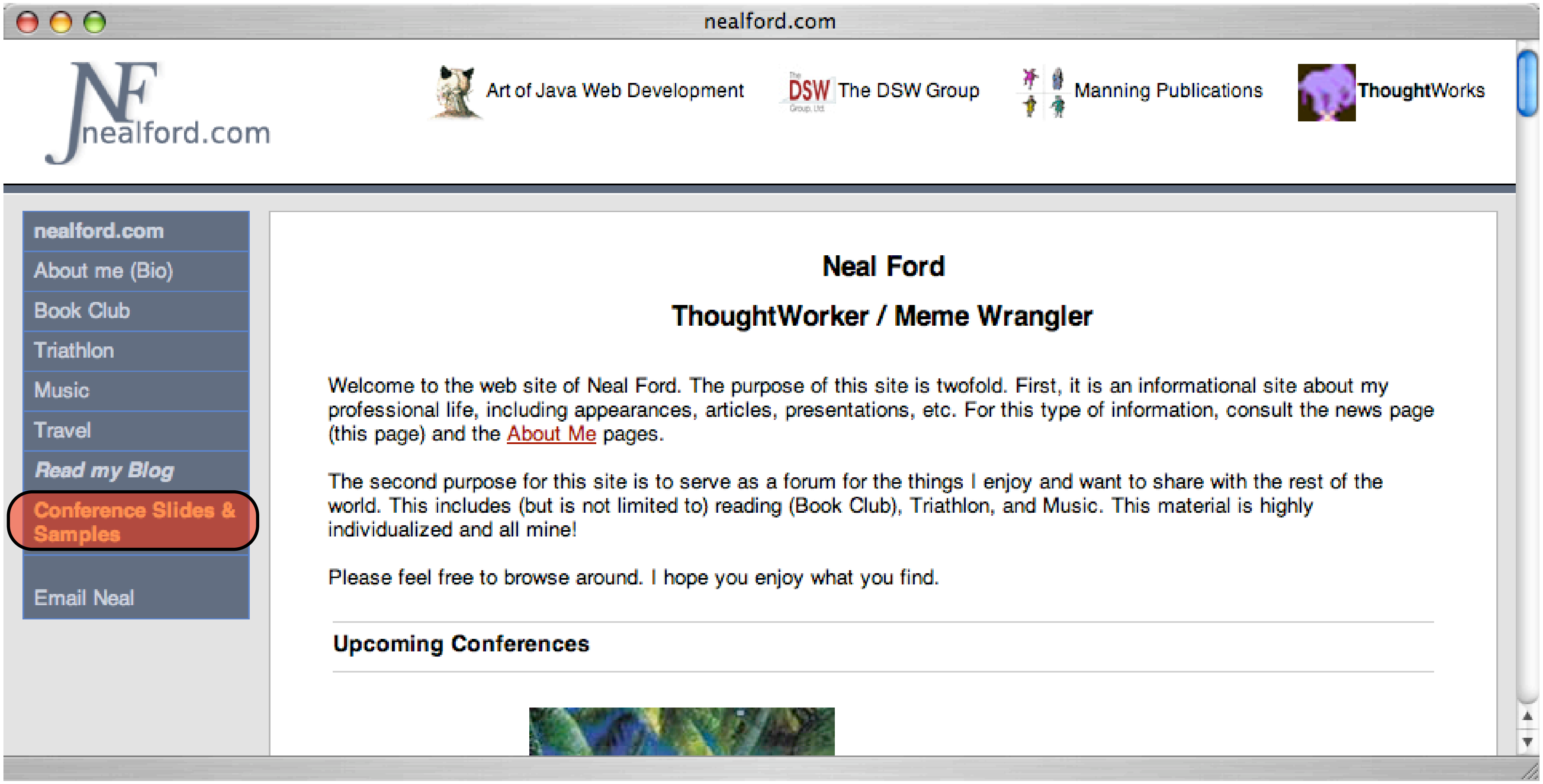
real-world refactoring

NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com



nealford.com

nealford.com

Art of Java Web Development

The DSW Group

Manning Publications

ThoughtWorks

- nealford.com
- About me (Bio)
- Book Club
- Triathlon
- Music
- Travel
- *Read my Blog*
- Conference Slides & Samples
- Email Neal

Neal Ford

ThoughtWorker / Meme Wrangler

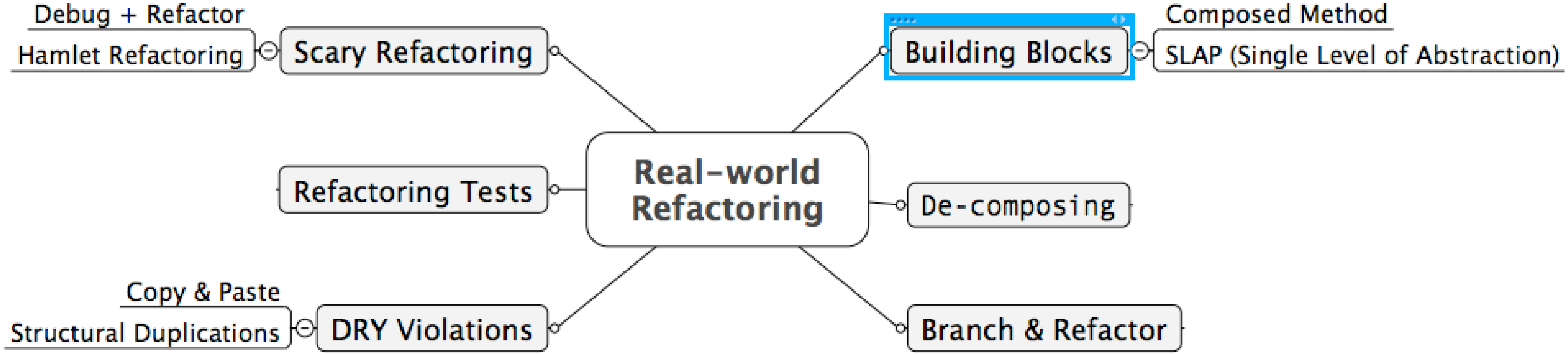
Welcome to the web site of Neal Ford. The purpose of this site is twofold. First, it is an informational site about my professional life, including appearances, articles, presentations, etc. For this type of information, consult the news page (this page) and the About Me pages.

The second purpose for this site is to serve as a forum for the things I enjoy and want to share with the rest of the world. This includes (but is not limited to) reading (Book Club), Triathlon, and Music. This material is highly individualized and all mine!

Please feel free to browse around. I hope you enjoy what you find.

Upcoming Conferences

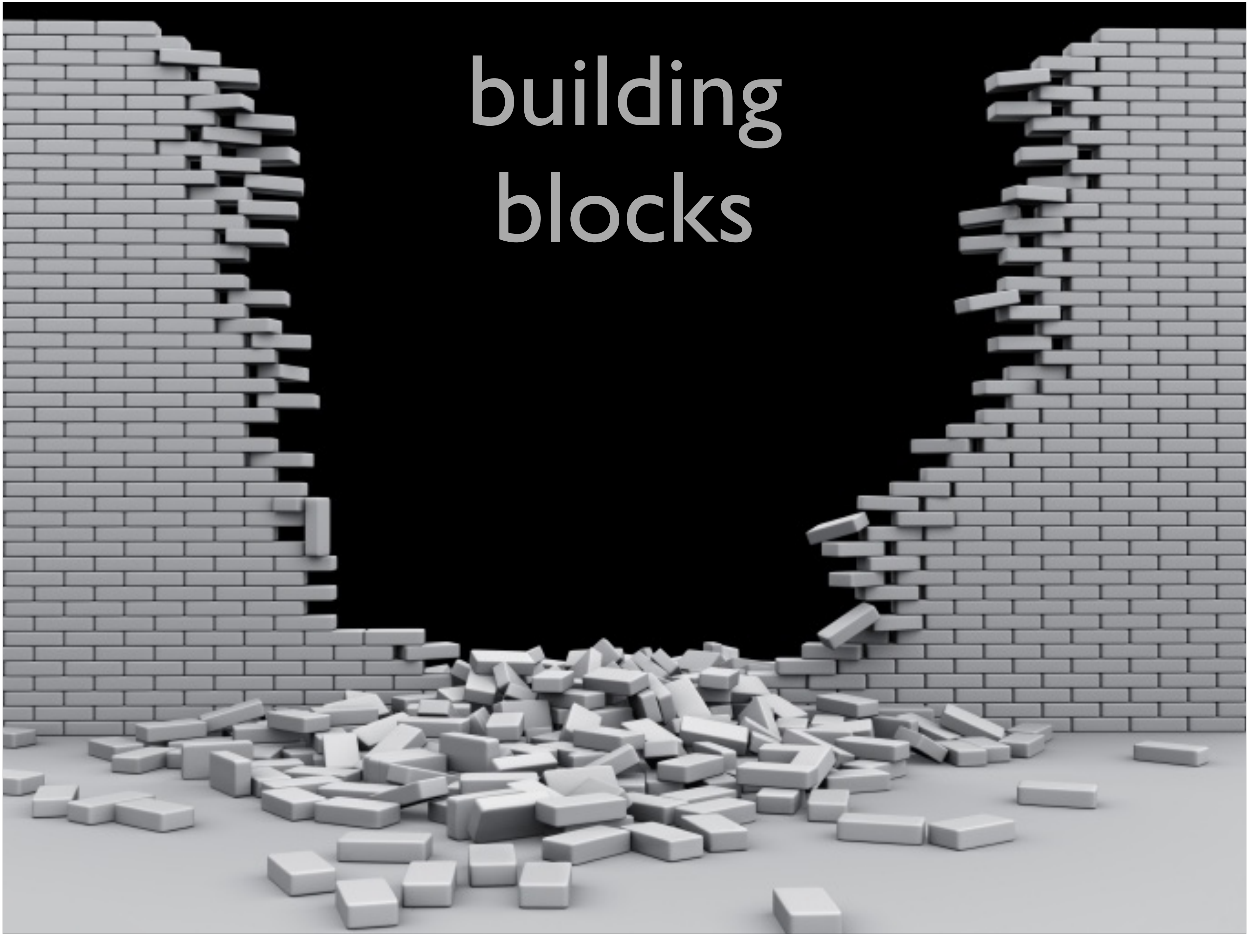
what i cover:

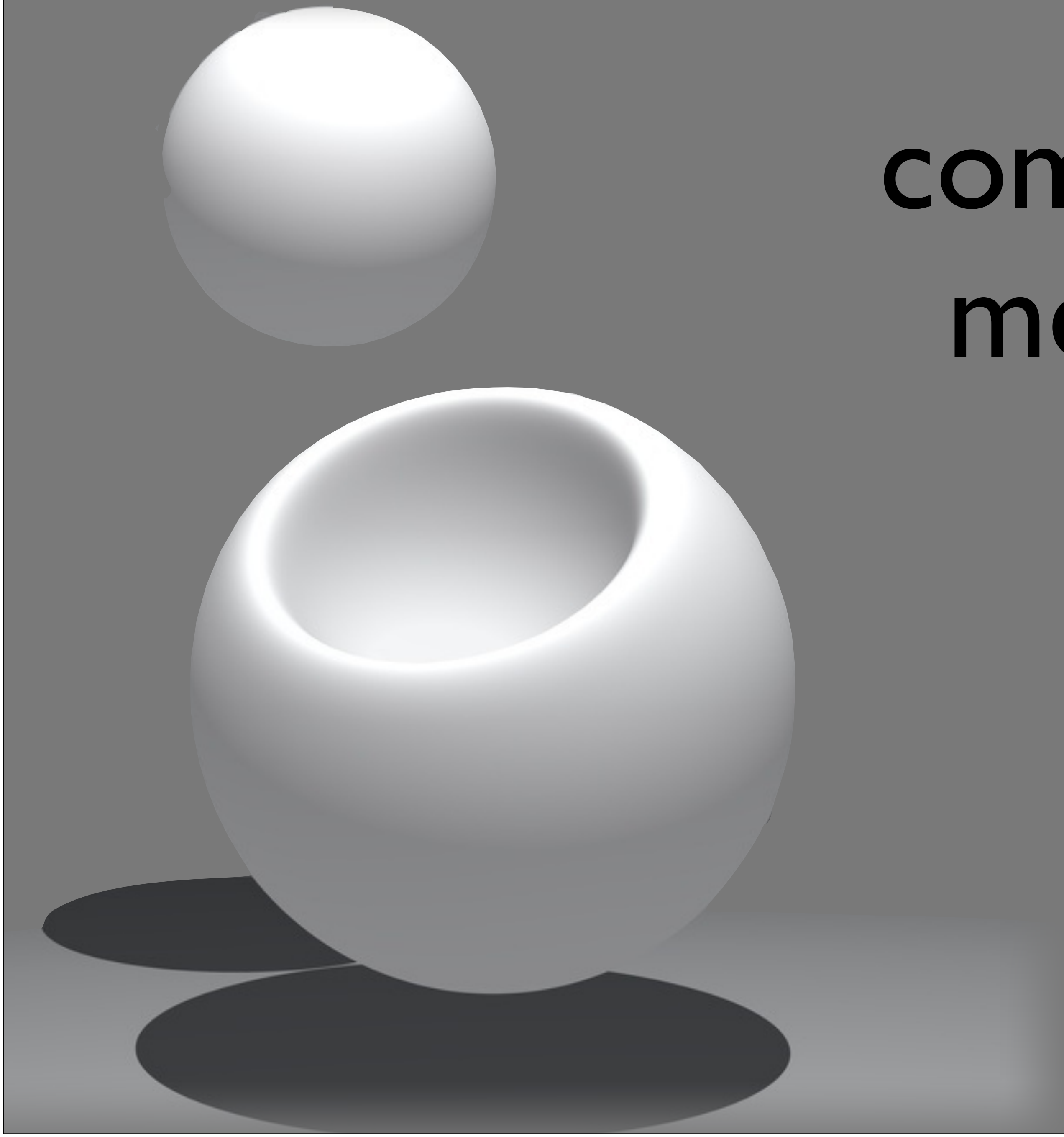


Real-world Refactoring

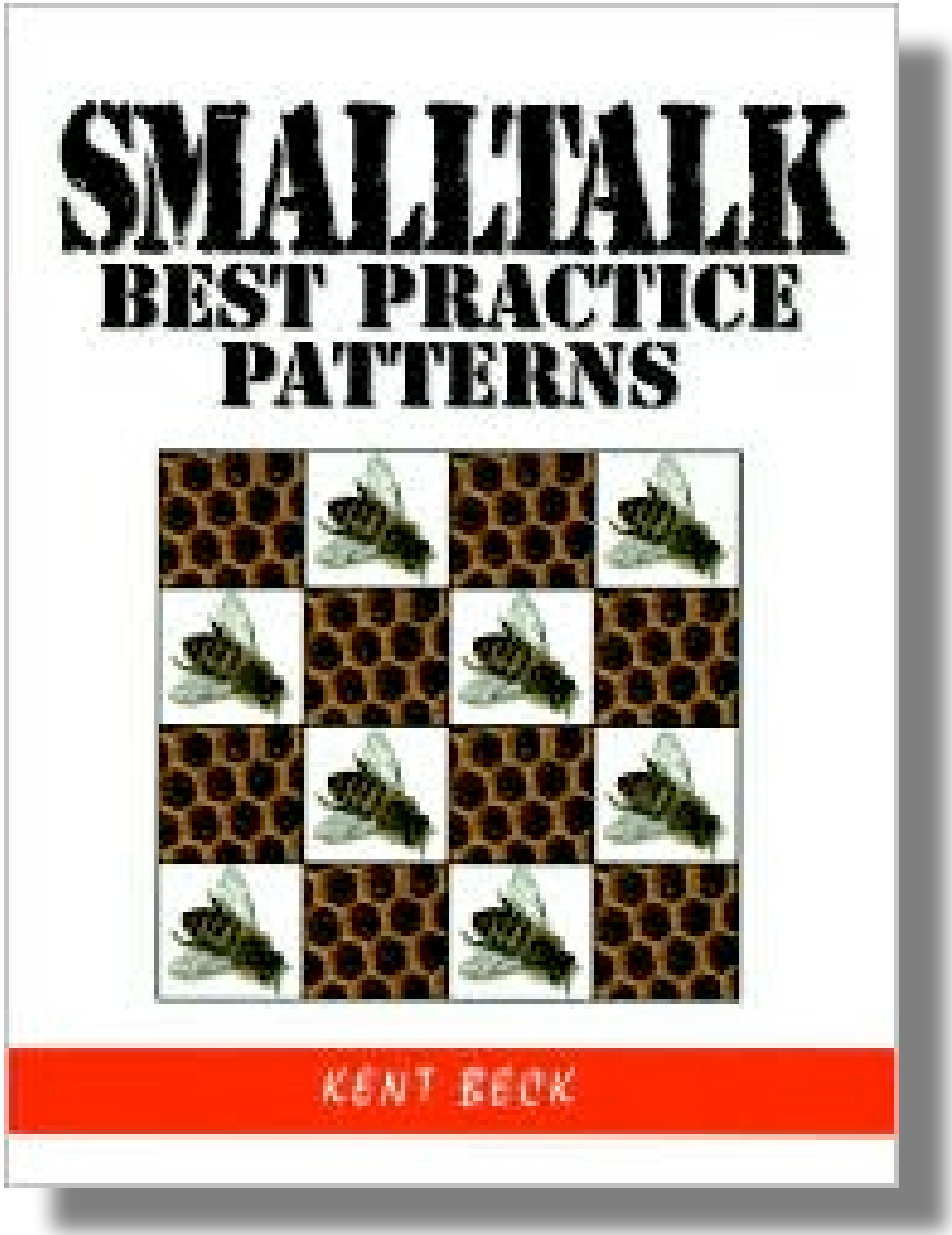
- Scary Refactoring
  - Debug + Refactor
  - Hamlet Refactoring
- Refactoring Tests
- DRY Violations
  - Copy & Paste
  - Structural Duplications
- Building Blocks
  - Composed Method
  - SLAP (Single Level of Abstraction)
- De-composing
- Branch & Refactor

building blocks





composed method


SMALLTALK
BEST PRACTICE
PATTERNS
KENT BECK

composed method

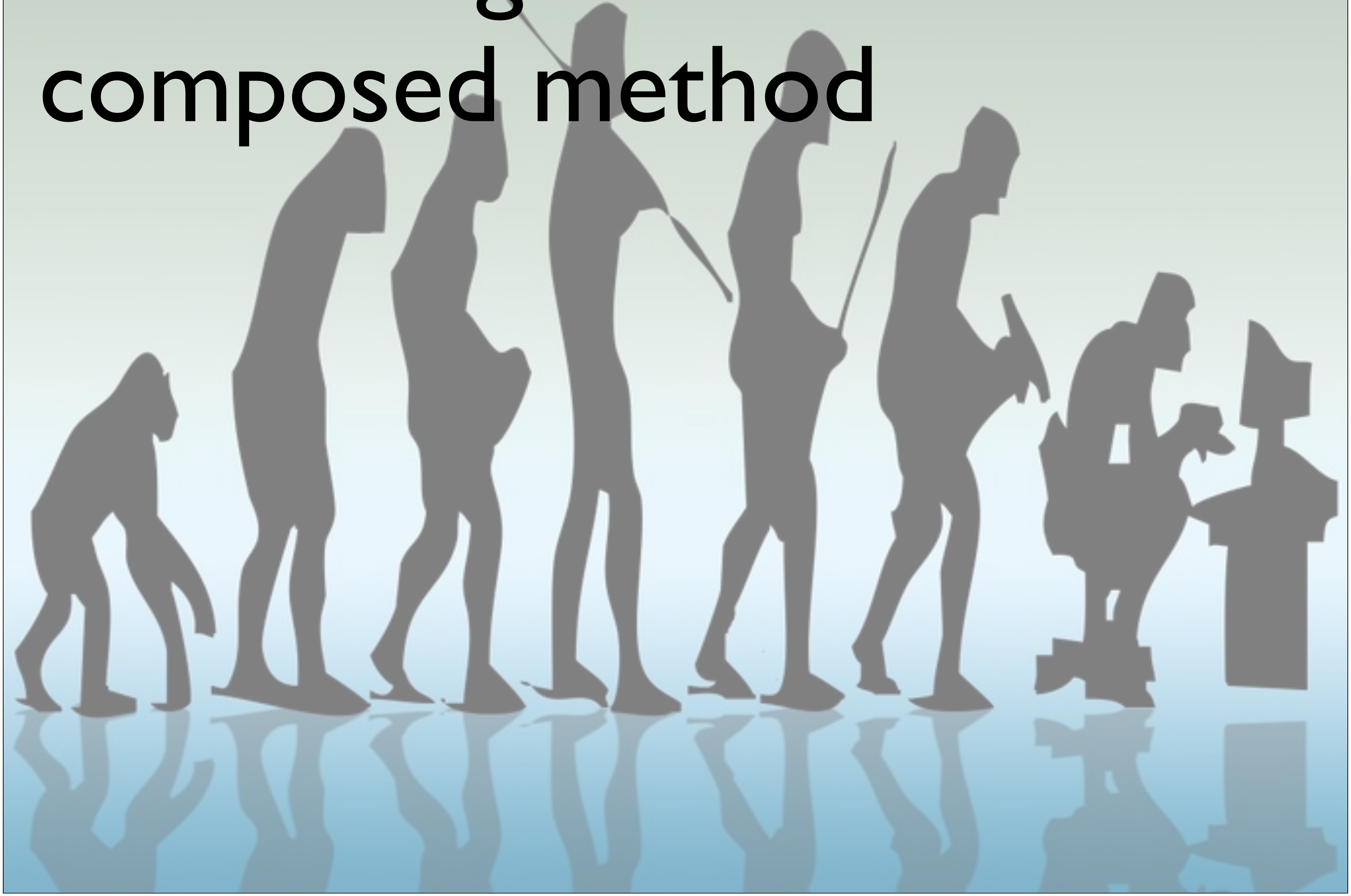
Divide your program into methods that perform one identifiable task.

Divide your program into methods that perform one identifiable task.

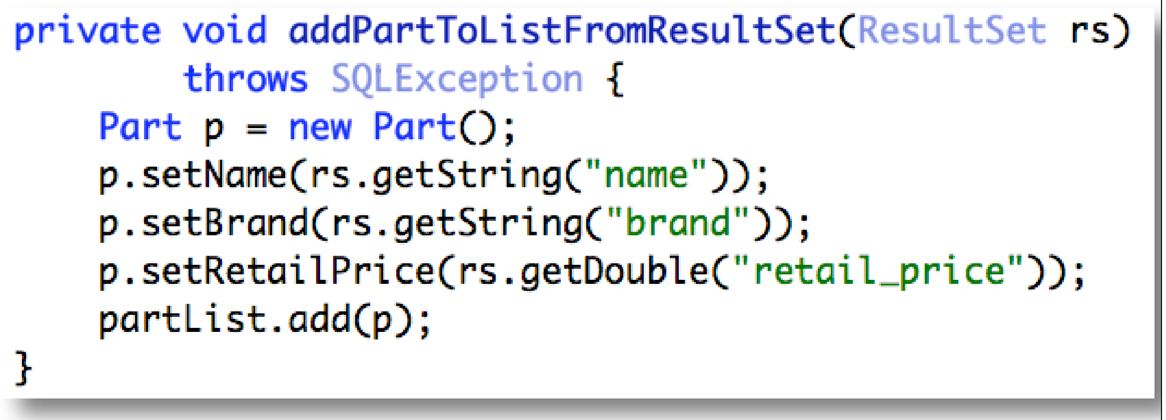
Keep all of the operations in a method at the same level of abstraction.

This will naturally result in programs with many small methods, each a few lines long.

refactoring to composed method

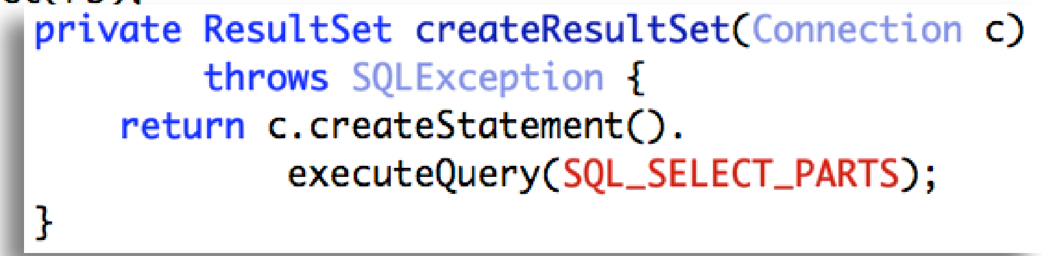


```java
public void populate() throws Exception  {
    Connection c = null;
    try {
        Class.forName(DRIVER_CLASS);
        c = DriverManager.getConnection(DB_URL, USER, PASSWORD);
        Statement stmt = c.createStatement();
        ResultSet rs = stmt.executeQuery(SQL_SELECT_PARTS);
        while (rs.next()) {
            Part p = new Part();
            p.setName(rs.getString("name"));
            p.setBrand(rs.getString("brand"));
            p.setRetailPrice(rs.getDouble("retail_price"));
            partList.add(p);
        }
    } finally {
        c.close();
    }
}
```

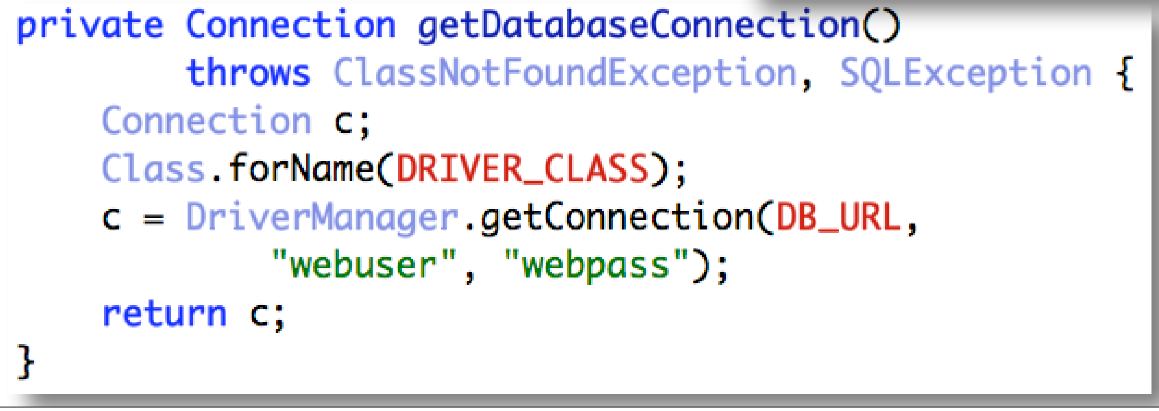


```
private void addPartToListFromResultSet(ResultSet rs)
        throws SQLException {
    Part p = new Part();
    p.setName(rs.getString("name"));
    p.setBrand(rs.getString("brand"));
    p.setRetailPrice(rs.getDouble("retail_price"));
    partList.add(p);
}
```

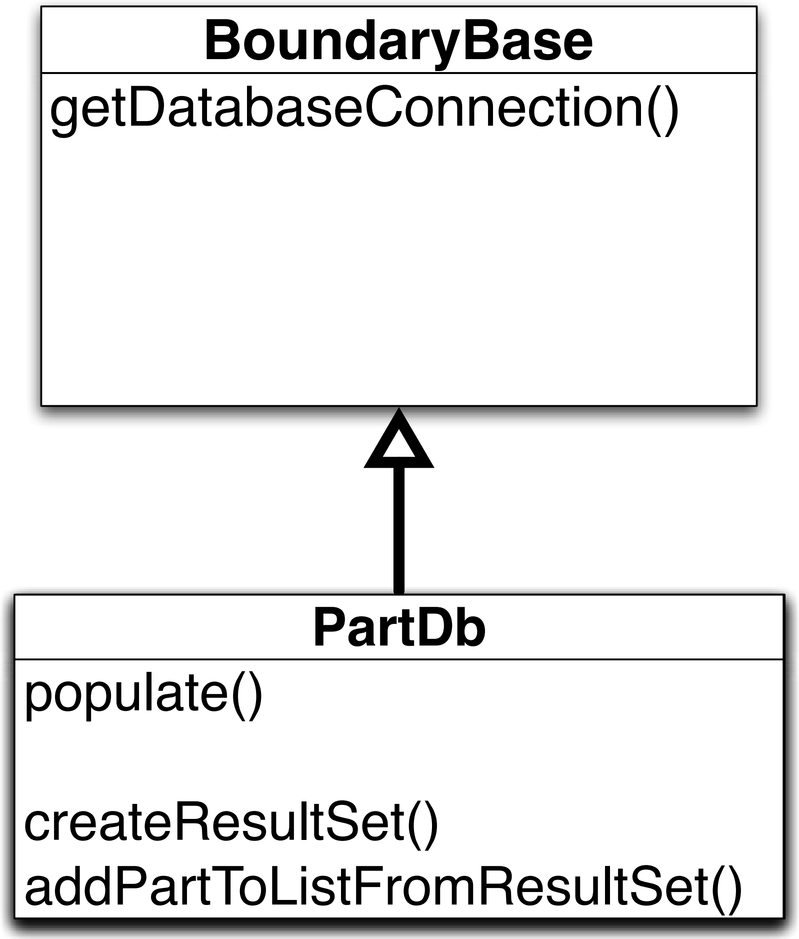
```
public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addPartToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```



```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```



```
private Connection getDatabaseConnection()
        throws ClassNotFoundException, SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL,
            "webuser", "webpass");
    return c;
}
```



BoundaryBase

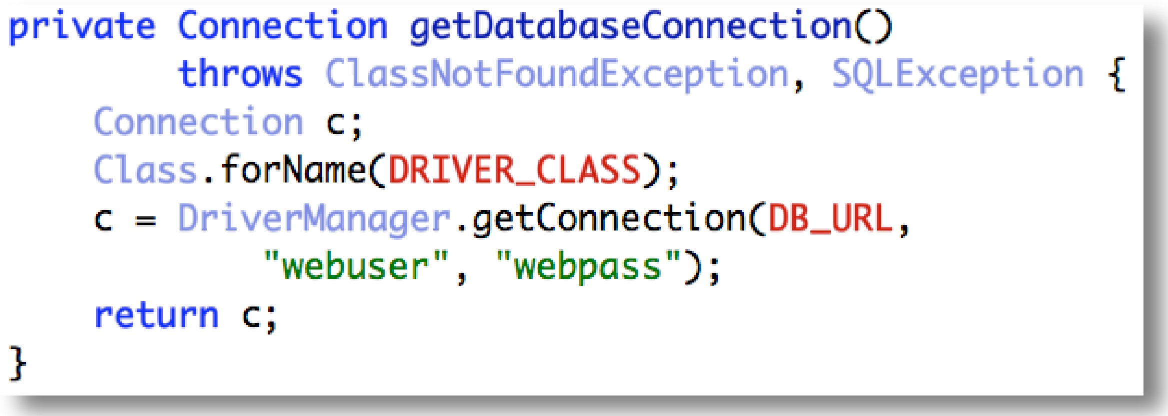
getDatabaseConnection()

PartDb

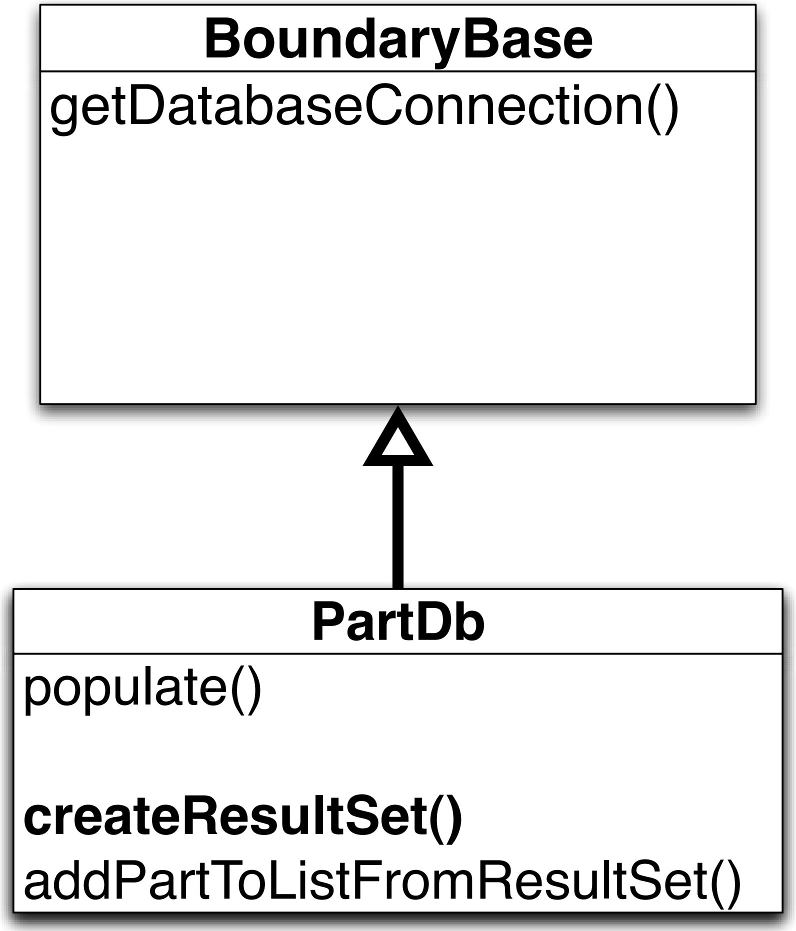
populate()

createResultSet()

addPartToListFromResultSet()



```
private Connection getDatabaseConnection()
        throws ClassNotFoundException, SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL,
            "webuser", "webpass");
    return c;
}
```



BoundaryBase

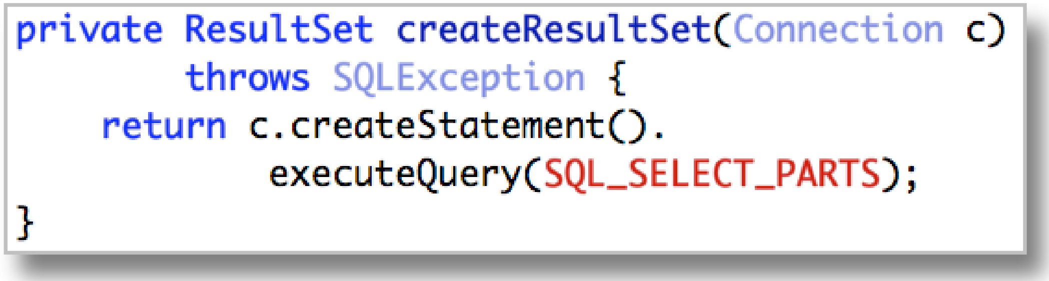
getDatabaseConnection()

PartDb

populate()

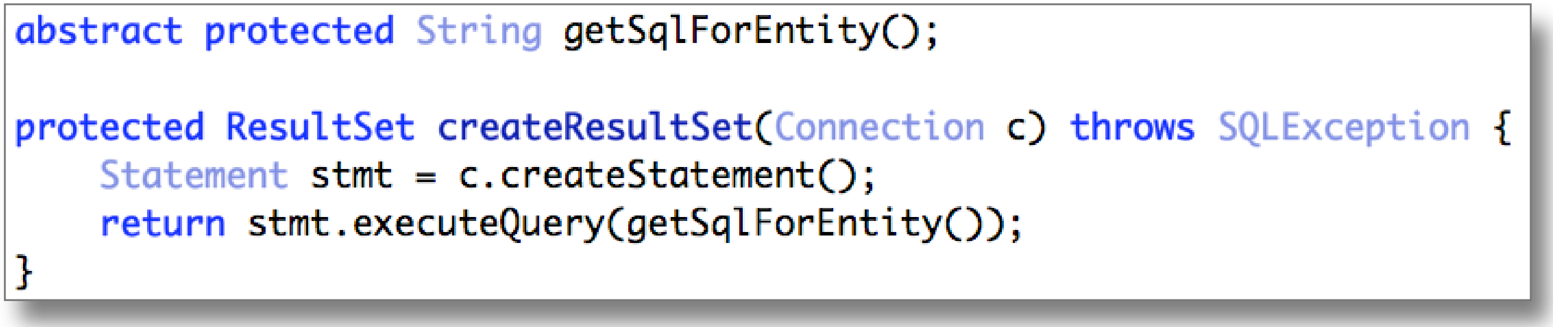
createResultSet()

addPartToListFromResultSet()

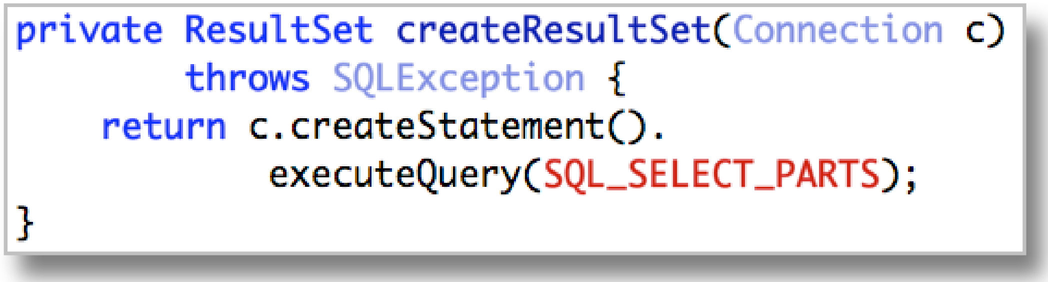


```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```

BoundaryBase

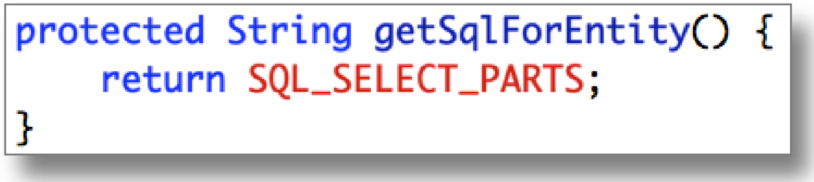


```
abstract protected String getSqlForEntity();

protected ResultSet createResultSet(Connection c) throws SQLException {
    Statement stmt = c.createStatement();
    return stmt.executeQuery(getSqlForEntity());
}
```

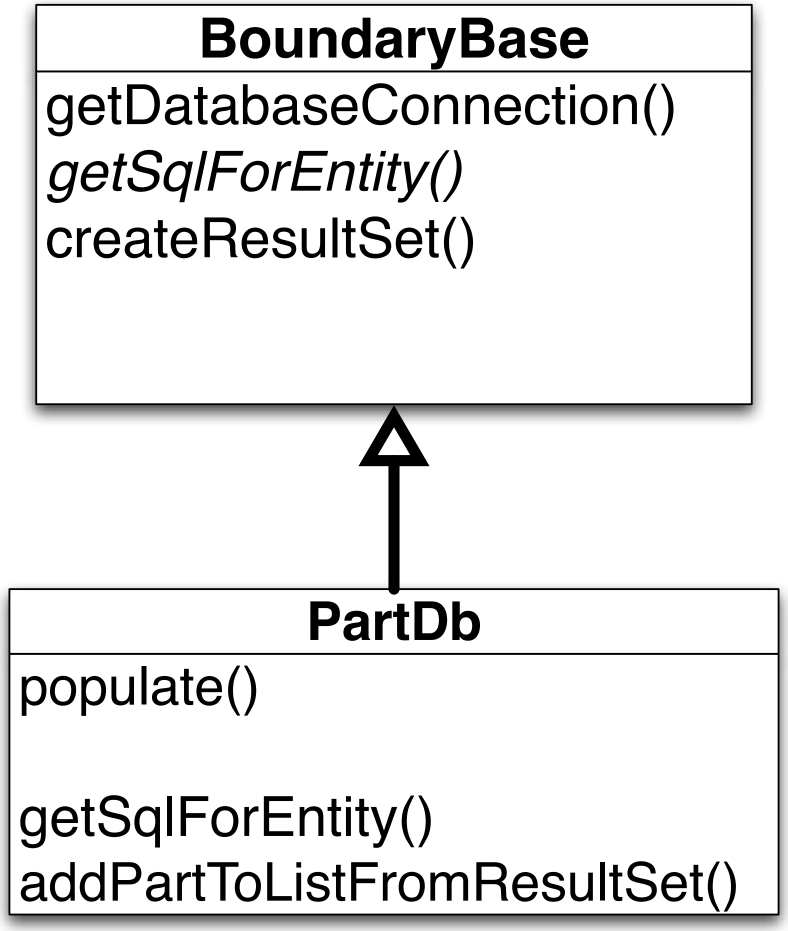


```
private ResultSet createResultSet(Connection c)
        throws SQLException {
    return c.createStatement().
            executeQuery(SQL_SELECT_PARTS);
}
```

PartDb



```
protected String getSqlForEntity() {
    return SQL_SELECT_PARTS;
}
```



BoundaryBase

getDatabaseConnection()
getSqlForEntity()
createResultSet()

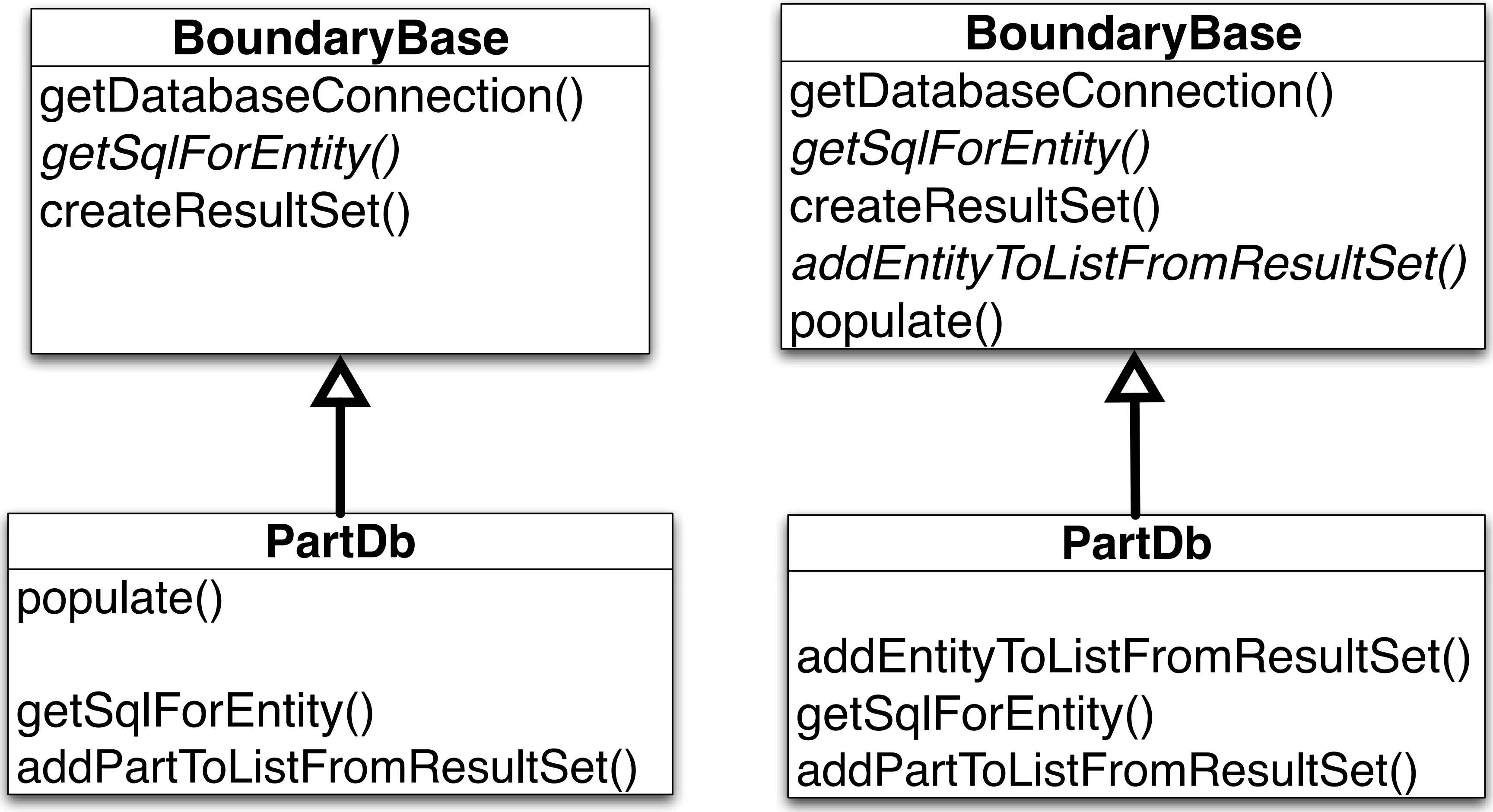
PartDb

populate()

getSqlForEntity()
addPartToListFromResultSet()

```
public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addPartToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

```
abstract protected void addEntityToListFromResultSet(ResultSet rs)
        throws SQLException;

public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addEntityToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```



BoundaryBase

getDatabaseConnection()
getSqlForEntity()
createResultSet()

PartDb

populate()

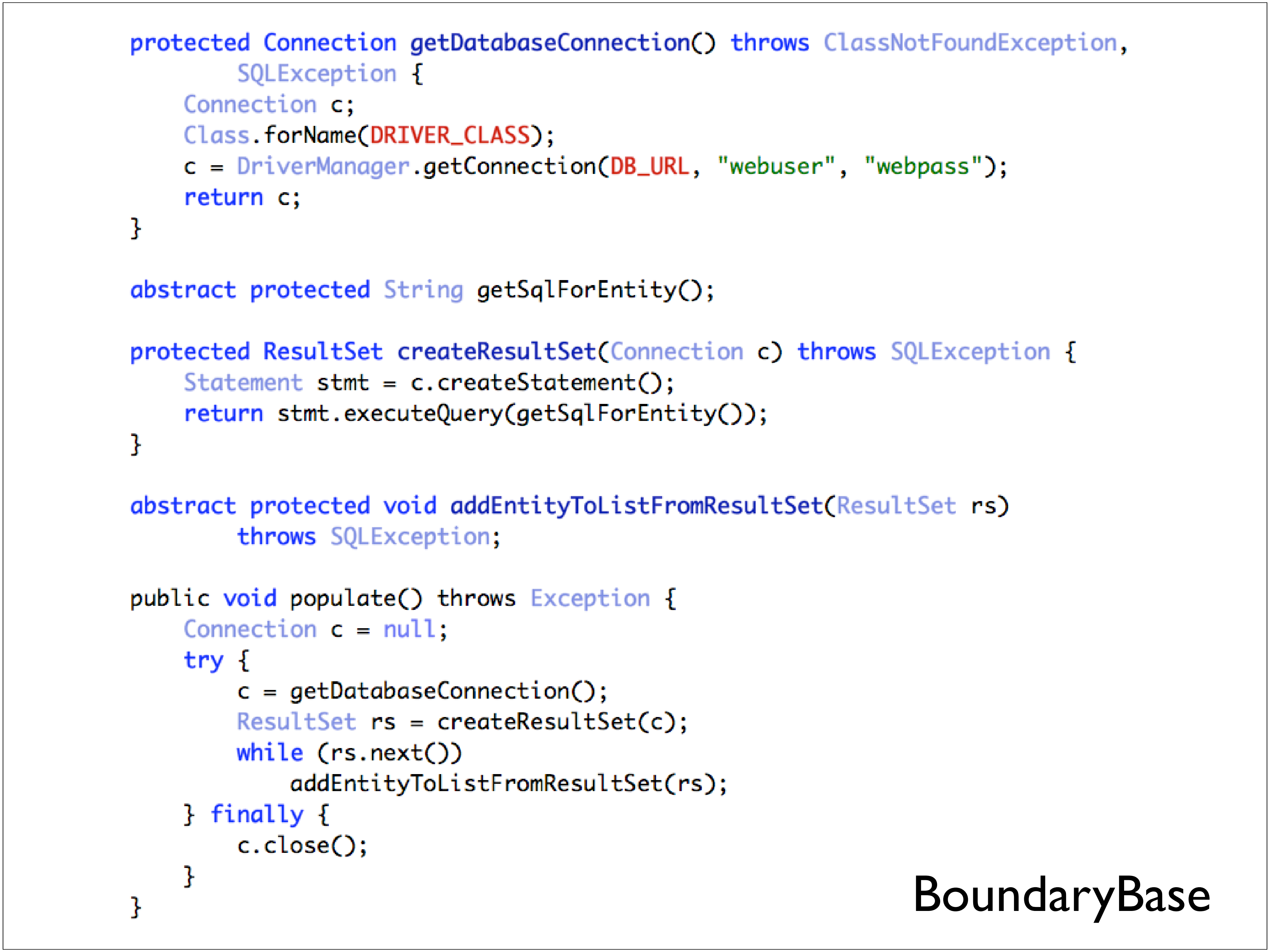
getSqlForEntity()
addPartToListFromResultSet()

BoundaryBase

getDatabaseConnection()
getSqlForEntity()
createResultSet()
addEntityToListFromResultSet()
populate()

PartDb

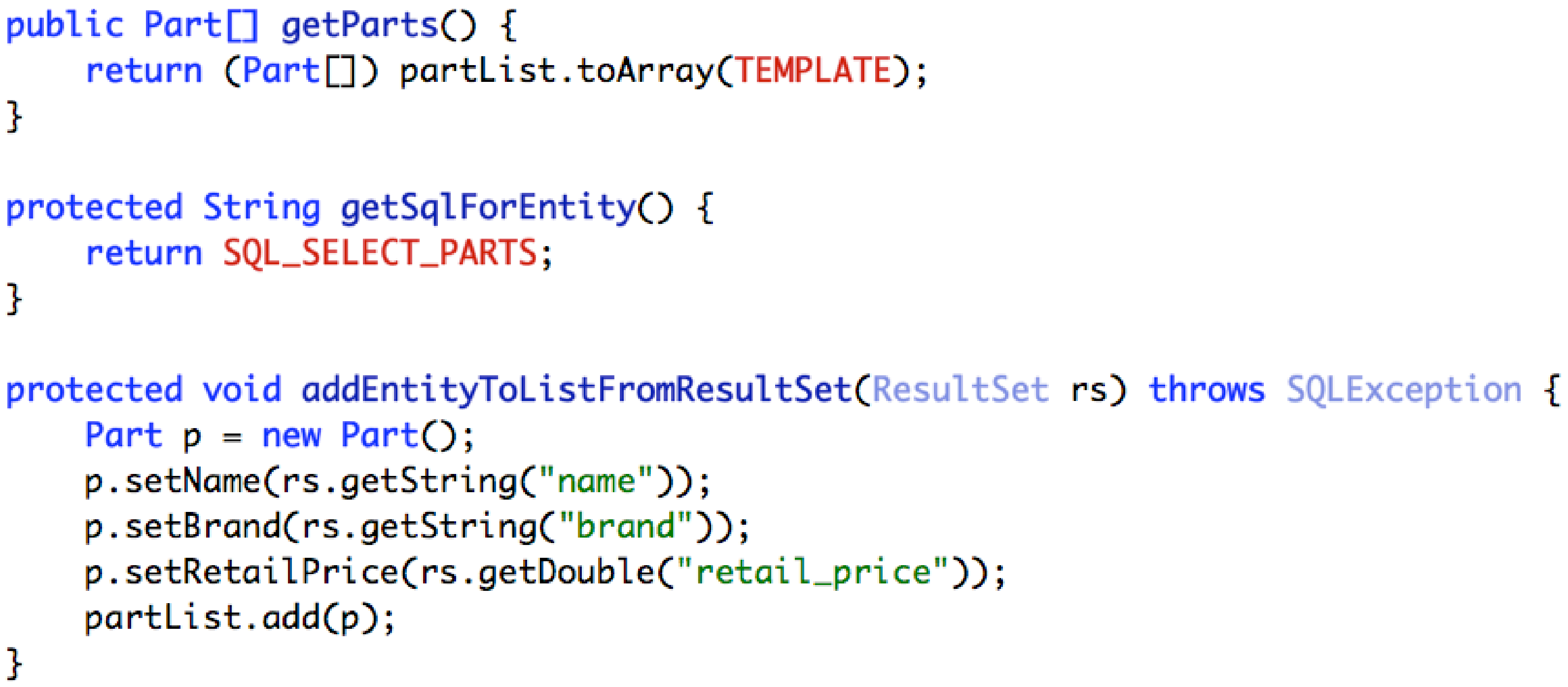
addEntityToListFromResultSet()
getSqlForEntity()
addPartToListFromResultSet()



```
protected Connection getDatabaseConnection() throws ClassNotFoundException,
        SQLException {
    Connection c;
    Class.forName(DRIVER_CLASS);
    c = DriverManager.getConnection(DB_URL, "webuser", "webpass");
    return c;
}

abstract protected String getSqlForEntity();

protected ResultSet createResultSet(Connection c) throws SQLException {
    Statement stmt = c.createStatement();
    return stmt.executeQuery(getSqlForEntity());
}

abstract protected void addEntityToListFromResultSet(ResultSet rs)
        throws SQLException;

public void populate() throws Exception {
    Connection c = null;
    try {
        c = getDatabaseConnection();
        ResultSet rs = createResultSet(c);
        while (rs.next())
            addEntityToListFromResultSet(rs);
    } finally {
        c.close();
    }
}
```

BoundaryBase

PartDb



```
public Part[] getParts() {
    return (Part[]) partList.toArray(TEMPLATE);
}

protected String getSqlForEntity() {
    return SQL_SELECT_PARTS;
}

protected void addEntityToListFromResultSet(ResultSet rs) throws SQLException {
    Part p = new Part();
    p.setName(rs.getString("name"));
    p.setBrand(rs.getString("brand"));
    p.setRetailPrice(rs.getDouble("retail_price"));
    partList.add(p);
}
```

benefits of composed method

shorter methods easier to test

method names become documentation

large number of very cohesive methods

discover reusable assets that you didn’t know were there

SLAP

single level of abstraction principle

composed method

Divide your program into methods that perform one identifiable task.

Keep all of the operations in a method at the same level of abstraction.

Keep all of the operations in a method at the same level of abstraction.

This will naturally result in programs with many small methods, each a few lines long.

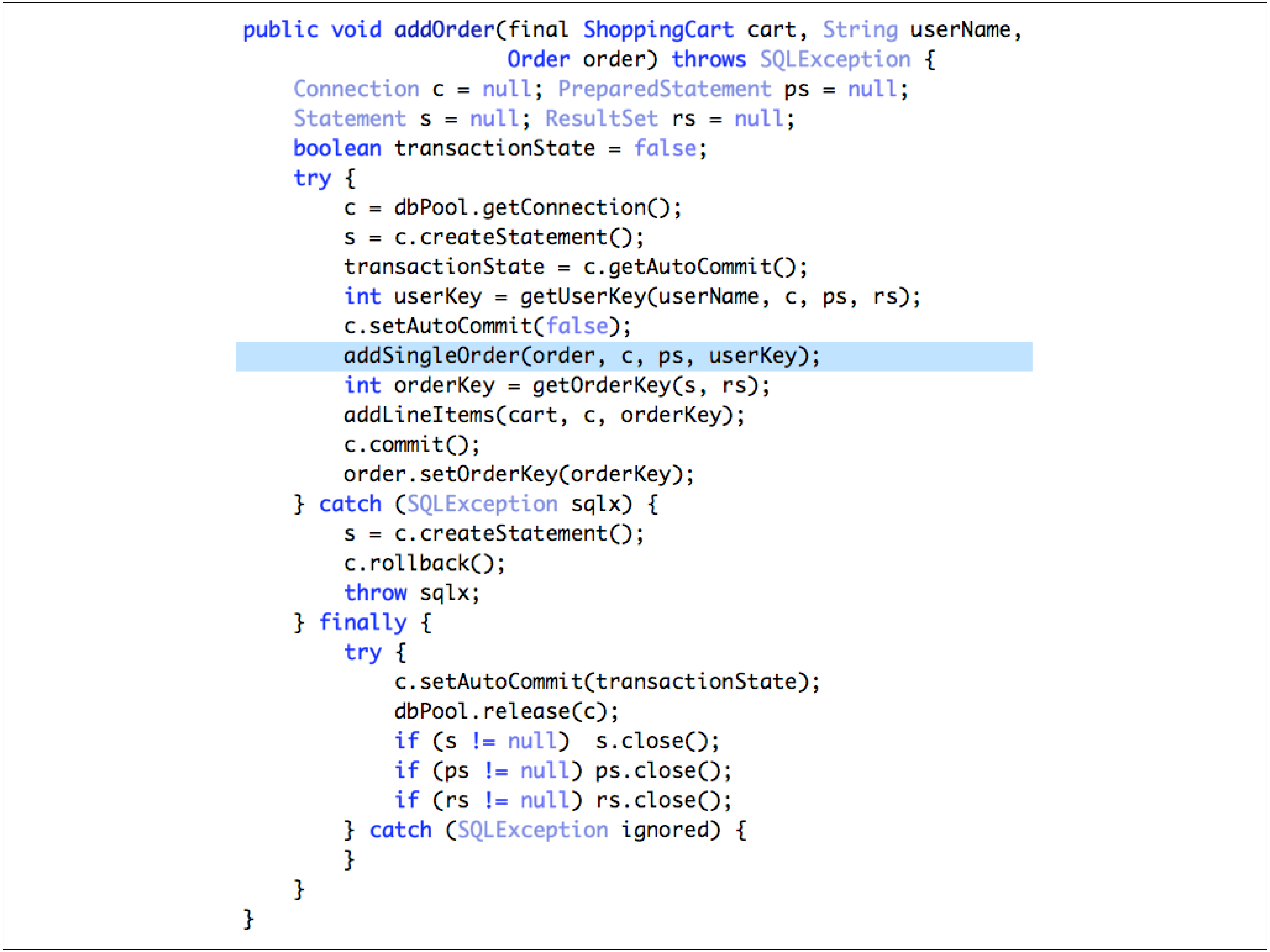
slap

jumping abstraction layers makes code hard to understand

composed method => slap

refactor to slap

even if it means single-line methods



```java
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection c = null; PreparedStatement ps = null;
    Statement s = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        c = dbPool.getConnection();
        s = c.createStatement();
        transactionState = c.getAutoCommit();
        int userKey = getUserKey(userName, c, ps, rs);
        c.setAutoCommit(false);
        addSingleOrder(order, c, ps, userKey);
        int orderKey = getOrderKey(s, rs);
        addLineItems(cart, c, orderKey);
        c.commit();
        order.setOrderKey(orderKey);
    } catch (SQLException sqlx) {
        s = c.createStatement();
        c.rollback();
        throw sqlx;
    } finally {
        try {
            c.setAutoCommit(transactionState);
            dbPool.release(c);
            if (s != null)  s.close();
            if (ps != null) ps.close();
            if (rs != null) rs.close();
        } catch (SQLException ignored) {
        }
    }
}
```

```java
public void addOrder(final ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection connection = null; PreparedStatement ps = null;
    Statement statement = null; ResultSet rs = null;
    boolean transactionState = false;
    try {
        connection = dbPool.getConnection();
        statement = connection.createStatement();
        transactionState = setupTransactionStateFor(connection, transactionState);
        addSingleOrder(order, connection, ps, userKeyFor(userName, connection));
        order.setOrderKey(generateOrderKey(statement, rs));
        addLineItems(cart, connection, order.getOrderKey());
        completeTransaction(connection);
    } catch (SQLException sqlx) {
        rollbackTransactionFor(connection);
        throw sqlx;
    } finally {
        cleanUpDatabaseResources(connection, transactionState, statement, ps, rs);
    }
}
```

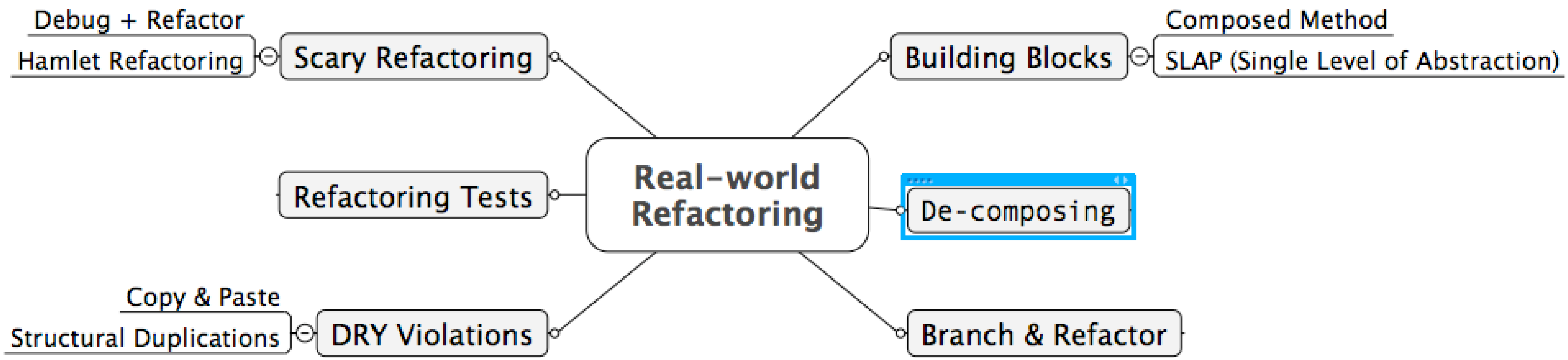
```java
public void addOrderFrom(ShoppingCart cart, String userName,
                         Order order) throws SQLException {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        rollbackTransaction();
        throw sqlx;
    } finally {
        cleanUp();
    }
}
```

```java
private void completeTransaction() throws SQLException {
    ((Connection)_db.get("connection")).commit();
}
```

```java
ivate void setupDataInfrastructure() throws SQLException {
 _db = new HashMap();
 private void
 _db.put("Conn
    PreparedStatement                prepareStatement(SQL_INSERT_ORDER);
    ps.setInt(
    ps.setString(2, order.getCcType());
    ps.setString(3, order.getCcNum());
    ps.setString(4, order.getCcExp());
    int result = ps.executeUpdate();
    if (result != 1) {
        throw new SQLException(
                "Order.add(): order insert failed");
    }
    dbInfrastructure.put("prepared statement", ps);
```

```
public void addOrder(ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    Connection c = null;
    PreparedStatement ps = null;
    Statement s = null;
    ResultSet rs = null;
    boolean transactionState = false;
    try {
        s = c.createStatement();
        transactionState = c.getAutoCommit();
        int userKey = getUserKey(userName, c, ps, rs);
        c.setAutoCommit(false);
        addSingleOrder(order, c, ps, userKey);
        int orderKey = getOrderKey(s, rs);
        addLineItems(cart, c, orderKey);
        c.commit();
        order.setOrderKeyFrom(orderKey);
    } catch (SQLException sqlx) {
        s = c.createStatement();
        c.rollback();
        throw sqlx;
    } finally {
        try {
            c.setAutoCommit(transactio
            dbPool.release(c);
            if (s != null)
                s.close();
            if (ps != null)
                ps.close();
            if (rs != null)
                rs.close();
        } catch (SQLException ignored)
        }
    }
}
```

```
public void addOrderFrom(ShoppingCart cart, String userName,
                     Order order) throws SQLException {
    setupDataInfrastructure();
    try {
        add(order, userKeyBasedOn(userName));
        addLineItemsFrom(cart, order.getOrderKey());
        completeTransaction();
    } catch (SQLException sqlx) {
        rollbackTransaction();
        throw sqlx;
    } finally {
        cleanUp();
    }
}
```



Real-world Refactoring

Scary Refactoring

Debug + Refactor

Hamlet Refactoring

Refactoring Tests

DRY Violations

Copy & Paste

Structural Duplications

Building Blocks

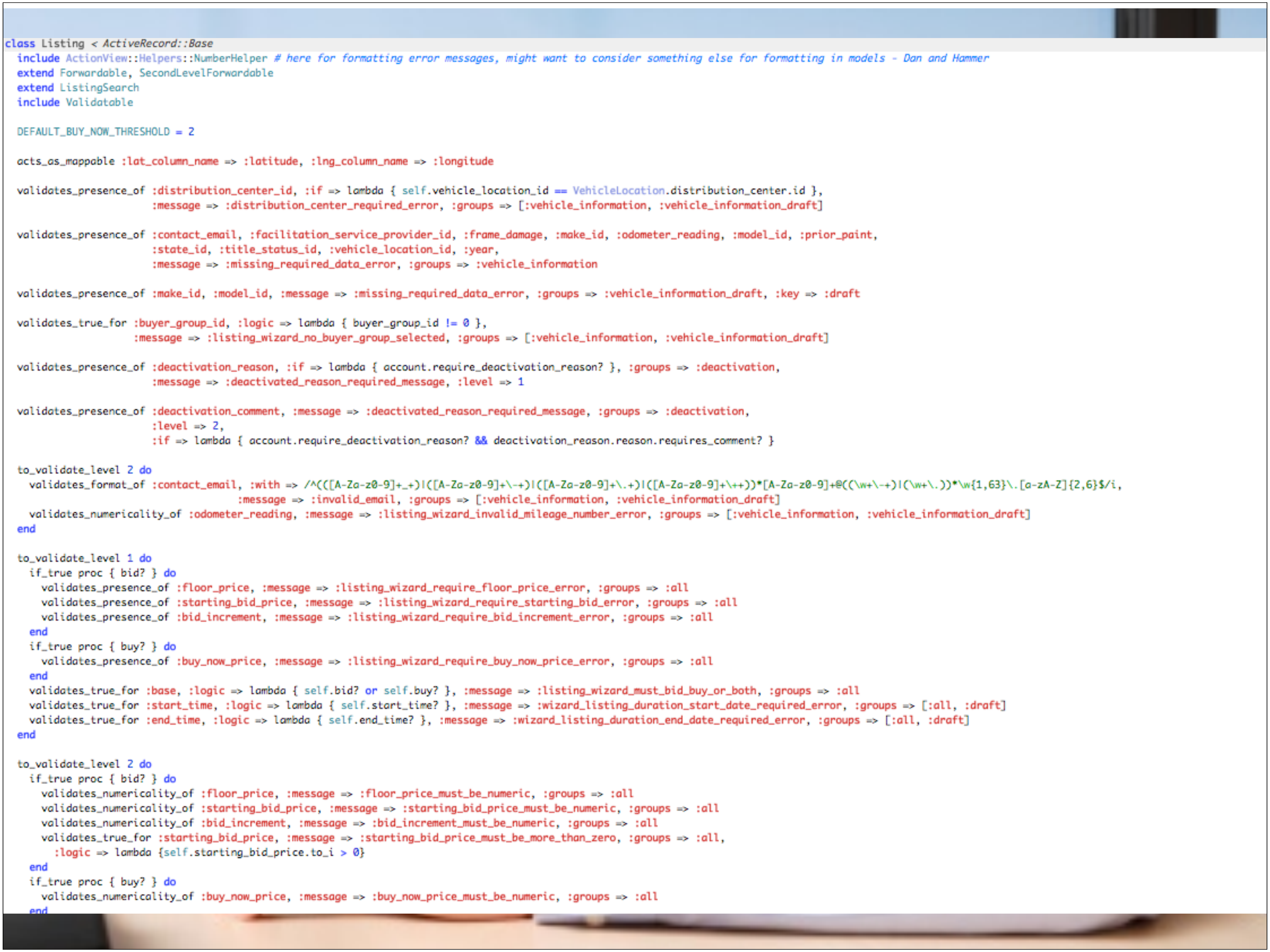
Composed Method

SLAP (Single Level of Abstraction)

De-composing

Branch & Refactor

de-composing



```ruby
class Listing < ActiveRecord::Base
  include ActionView::Helpers::NumberHelper # here for formatting error messages, might want to consider something else for formatting in models - Dan and Hammer
  extend Forwardable, SecondLevelForwardable
  extend ListingSearch
  include Validatable

  DEFAULT_BUY_NOW_THRESHOLD = 2

  acts_as_mappable :lat_column_name => :latitude, :lng_column_name => :longitude

  validates_presence_of :distribution_center_id, :if => lambda { self.vehicle_location_id == VehicleLocation.distribution_center.id },
                        :message => :distribution_center_required_error, :groups => [:vehicle_information, :vehicle_information_draft]

  validates_presence_of :contact_email, :facilitation_service_provider_id, :frame_damage, :make_id, :odometer_reading, :model_id, :prior_paint,
                        :state_id, :title_status_id, :vehicle_location_id, :year,
                        :message => :missing_required_data_error, :groups => :vehicle_information

  validates_presence_of :make_id, :model_id, :message => :missing_required_data_error, :groups => :vehicle_information_draft, :key => :draft

  validates_true_for :buyer_group_id, :logic => lambda { buyer_group_id != 0 },
                     :message => :listing_wizard_no_buyer_group_selected, :groups => [:vehicle_information, :vehicle_information_draft]

  validates_presence_of :deactivation_reason, :if => lambda { account.require_deactivation_reason? }, :groups => :deactivation,
                        :message => :deactivated_reason_required_message, :level => 1

  validates_presence_of :deactivation_comment, :message => :deactivated_reason_required_message, :groups => :deactivation,
                        :level => 2,
                        :if => lambda { account.require_deactivation_reason? && deactivation_reason.reason.requires_comment? }

  to_validate_level 2 do
    validates_format_of :contact_email, :with => /^(([A-Za-z0-9]+_+)|([A-Za-z0-9]+\-+)|([A-Za-z0-9]+\.+)|([A-Za-z0-9]+\++))*[A-Za-z0-9]+@((\w+\-+)|(\w+\.))*\w{1,63}\.[a-zA-Z]{2,6}$/i,
                                         :message => :invalid_email, :groups => [:vehicle_information, :vehicle_information_draft]
    validates_numericality_of :odometer_reading, :message => :listing_wizard_invalid_mileage_number_error, :groups => [:vehicle_information, :vehicle_information_draft]
  end

  to_validate_level 1 do
    if_true proc { bid? } do
      validates_presence_of :floor_price, :message => :listing_wizard_require_floor_price_error, :groups => :all
      validates_presence_of :starting_bid_price, :message => :listing_wizard_require_starting_bid_error, :groups => :all
      validates_presence_of :bid_increment, :message => :listing_wizard_require_bid_increment_error, :groups => :all
    end
    if_true proc { buy? } do
      validates_presence_of :buy_now_price, :message => :listing_wizard_require_buy_now_price_error, :groups => :all
    end
    validates_true_for :base, :logic => lambda { self.bid? or self.buy? }, :message => :listing_wizard_must_bid_buy_or_both, :groups => :all
    validates_true_for :start_time, :logic => lambda { self.start_time? }, :message => :wizard_listing_duration_start_date_required_error, :groups => [:all, :draft]
    validates_true_for :end_time, :logic => lambda { self.end_time? }, :message => :wizard_listing_duration_end_date_required_error, :groups => [:all, :draft]
  end

  to_validate_level 2 do
    if_true proc { bid? } do
      validates_numericality_of :floor_price, :message => :floor_price_must_be_numeric, :groups => :all
      validates_numericality_of :starting_bid_price, :message => :starting_bid_price_must_be_numeric, :groups => :all
      validates_numericality_of :bid_increment, :message => :bid_increment_must_be_numeric, :groups => :all
      validates_true_for :starting_bid_price, :message => :starting_bid_price_must_be_more_than_zero, :groups => :all,
        :logic => lambda {self.starting_bid_price.to_i > 0}
    end
    if_true proc { buy? } do
      validates_numericality_of :buy_now_price, :message => :buy_now_price_must_be_numeric, :groups => :all
    end
```

decomposition

should you?

look for natural partitions

one-way dependencies

extract related items...

...probably coupled via interfaces *to your classes*

coupling to infrastructure

don’t tie yourself into infrastructure

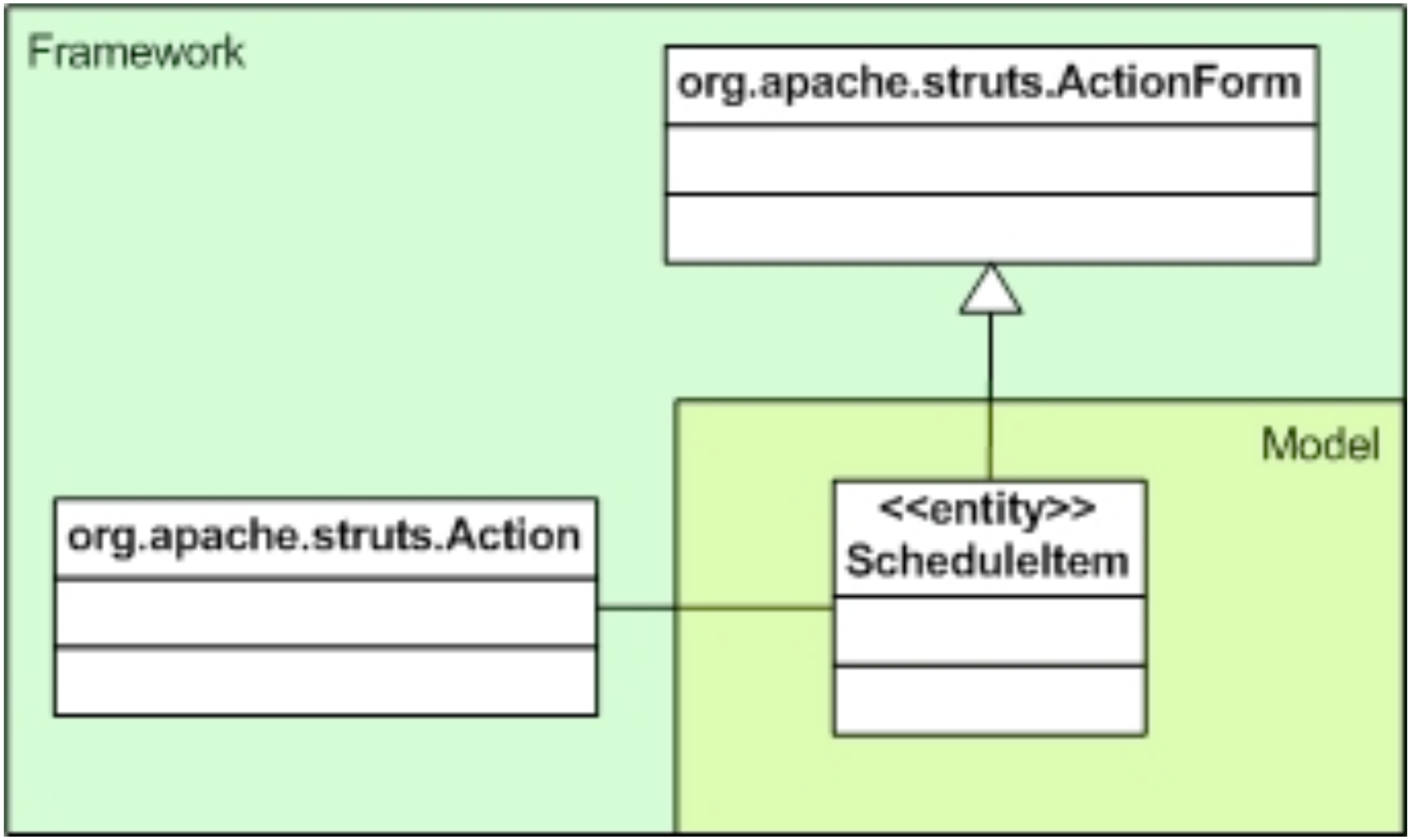
```
import com.giantvendor.seductiveclasses.*
```

pay attention to dependencies

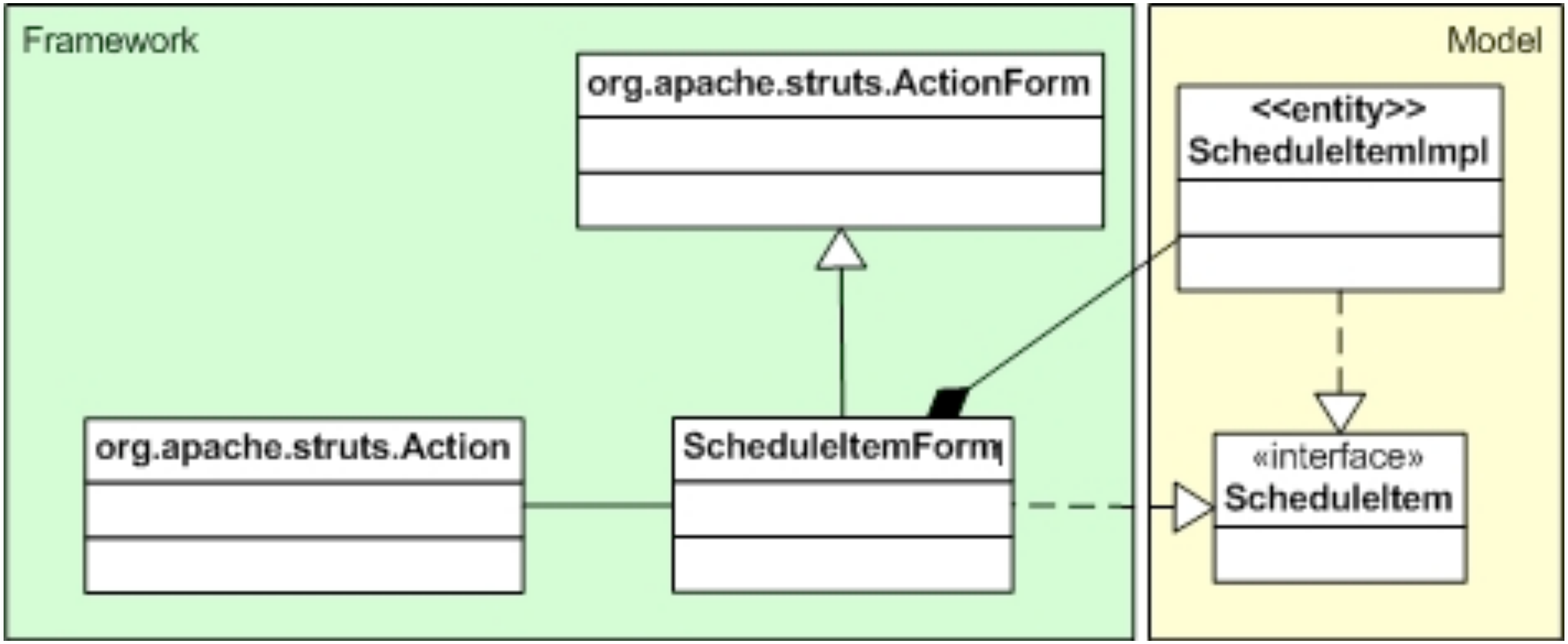
don’t extend library/framework classes

compose instead

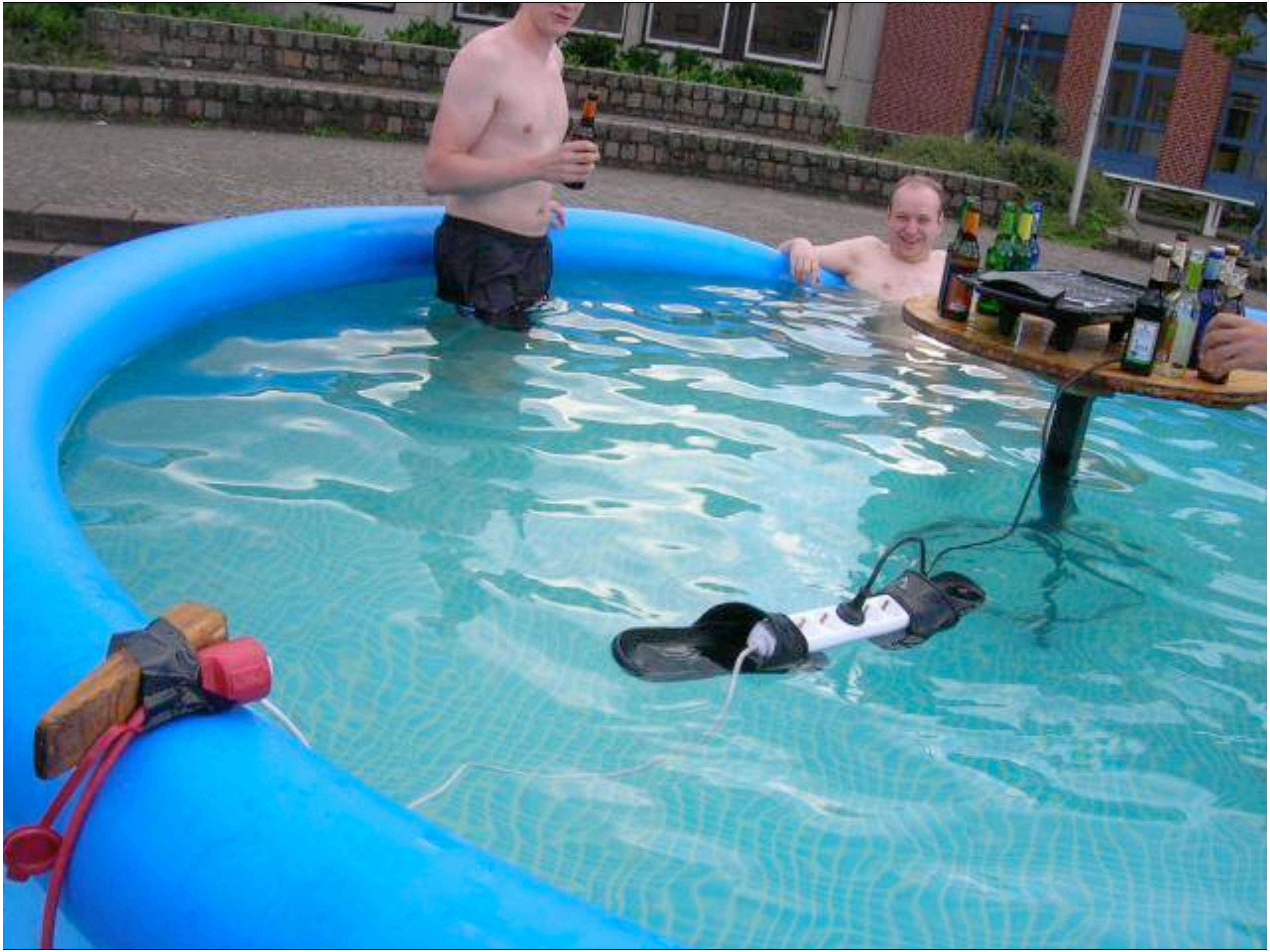
struts `ActionForm`



decoupling from struts

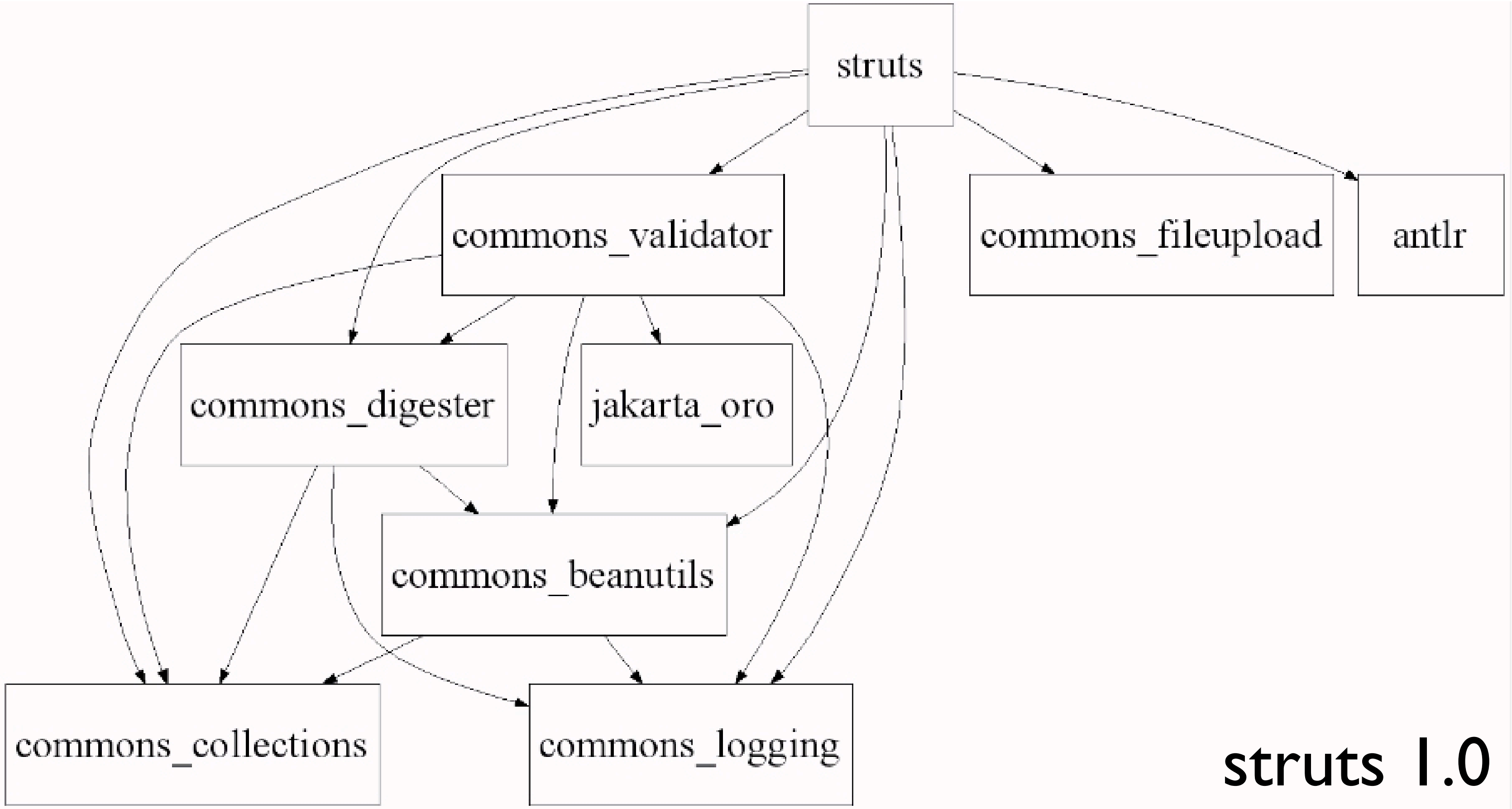


don’t decompose large things just because you can

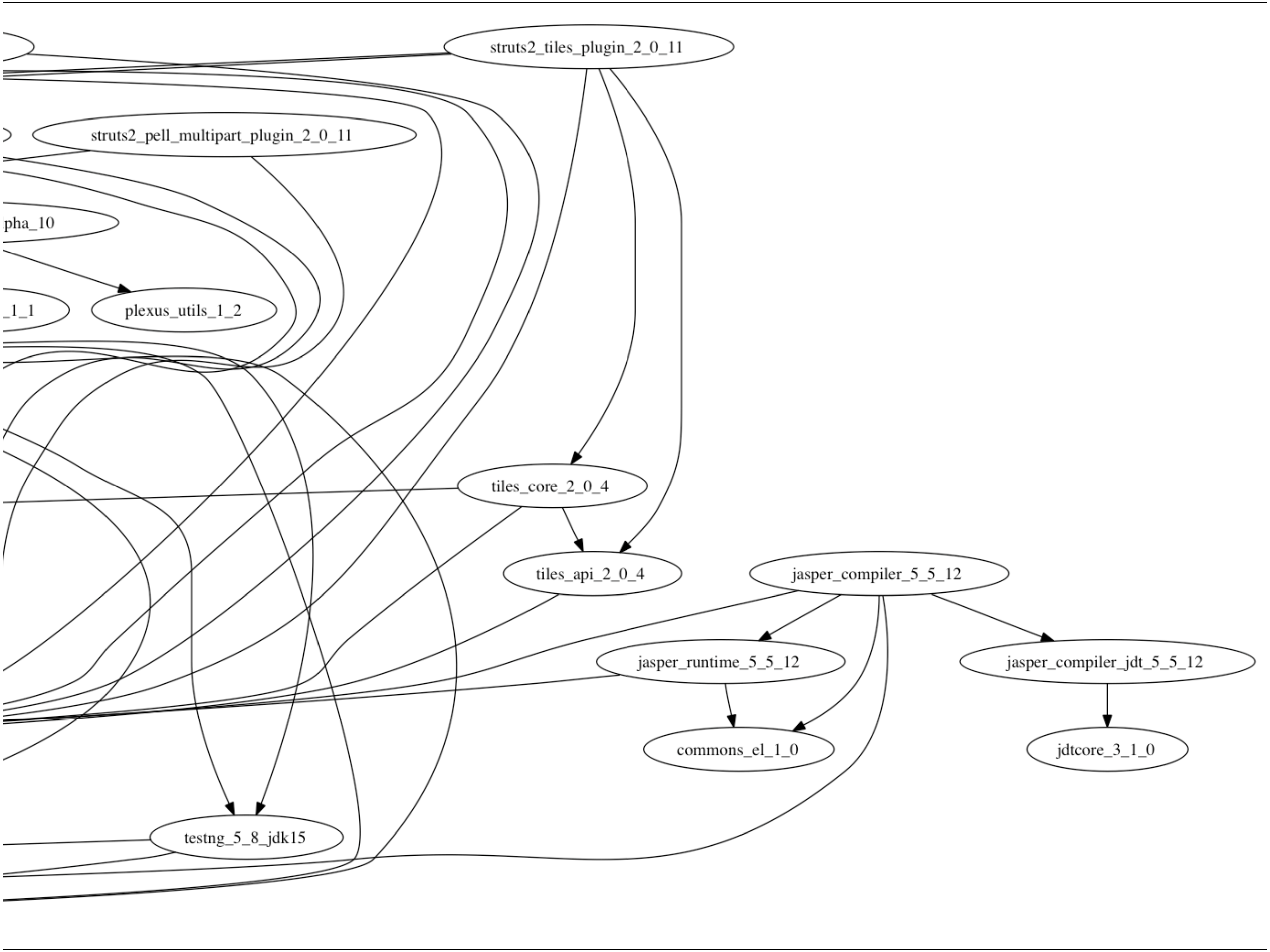


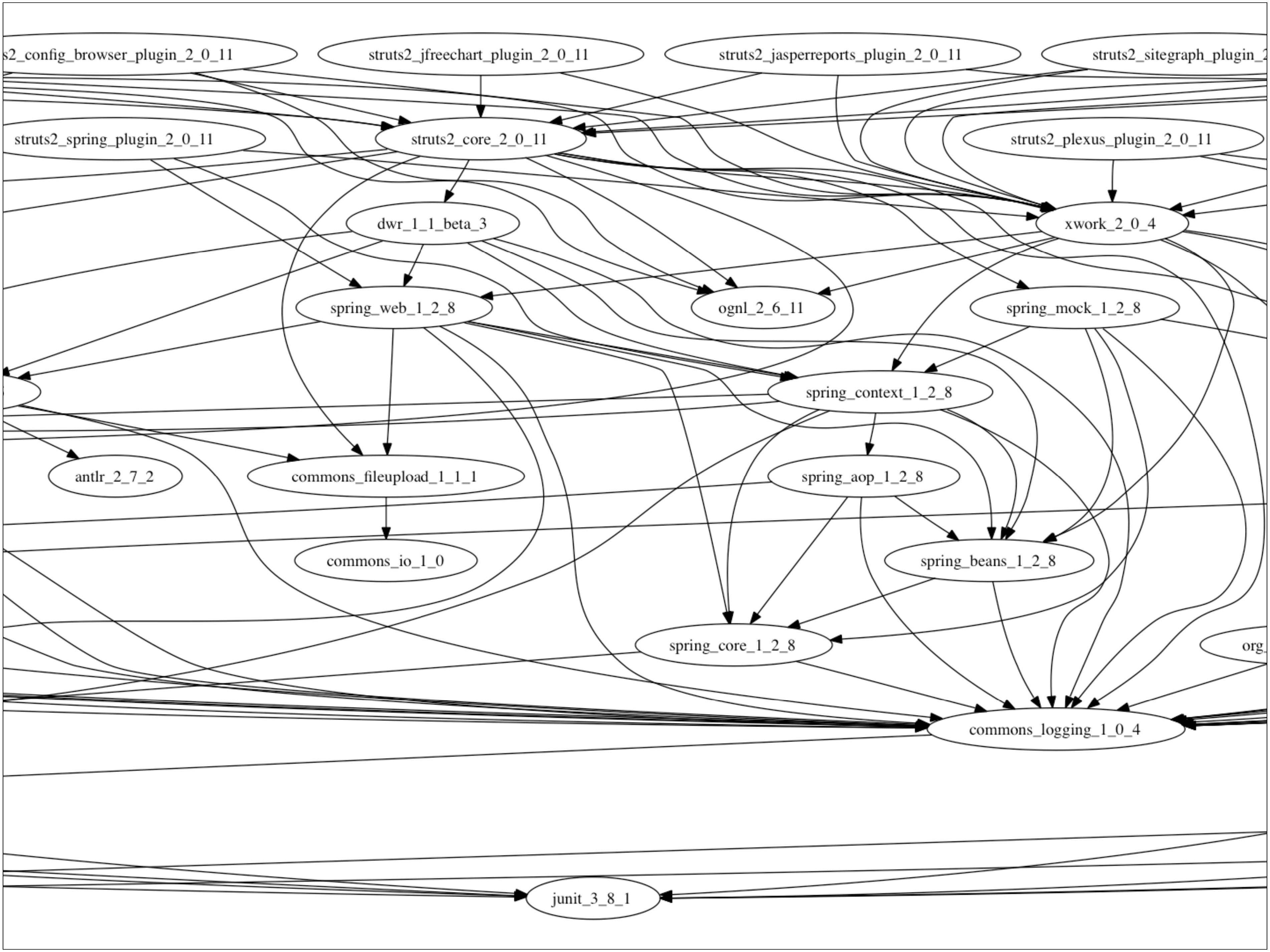
dependencies are killers

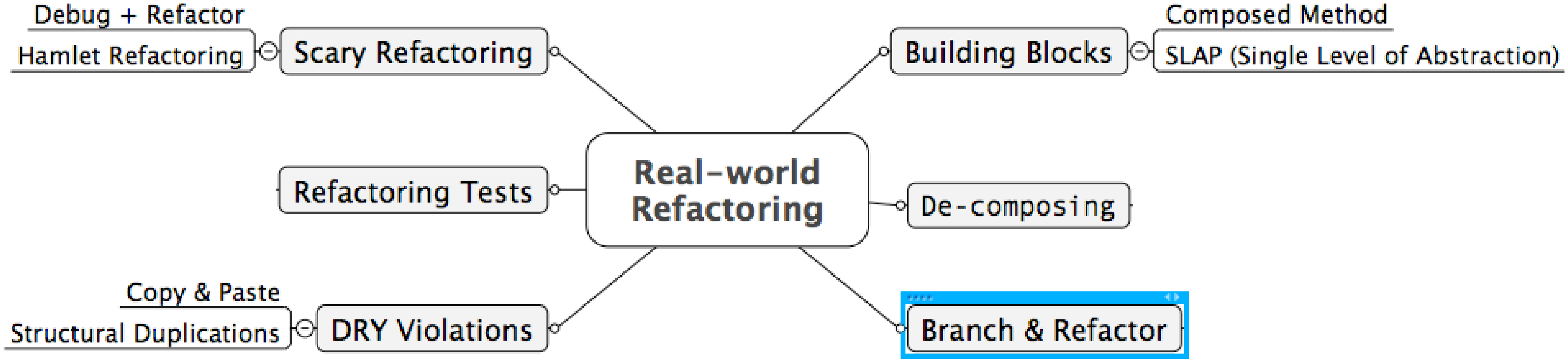
dependencies



struts 1.0


struts2_tiles_plugin_2_0_11
struts2_pell_multipart_plugin_2_0_11
pha_10
_1_1
plexus_utils_1_2
tiles_core_2_0_4
tiles_api_2_0_4
jasper_compiler_5_5_12
jasper_runtime_5_5_12
jasper_compiler_jdt_5_5_12
commons_el_1_0
jdtcore_3_1_0
testng_5_8_jdk15

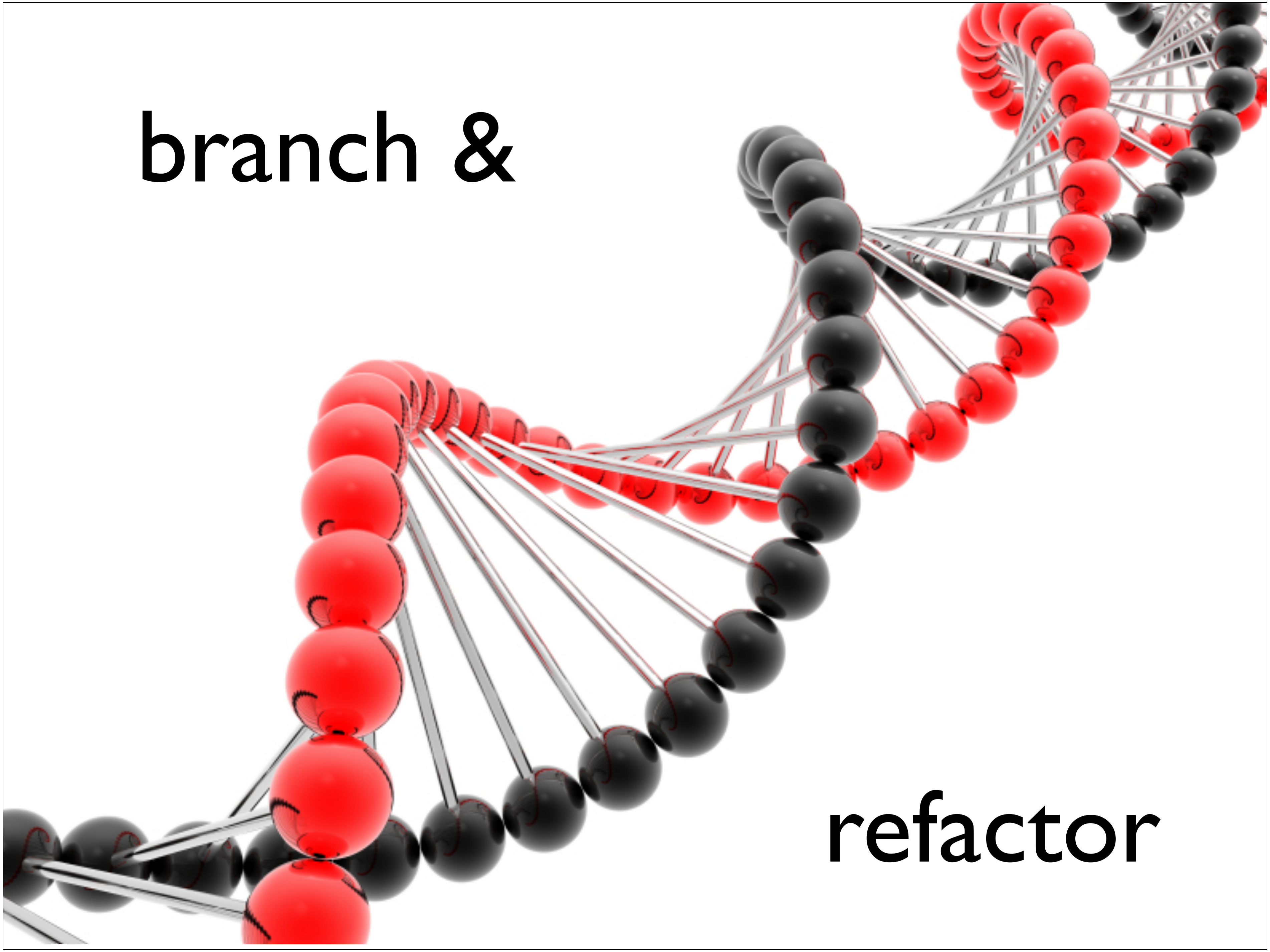

s2_config_browser_plugin_2_0_11
struts2_jfreechart_plugin_2_0_11
struts2_jasperreports_plugin_2_0_11
struts2_sitegraph_plugin_2
struts2_spring_plugin_2_0_11
struts2_core_2_0_11
struts2_plexus_plugin_2_0_11
dwr_1_1_beta_3
xwork_2_0_4
spring_web_1_2_8
ognl_2_6_11
spring_mock_1_2_8
spring_context_1_2_8
antlr_2_7_2
commons_fileupload_1_1_1
spring_aop_1_2_8
commons_io_1_0
spring_beans_1_2_8
spring_core_1_2_8
org_
commons_logging_1_0_4
junit_3_8_1



Real-world Refactoring

- Scary Refactoring
  - Debug + Refactor
  - Hamlet Refactoring
- Refactoring Tests
- DRY Violations
  - Copy & Paste
  - Structural Duplications
- Building Blocks
  - Composed Method
  - SLAP (Single Level of Abstraction)
- De-composing
- Branch & Refactor

branch &



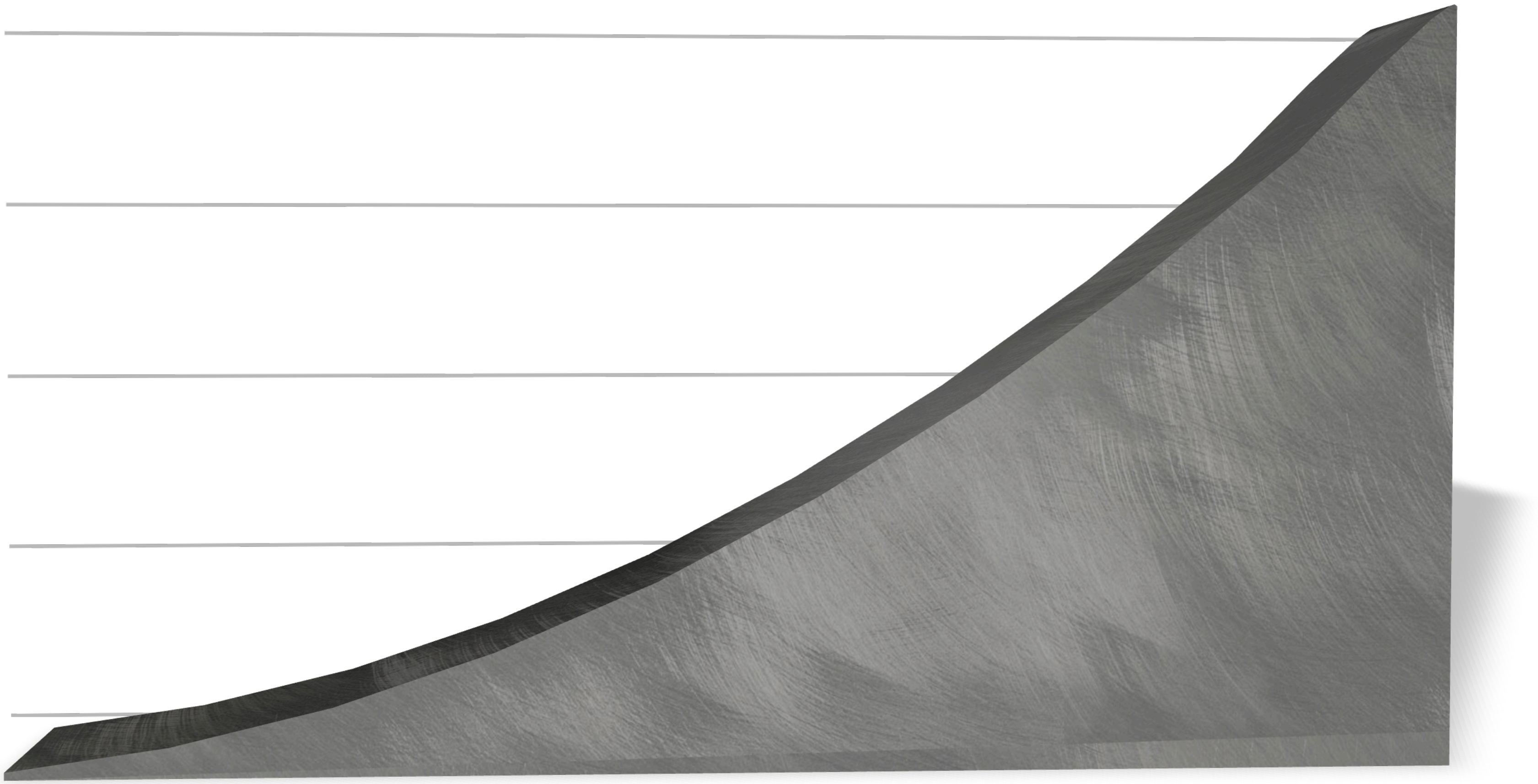
refactor

multi-day refactorings

bite the bullet

paying back technical debt

- like interest, debt payback doesn't touch the principle
- starts eating up a lot of useful time
- the longer the delay, the higher the price



the longer you put it off, the worse it gets

step

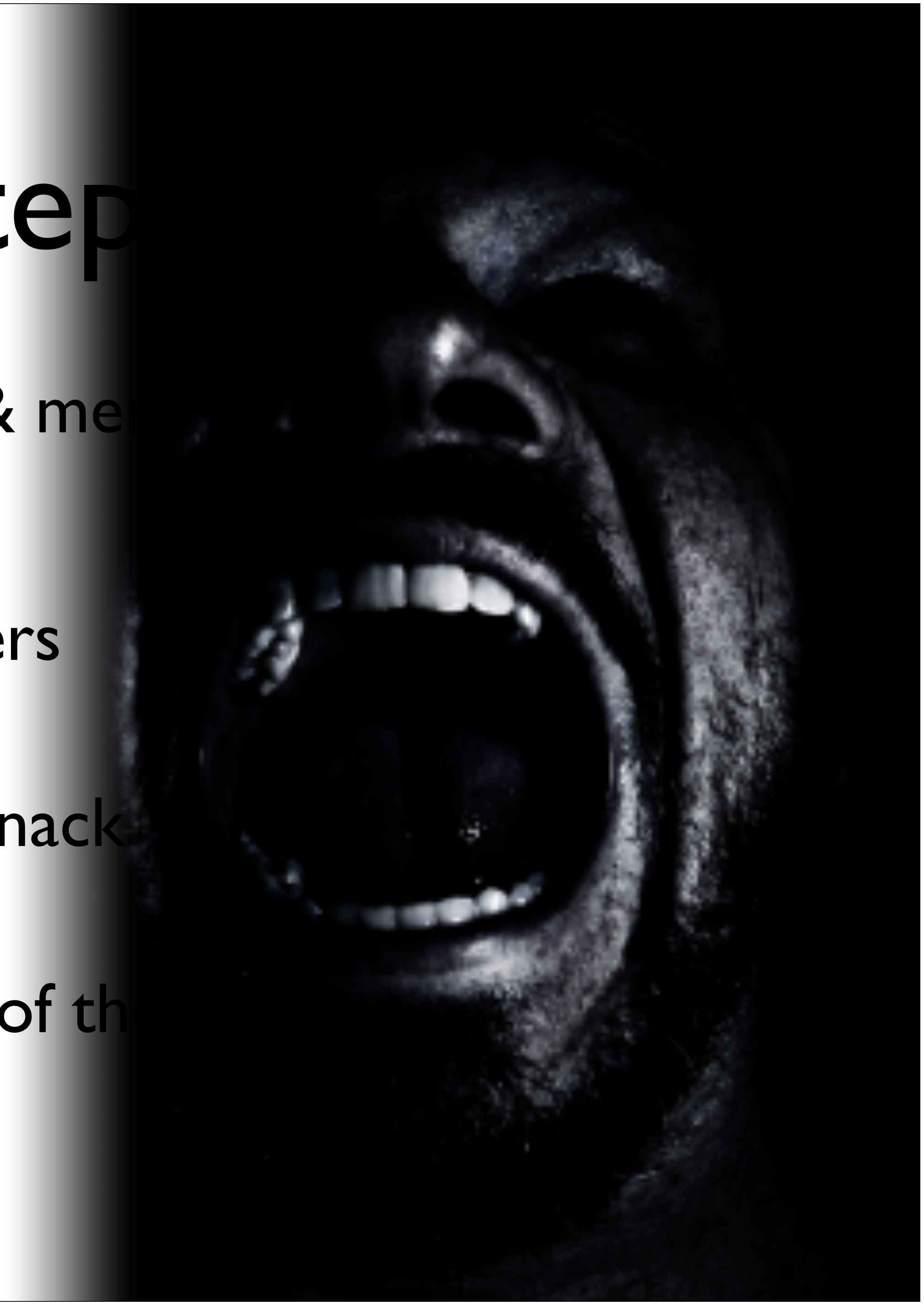
learn how branching & me
version control

find a pair of developers

get them caffeinated snack

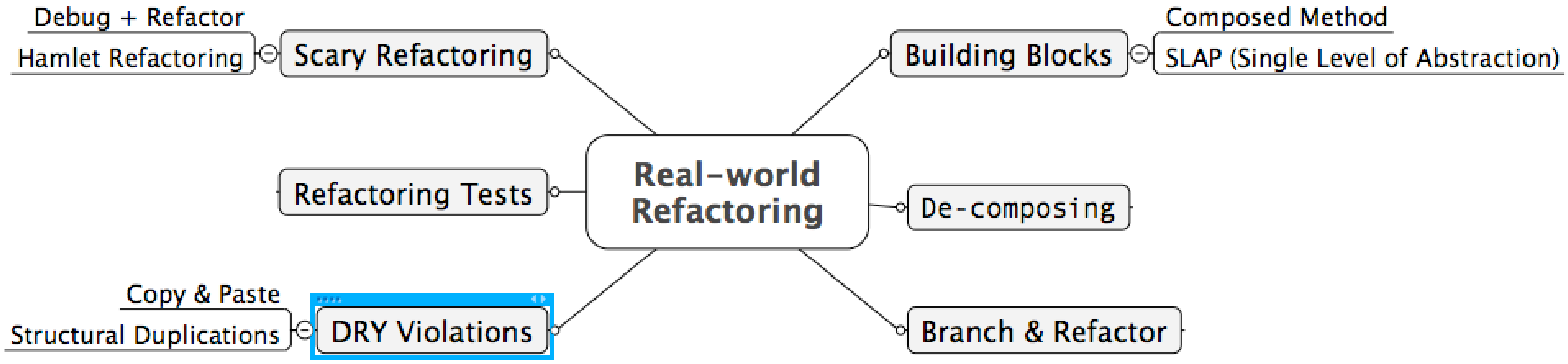
gently avoid that part of th

merge hell



time box

don’t be afraid to fight another day



Real-world Refactoring

Scary Refactoring

Debug + Refactor

Hamlet Refactoring

Refactoring Tests

DRY Violations

Copy & Paste

Structural Duplications

Building Blocks

Composed Method

SLAP (Single Level of Abstraction)

De-composing

Branch & Refactor

DRY violations

copy & paste

cpd

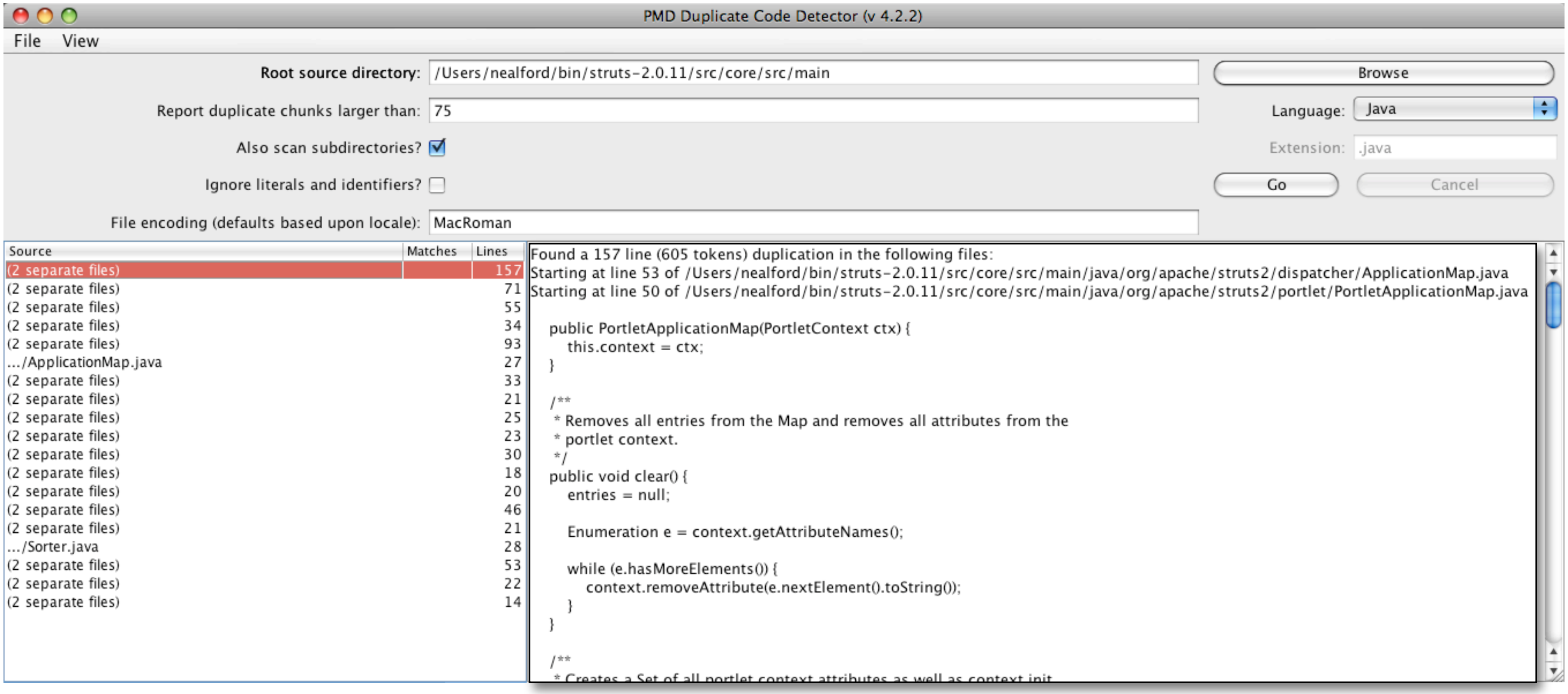
part of the source-code analysis tool **pmd**

configurable window of number of duplicate tokens

pre-configured with several languages

easy to add new language support

also simian (commercial)



Root source directory: /Users/nealford/bin/struts-2.0.11/src/core/src/main

Report duplicate chunks larger than: 75

Language: Java

Also scan subdirectories? ☑

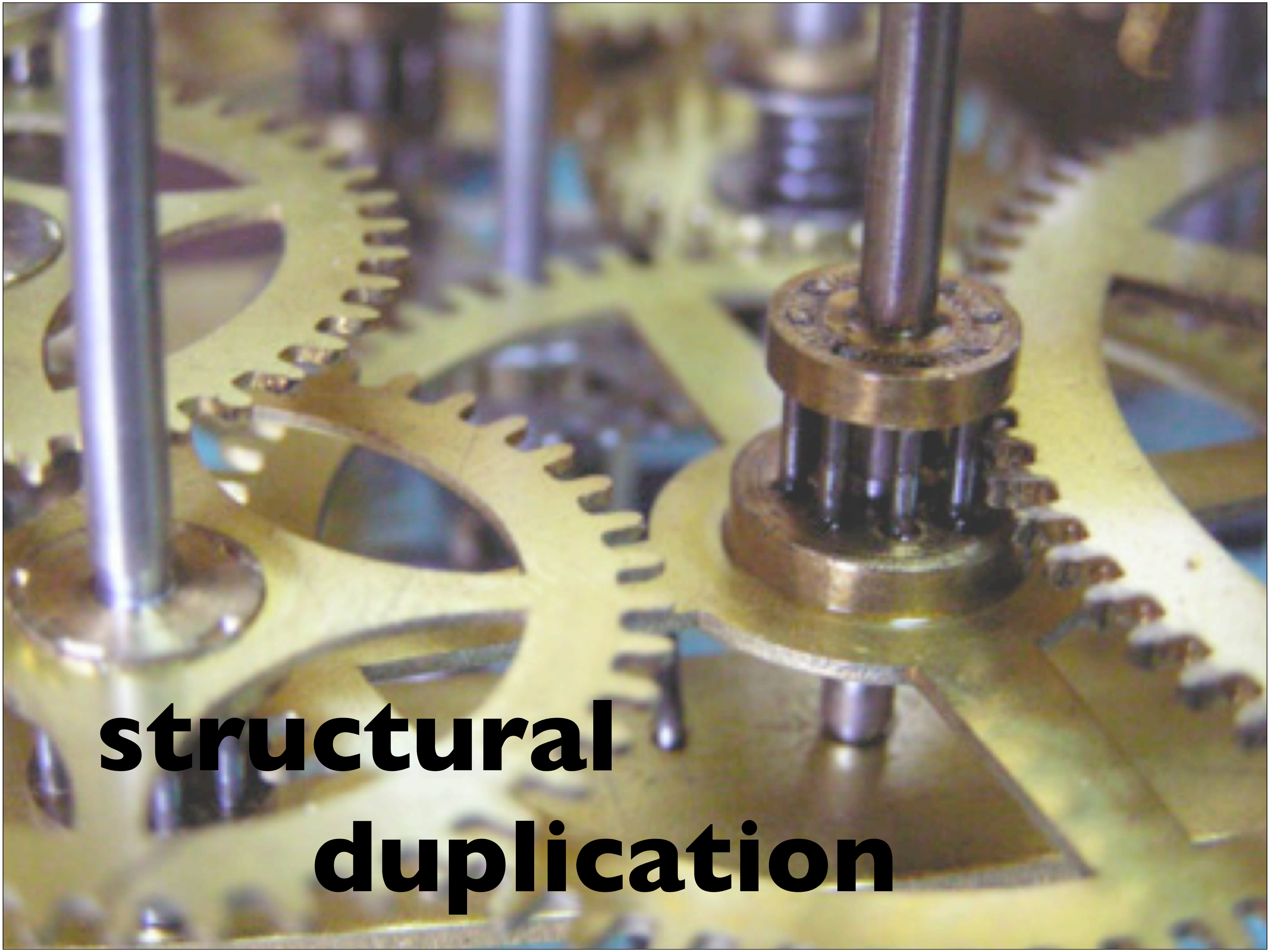
Extension: .java

Ignore literals and identifiers? □

File encoding (defaults based upon locale): MacRoman

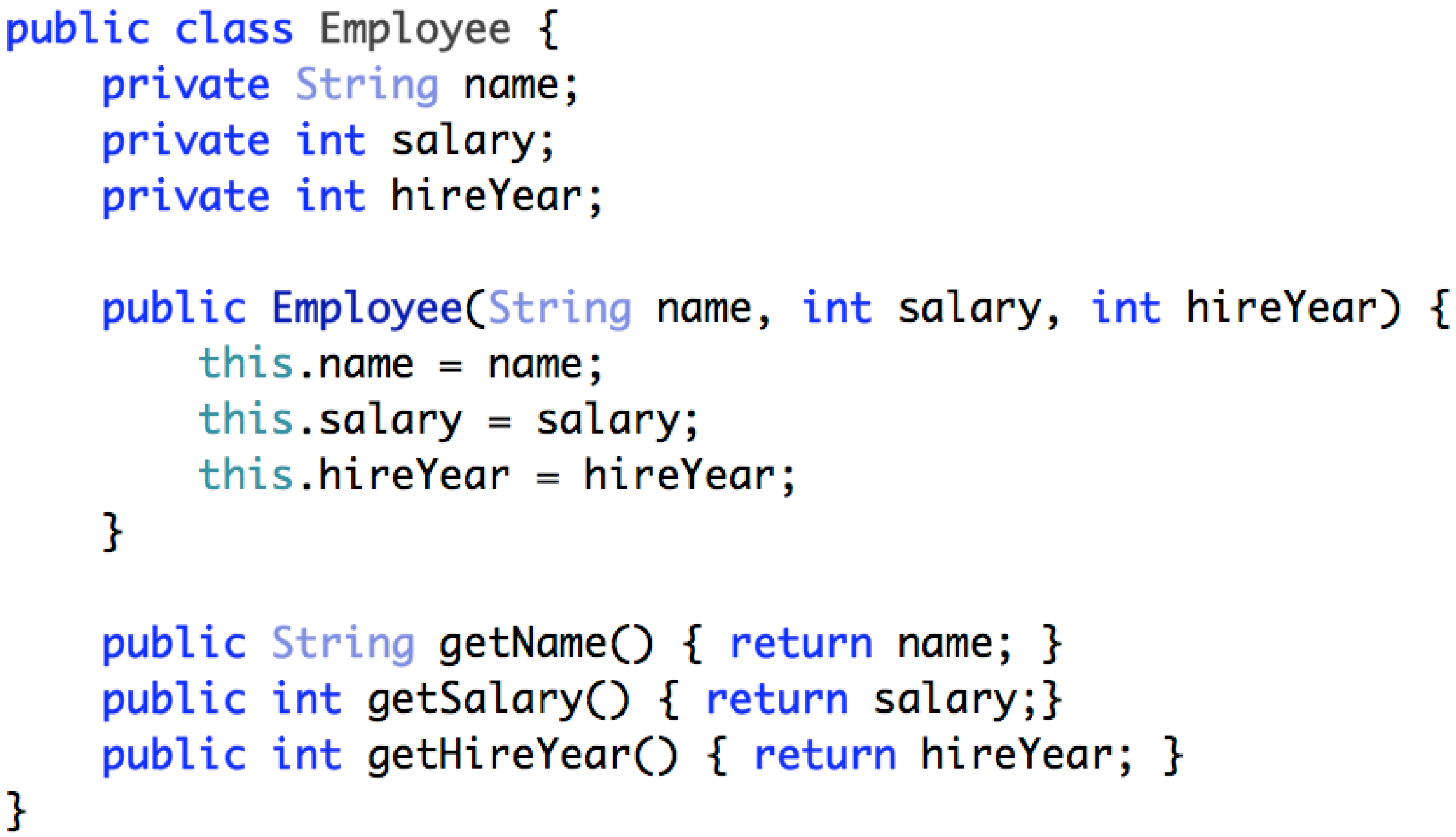
Source	Matches	Lines
(2 separate files)		157
(2 separate files)		71
(2 separate files)		55
(2 separate files)		34
(2 separate files)		93
.../ApplicationMap.java		27
(2 separate files)		33
(2 separate files)		21
(2 separate files)		25
(2 separate files)		23
(2 separate files)		30
(2 separate files)		18
(2 separate files)		20
(2 separate files)		46
(2 separate files)		21
.../Sorter.java		28
(2 separate files)		53
(2 separate files)		22
(2 separate files)		14

```
Found a 157 line (605 tokens) duplication in the following files:
Starting at line 53 of /Users/nealford/bin/struts-2.0.11/src/core/src/main/java/org/apache/struts2/dispatcher/ApplicationMap.java
Starting at line 50 of /Users/nealford/bin/struts-2.0.11/src/core/src/main/java/org/apache/struts2/portlet/PortletApplicationMap.java

  public PortletApplicationMap(PortletContext ctx) {
     this.context = ctx;
  }

  /**
   * Removes all entries from the Map and removes all attributes from the
   * portlet context.
   */
  public void clear() {
     entries = null;

     Enumeration e = context.getAttributeNames();

     while (e.hasMoreElements()) {
        context.removeAttribute(e.nextElement().toString());
     }
  }

  /**
   * Creates a Set of all portlet context attributes as well as context init
```



structural duplication

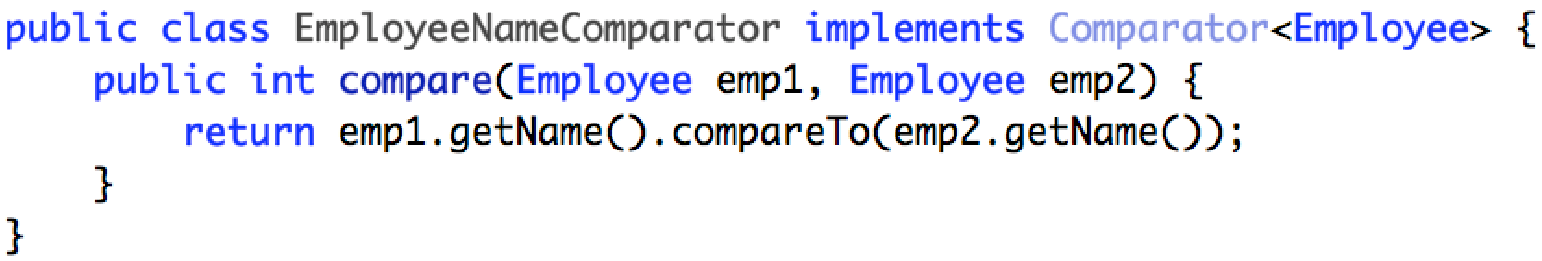
given:



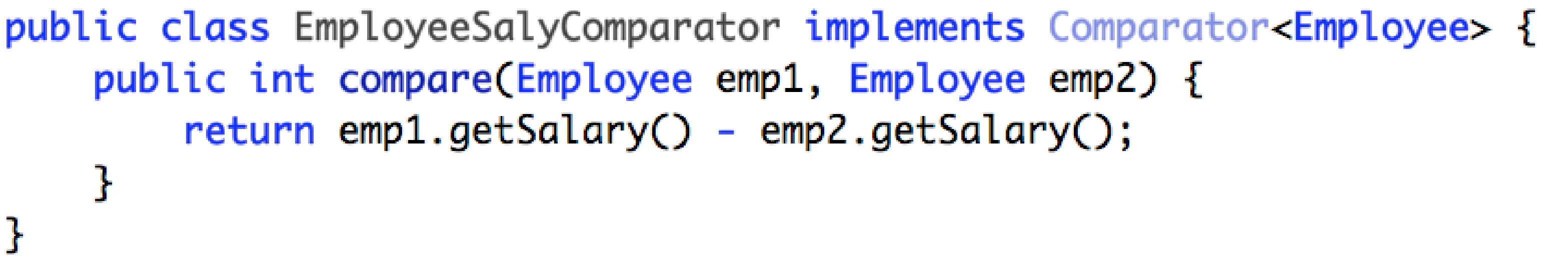
```java
public class Employee {
    private String name;
    private int salary;
    private int hireYear;

    public Employee(String name, int salary, int hireYear) {
        this.name = name;
        this.salary = salary;
        this.hireYear = hireYear;
    }

    public String getName() { return name; }
    public int getSalary() { return salary;}
    public int getHireYear() { return hireYear; }
}
```

goal: sort on any property

comparator mania!

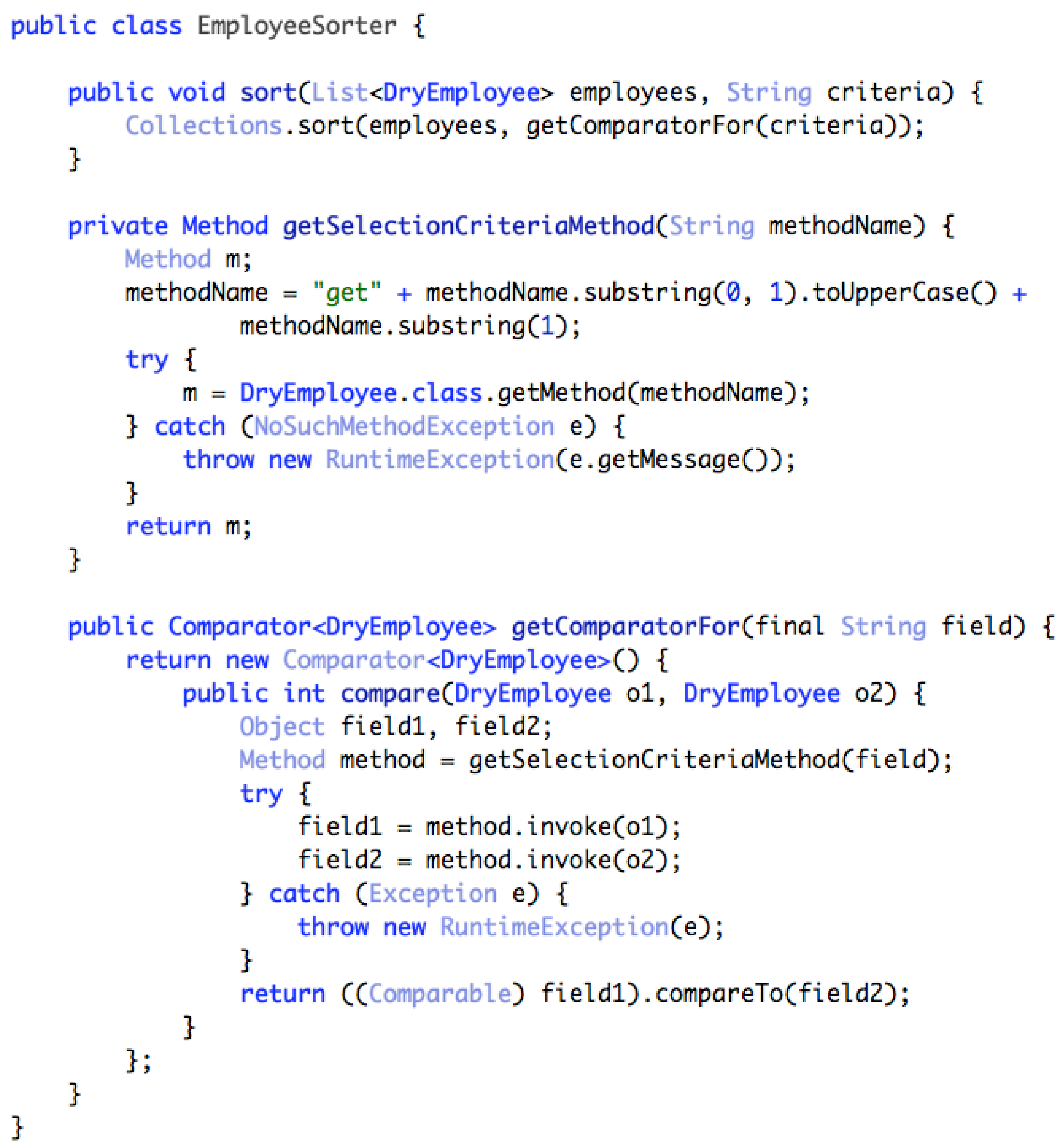


```
public class EmployeeNameComparator implements Comparator<Employee> {
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getName().compareTo(emp2.getName());
    }
}
```



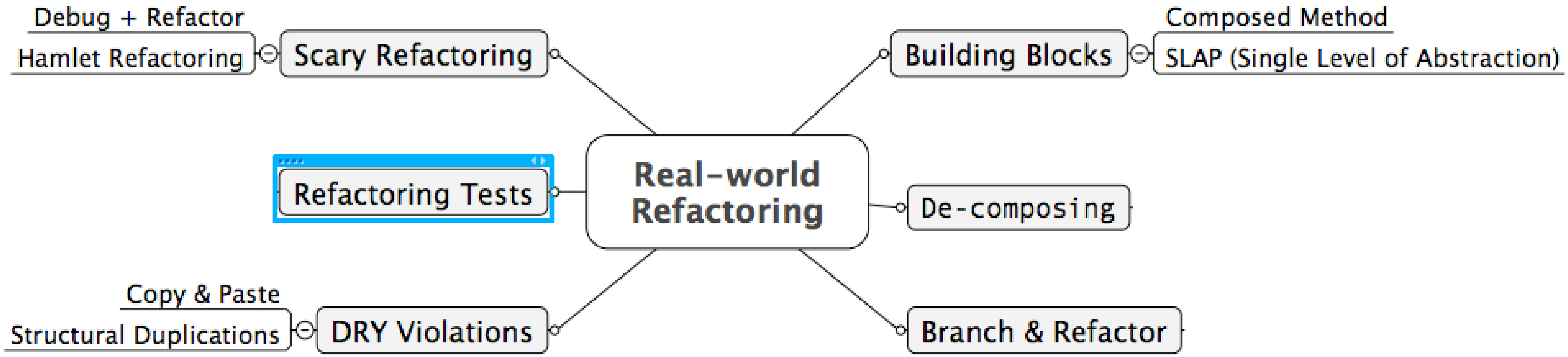
```
public class EmployeeSalyComparator implements Comparator<Employee> {
    public int compare(Employee emp1, Employee emp2) {
        return emp1.getSalary() - emp2.getSalary();
    }
}
```

same whitespace, different values



```
public class EmployeeSorter {

    public void sort(List<DryEmployee> employees, String criteria) {
        Collections.sort(employees, getComparatorFor(criteria));
    }

    private Method getSelectionCriteriaMethod(String methodName) {
        Method m;
        methodName = "get" + methodName.substring(0, 1).toUpperCase() +
                methodName.substring(1);
        try {
            m = DryEmployee.class.getMethod(methodName);
        } catch (NoSuchMethodException e) {
            throw new RuntimeException(e.getMessage());
        }
        return m;
    }

    public Comparator<DryEmployee> getComparatorFor(final String field) {
        return new Comparator<DryEmployee>() {
            public int compare(DryEmployee o1, DryEmployee o2) {
                Object field1, field2;
                Method method = getSelectionCriteriaMethod(field);
                try {
                    field1 = method.invoke(o1);
                    field2 = method.invoke(o2);
                } catch (Exception e) {
                    throw new RuntimeException(e);
                }
                return ((Comparable) field1).compareTo(field2);
            }
        };
    }
}
```

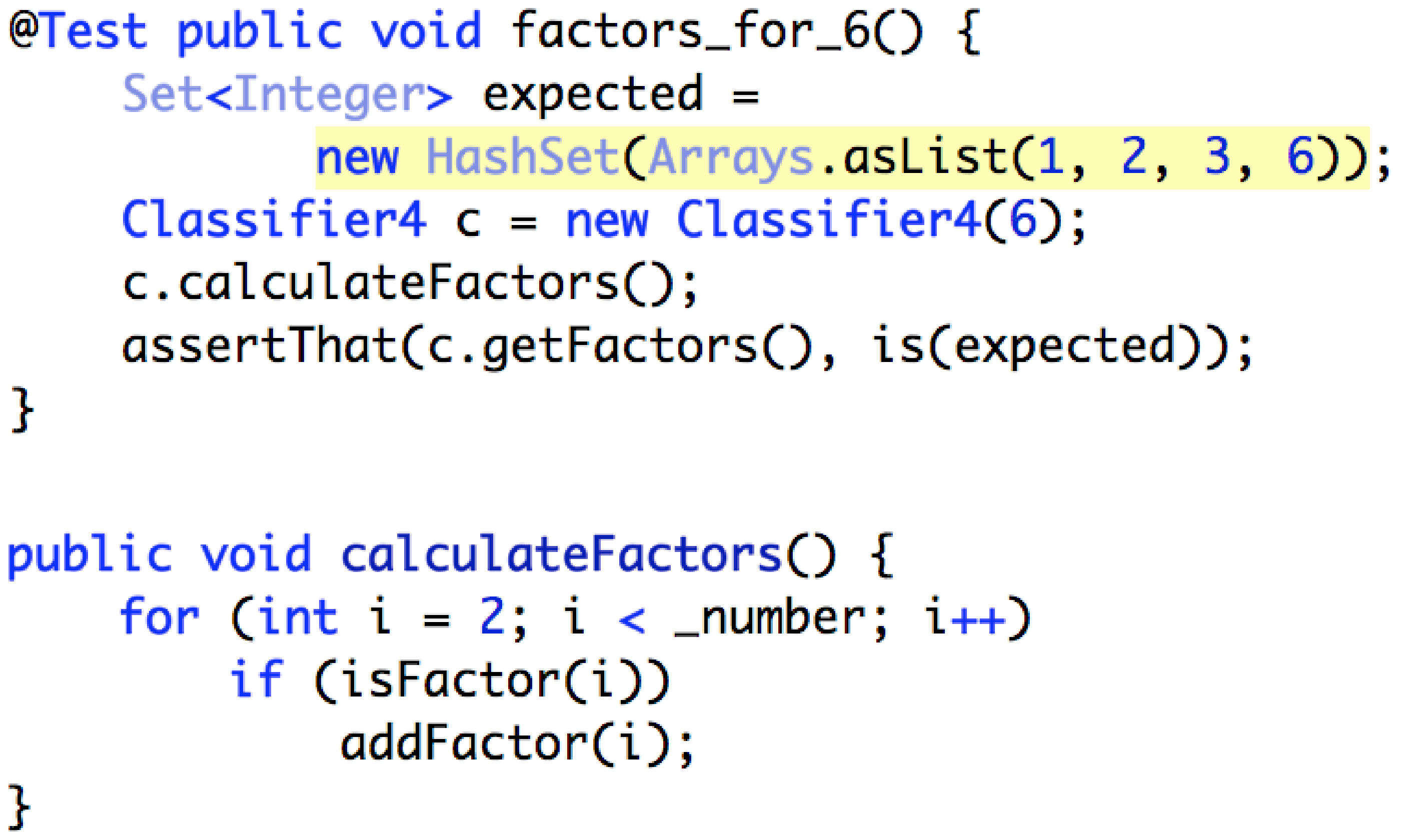
```java
@Before public void setup() {
    _sorter = new EmployeeSorter();
    _list = new ArrayList<DryEmployee>();
    _list.add(new DryEmployee("Homer", 20000, 1975));
    _list.add(new DryEmployee("Smithers", 150000, 1980));
    _list.add(new DryEmployee("Lenny", 100000, 1982));
}

@Test public void name_comparisons() {
    _sorter.sort(_list, "name");
    assertThat(_list.get(0).getName(), is("Homer"));
    assertThat(_list.get(1).getName(), is("Lenny"));
    assertThat(_list.get(2).getName(), is("Smithers"));
}

@Test public void salary_comparisons() {
    _sorter.sort(_list, "salary");
    assertThat(_list.get(0).getSalary(), is(20000));
    assertThat(_list.get(1).getSalary(), is(100000));
    assertThat(_list.get(2).getSalary(), is(150000));
}

@Test public void hireYearComparison() {
    _sorter.sort(_list, "hireYear");
    assertThat(_list.get(0).getHireYear(), is(1975));
    assertThat(_list.get(1).getHireYear(), is(1980));
    assertThat(_list.get(2).getHireYear(), is(1982));
}
```



Real-world Refactoring

- Scary Refactoring
  - Debug + Refactor
  - Hamlet Refactoring
- Refactoring Tests
- DRY Violations
  - Copy & Paste
  - Structural Duplications
- Building Blocks
  - Composed Method
  - SLAP (Single Level of Abstraction)
- De-composing
- Branch & Refactor

test:
calculateFactors()



```
@Test public void factors_for_6() {
    Set<Integer> expected =
            new HashSet(Arrays.asList(1, 2, 3, 6));
    Classifier4 c = new Classifier4(6);
    c.calculateFactors();
    assertThat(c.getFactors(), is(expected));
}

public void calculateFactors() {
    for (int i = 2; i < _number; i++)
        if (isFactor(i))
            addFactor(i);
}
```

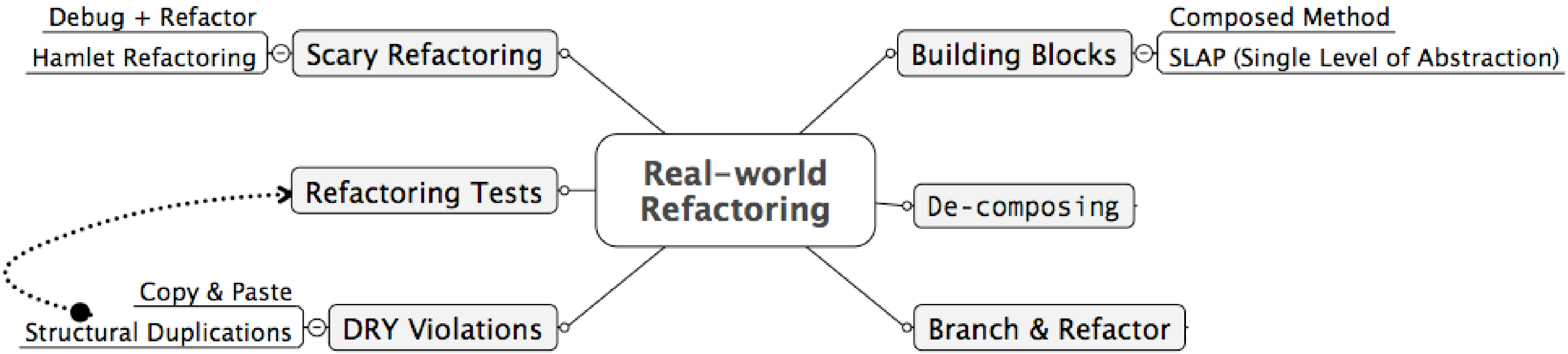
ok for tests to be moist...

...but not drenched

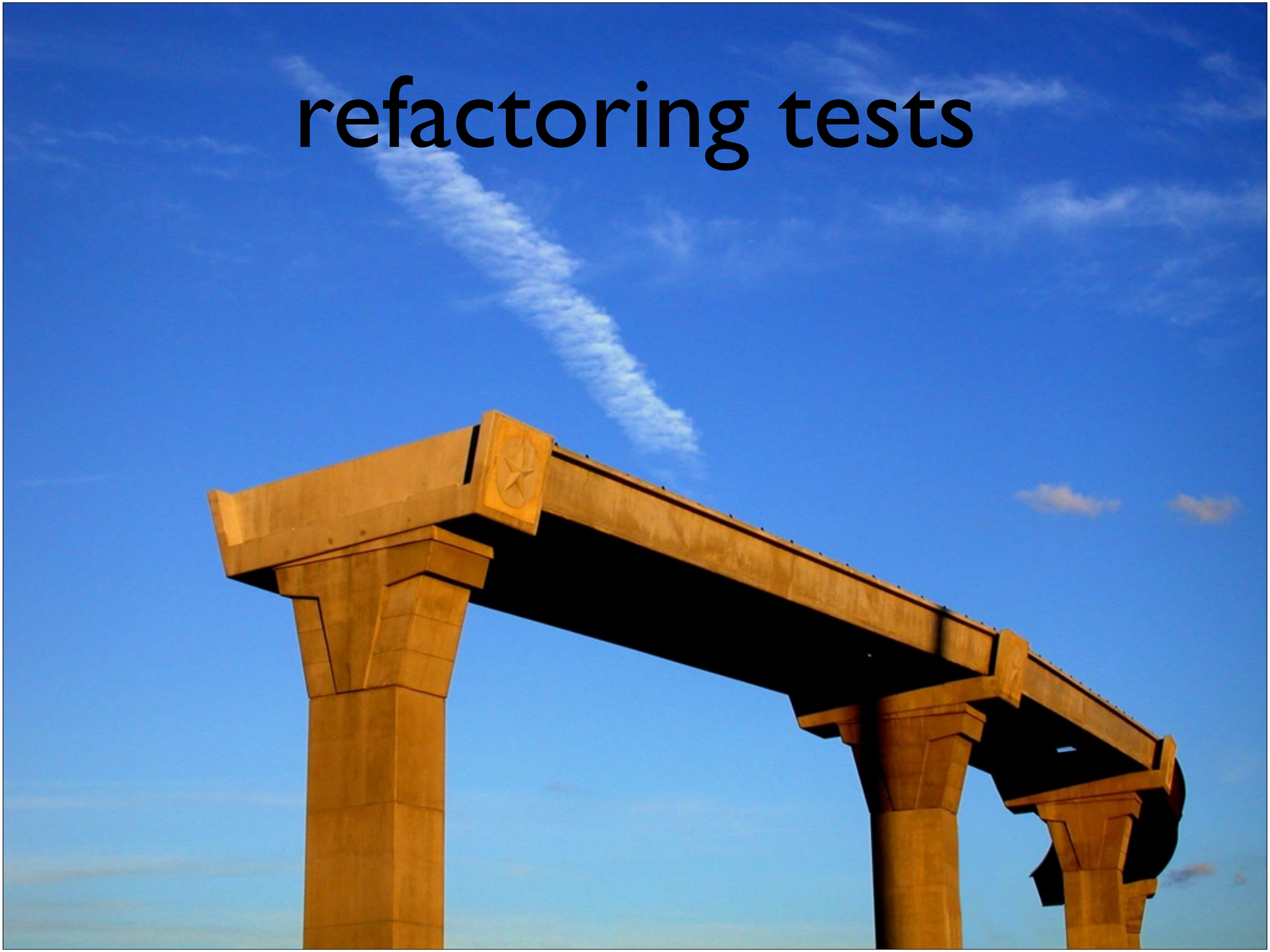
refactor

```
private Set<Integer> expectationSetWith(Integer... numbers) {
    return new HashSet<Integer>(Arrays.asList(numbers));
}
```

```
@Test public void factors_for_6() {
    Set<Integer> expected = expectationSetWith(1, 2, 3, 6);
    Classifier4 c = new Classifier4(6);
    c.calculateFactors();
    assertThat(c.getFactors(), is(expected));
}
```

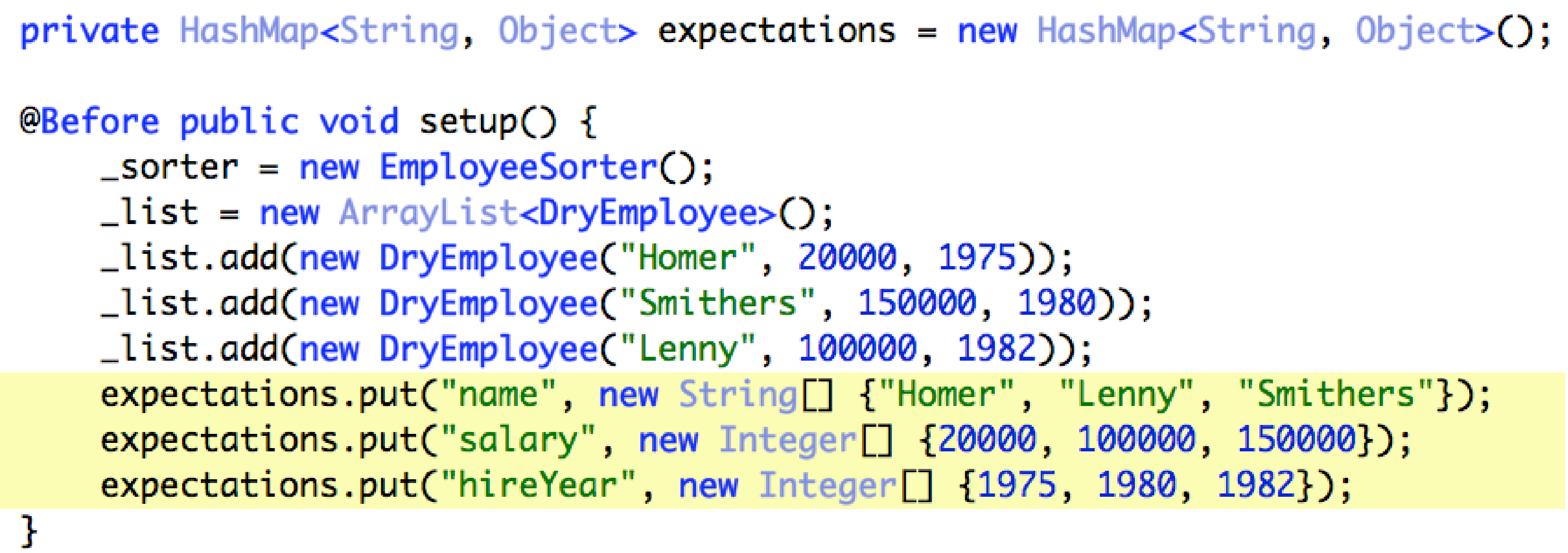

Real-world Refactoring
Scary Refactoring
Debug + Refactor
Hamlet Refactoring
Refactoring Tests
DRY Violations
Copy & Paste
Structural Duplications
Building Blocks
Composed Method
SLAP (Single Level of Abstraction)
De-composing
Branch & Refactor

refactoring tests



```
@Before public void setup() {
    _sorter = new EmployeeSorter();
    _list = new ArrayList<DryEmployee>();
    _list.add(new DryEmployee("Homer", 20000, 1975));
    _list.add(new DryEmployee("Smithers", 150000, 1980));
    _list.add(new DryEmployee("Lenny", 100000, 1982));
}

@Test public void name_comparisons() {
    _sorter.sort(_list, "name");
    assertThat(_list.get(0).getName(), is("Homer"));
    assertThat(_list.get(1).getName(), is("Lenny"));
    assertThat(_list.get(2).getName(), is("Smithers"));
}

@Test public void salary_comparisons() {
    _sorter.sort(_list, "salary");
    assertThat(_list.get(0).getSalary(), is(20000));
    assertThat(_list.get(1).getSalary(), is(100000));
    assertThat(_list.get(2).getSalary(), is(150000));
}

@Test public void hireYearComparison() {
    _sorter.sort(_list, "hireYear");
    assertThat(_list.get(0).getHireYear(), is(1975));
    assertThat(_list.get(1).getHireYear(), is(1980));
    assertThat(_list.get(2).getHireYear(), is(1982));
}
```

generic tests



```
private HashMap<String, Object> expectations = new HashMap<String, Object>();

@Before public void setup() {
    _sorter = new EmployeeSorter();
    _list = new ArrayList<DryEmployee>();
    _list.add(new DryEmployee("Homer", 20000, 1975));
    _list.add(new DryEmployee("Smithers", 150000, 1980));
    _list.add(new DryEmployee("Lenny", 100000, 1982));
    expectations.put("name", new String[] {"Homer", "Lenny", "Smithers"});
    expectations.put("salary", new Integer[] {20000, 100000, 150000});
    expectations.put("hireYear", new Integer[] {1975, 1980, 1982});
}
```

```java
private String[] FIELDS = new String[] {"name", "salary", "hireYear"};

@Test public void all_comparators() {
    for (String field : FIELDS) {
        sorter.sort(_list, field);
```

```java
private HashMap<String, Object> expectations = new HashMap<String, Object>();

expectations.put("name", new String[] {"Homer", "Lenny", "Smithers"});
expectations.put("salary", new Integer[] {20000, 100000, 150000});
expectations.put("hireYear", new Integer[] {1975, 1980, 1982});
```

```java
                        class.
                        getDeclaredMethod("get" + methodNameFromString(field));
                for (int i = 0; i < o.length; i++)
                    assertThat(m.invoke(_list.get(i)), is(o[i]));
            } catch (Exception e) {
                fail();
            }
        }
    }
}

private static String methodNameFromString(String s) {
    return s.substring(0, 1).toUpperCase() + s.substring(1);
}
```

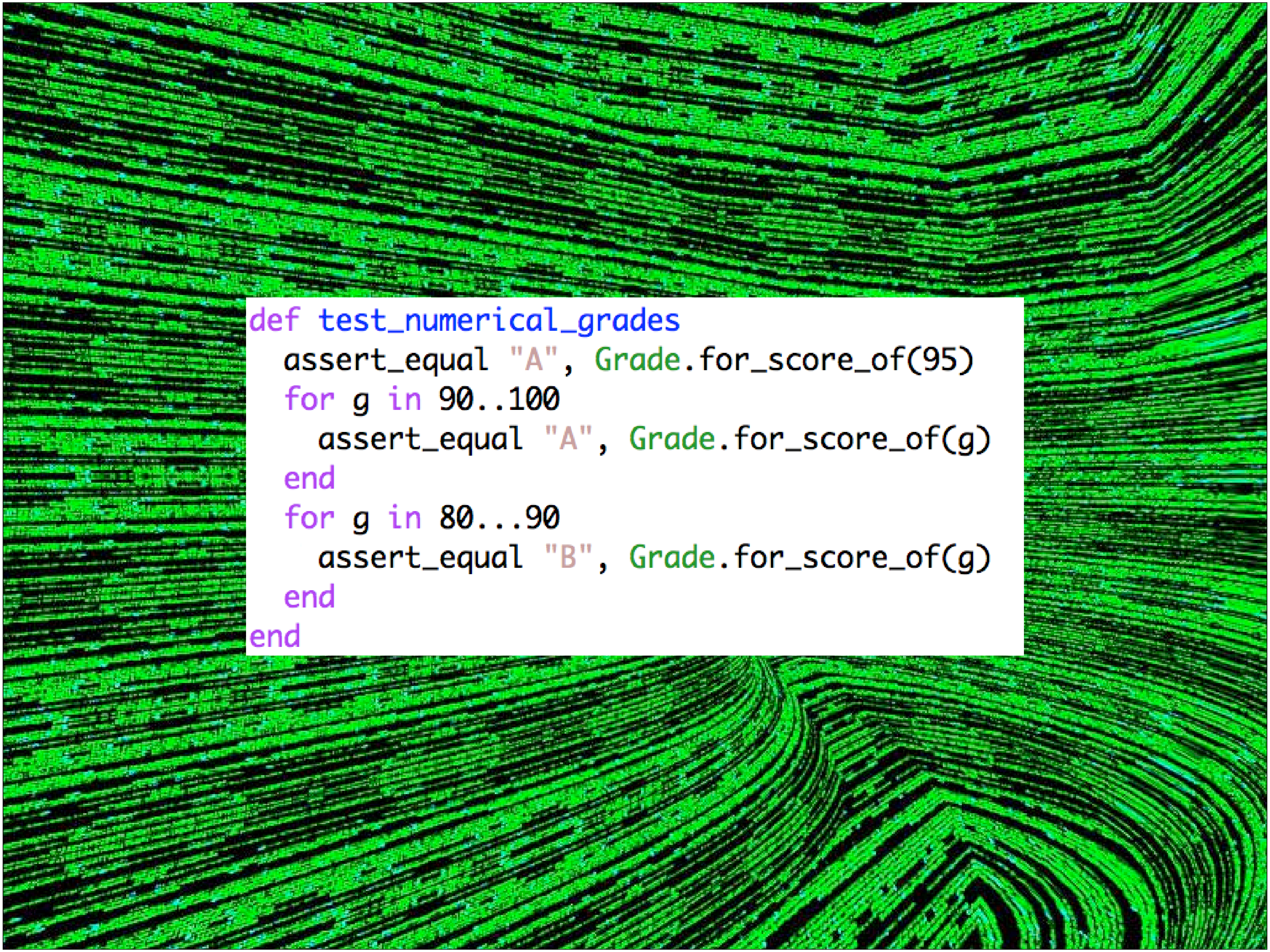
is this a good idea?

how far is *too* far?

reflection potion

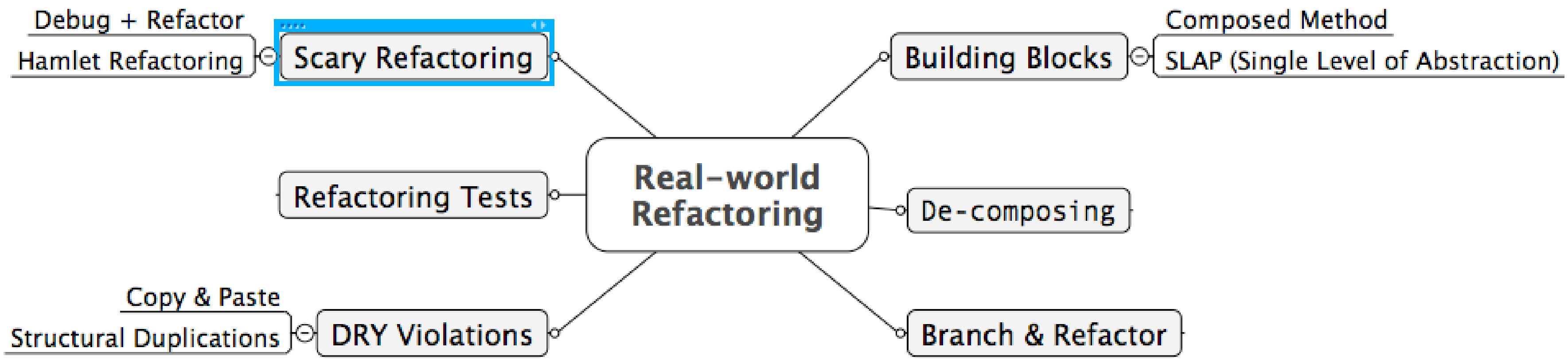
don’t fear powerful things

```ruby
class Grade
  class << self
    def for_score_of(grade)
      case grade
        when 90..100: 'A'
        when 80..90 : 'B'
        when 70..80 : 'C'
        when 60..70 : 'D'
        when Integer: 'F'
        when /[A-D]/, /[F]/ : grade
        else raise "Not a grade: #{grade}"
      end
    end
  end
end
```



```
def test_numerical_grades
  assert_equal "A", Grade.for_score_of(95)
  for g in 90..100
    assert_equal "A", Grade.for_score_of(g)
  end
  for g in 80...90
    assert_equal "B", Grade.for_score_of(g)
  end
end
```

```
TestGrades.class_eval do
  grade_range = {
    'A' => 90..100,
    'B' => 80...90,
    'C' => 70...80,
    'D' => 60...70,
    'F' => 0...60}

  grade_range.each do |k, v|
    method_name = ("test_" + k + "_letter_grade").to_sym
    define_method method_name do
      for g in v
        assert_equal k, Grade.for_score_of(g)
      end
    end
  end
end
```



Real-world Refactoring

Scary Refactoring

Debug + Refactor

Hamlet Refactoring

Refactoring Tests

DRY Violations

Copy & Paste

Structural Duplications

Building Blocks

Composed Method

SLAP (Single Level of Abstraction)

De-composing

Branch & Refactor

scary refactoring

debug + refactor

the problem:

aging code base

no tests

lots of bugs

strong desire to refactor...

...plus gut-wrenching fear

attack plan

draw a line in the sand:

starting next thursday, our test coverage will always go up

every time you add a feature, write tests

every time you fix a bug, write a test

BUT! tons of looooooooooong methods

refactoring attack

refactor to composed method using extract method

(you’re debugging anyway)

once you extract the buggy code...

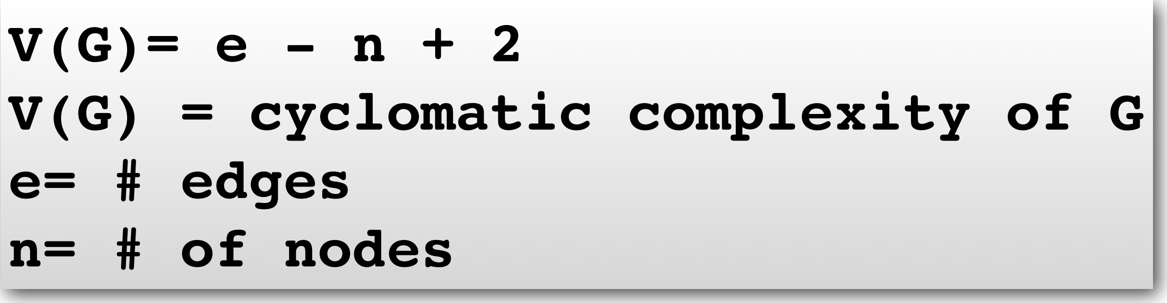
...write tests for it

tests grow around most fragile code first

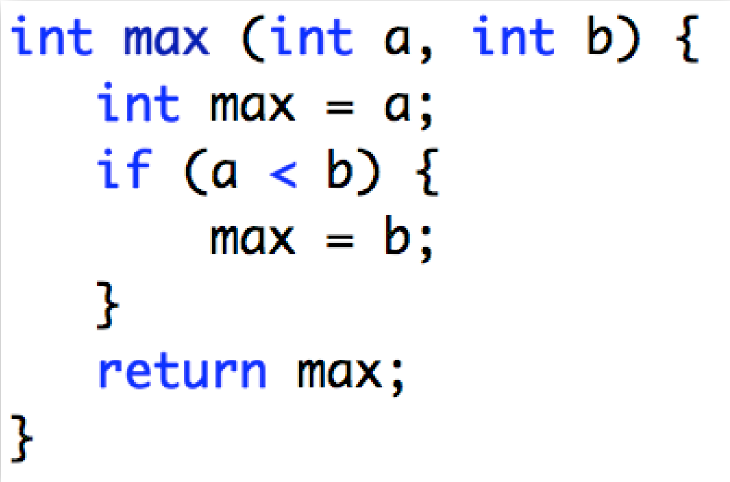
to
refactor
or not
to
refactor?

cyclomatic complexity

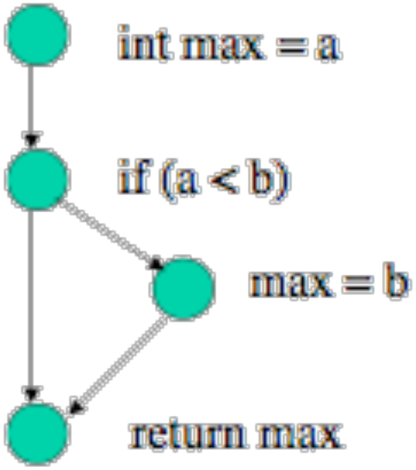
measures complexity of a function



```
V(G)= e - n + 2
V(G) = cyclomatic complexity of G
e= # edges
n= # of nodes
```



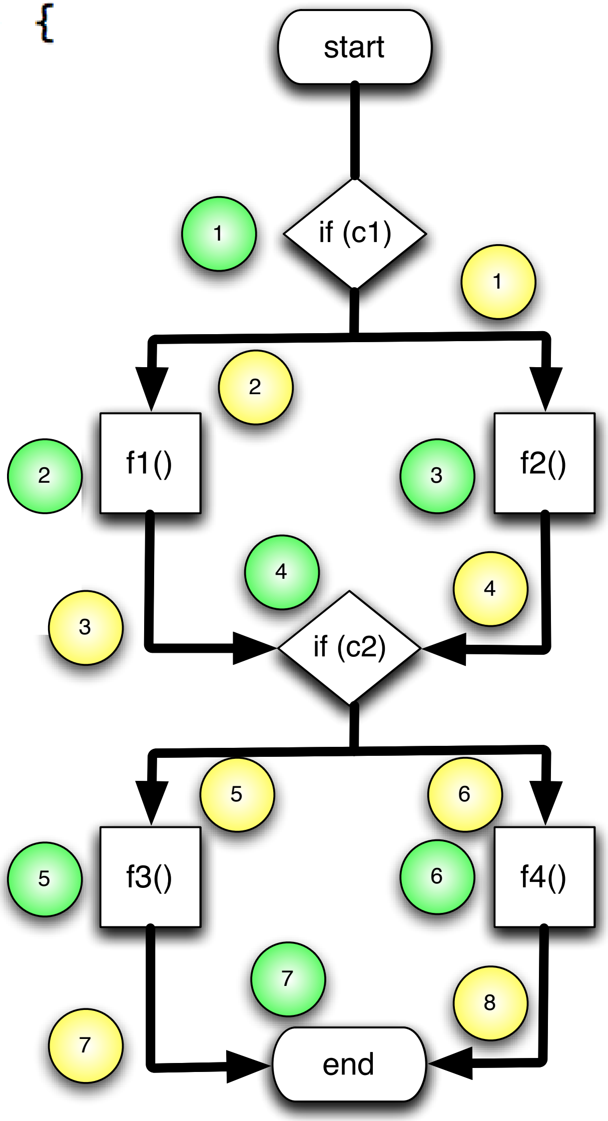
```
int max (int a, int b) {
   int max = a;
   if (a < b) {
       max = b;
   }
   return max;
}
```



```
public void doIt() {
    if (c1) {
        f1();
    } else {
        f2();
    }
    if (c2) {
        f3();
    } else {
        f4();
    }
}
```

🟩 nodes

🟨 edges



afferent coupling

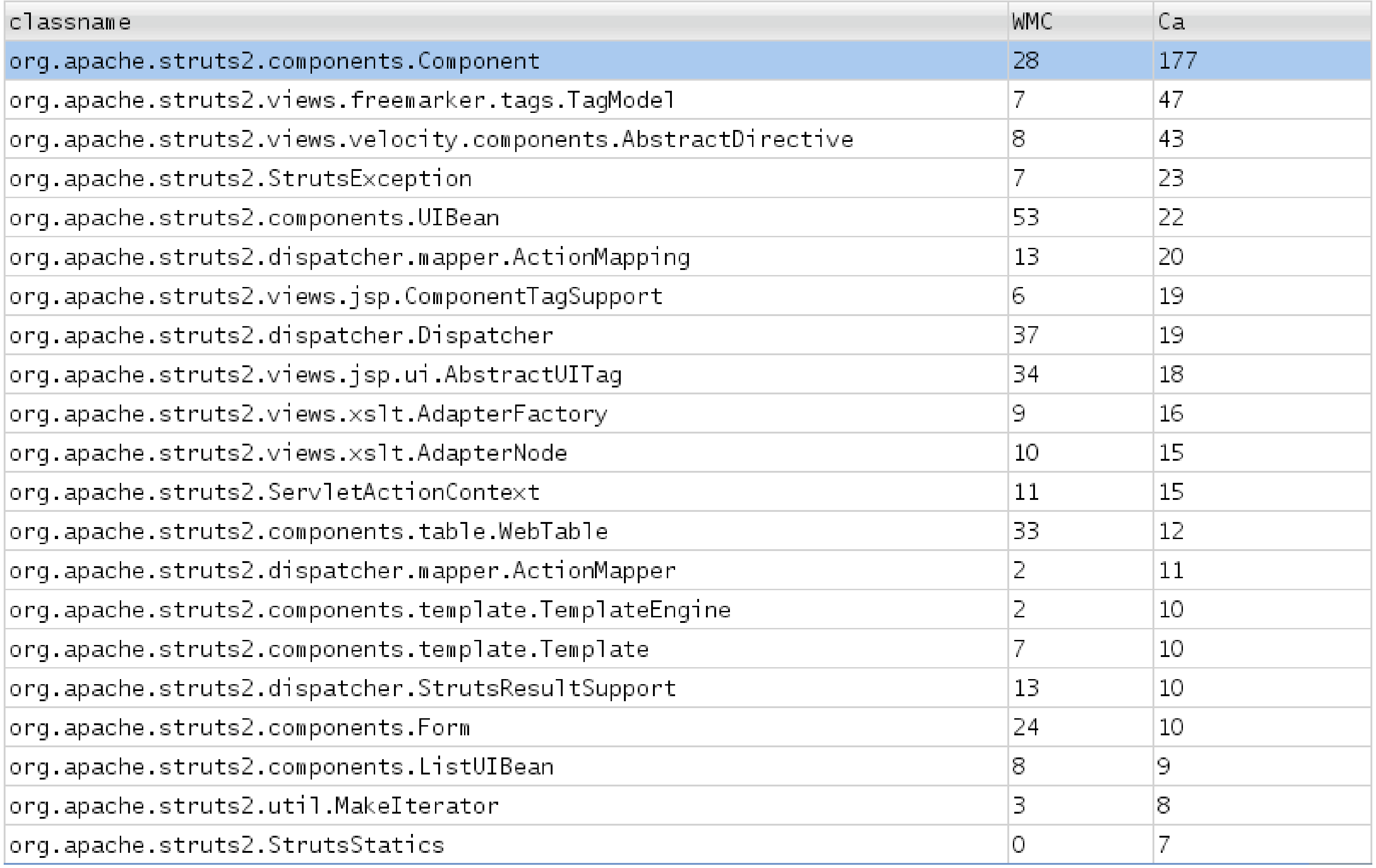
∑ of how many classes use this class

incoming calls

determines what is the “hard, crunchy center” of your code base

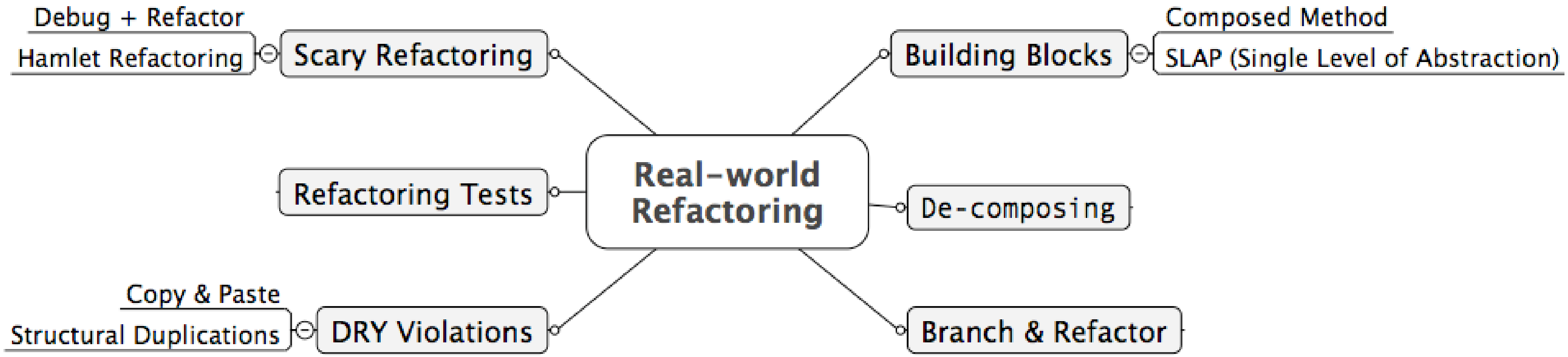
measure with CKJM, other metrics tools

struts 2.x



classname	WMC	Ca
org.apache.struts2.components.Component	28	177
org.apache.struts2.views.freemarker.tags.TagModel	7	47
org.apache.struts2.views.velocity.components.AbstractDirective	8	43
org.apache.struts2.StrutsException	7	23
org.apache.struts2.components.UIBean	53	22
org.apache.struts2.dispatcher.mapper.ActionMapping	13	20
org.apache.struts2.views.jsp.ComponentTagSupport	6	19
org.apache.struts2.dispatcher.Dispatcher	37	19
org.apache.struts2.views.jsp.ui.AbstractUITag	34	18
org.apache.struts2.views.xslt.AdapterFactory	9	16
org.apache.struts2.views.xslt.AdapterNode	10	15
org.apache.struts2.ServletActionContext	11	15
org.apache.struts2.components.table.WebTable	33	12
org.apache.struts2.dispatcher.mapper.ActionMapper	2	11
org.apache.struts2.components.template.TemplateEngine	2	10
org.apache.struts2.components.template.Template	7	10
org.apache.struts2.dispatcher.StrutsResultSupport	13	10
org.apache.struts2.components.Form	24	10
org.apache.struts2.components.ListUIBean	8	9
org.apache.struts2.util.MakeIterator	3	8
org.apache.struts2.StrutsStatics	0	7

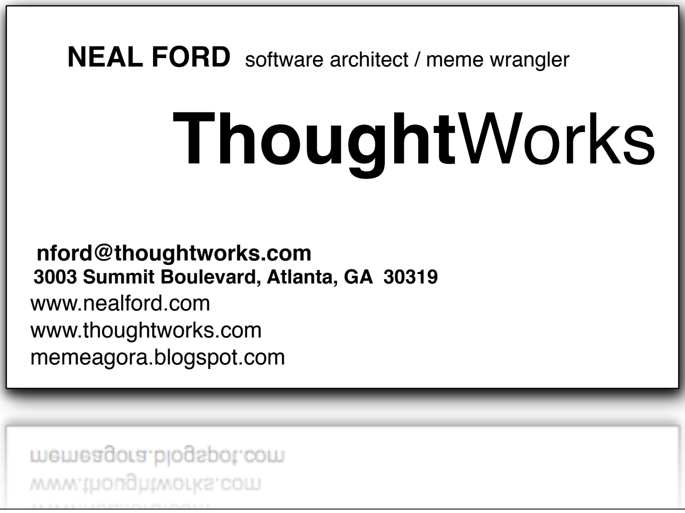
summary


Debug + Refactor
Hamlet Refactoring
Scary Refactoring
Building Blocks
Composed Method
SLAP (Single Level of Abstraction)
Refactoring Tests
Real-world Refactoring
De-composing
Copy & Paste
Structural Duplications
DRY Violations
Branch & Refactor

questions?

please fill out the session evaluations

slides & samples available at nealford.com



NEAL FORD software architect / meme wrangler

ThoughtWorks

nford@thoughtworks.com
3003 Summit Boulevard, Atlanta, GA 30319
www.nealford.com
www.thoughtworks.com
memeagora.blogspot.com

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 2.5 License.

http://creativecommons.org/licenses/by-nc-sa/2.5/

resources

Text

Text

Text

Text

Text