

# **Re-Engineering Compiled Java Code**

by Adrian Kuhn  
PhD Cand, University of Bern

# Context

- We have a Java application
- There is a feature request
- **But,** the sources are not available!

# Outline

- This talk presents four tools:
  - Jode, to decompile class files
  - Javassist, to modify class files
  - Moose/Hapax, to locate concepts
  - JOI, to browse objects at runtime

# Example: Epoquenet

- The application:
  - A client to search and browse documents on a central server.
- The feature request:
  - Save all selections, text as well as images, of the current document.

# Viewer

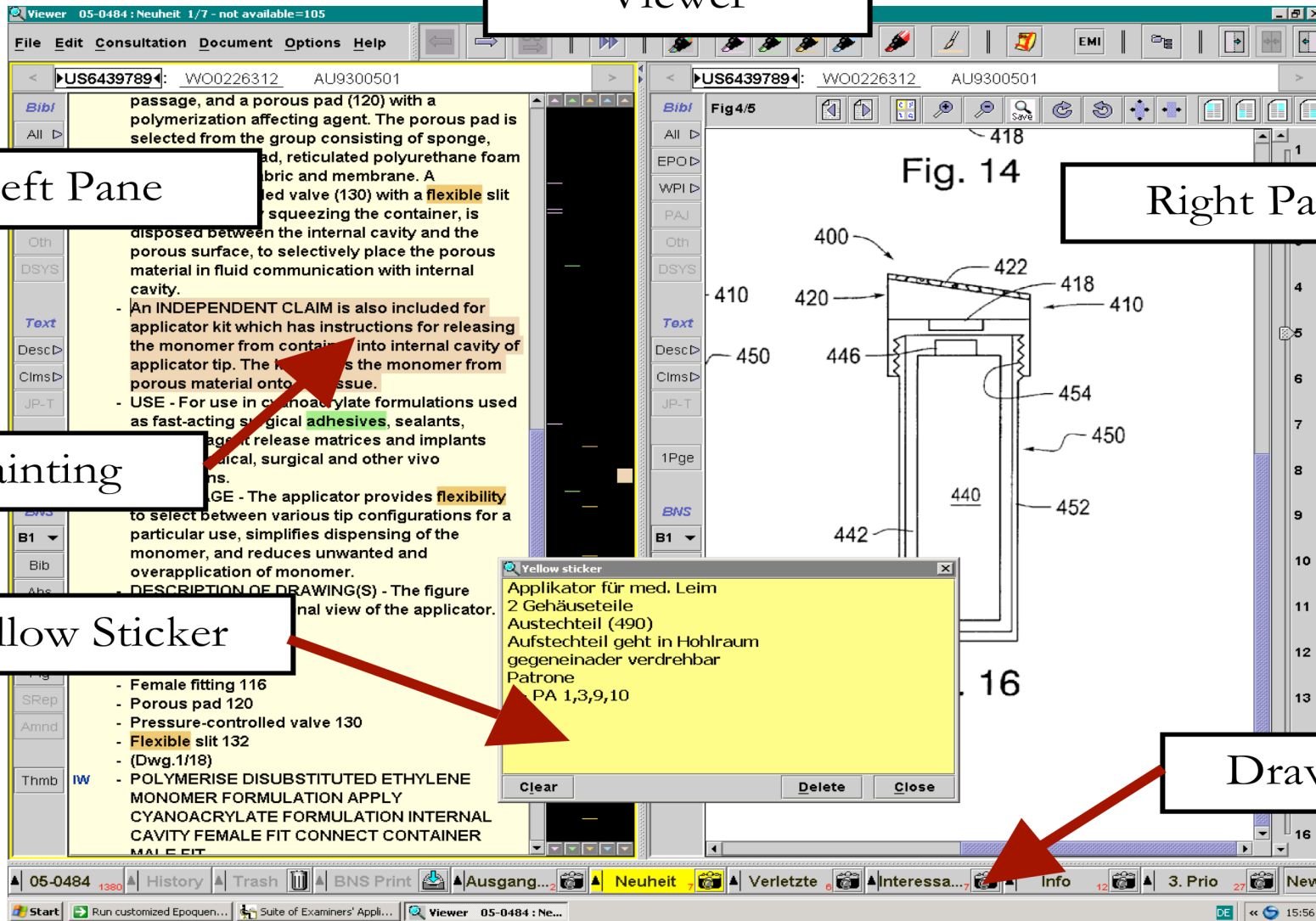
## Left Pane

## Painting

## Yellow Sticker

## Right Pane

## Drawer



# A first solution

- Decompile
- Change the source code
- Re-compile

→ [jode.sourceforge.net](http://jode.sourceforge.net)

# What's in a class file?

- The full qualified name of
  - All its methods and attributes
  - All types and all called methods
  - But no local variable names
- That is most of the identifiers

# However, this fails

- Decompilation works fine on toy examples written in Java 1.0
  - But fails using later Java versions due to all the syntactic sugar
  - Failed on 5% out of 1000 classes

# A better solution

- Do not decompile
- Do not re-compile
- Extend the compiled code

**But,** that's byte code - help!

# But, that's byte code!

- This is true
- However, you don't need any byte code knowledge at all
- There is Javassist

→ [www.jboss.org/products/javassist](http://www.jboss.org/products/javassist)

# Javassist

- Declare changes in Java syntax
- Apply them at byte code level
  - Add a new method
  - Replace a given method
  - Insert code into a method

# But first...

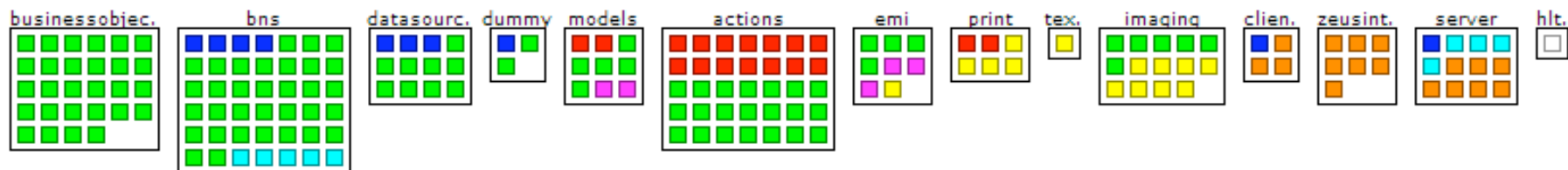
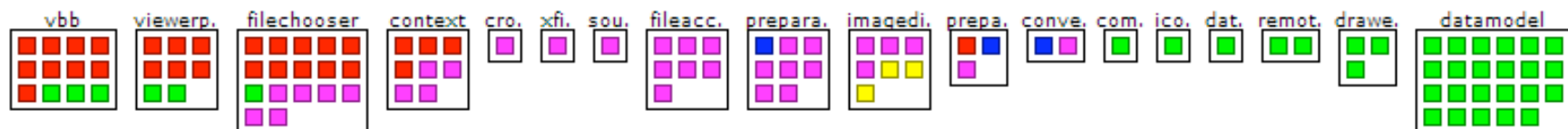
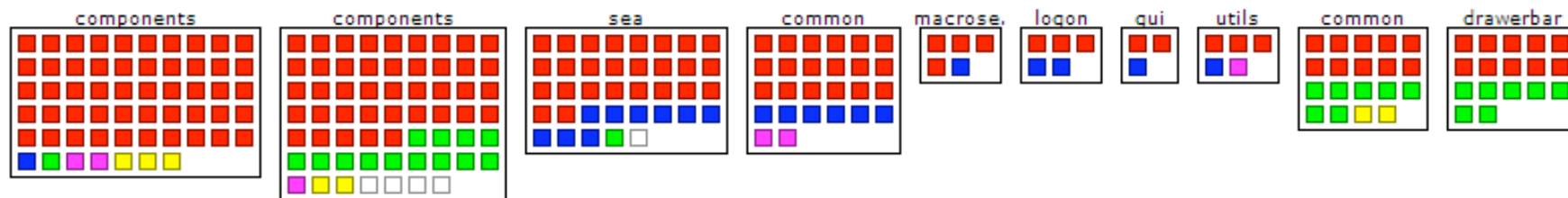
- Concept location
  - Where and what to change?
  - Find a “Point of Attack”.
- Inspect objects at runtime
  - Browse the data model.

# Concept Location

- Simple points of attack
  - GUI elements
  - Main methods
  - Test classes
- More complex
  - Semantic Clustering

# Semantic Clustering








- Based in these observations
  - Developers use meaningful names that convey domain concepts.
  - Classes that use similar identifier names belong to the same concept.
- Uses search engine technology.

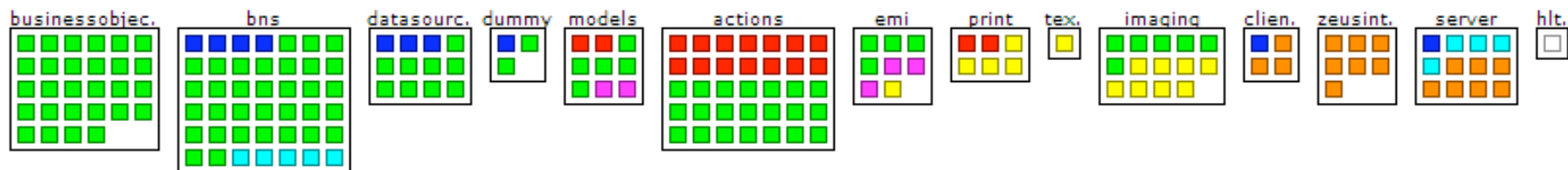
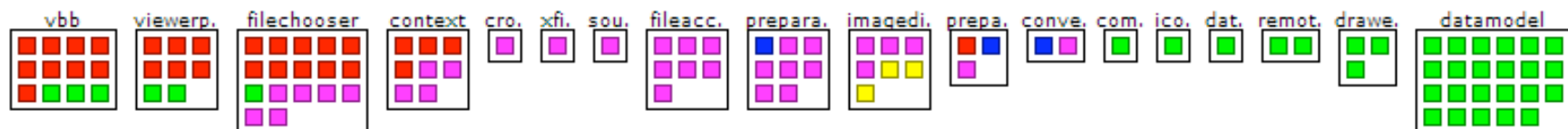
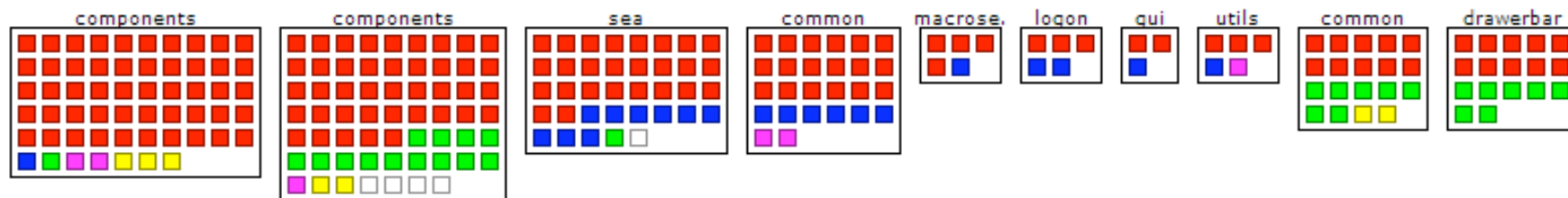


# What does this mean?

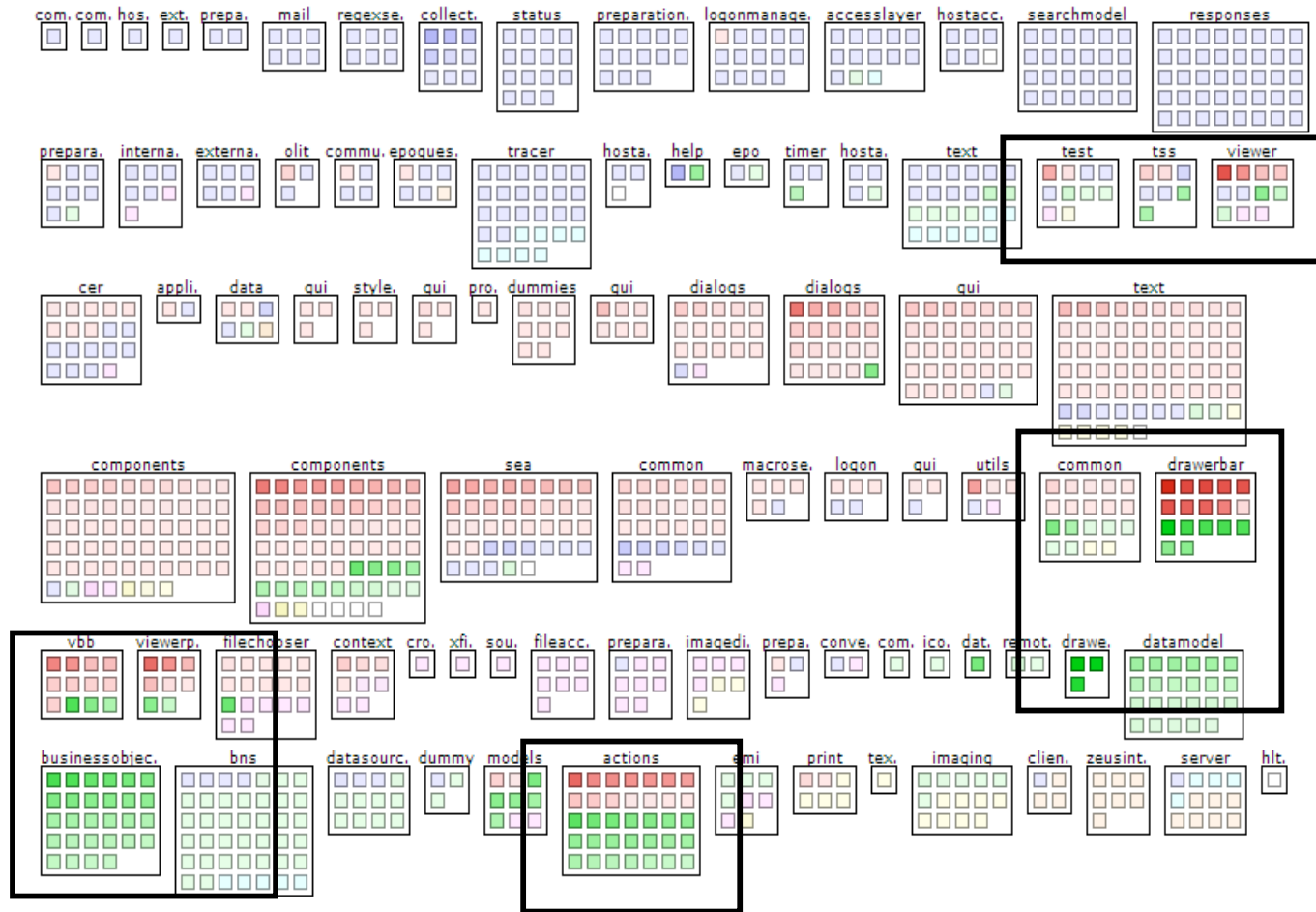
- Find most significant terms:
  - “show swing visible component gui update dispose empty perform javax”
  - Thus, concept red is the GUI layer
- Do this for each concept

# All Concepts

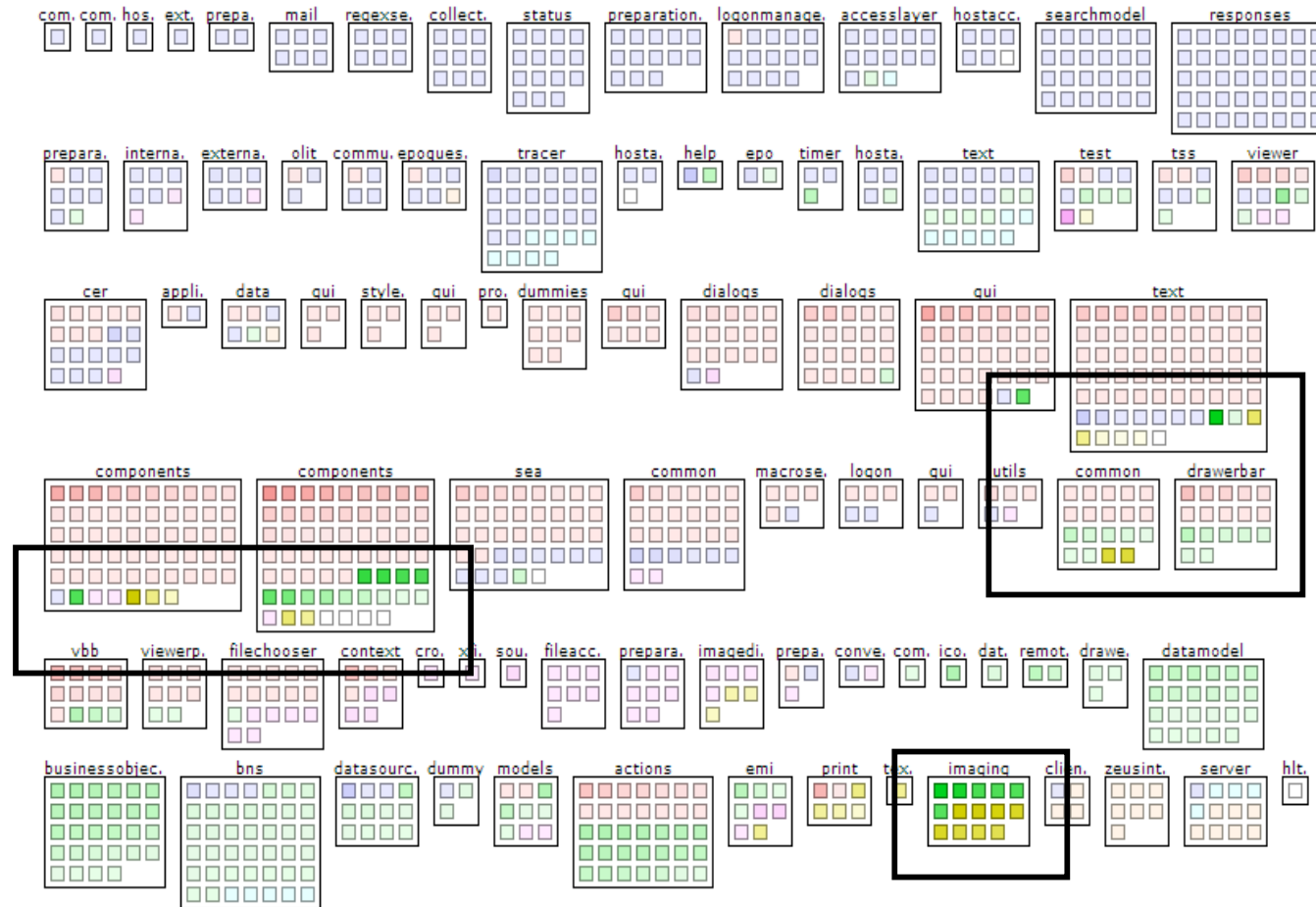
-  – Red, GUI layer
-  – Green, business model
-  – Blue, database layer
-  – Cyan, RMI remote access
-  – Orange, Zeus server access
-  – Magenta, file access
-  – Yellow, image processing



# Search: "drawer"



# “save cropped image”



# Semantic Clustering

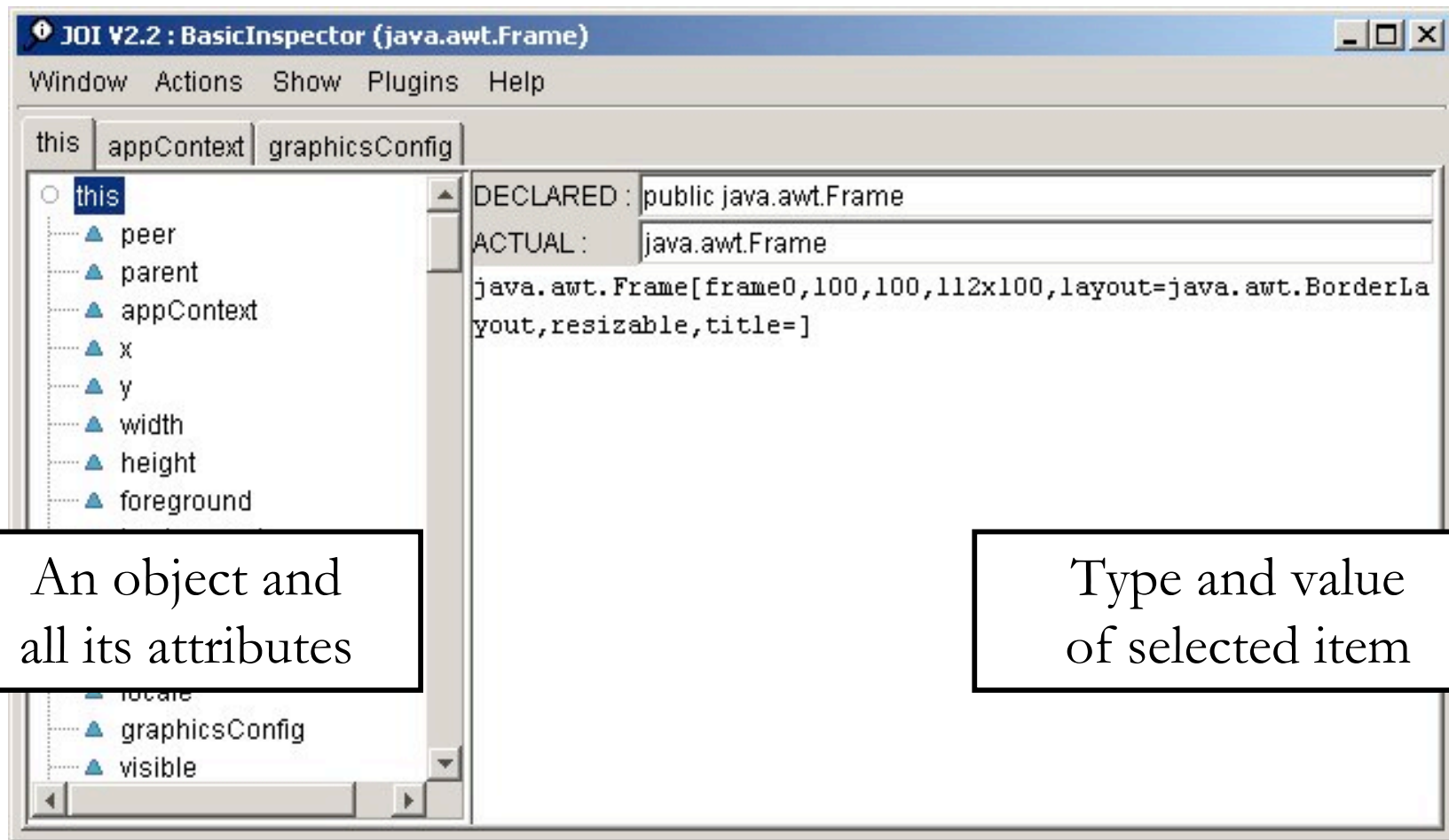
- Current research topic.
- Proof-of-concept implementation done as part of my masters thesis
- Eclipse plug-in is planned for summer 2006.

# Inspect objects at runtime

- Naïve approach
  - `System.out.println(anObject);`
- Clever approach
  - `Inspector.inspect(anObject);`

→ [www.programmers-friend.org/JOI](http://www.programmers-friend.org/JOI)

# Inspector window



# Append a menu item

```
ClassPool pool = ClassPool.getDefault();
 CtClass cclass = pool.get("...SnapshotDrawerPopupMenu" );
 CtMethod method = cclass.getMethod( ..signature.. );
method.insertAfter(
    "add(new javax.swing.JPopupMenu.Separator());"+
    "add(new InspectorAction(this.$0.getDrawer()));");
cclass.writeFile( "output/" );
```

- Get class and method
- Append code in Java syntax
- Write new class file

# Insert into class path

- If a class appears twice in the class path, the first one is used
- Therefore, insert new classes before the original jar in the path
  - Java -cp "patch.jar;original.jar"

# How classes are found

- Bootstrap classes
  - Java -Xbootclasspath "patch.jar"
  - The files rt.jar and i18n.jar
- Extension classes
  - All jars in jre/lib/ext/
- User classes
  - In order of appearance

# Build process

- First, compile new classes
- Then, patch original classes
- **But**, what if the new classes depend on the patches?

# Resolve dependencies

- New class A depends on a method added with patch B
- Patch B depends on new class A
- **Solution:** use an interface to access method B from class A
  - `((FooInterface) anObject).foo(args);`

# Example: add accessor

```
CtClass string = pool.get( "String" );  
CtMethod method = CtNewMethod.make(  
    "public char[] getCharArray() {" +  
    "    return value;}" , point );  
string.addMethod(method);  
string.writeFile( "output/" );
```

- Create new method
- Add method to class

# Example: make public

```
CtClass cclass = pool.get( "SomeClass" );  
CtMethod method = cclass.getMethod( ..signature.. );  
method.setModifiers(Modifier.setPublic(  
    method.getModifiers()));  
cclass.writeFile( "output/" );
```

- Get class and method
- Change modifier to public

# Example: catch exception

```
CtClass cclass = pool.get( "SomeClass" );  
CtMethod method = cclass.getMethod( ..signature.. );  
CtClass etype = pool.get( "java.io.IOException" );  
method.addCatch( "$e.printStackTrace();", etype );  
cclass.writeFile( "output/" );
```

- Get class and method
- Get exception type
- Add catch block to method
  - Access exception as \$e

# Tricky: get outer class

```
CtClass cclass = pool.get( "SomeClass$Inner" );  
CtMethod method = CtNewMethod.make(  
    "public Object getOuterClass() {" +  
    "    return this$0;}", cclass );  
cclass.addMethod(method);  
cclass.writeFile( "output/" );
```

- Get method in inner class
- Add new method
  - Access outer class as this\$0

# Tricky: recursive calls

```
CtMethod foo = CtNewMethod.make( "public abstract int foo(int n)",  
cc );  
CtMethod bar = CtNewMethod.make( "public abstract int bar(int  
n)", cc);  
cc.addMethod(foo);  
cc.addMethod(bar);  
foo.setBody( "{ return bar($1); }" );  
bar.setBody( "{ return foo($1); }" );  
cc.setModifiers( cc.getModifiers() & ~Modifier.ABSTRACT );
```

- First, add abstract methods
- Then, add method bodies
  - Access arguments as \$1..\$n
- Remove abstract flag

# Expert: search & replace

```
CtMethod method = cclass.getMethod( ..signature.. );
method.instrument( new ExprEditor() {
    public void edit( MethodCall call ) {
        if ( call.getClassName().equals( "Point" )
            && call.getMethodName().equals( "move" )) {
            call.replace( "{ $1 = 0; $_ = $process($$);
        }" );
    }
});
cclass.writeFile( "output/" );
```

- Get class and method
- Replace all `move(x,y)` with `move(0,y)`
  - Use meta variables such as `$_` or `$$`

# Summary

- Decompilation fails
- Better: patch compiled class files
- Use concept location
- Inspect, do not `System.out.println`

# Questions?