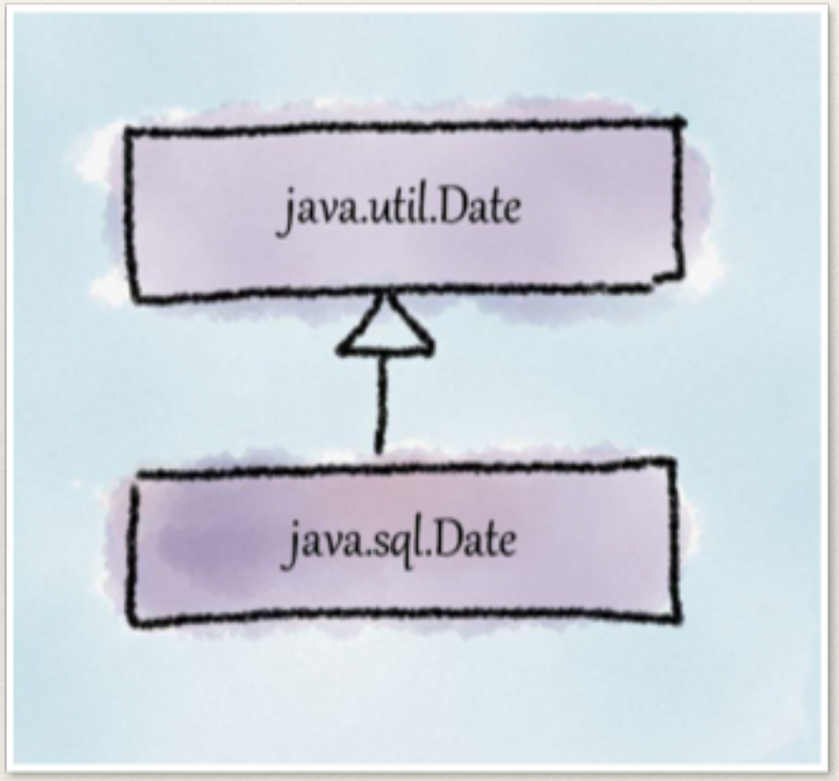
Refactoring

Replace inheritance with delegation

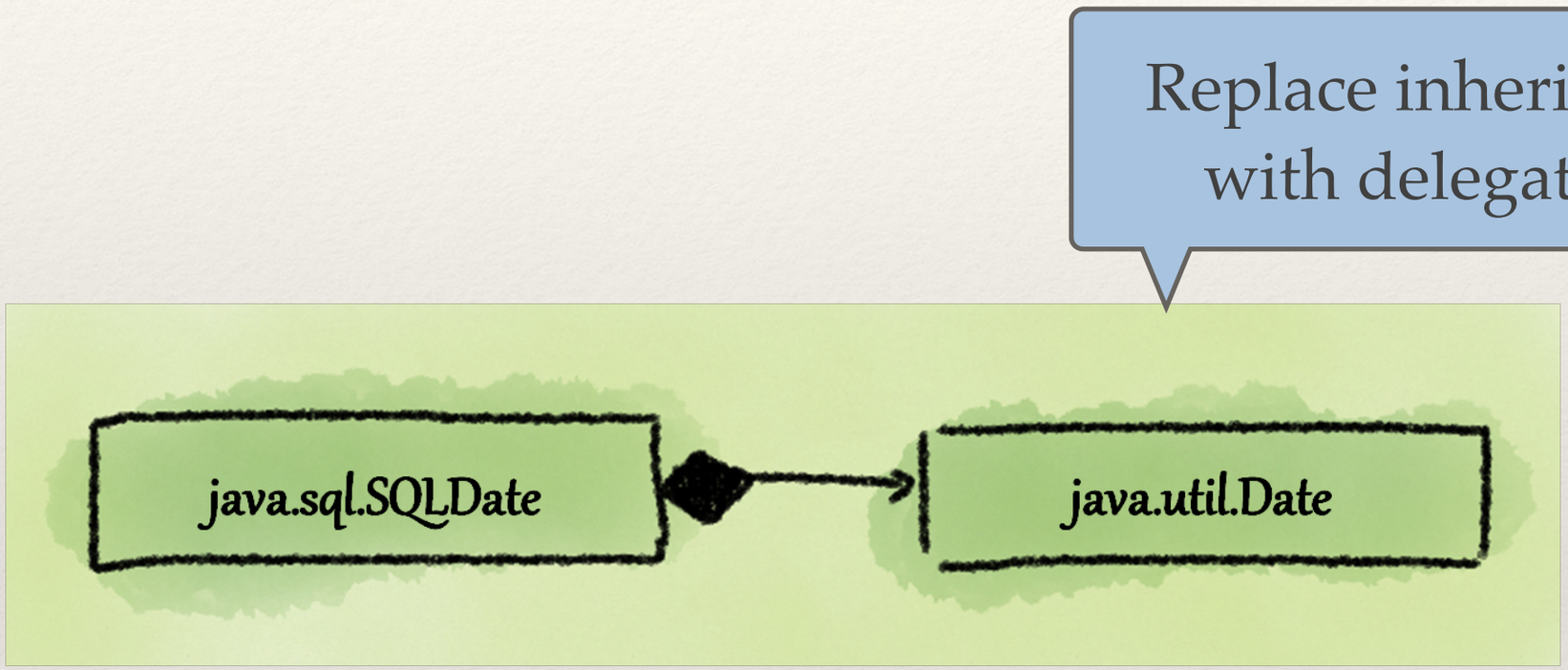




What's that smell?



Refactoring for Date



What's that smell?

javax.swing.Timer	• executing objects of type ActionEvent at specified intervals
java.util.Timer	• scheduling a thread to execute in the future as a background thread
javax.management.timer.Timer	• sending out an alarm to wake-up the listeners that have registered to get timer notifications

What's that smell?

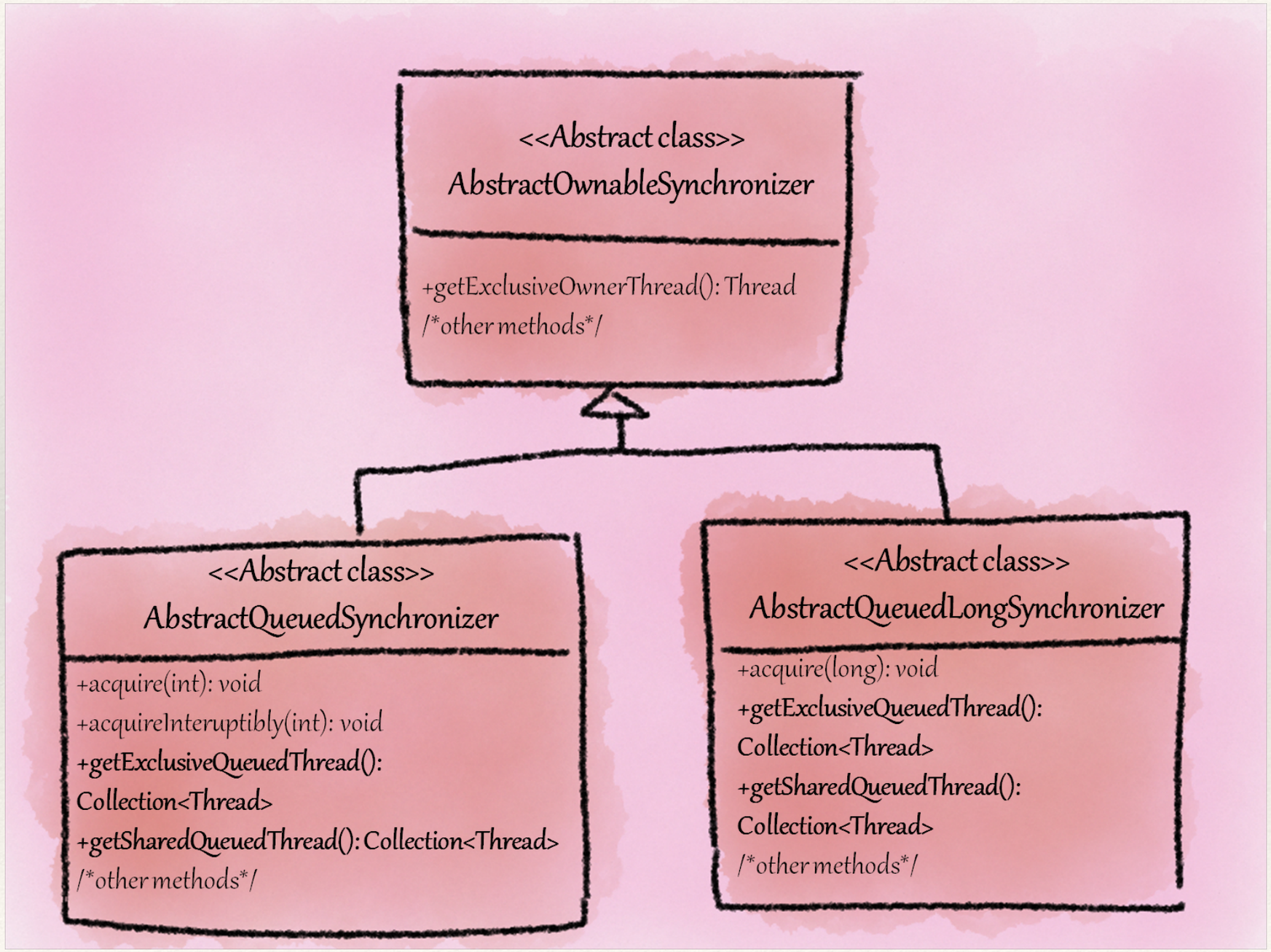
Rectangle2D class

```
/*
* Note on casts to double below.  If the arithmetic of
* x+w or y+h is done in float, then some bits may be
* lost if the binary exponents of x/y and w/h are not
* similar.  By converting to double before the addition
* we force the addition to be carried out in double to
* avoid rounding error in the comparison.
*
* See bug 4320890 for problems that this inaccuracy causes.
*/
```

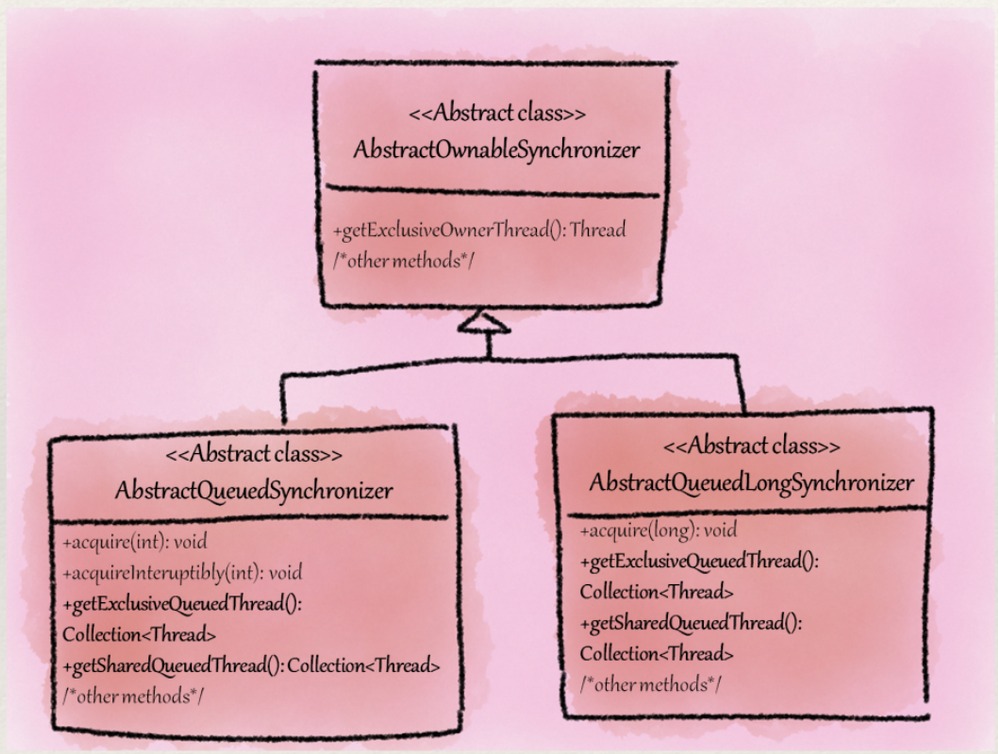
Rectangle class

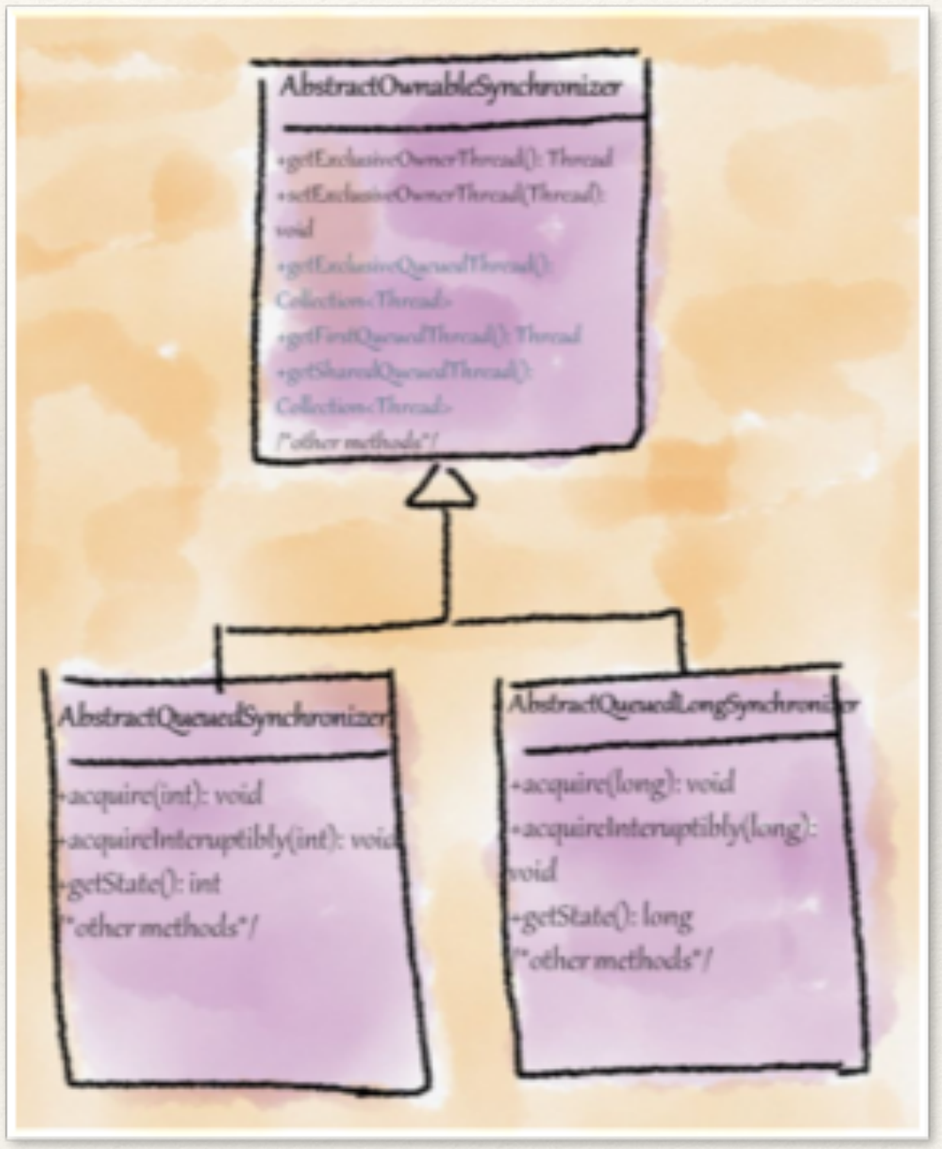
```
/*
* Note on casts to double below.  If the arithmetic of
* x+w or y+h is done in int, then we may get integer
* overflow. By converting to double before the addition
* we force the addition to be carried out in double to
* avoid overflow in the comparison.
*
* See bug 4320890 for problems that this can cause.
*/
```

What's that smell?

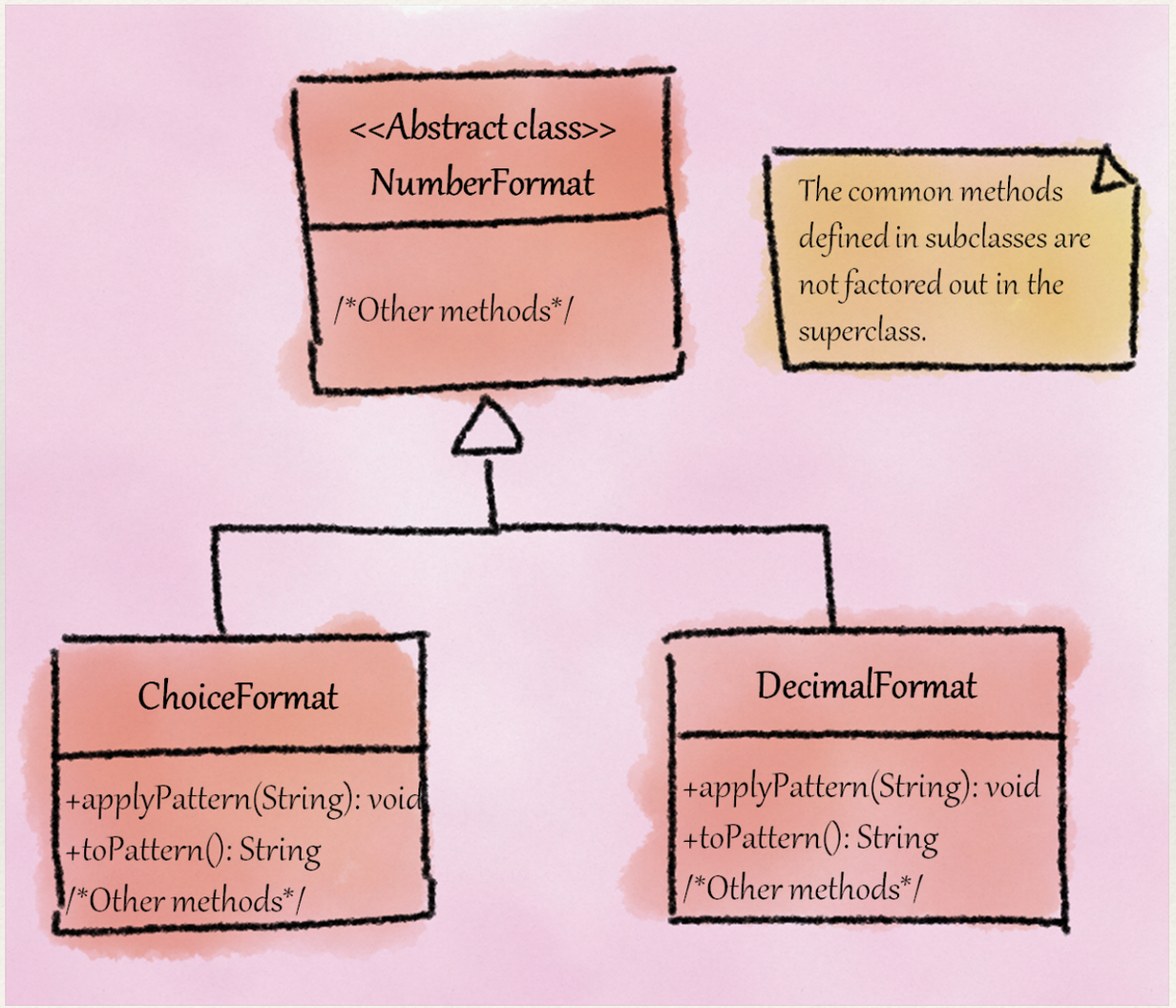


Refactoring

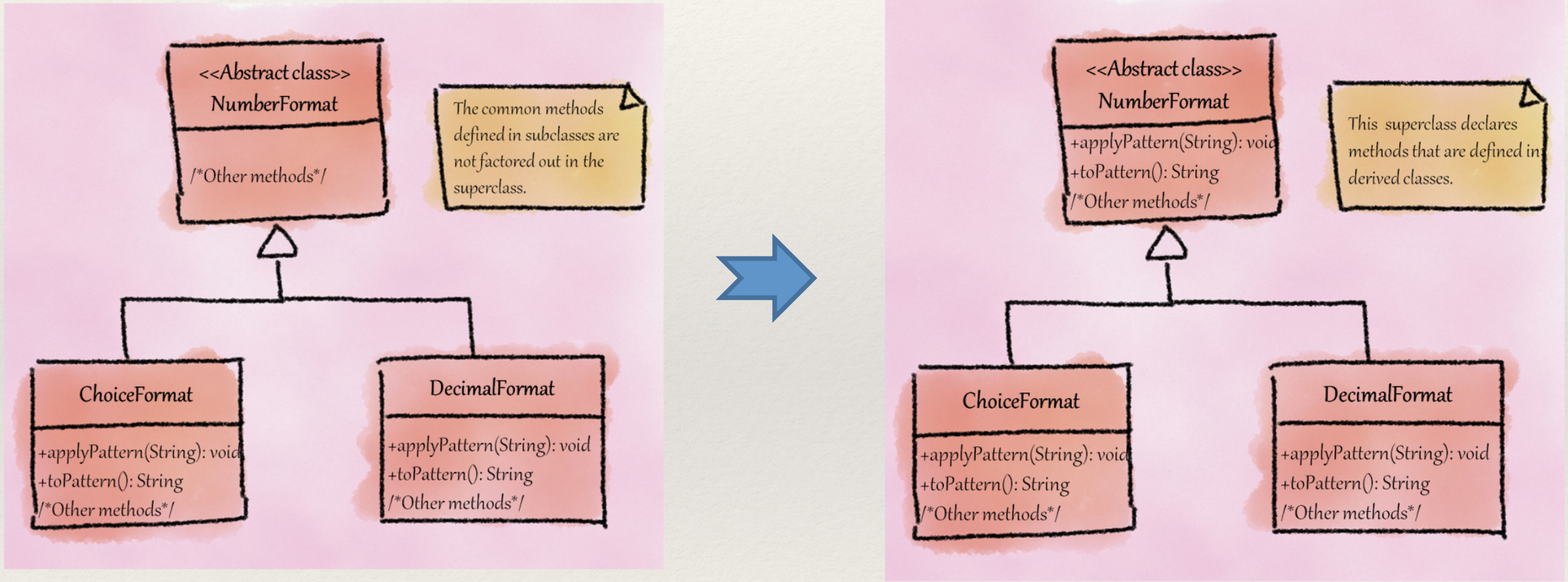




What's that smell?



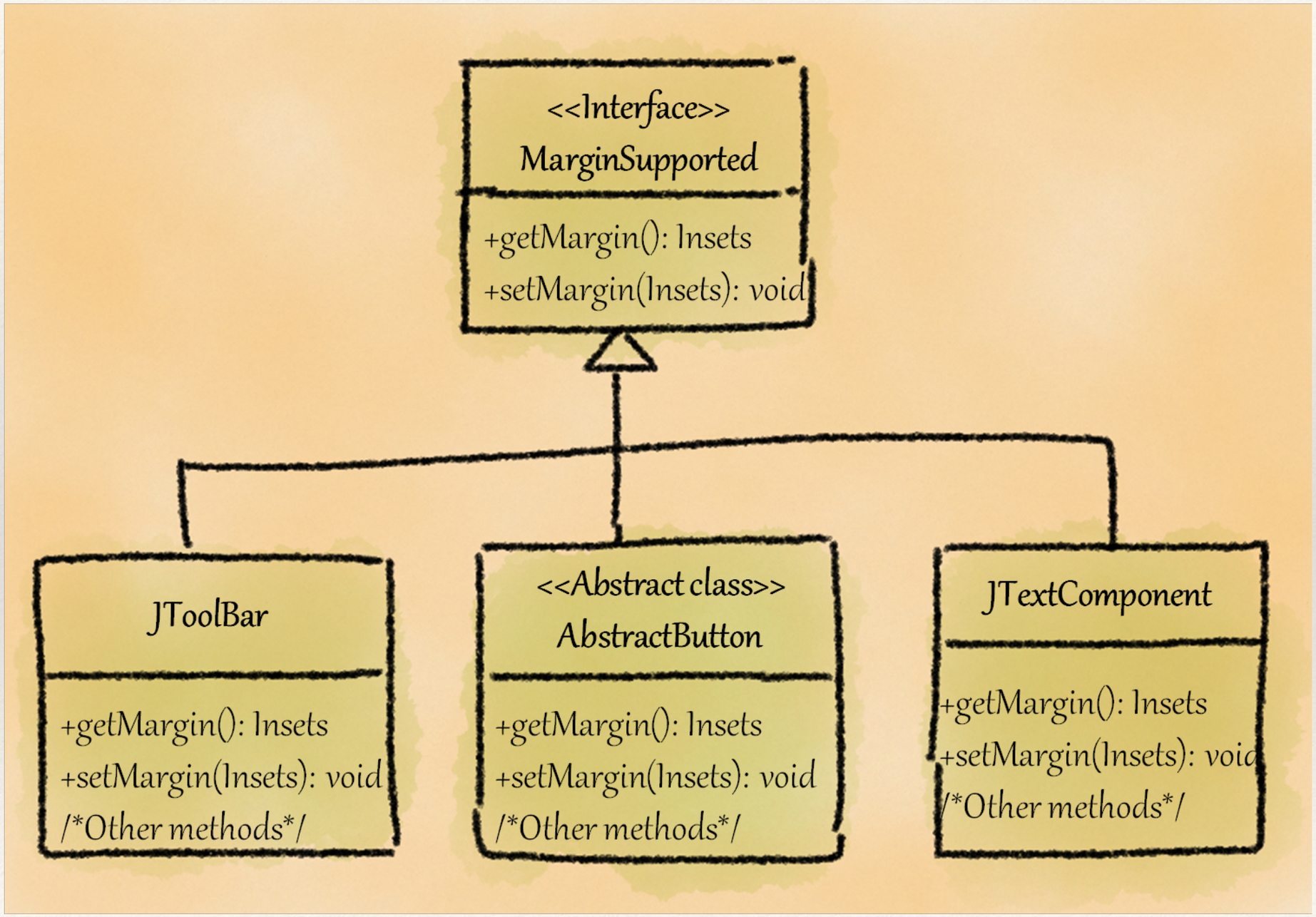
Refactoring



What's that smell?

```
public Insets getBorderInsets(Component c, Insets insets){
        Insets margin = null;
         // Ideally we'd have an interface defined for classes which
        // support margins (to avoid this hackery), but we've
        // decided against it for simplicity
        //
        if (c instanceof AbstractButton) {
                margin = ((AbstractButton)c).getMargin();
        } else if (c instanceof JToolBar) {
                margin = ((JToolBar)c).getMargin();
        } else if (c instanceof JTextComponent) {
                margin = ((JTextComponent)c).getMargin();
        }
        // rest of the code omitted …
```

Refactoring

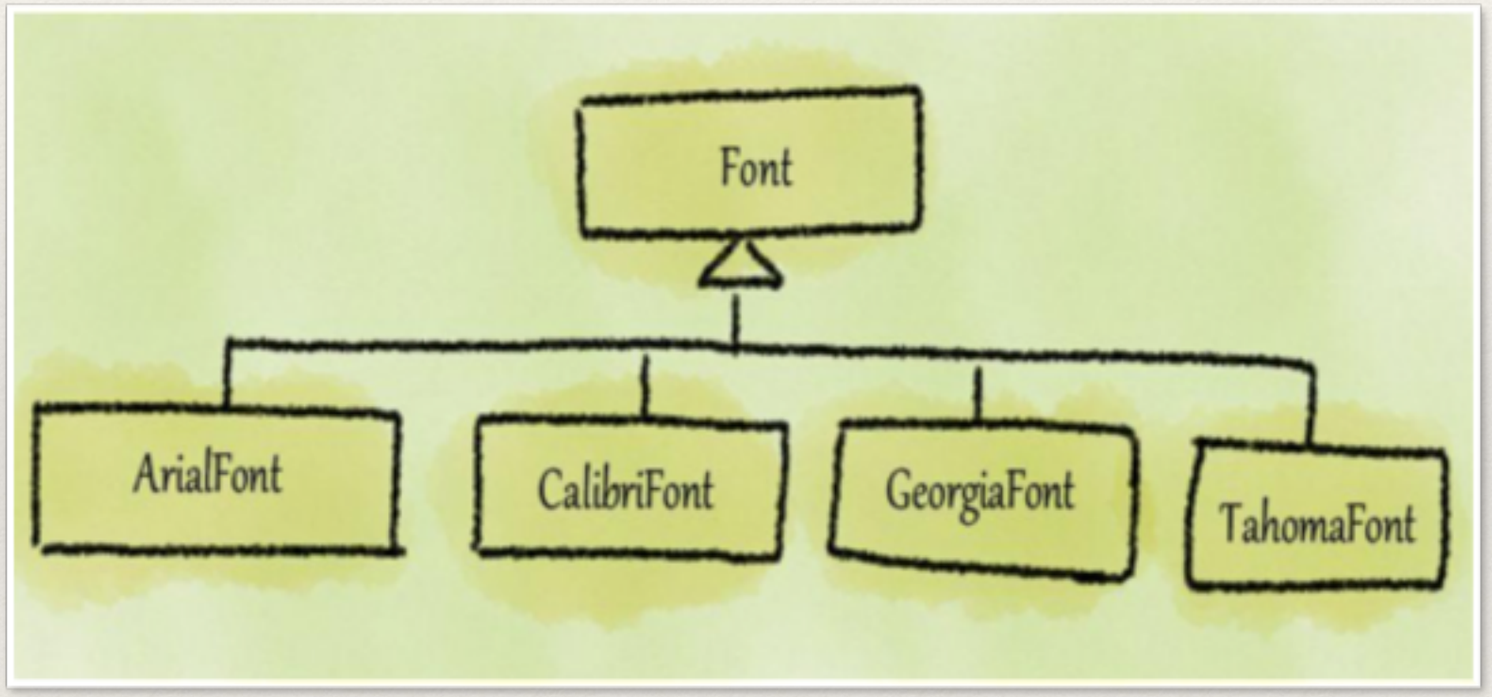


Refactoring

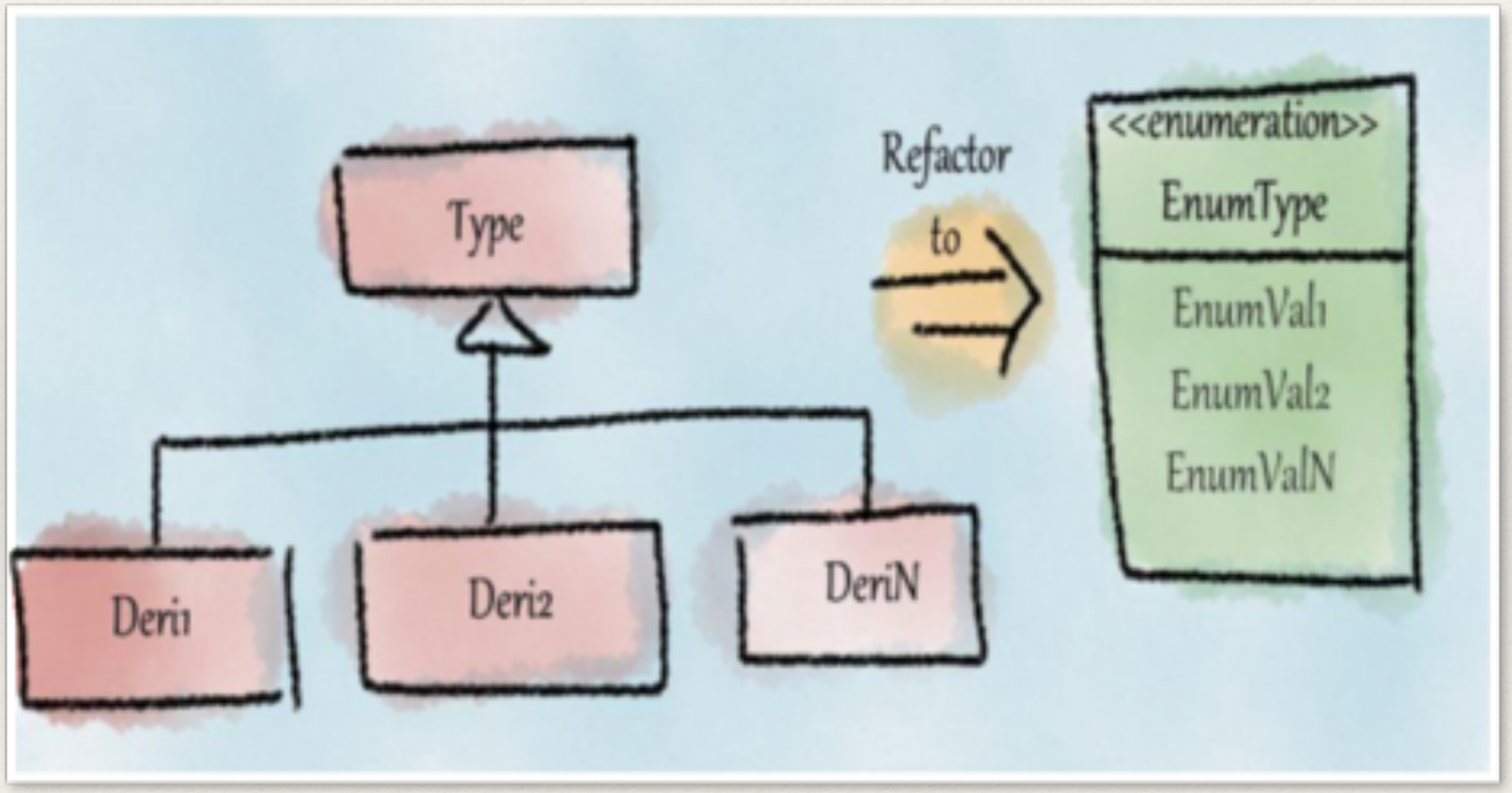
```
if (c instanceof AbstractButton) {
        margin = ((AbstractButton)c).getMargin();
} else if (c instanceof JToolBar) {
        margin = ((JToolBar)c).getMargin();
} else if (c instanceof JTextComponent) {
        margin = ((JTextComponent)c).getMargin();
}
```

```
margin = c.getMargin();
```

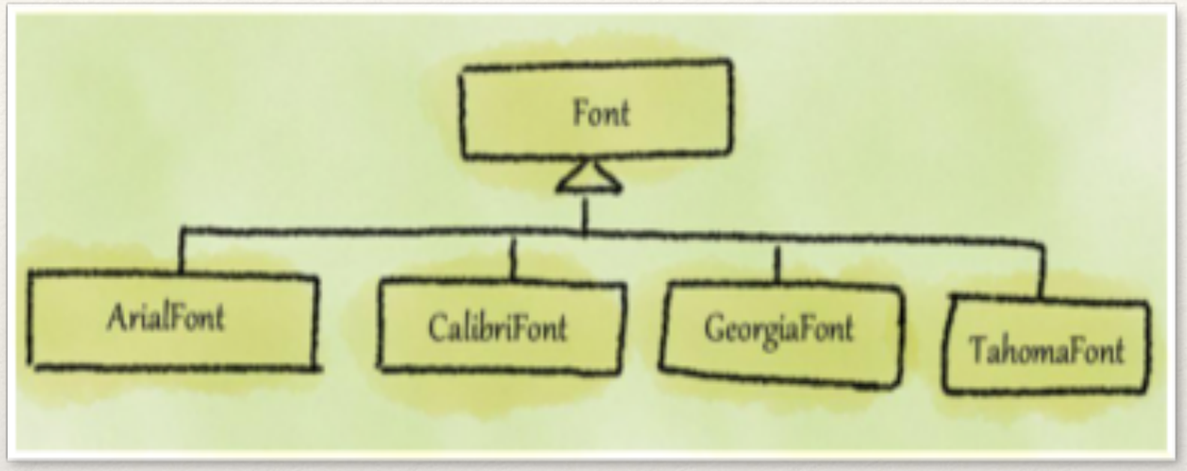
What's that smell?

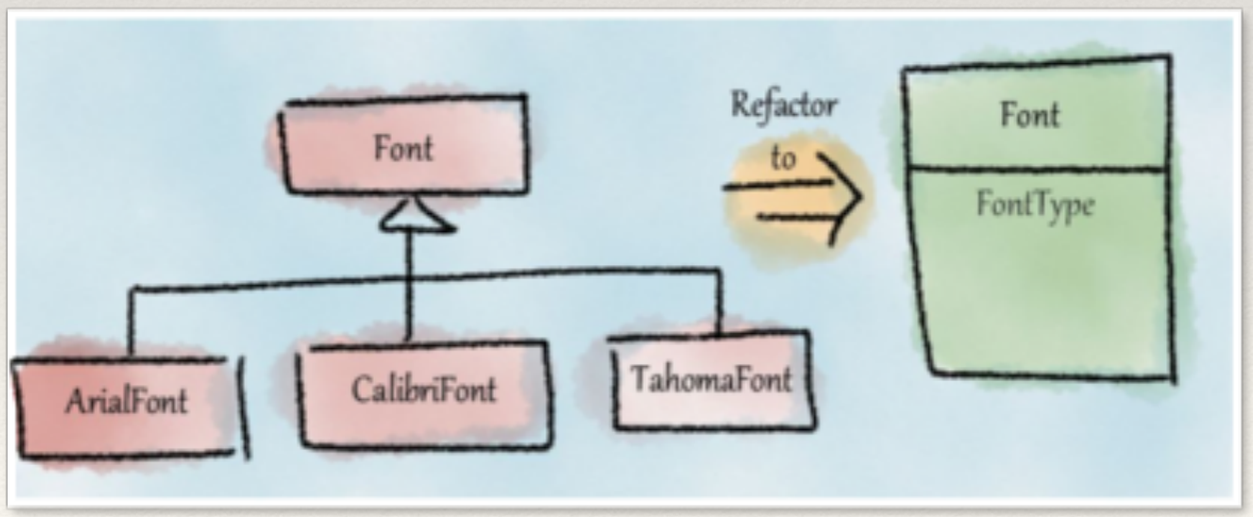


Refactoring

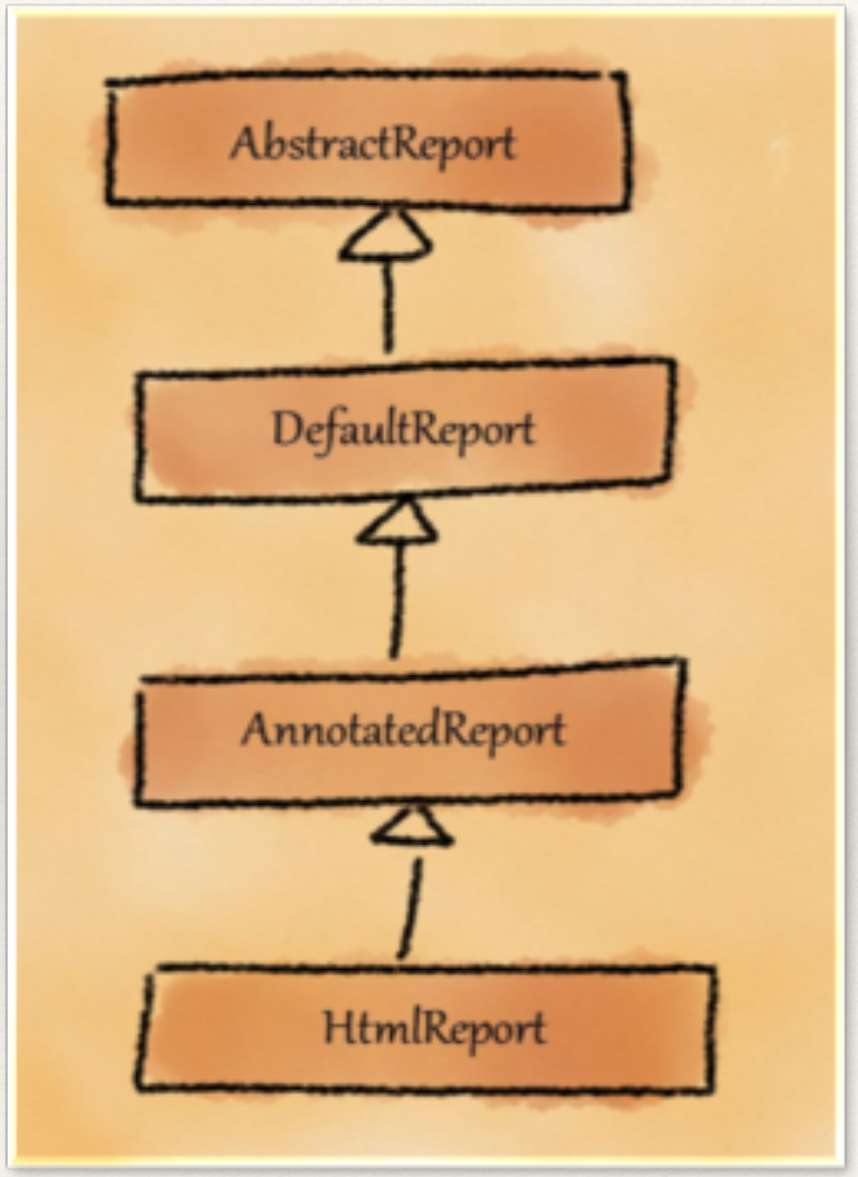


Refactoring

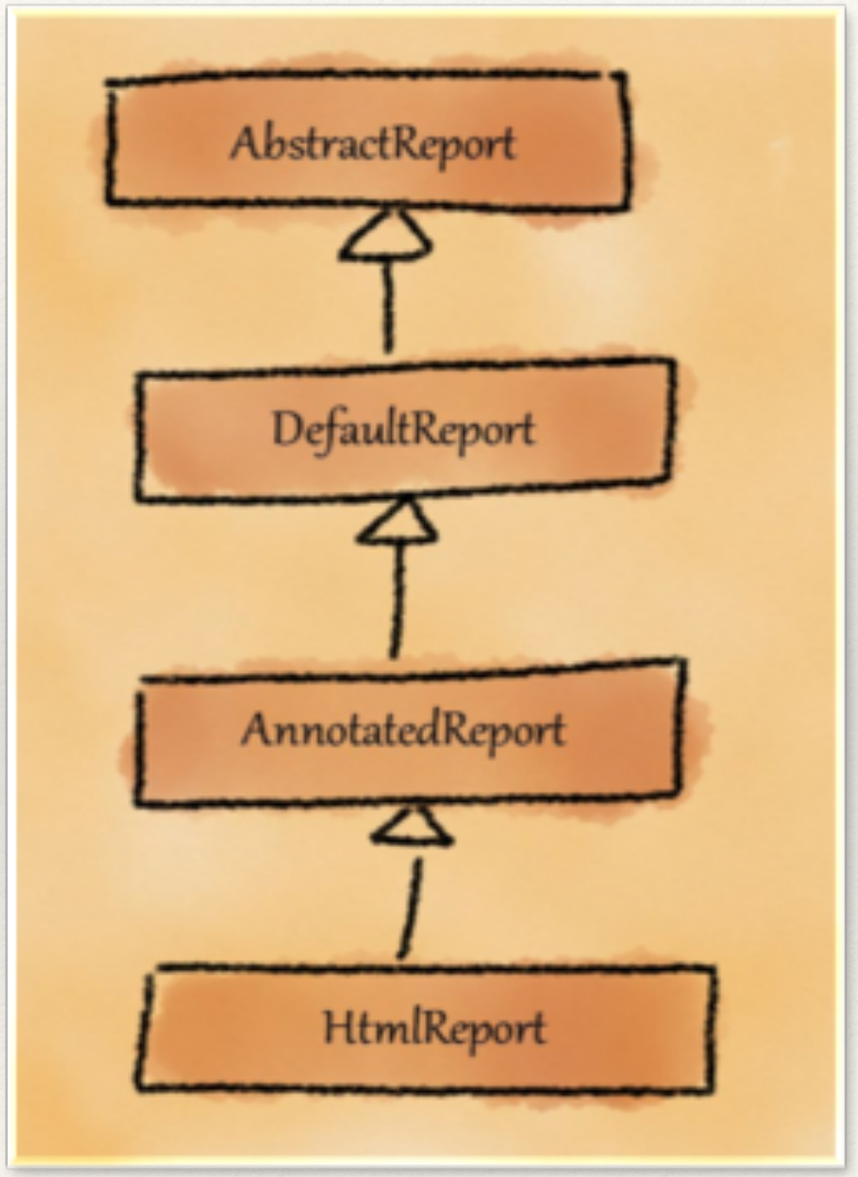




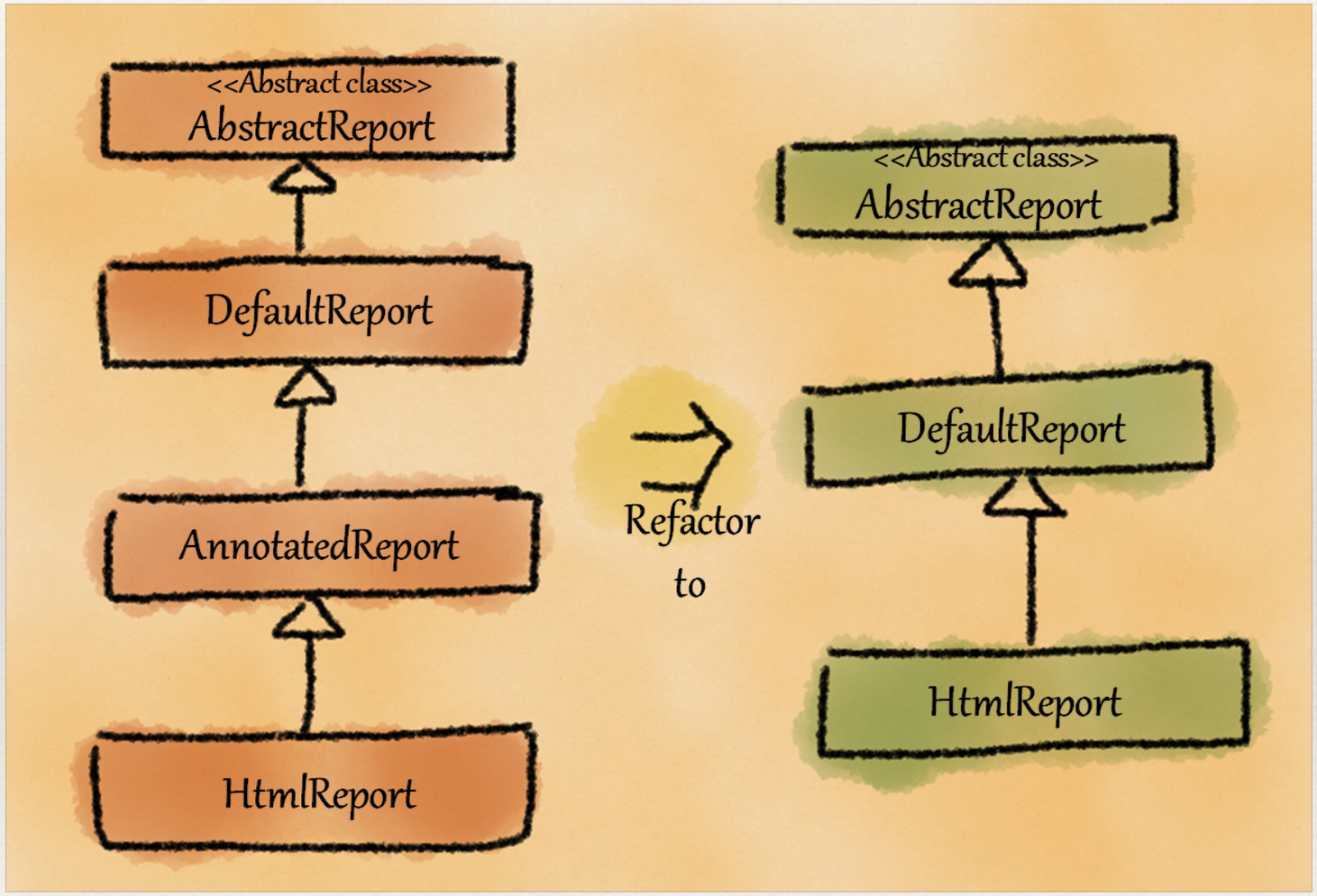
What's that smell?



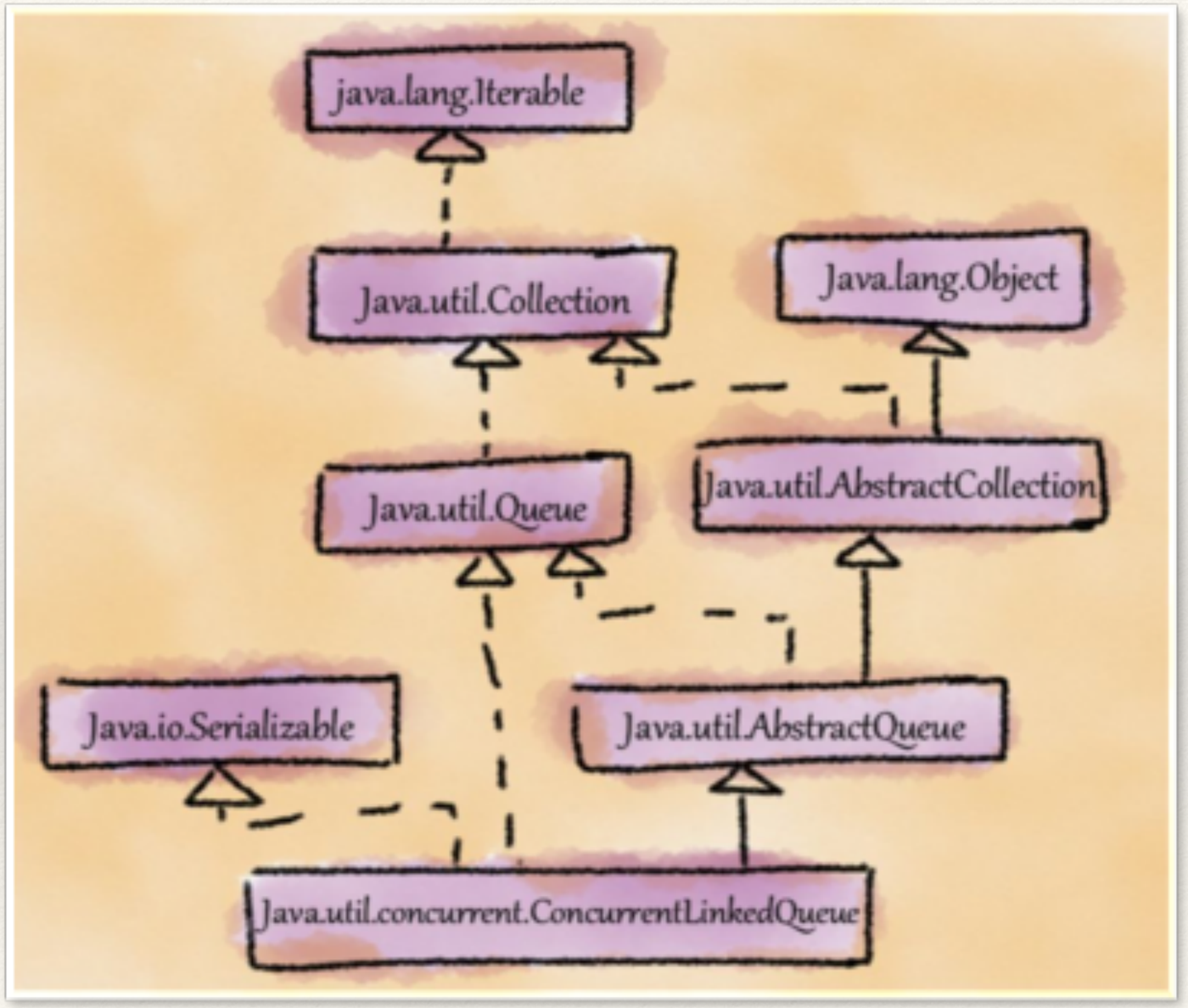
What's that smell?



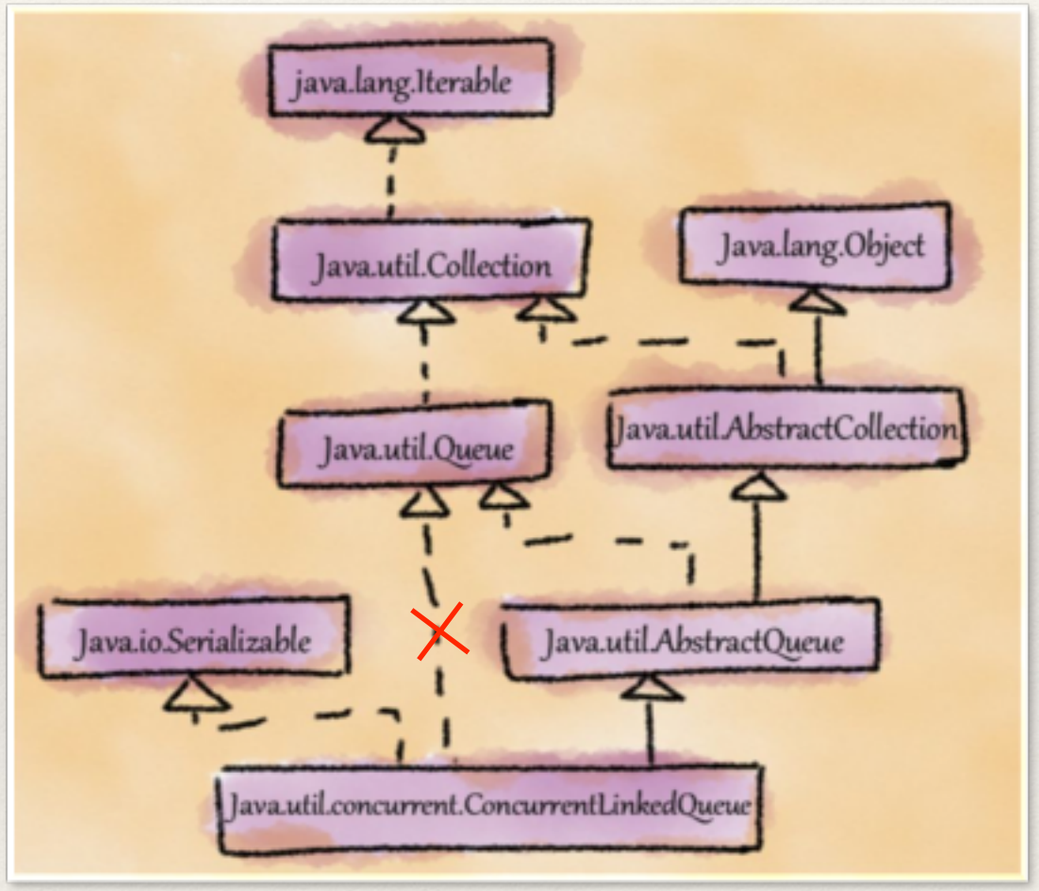
Refactoring “speculative generality”



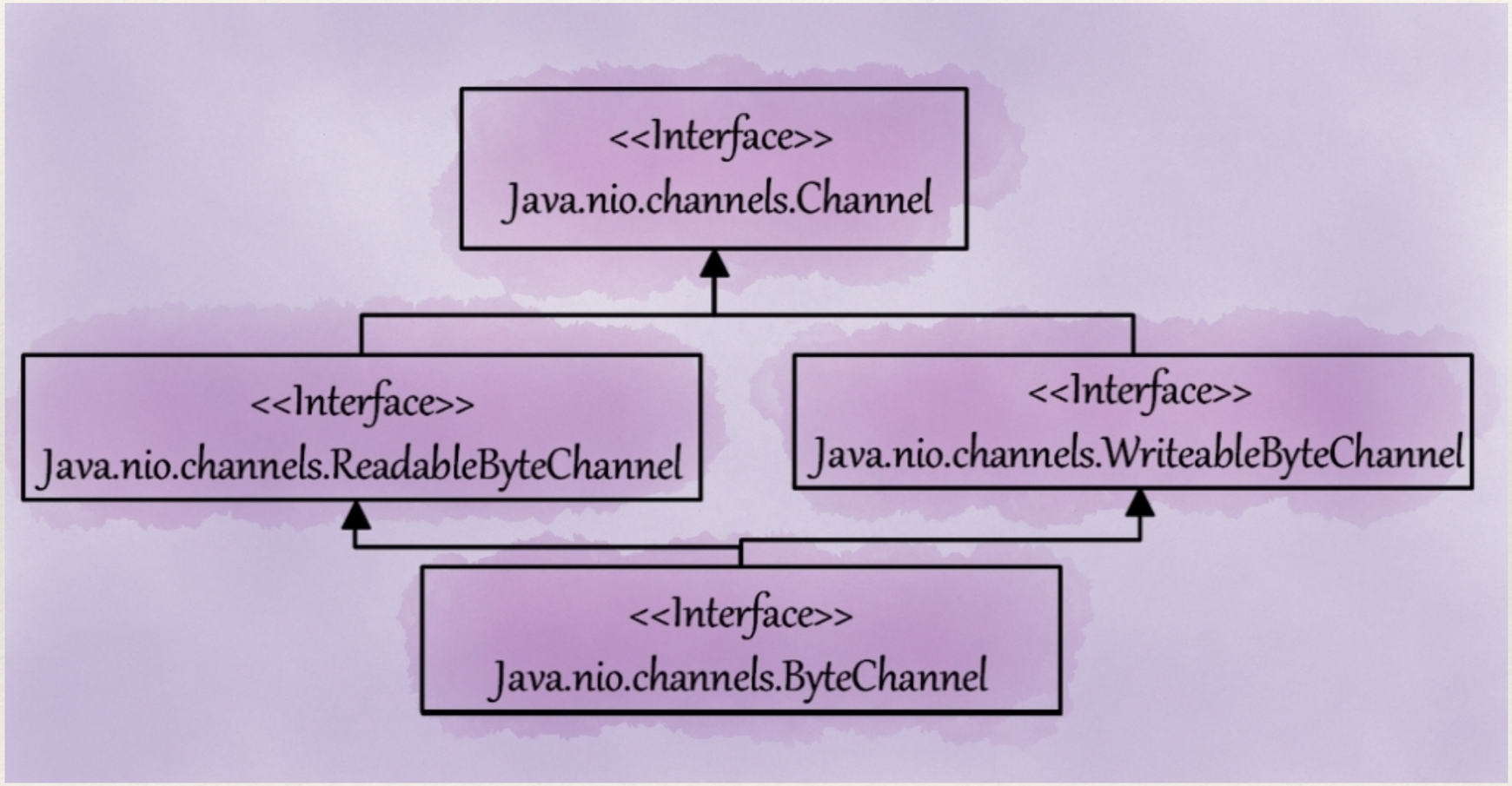
What's that smell?



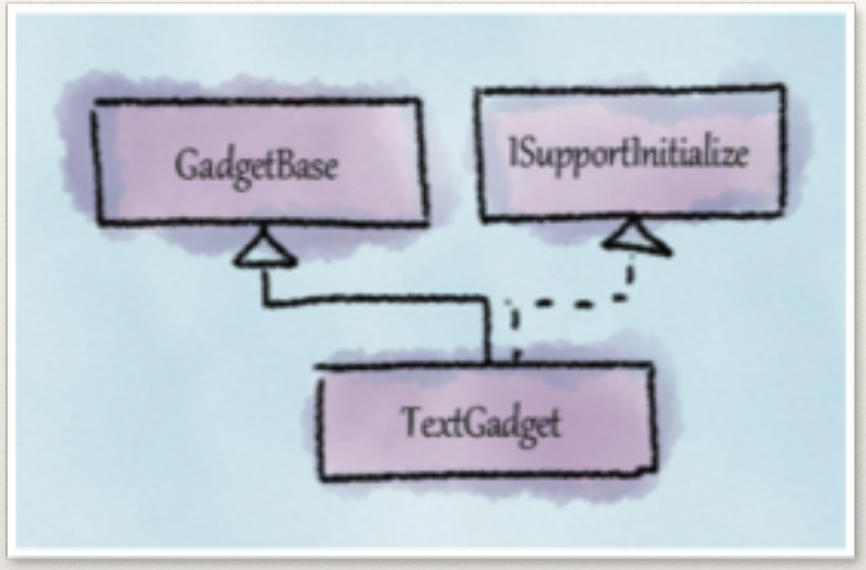
Refactoring

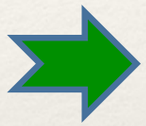


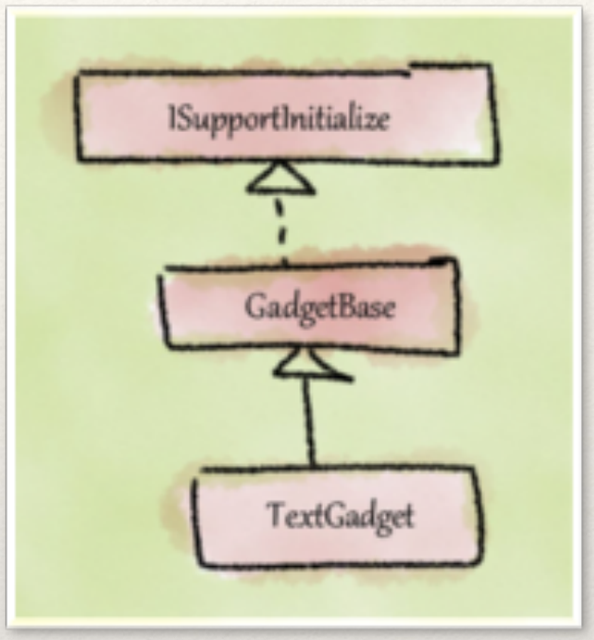
How about multiple interface inheritance?

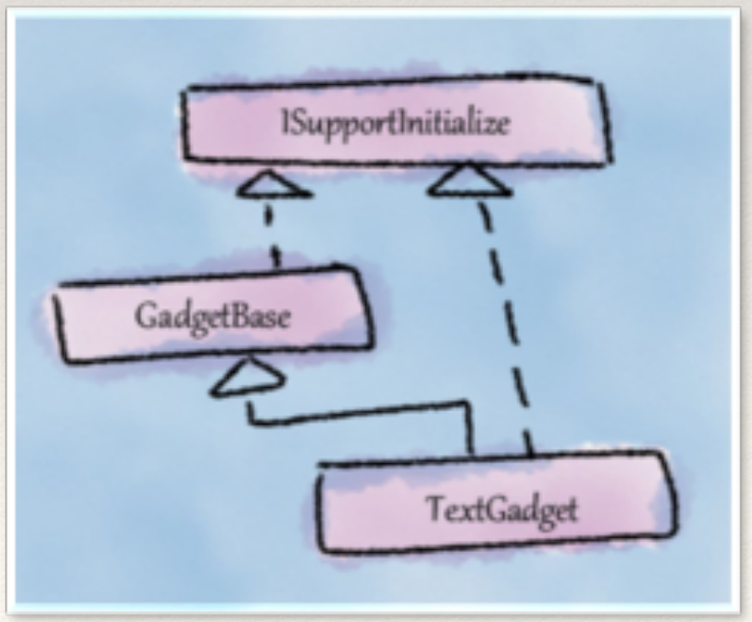


Case study


GadgetBase
ISupportInitialize
TextGadget




ISupportInitialize
GadgetBase
TextGadget

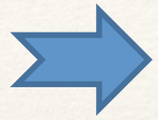

ISupportInitialize
GadgetBase
TextGadget

What's that smell?

```
switch (transferType) {
        case DataBuffer.TYPE_BYTE:
                byte bdata[] = (byte[])inData;
                pixel = bdata[0] & 0xff;
                length = bdata.length;
                break;
        case DataBuffer.TYPE_USHORT:
                short sdata[] = (short[])inData;
                pixel = sdata[0] & 0xffff;
                length = sdata.length;
                break;
        case DataBuffer.TYPE_INT:
                int idata[] = (int[])inData;
                pixel = idata[0];
                length = idata.length;
                break;
        default:
                throw new UnsupportedOperationException("This method has not been "+ "implemented
                for transferType " + transferType);
}
```

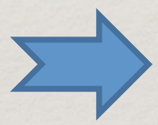
Replace conditional with polymorphism

protected int transferType;



protected DataBuffer dataBuffer;

```
switch (transferType) {
        case DataBuffer.TYPE_BYTE:
                byte bdata[] = (byte[])inData;
                pixel = bdata[0] & 0xff;
                length = bdata.length;
                break;
        case DataBuffer.TYPE_USHORT:
                short sdata[] = (short[])inData;
                pixel = sdata[0] & 0xffff;
                length = sdata.length;
                break;
        case DataBuffer.TYPE_INT:
                int idata[] = (int[])inData;
                pixel = idata[0];
                length = idata.length;
                break;
        default:
                throw new UnsupportedOperationException("This method has not been "+ "implemented for transferType " + transferType);
}
```



```
pixel = dataBuffer.getPixel();
length = dataBuffer.getSize();
```

Scenario

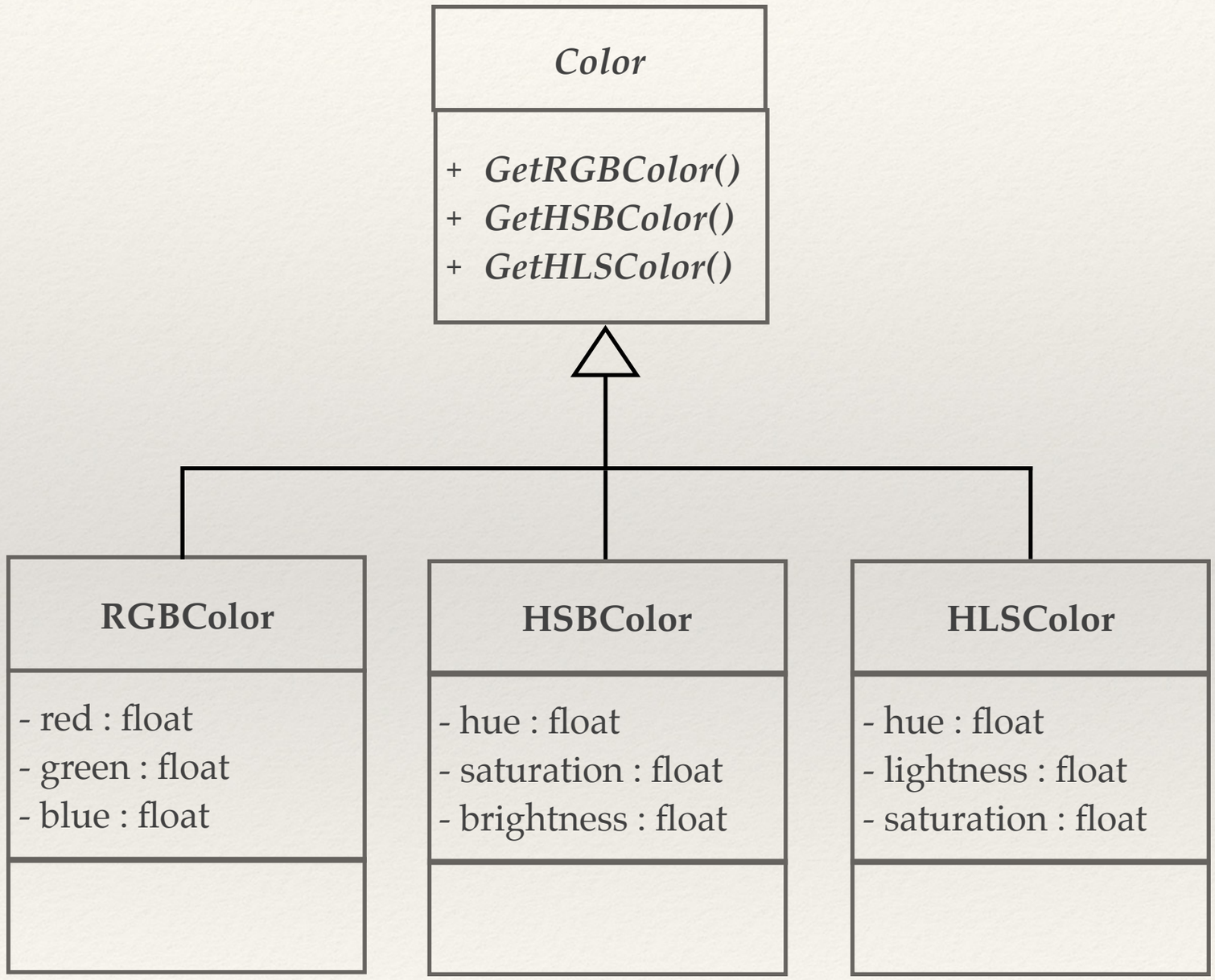
- Assume that you need to support different Color schemes in your software
  - RGB (Red, Green, Blue), HSB (Hue, Saturation, Brightness), and HLS (Hue, Lightness, and Saturation) schemes
- Overloading constructors and differentiating them using enums can become confusing
- What could be a better design?

```
enum ColorScheme { RGB, HSB, HLS, CMYK }
class Color {
        private float red, green, blue;                         // for supporting RGB scheme
        private float hue1, saturation1, brightness1;       // for supporting HSB scheme
        private float hue2, lightness2, saturation2;        // for supporting HLS scheme
        public Color(float arg1, float arg2, float arg3, ColorScheme  cs) {
                switch (cs) {
                        // initialize arg1, arg2, and arg3 based on ColorScheme value
                }
        }
}
```

Hands-on exercise

- Refactor Color.java
- Hint: Use “factory method” design pattern

A solution using factory method pattern



Factory method pattern: Discussion

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses.

- A class cannot anticipate the class of objects it must create
- A class wants its subclasses to specify the objects it creates

- Delegate the responsibility to one of the several helper subclasses
- Also, localize the knowledge of which subclass is the delegate

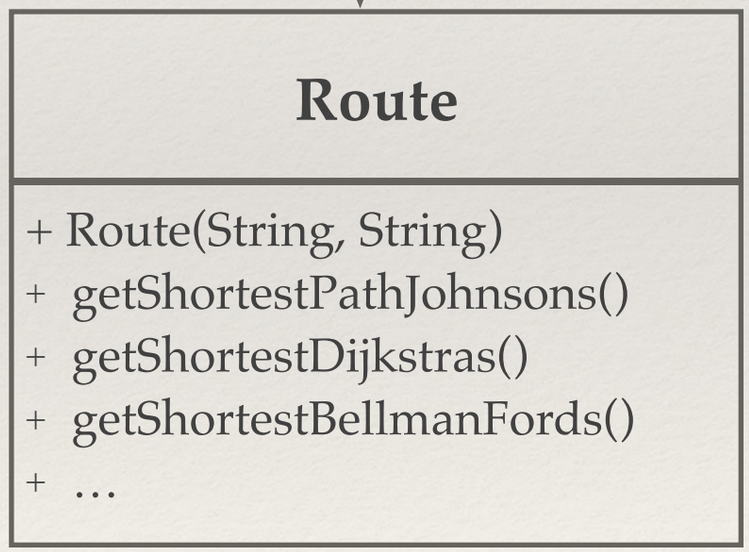
Factory method: Java library example

```
Logger logger = Logger.getLogger(TestFileLogger.class.getName());
```

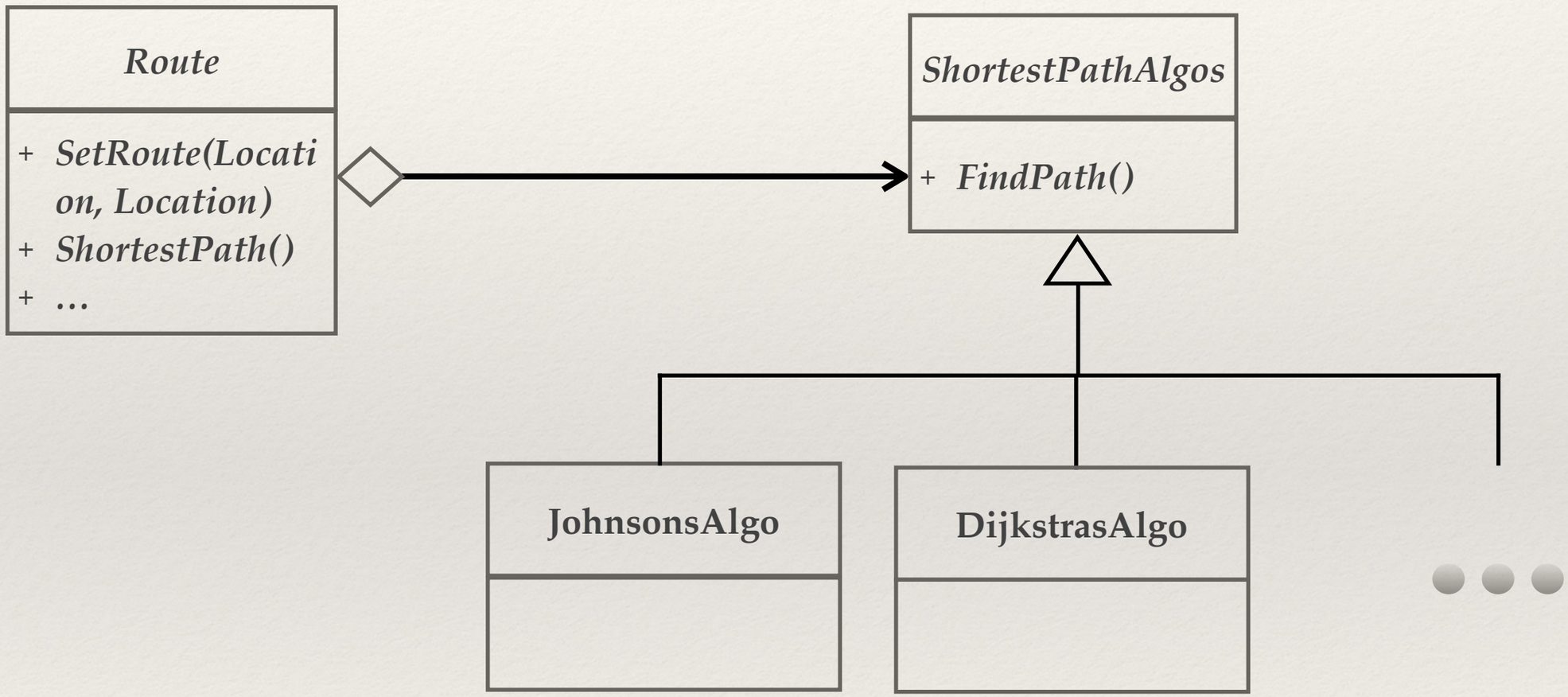
Scenario

- Consider a Route class in an application like Google Maps
- For finding shortest path from source to destination, many algorithms can be used
- The problem is that these algorithms get embedded into Route class and cannot be reused easily (smell!)

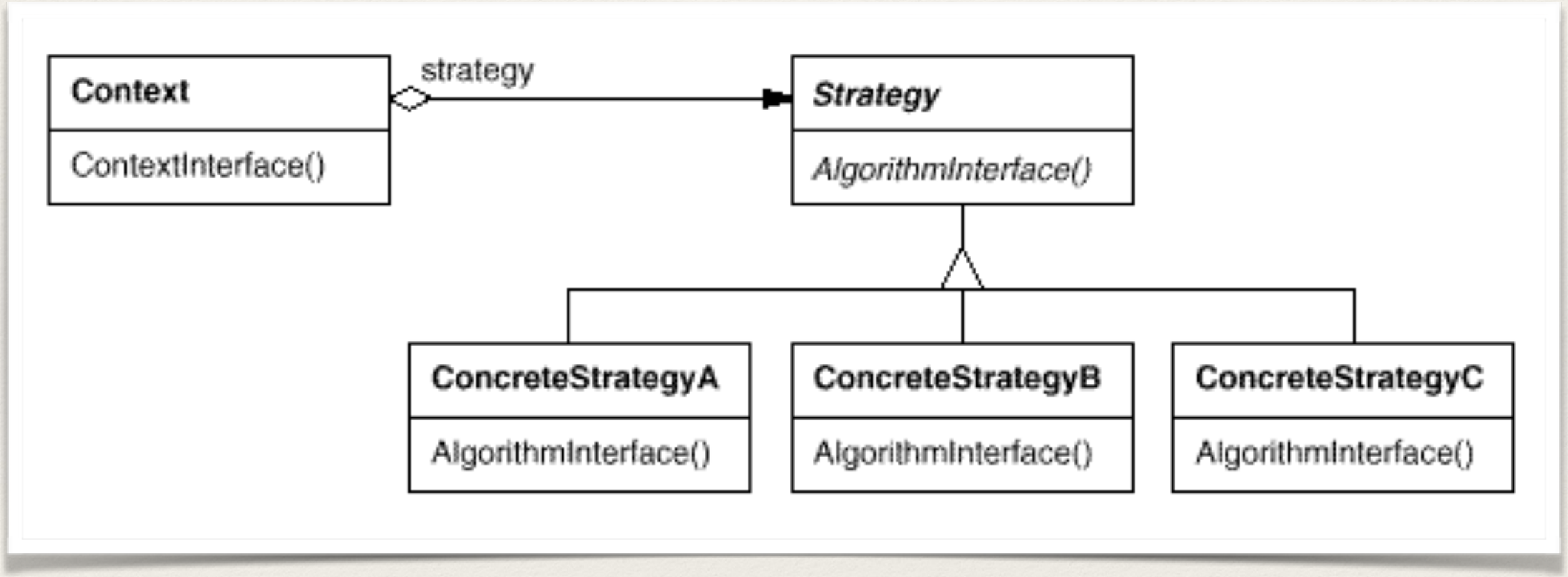
How will you refactor such that
a) Support for shortest path algorithm can be added easily?
b) Separate path finding logic from dealing with location information.



How about this solution?



You're right: Its Strategy pattern!

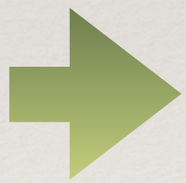


Source: "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley,1994

Strategy pattern: Discussion

Defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it

- Useful when there is a set of related algorithms and a client object needs to be able to dynamically pick and choose an algorithm that suits its current need



- The implementation of each of the algorithms is kept in a separate class referred to as a strategy.
- An object that uses a Strategy object is referred to as a context object.
- Changing the behavior of a Context object is a matter of changing its Strategy object to the one that implements the required algorithm

Scenario

```
class Plus extends Expr {
        private Expr left, right;
        public Plus(Expr arg1, Expr arg2) {
                left = arg1;
                right = arg2;
        }
        public void genCode() {
                left.genCode();
                right.genCode();
                if(t == Target.JVM) {
                        System.out.println("iadd");
                }
                else { / / DOTNET
                        System.out.println("add");
                }
        }
}
```

How to separate:

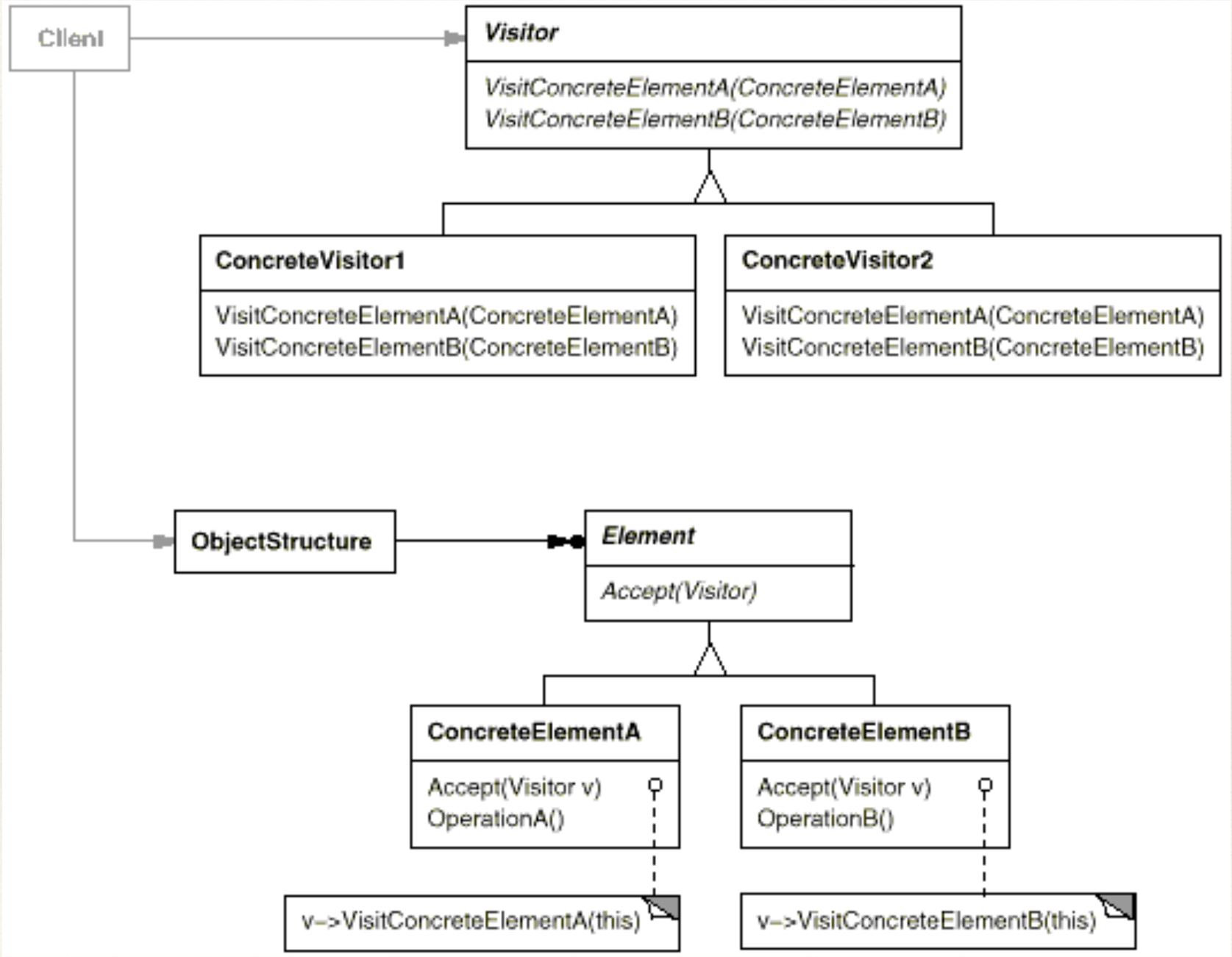
a) code generation logic from node types?
b) how to support different target types?

A solution using Visitor pattern

```
class Plus extends Expr {
        private Expr left, right;
        public Plus(Expr arg1, Expr arg2) {
                left = arg1;
                right = arg2;
        }
        public Expr getLeft() {
                return left;
        }
        public Expr getRight() {
                return right;
        }
        public void accept(Visitor v) {
                v.visit(this);
        }
}
```

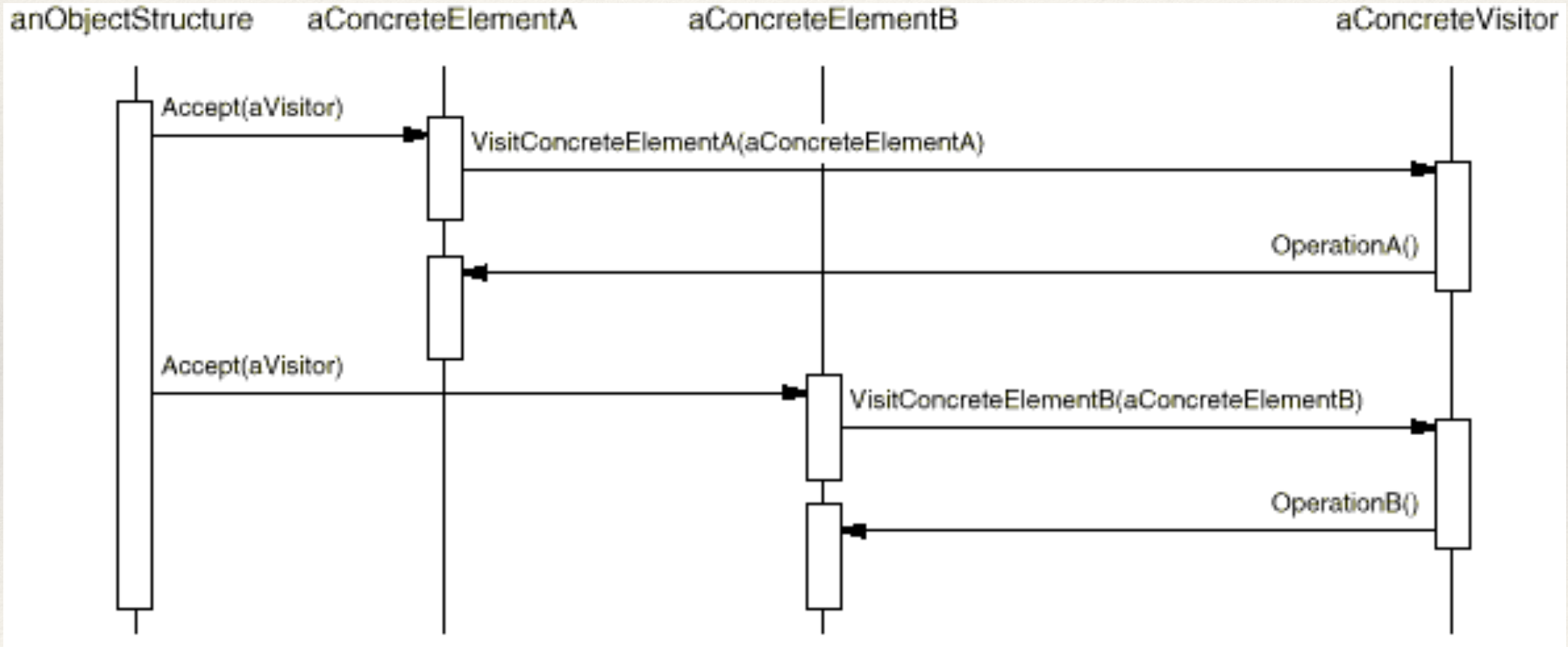
```
class DOTNETVisitor extends Visitor {
        public void visit(Constant arg) {
                System.out.println("ldarg " + arg.getVal());
        }
        public void visit(Plus plus) {
                genCode(plus.getLeft());
                genCode(plus.getRight());
                System.out.println("add");
        }
        public void visit(Sub sub) {
                genCode(sub.getLeft());
                genCode(sub.getRight());
                System.out.println("sub");
        }
        public void genCode(Expr expr) {
                expr.accept(this);
        }
}
```

Visitor pattern: structure



Source: "Design Patterns: Elements of Reusable Object-Oriented Software", Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley,1994

Visitor pattern: call sequence

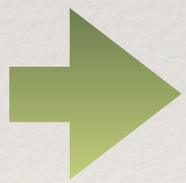


Source: “Design Patterns: Elements of Reusable Object-Oriented Software”, Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides, Addison-Wesley,1994

Visitor pattern: Discussion

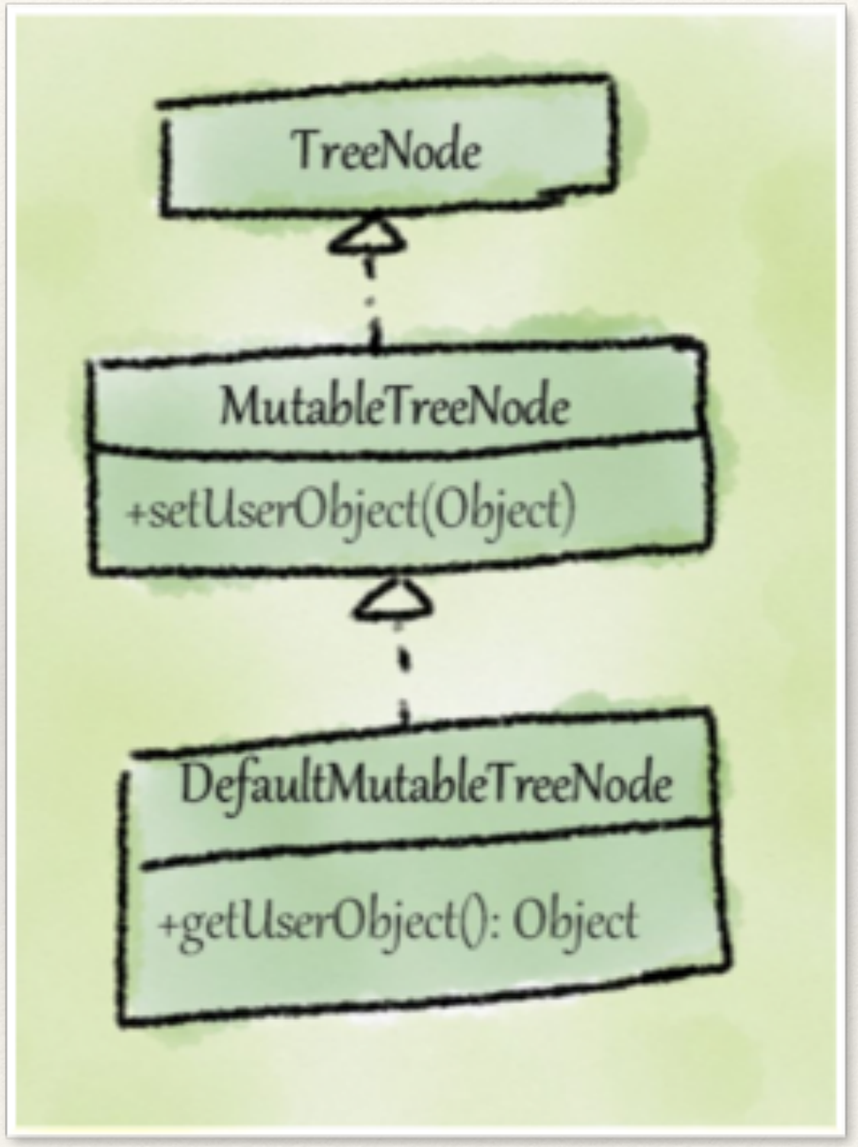
Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operations

- Many distinct and unrelated operations need to be performed on objects in an object structure, and you want to avoid "polluting" their classes with these operations

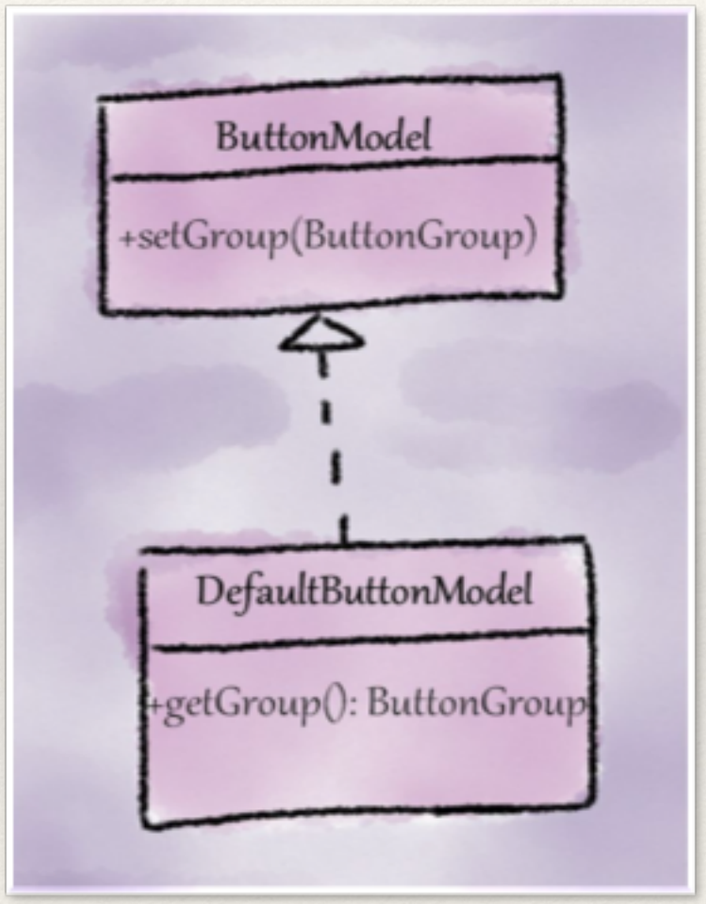


- Create two class hierarchies:
  - One for the elements being operated on
  - One for the visitors that define operations on the elements

What's that smell?



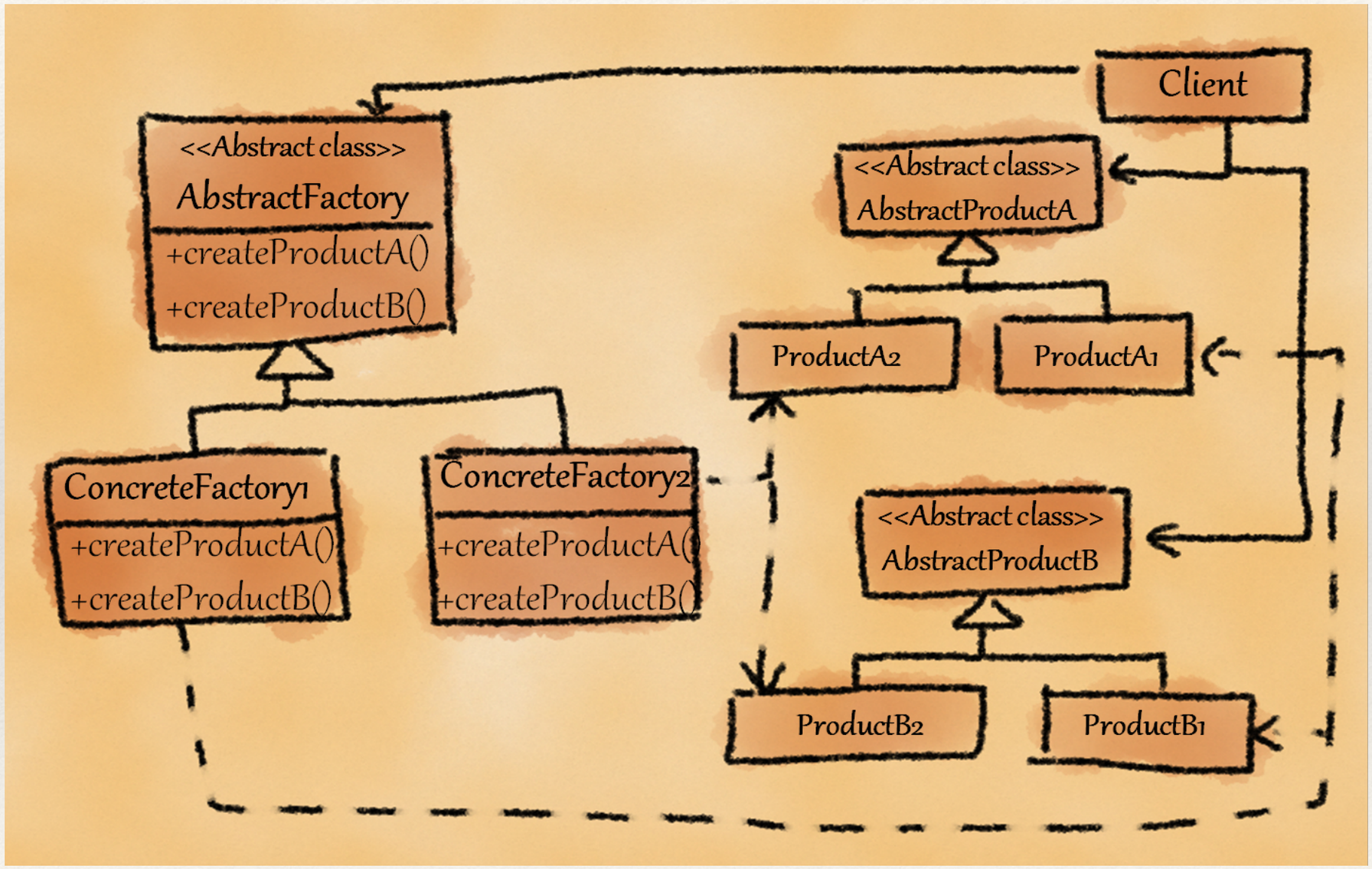
What's that smell?



Refactoring "incomplete library classes" smell

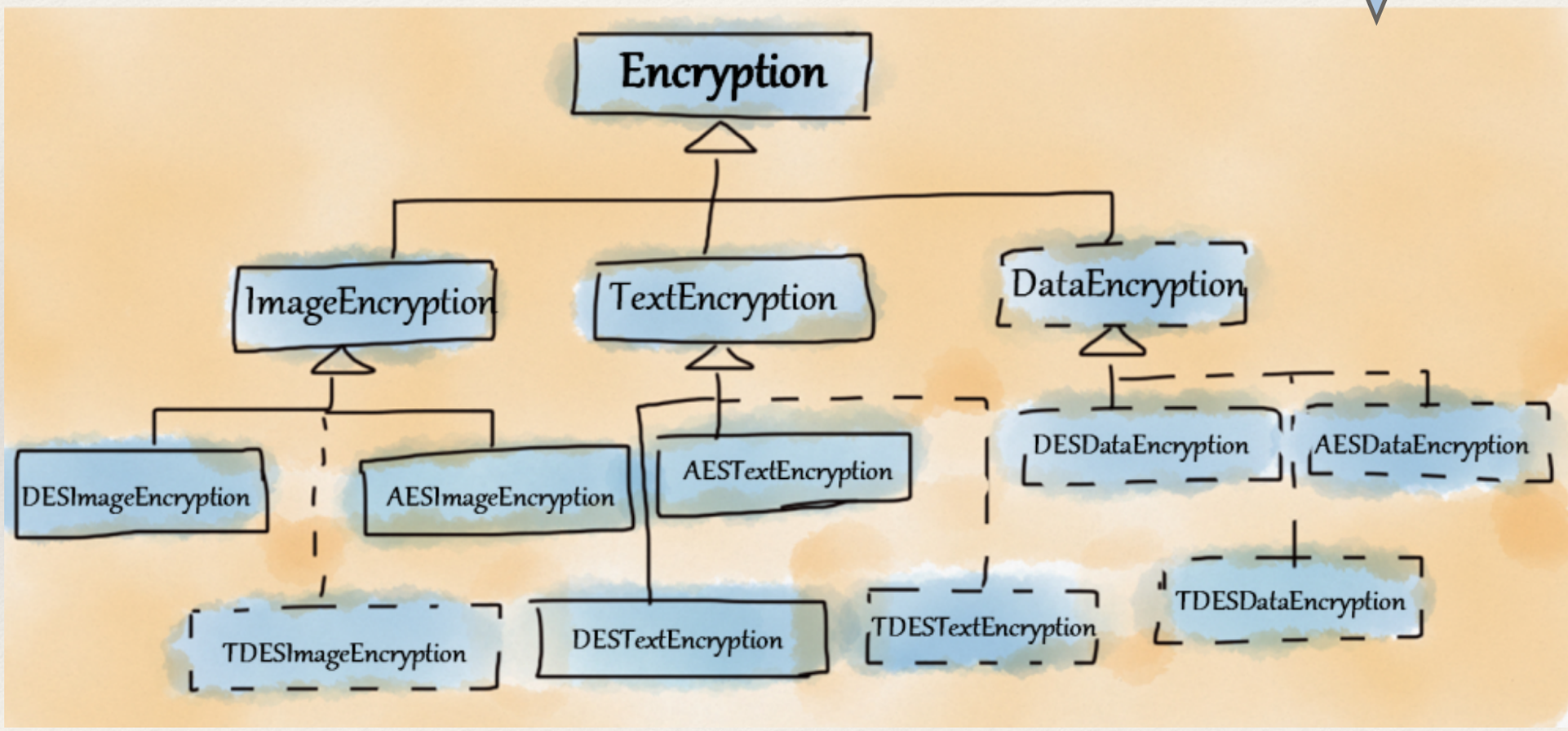
min/max	open/close	create/destroy	get/set
read/write	print/scan	first/last	begin/end
start/stop	lock/unlock	show/hide	up/down
source/target	insert/delete	first/last	push/pull
enable/disable	acquire/release	left/right	on/off

Abstract factory pattern

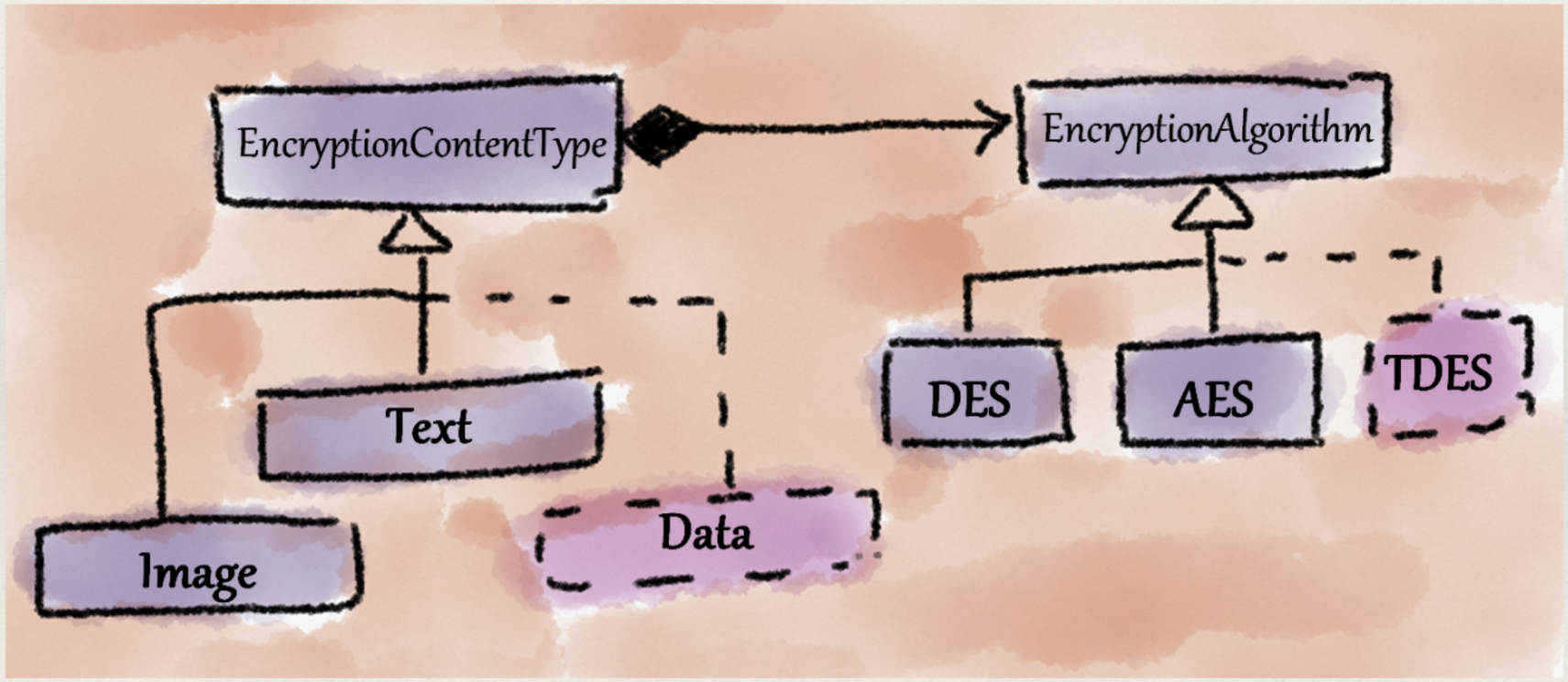


What's that smell?

"Combination explosion" smell



Refactoring using Bridge structure



Refactoring with Lambdas

```
File[] files = new File(".").listFiles(
     new FileFilter() {
          public boolean accept(File f) { return f.isFile(); }
     }
);
for(File file: files) {
     System.out.println(file);
}
```

```
Arrays.stream(new File(“.”)
     .listFiles(file -> file.isFile()))
     .forEach(System.out::println);
```

Refactoring APIs

```
public class Throwable {
        // following method is available from Java 1.0 version.
        // Prints the stack trace as a string to standard output
        // for processing a stack trace,
        // we need to write regular expressions
        public void printStackTrace();
        // other methods omitted
}
```

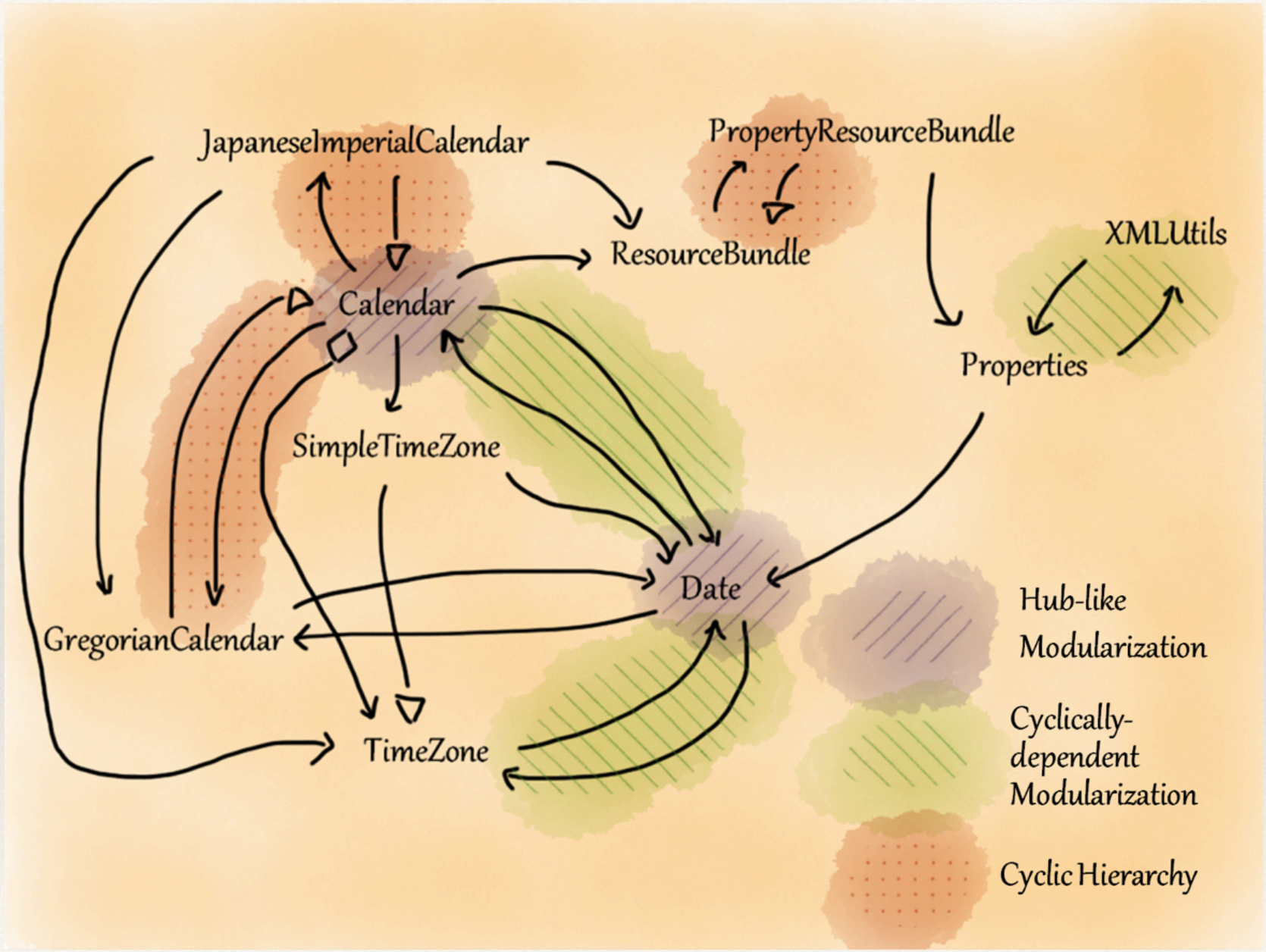
Refactoring APIs

```
public class Throwable {
        // following method is available from Java 1.0 version.
        // Prints the stack trace as a string to standard output
        // for processing a stack trace,
        // we need to write regular expressions
        public void printStackTrace();
        // other methods omitted
}
```

```
public class Throwable {
        public void printStackTrace();
        public StackTraceElement[] getStackTrace(); // Since 1.4
        // other methods omitted
}
```

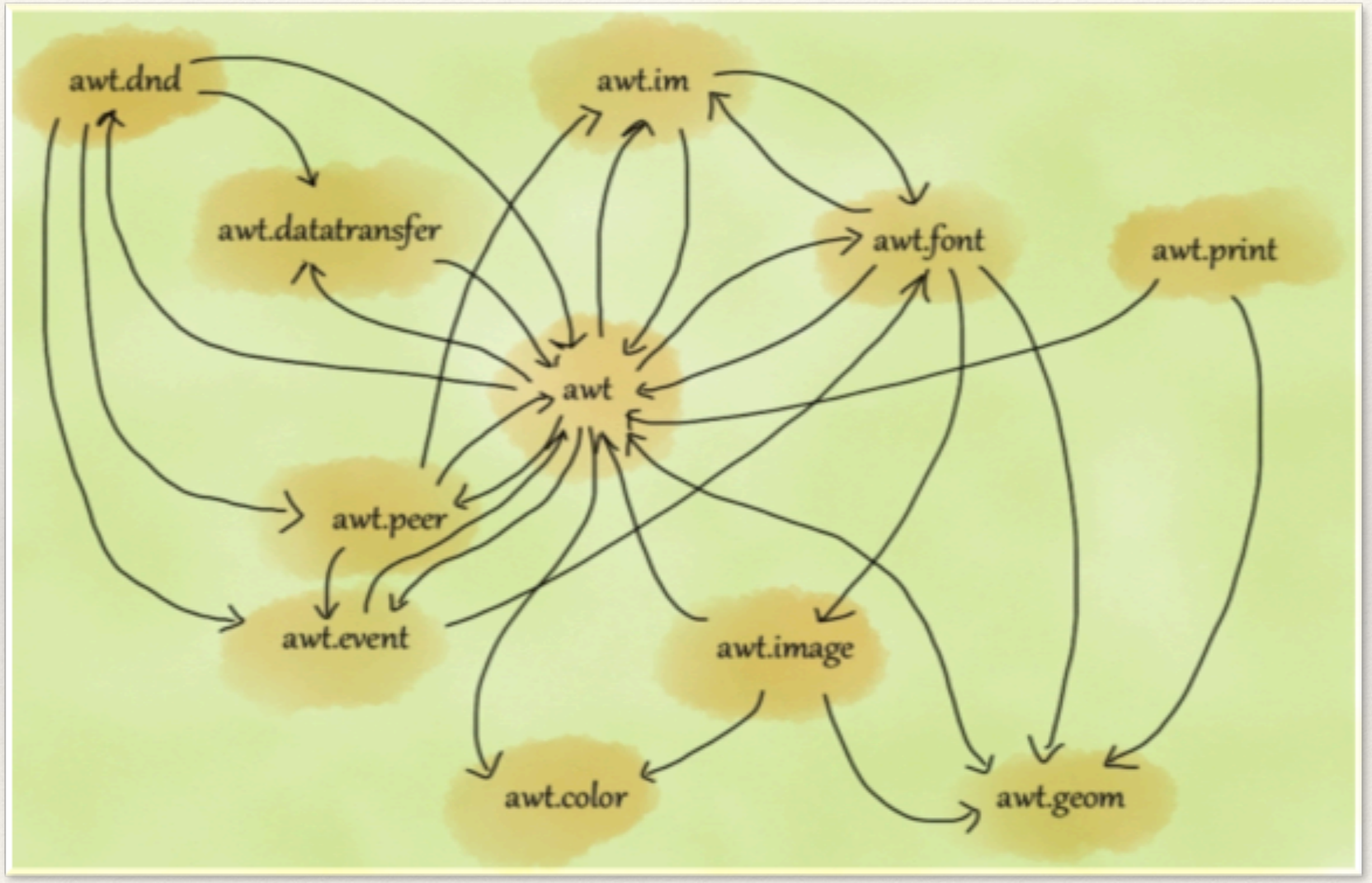
```
public final class StackTraceElement {
        public String getFileName();
        public int getLineNumber();
        public String getClassName();
        public String getMethodName();
        public boolean isNativeMethod();
}
```

Smells tend to “co-occur”



Amplification

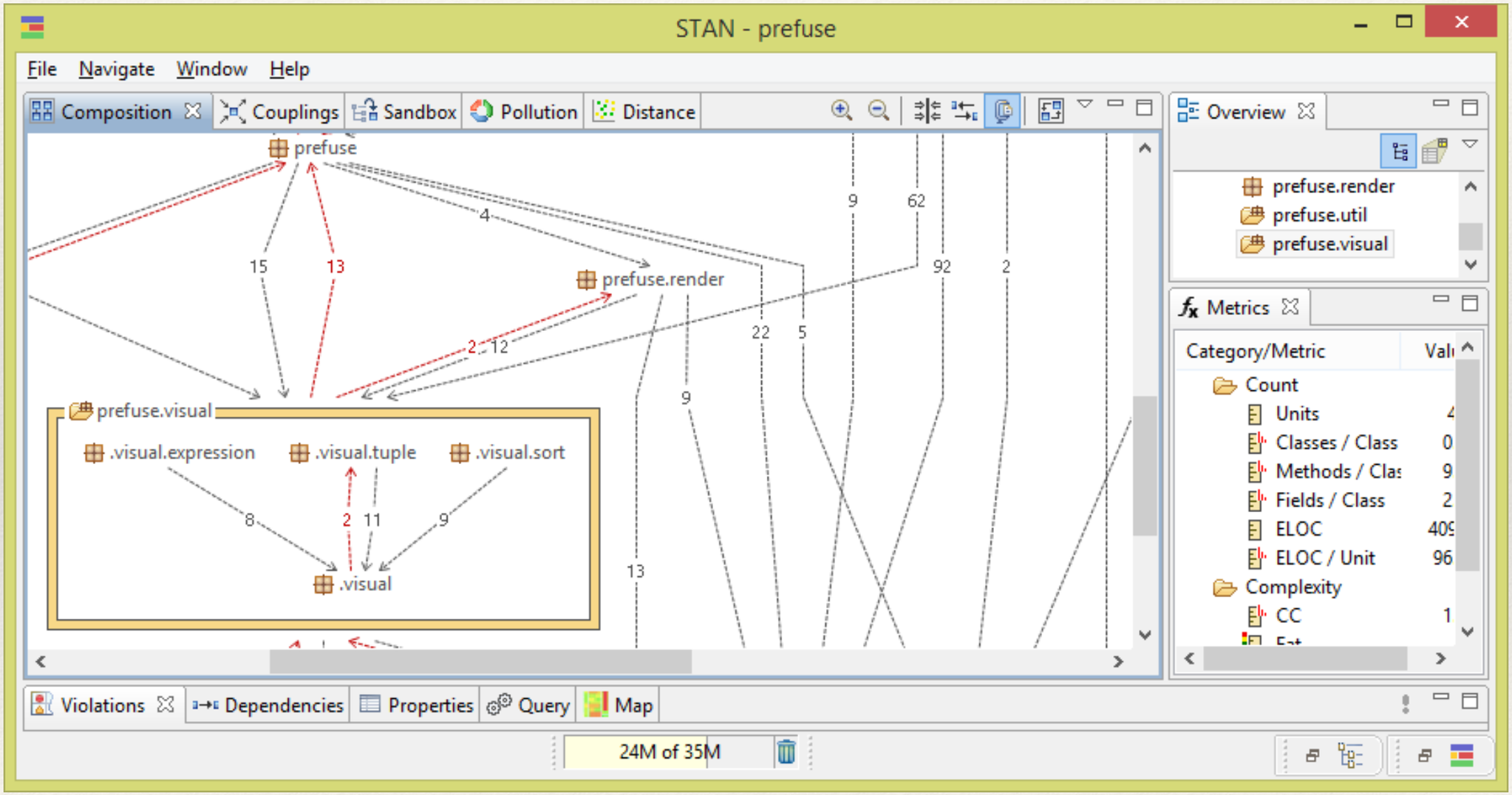
Architecture smells



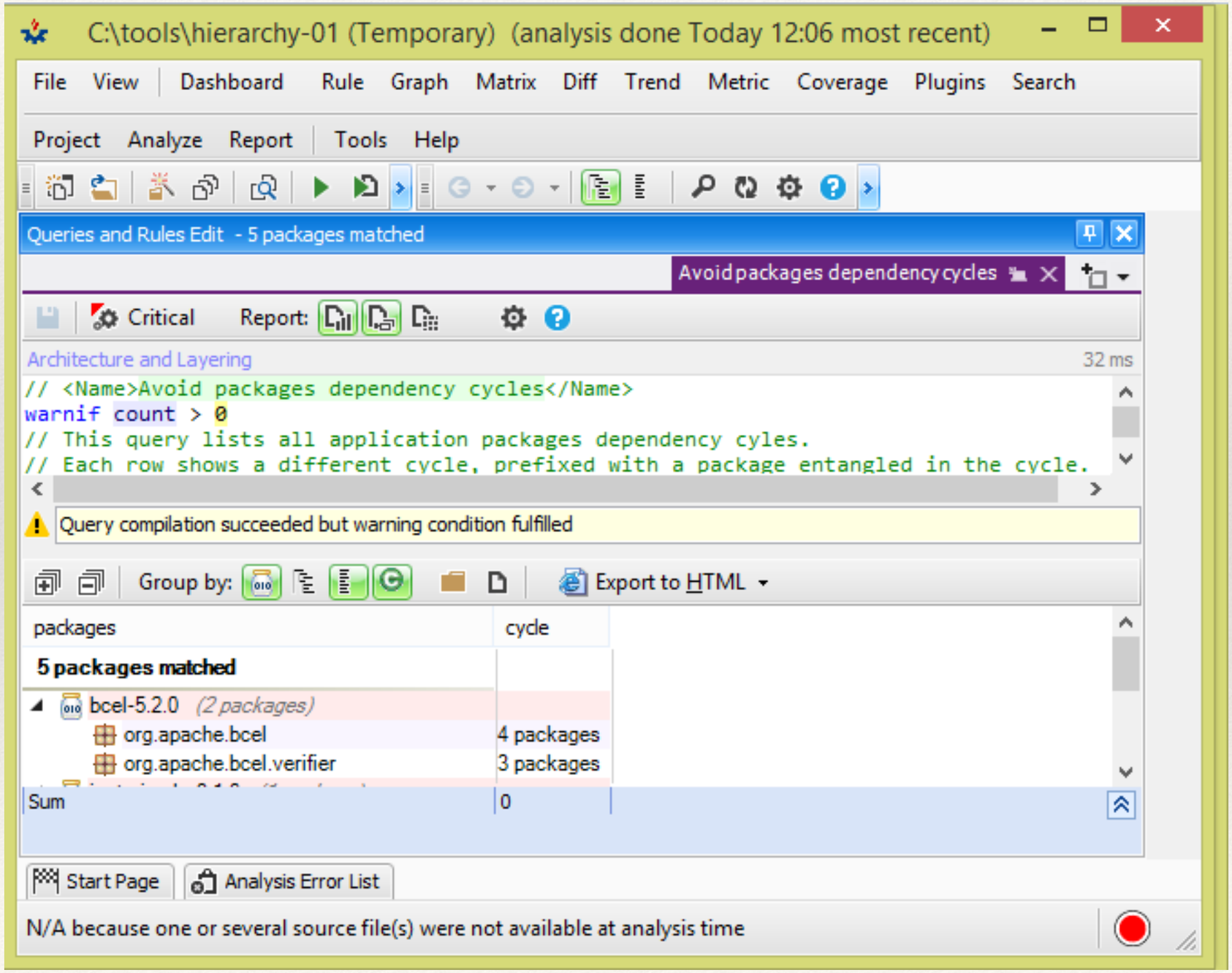
Agenda

- Refactoring Foundations
- Refactoring - Principles
- Refactoring Bad Smells
- **Refactoring Tools**

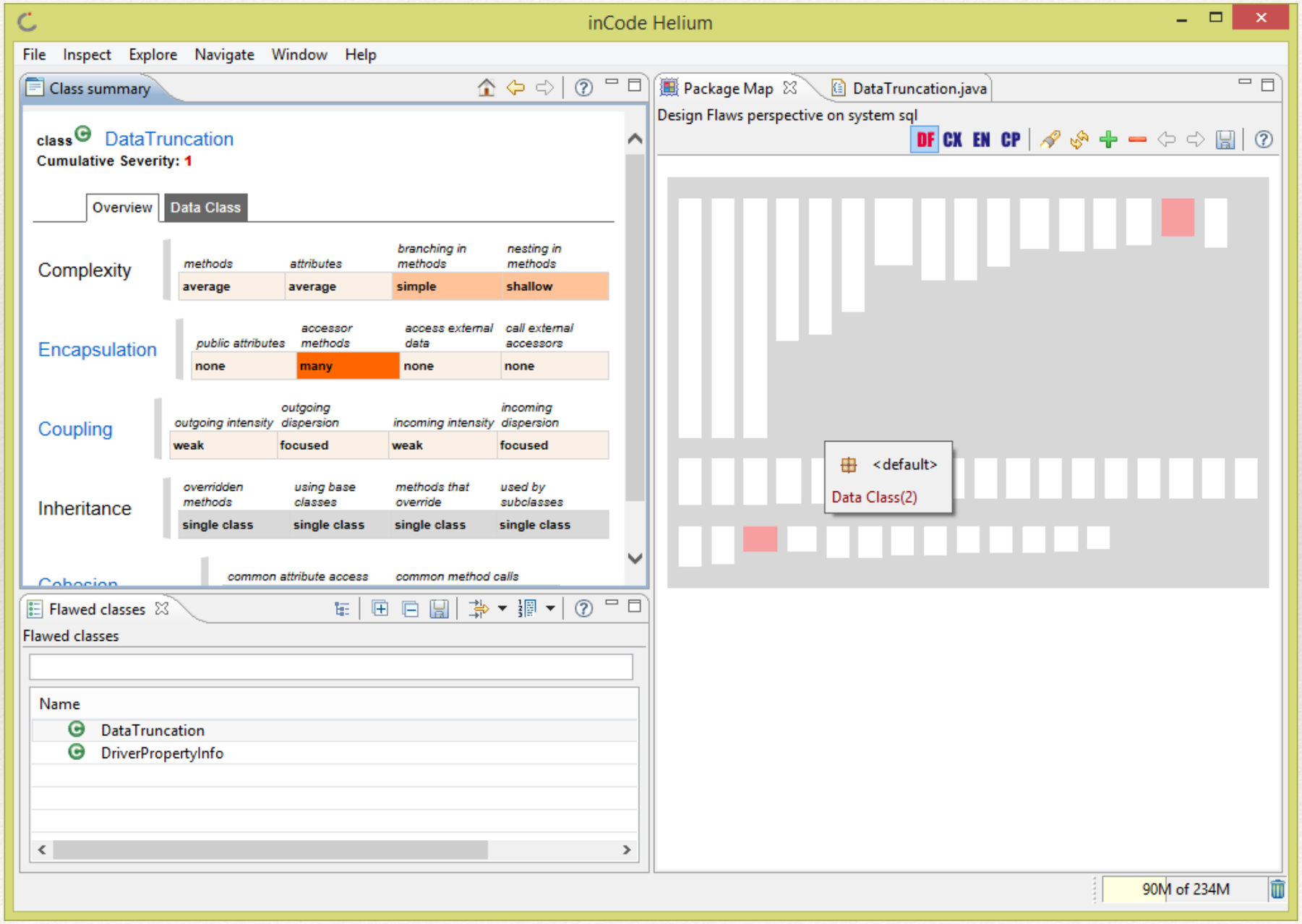
Structural Analysis for Java (stan4j)



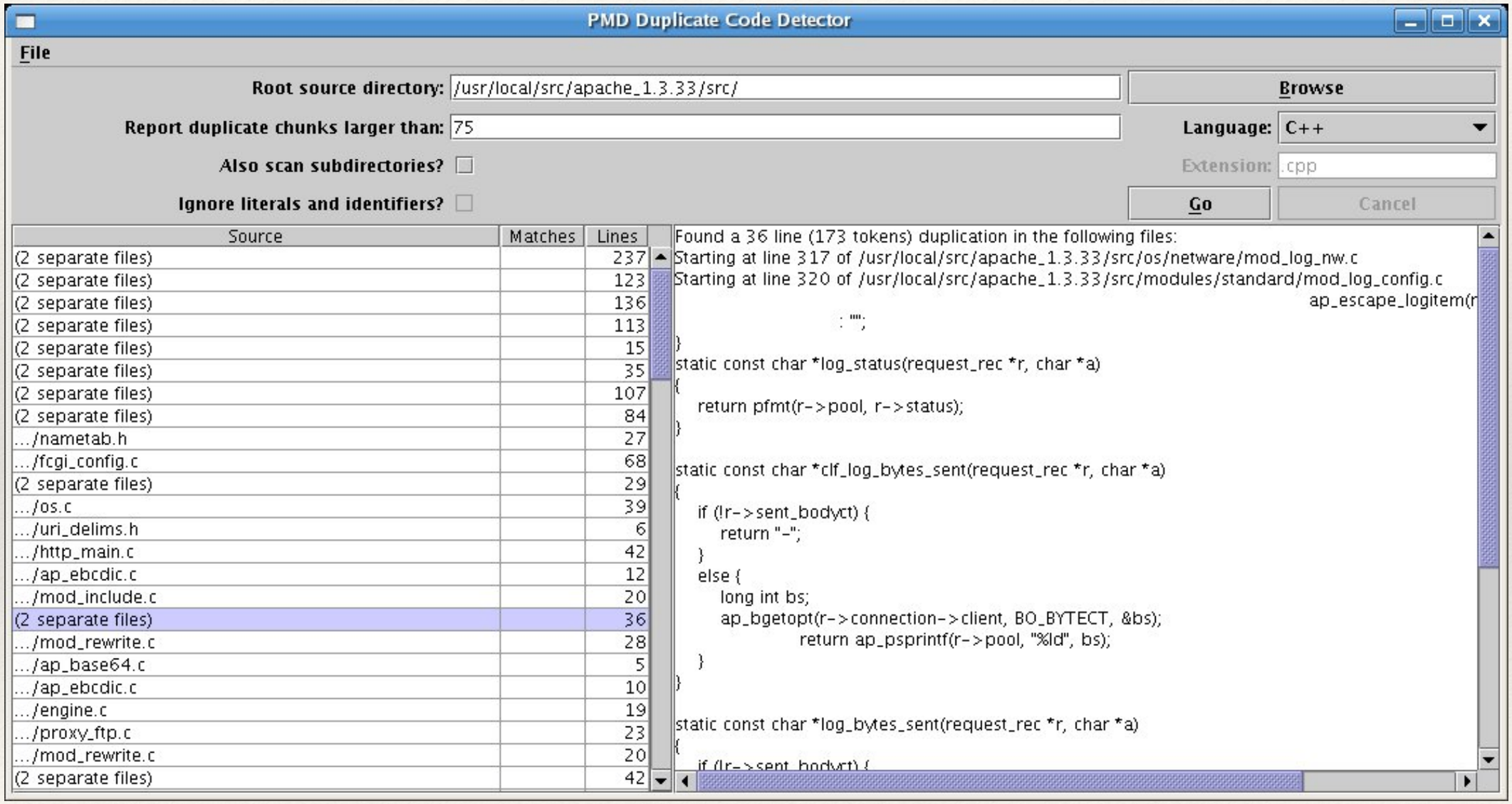
JArchitect


C:\tools\hierarchy-01 (Temporary) (analysis done Today 12:06 most recent)
File View Dashboard Rule Graph Matrix Diff Trend Metric Coverage Plugins Search
Project Analyze Report Tools Help
Queries and Rules Edit - 5 packages matched
Avoid packages dependency cycles
Critical Report:
Architecture and Layering
32 ms
// <Name>Avoid packages dependency cycles</Name>
warnif count > 0
// This query lists all application packages dependency cyles.
// Each row shows a different cycle, prefixed with a package entangled in the cycle.
Query compilation succeeded but warning condition fulfilled
Group by:
Export to HTML
packages
cycle
5 packages matched
bcel-5.2.0 (2 packages)
org.apache.bcel
4 packages
org.apache.bcel.verifier
3 packages
Sum
0
Start Page
Analysis Error List
N/A because one or several source file(s) were not available at analysis time

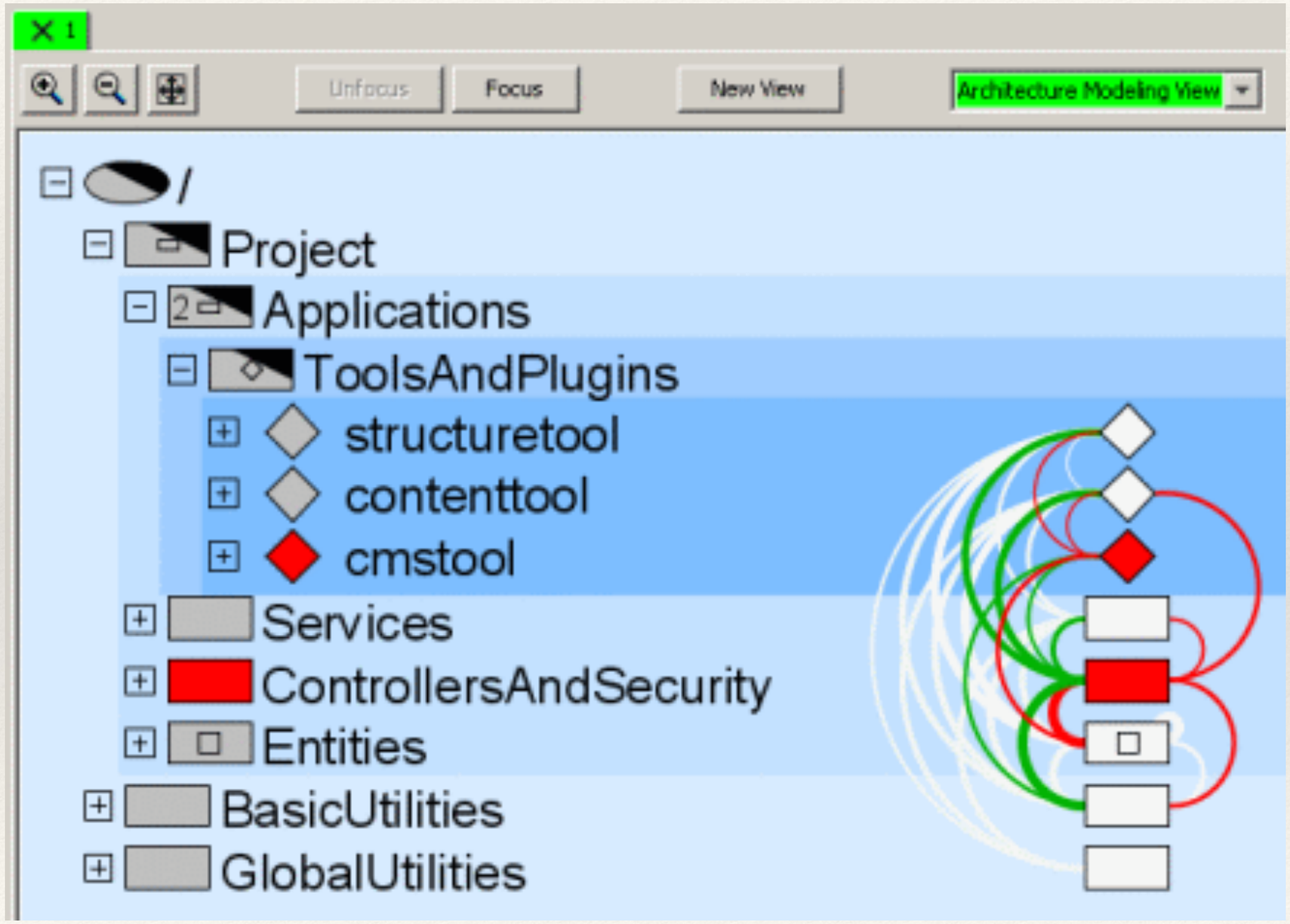
InFusion


inCode Helium
File Inspect Explore Navigate Window Help
Class summary
class DataTruncation
Cumulative Severity: 1
Overview
Data Class
Complexity
methods
attributes
branching in methods
nesting in methods
average
average
simple
shallow
Encapsulation
public attributes
accessor methods
access external data
call external accessors
none
many
none
none
Coupling
outgoing intensity
outgoing dispersion
incoming intensity
incoming dispersion
weak
focused
weak
focused
Inheritance
overridden methods
using base classes
methods that override
used by subclasses
single class
single class
single class
single class
Cohesion
common attribute access
common method calls
Flawed classes
Flawed classes
Name
DataTruncation
DriverPropertyInfo
Package Map
DataTruncation.java
Design Flaws perspective on system sql
DF CX EN CP
<default>
Data Class(2)
90M of 234M

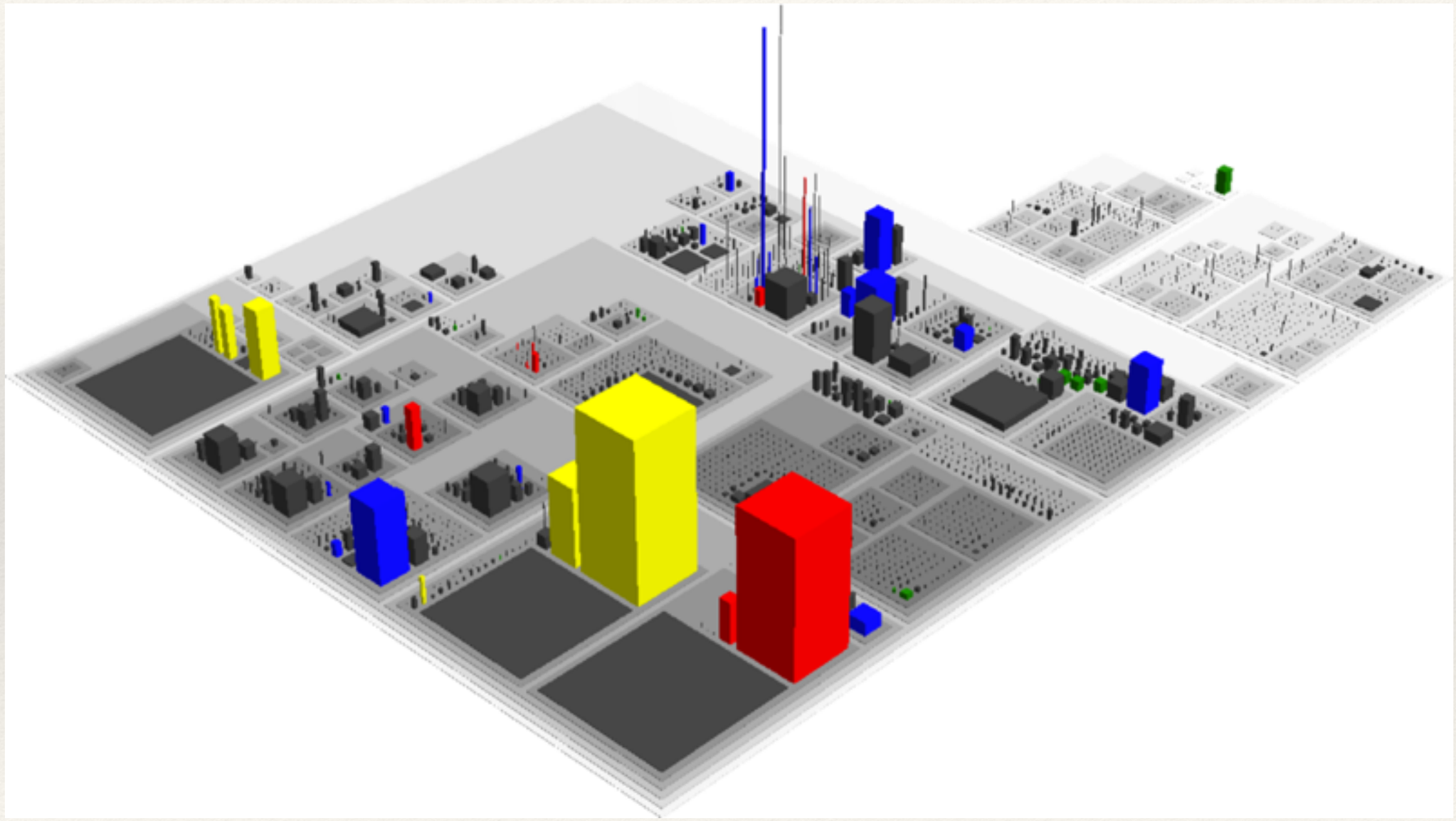
PMD CPD


PMD Duplicate Code Detector
File
Root source directory: /usr/local/src/apache_1.3.33/src/
Browse
Report duplicate chunks larger than: 75
Language: C++
Also scan subdirectories?
Extension: .cpp
Ignore literals and identifiers?
Go
Cancel
Source
Matches
Lines
(2 separate files) 237
(2 separate files) 123
(2 separate files) 136
(2 separate files) 113
(2 separate files) 15
(2 separate files) 35
(2 separate files) 107
(2 separate files) 84
.../nametab.h 27
.../fcgi_config.c 68
(2 separate files) 29
.../os.c 39
.../uri_delims.h 6
.../http_main.c 42
.../ap_ebcdic.c 12
.../mod_include.c 20
(2 separate files) 36
.../mod_rewrite.c 28
.../ap_base64.c 5
.../ap_ebcdic.c 10
.../engine.c 19
.../proxy_ftp.c 23
.../mod_rewrite.c 20
(2 separate files) 42
Found a 36 line (173 tokens) duplication in the following files:
Starting at line 317 of /usr/local/src/apache_1.3.33/src/os/netware/mod_log_nw.c
Starting at line 320 of /usr/local/src/apache_1.3.33/src/modules/standard/mod_log_config.c
ap_escape_logitem(r
: "";
}
static const char *log_status(request_rec *r, char *a)
{
return pfmt(r->pool, r->status);
}
static const char *clf_log_bytes_sent(request_rec *r, char *a)
{
if (!r->sent_bodyct) {
return "-";
}
else {
long int bs;
ap_bgetopt(r->connection->client, BO_BYTECT, &bs);
return ap_psprintf(r->pool, "%ld", bs);
}
}
static const char *log_bytes_sent(request_rec *r, char *a)
{

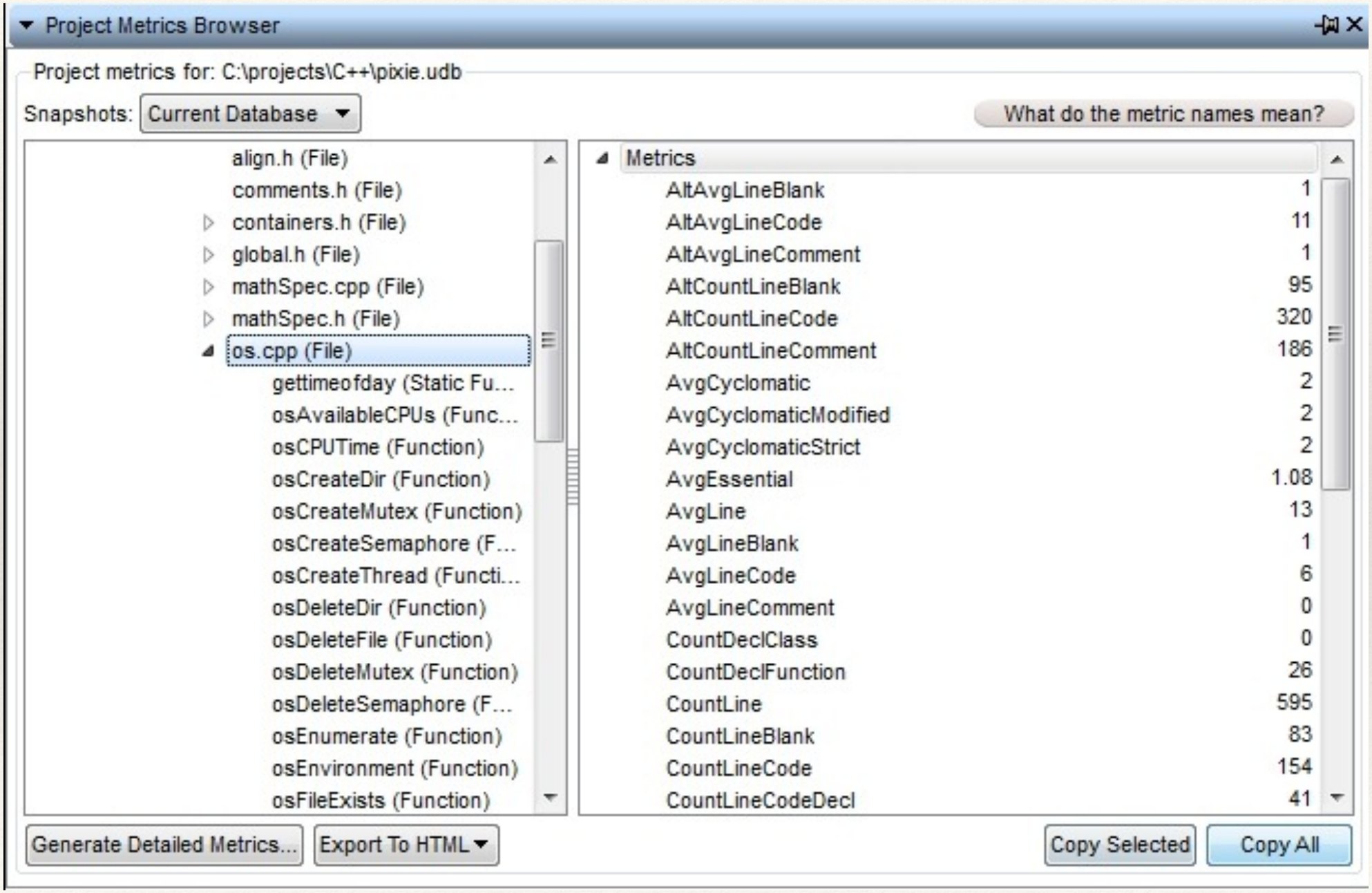
SotoArc


1
Unfocus
Focus
New View
Architecture Modeling View
/
Project
Applications
ToolsAndPlugins
structuretool
contenttool
cmstool
Services
ControllersAndSecurity
Entities
BasicUtilities
GlobalUtilities

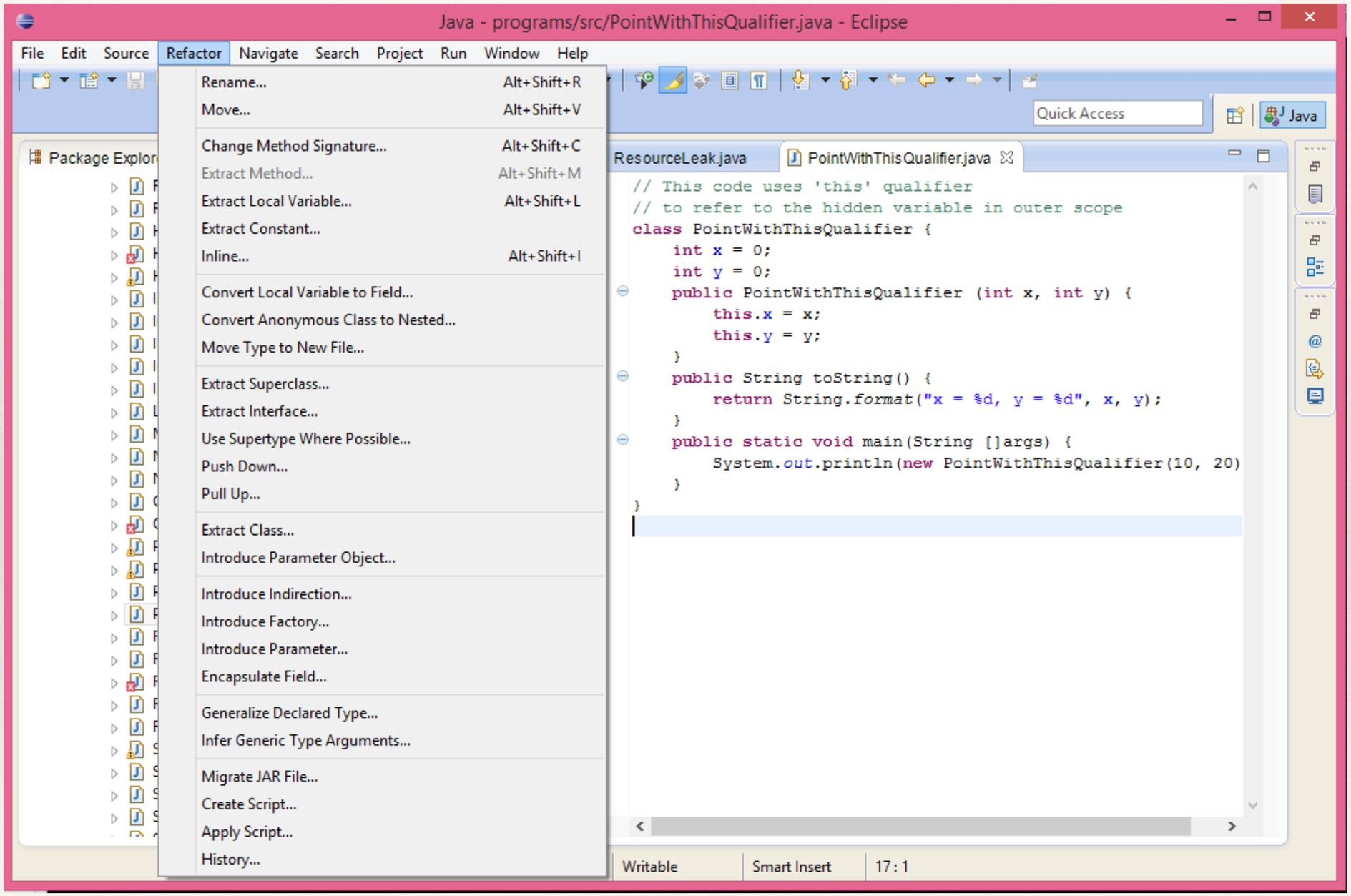
CodeCity



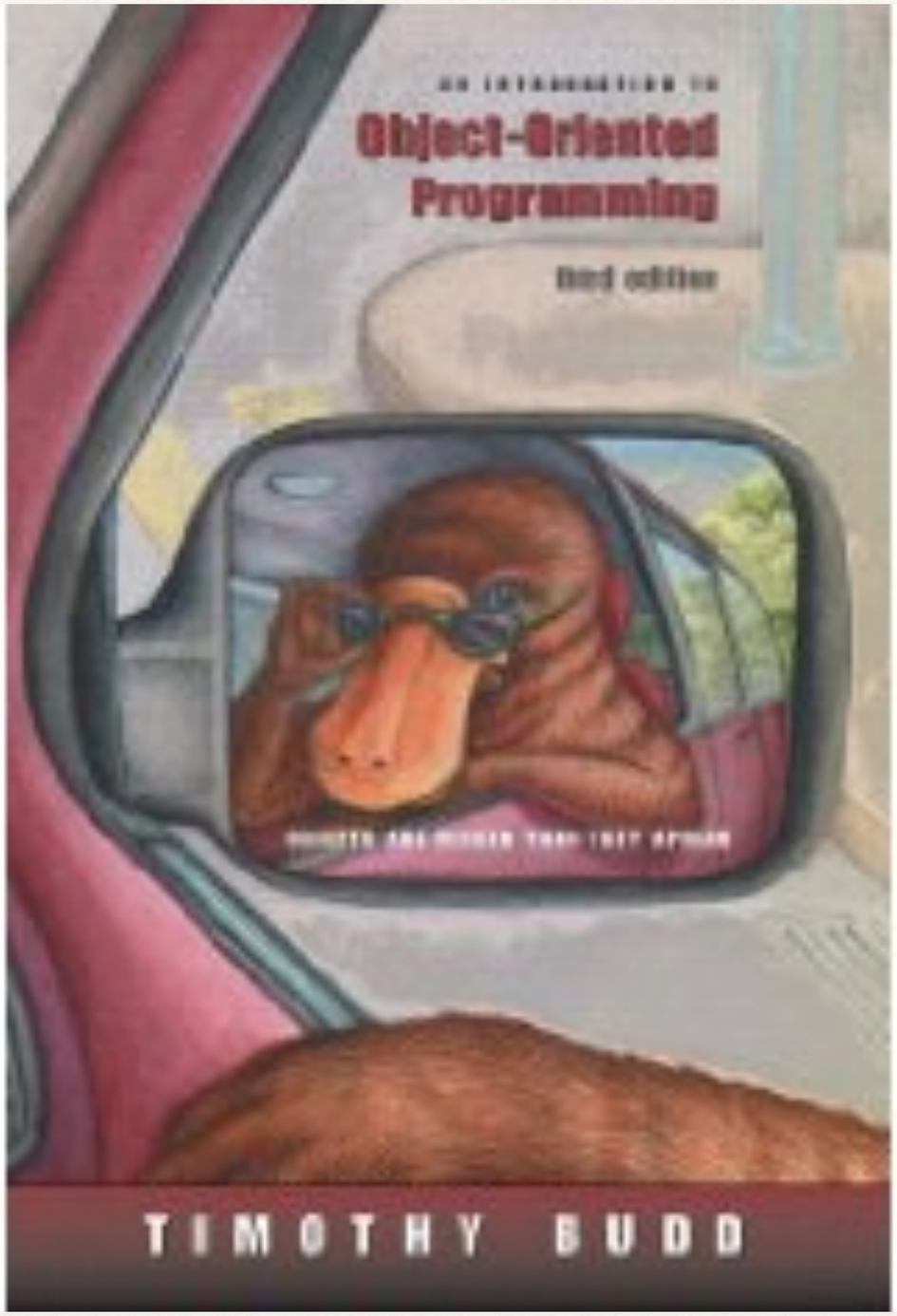
Understand


Project Metrics Browser
Project metrics for: C:\projects\C++\pixie.udb
Snapshots: Current Database
What do the metric names mean?
align.h (File)
comments.h (File)
containers.h (File)
global.h (File)
mathSpec.cpp (File)
mathSpec.h (File)
os.cpp (File)
gettimeofday (Static Fu...
osAvailableCPUs (Func...
osCPUTime (Function)
osCreateDir (Function)
osCreateMutex (Function)
osCreateSemaphore (F...
osCreateThread (Functi...
osDeleteDir (Function)
osDeleteFile (Function)
osDeleteMutex (Function)
osDeleteSemaphore (F...
osEnumerate (Function)
osEnvironment (Function)
osFileExists (Function)
Metrics
AltAvgLineBlank 1
AltAvgLineCode 11
AltAvgLineComment 1
AltCountLineBlank 95
AltCountLineCode 320
AltCountLineComment 186
AvgCyclomatic 2
AvgCyclomaticModified 2
AvgCyclomaticStrict 2
AvgEssential 1.08
AvgLine 13
AvgLineBlank 1
AvgLineCode 6
AvgLineComment 0
CountDeclClass 0
CountDeclFunction 26
CountLine 595
CountLineBlank 83
CountLineCode 154
CountLineCodeDecl 41
Generate Detailed Metrics...
Export To HTML
Copy Selected
Copy All

Eclipse IDE


Java - programs/src/PointWithThisQualifier.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Rename... Alt+Shift+R
Move... Alt+Shift+V
Change Method Signature... Alt+Shift+C
Extract Method... Alt+Shift+M
Extract Local Variable... Alt+Shift+L
Extract Constant...
Inline... Alt+Shift+I
Convert Local Variable to Field...
Convert Anonymous Class to Nested...
Move Type to New File...
Extract Superclass...
Extract Interface...
Use Supertype Where Possible...
Push Down...
Pull Up...
Extract Class...
Introduce Parameter Object...
Introduce Indirection...
Introduce Factory...
Introduce Parameter...
Encapsulate Field...
Generalize Declared Type...
Infer Generic Type Arguments...
Migrate JAR File...
Create Script...
Apply Script...
History...
Quick Access
Java
ResourceLeak.java
PointWithThisQualifier.java
// This code uses 'this' qualifier
// to refer to the hidden variable in outer scope
class PointWithThisQualifier {
int x = 0;
int y = 0;
public PointWithThisQualifier (int x, int y) {
this.x = x;
this.y = y;
}
public String toString() {
return String.format("x = %d, y = %d", x, y);
}
public static void main(String []args) {
System.out.println(new PointWithThisQualifier(10, 20)
}
}
Writable
Smart Insert
17 : 1

Books to read


Object-Oriented
Programming
TIMOTHY BUDD

REFACTORING

IMPROVING THE DESIGN OF EXISTING CODE

MARTIN FOWLER

With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts

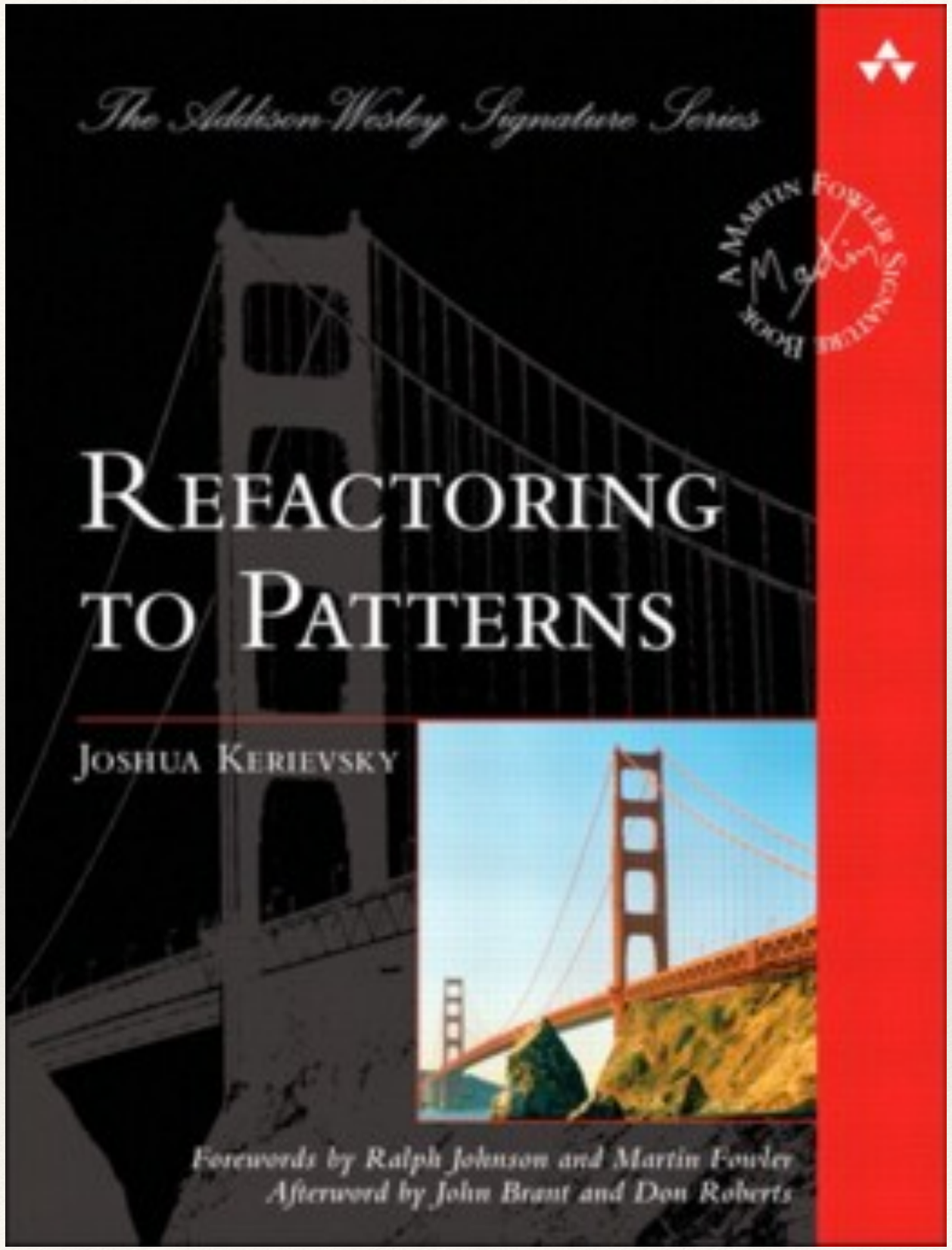
Foreword by Erich Gamma

Object Technology International Inc.

OBJECT TECHNOLOGY SERIES

BOOCH JACOBSON RUMBAUGH

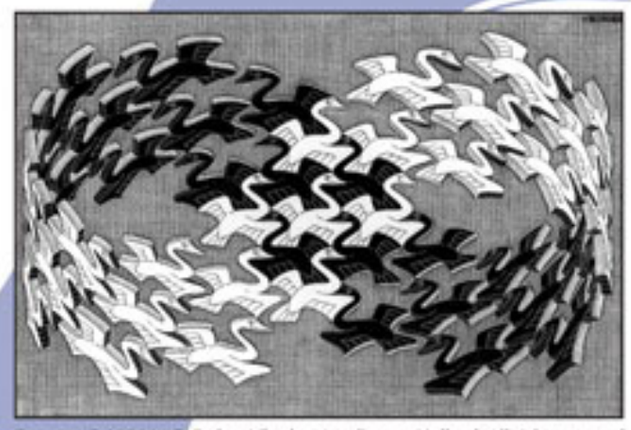
ADDISON-WESLEY SERIES EDITORS


The Addison-Wesley Signature Series
A Martin Fowler Signature Book
Refactoring to Patterns
Joshua Kerievsky
Forewords by Ralph Johnson and Martin Fowler
Afterword by John Brant and Don Roberts

Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides



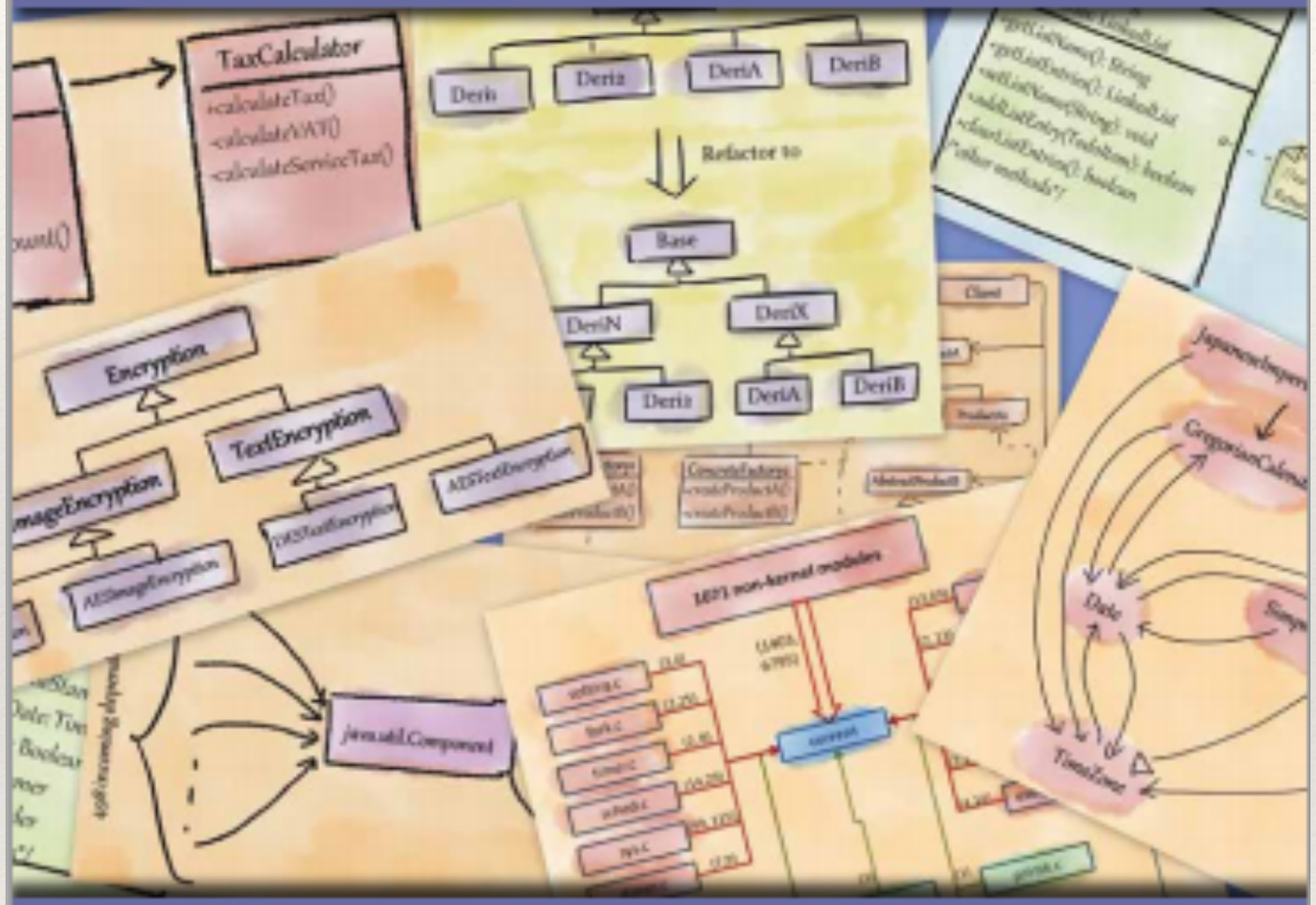
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch

ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Refactoring for Software Design Smells

Managing Technical Debt



Girish Suryanarayana,
Ganesh Samarthyam, Tushar Sharma

"Applying design principles is the key to creating high-quality software!"

Architectural principles:
Axis, symmetry, rhythm, datum, hierarchy, transformation

Image credits

- http://en.wikipedia.org/wiki/Fear_of_missing_out
- http://lesliejanemoran.blogspot.in/2010_05_01_archive.html
- http://javra.eu/wp-content/uploads/2013/07/angry_laptop2.jpg
- https://www.youtube.com/watch?v=5R8XHrfJkeg
- http://womenworld.org/image/052013/31/113745161.jpg
- http://www.fantom-xp.com/wallpapers/33/I'm_not_sure.jpg
- https://www.flickr.com/photos/31457017@N00/453784086
- https://www.gradtouch.com/uploads/images/question3.jpg
- http://gurujohn.files.wordpress.com/2008/06/bookcover0001.jpg
- http://upload.wikimedia.org/wikipedia/commons/d/d5/Martin_Fowler_-_Swipe_Conference_2012.jpg
- http://www.codeproject.com/KB/architecture/csdespat_2/dpcs_br.gif
- http://upload.wikimedia.org/wikipedia/commons/thumb/2/28/Bertrand_Meyer_IMG_2481.jpg/440px-Bertrand_Meyer_IMG_2481.jpg
- http://takeji-soft.up.n.seesaa.net/takeji-soft/image/GOF-OOPLSA-94-Color-75.jpg?d=a0
- https://developer.apple.com/library/ios/documentation/cocoa/Conceptual/OOP_ObjC/Art/watchcalls_35.gif
- http://www.pluspack.com/files/billeder/Newsletter/25/takeaway_bag.png
- http://cdn1.tnwcdn.com/wp-content/blogs.dir/1/files/2013/03/design.jpg

c-nnect

Ganesh Samarthyam

ganesh.samarthyam@gmail.com

@GSamarthyam

http://bit.ly/sgganesh